Randomness and Noise in Information Systems

Thesis by

Hongchao Zhou

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2013

(Defended June 1 2012)

© 2013

Hongchao Zhou

All Rights Reserved

To my maternal grandmother Zhenquan Hao.

Acknowledgments

I am deeply grateful to my advisor Professor Jehoshua (Shuki) Bruck. Since September 11, 2008, when I walked into Shuki's office for the first time, he has deeply influenced me in different aspects of my life, not only about his great guidance in research, but also his kindness, his encouragement, and his genuine concern for students. Shuki has introduced me to a variety of areas, taught me how to create important questions and how to appreciate the beauty and simplicity in real research. He has also devoted a great deal of time and energy to my personal growth. I feel extremely fortunate of being one of his students, as an old Chinese saying says: "The world has Bole (Bole was a legendary figure in the seventh century B.C. China, who was an authority on horses) then to have the splendid steed. The splendid steed is common, but Bole is not common."

My sincere gratitude goes to Professor Erik Winfree, Professor Chris Umans, Professor Babak Hassibi, and Professor Michelle Effros for being on my Ph.D. Defense and Candidacy Committees, and providing me their insight and suggestions on my research. In additional help and support, I must thank Professor Anxiao (Andrew) Jiang, Professor Erik Winfree, Professor Chris Umans, and Professor Xiaohong Guan.

I am grateful to my intelligent colleagues in the Paradise group. Special gratitude goes to some current and past group members for fruitful collaborations and intriguing discussions. I would like to thank Professor Anxiao (Andrew) Jiang, Zhiying Wang, Eyal En gad, Itzhak Tamo, Dr. Robert Mateescu, and Dr. Eitan Yaakobi, whom I worked with on coding for storage; Professor Marc Riedel, Professor Holin Chen, Professor Weikang Qian, Po-ling Loh, and David Lee, whom I worked with on stochastic system synthesis; Dan Wilhelm, and Dr. Lulu Qian, whom I worked with in the Molecular Programming Project. I was very fortunate to work with these smart people, and enjoyed

all the stimulating discussions and joyful moments that we shared.

Furthermore, I would like to thank all those who helped me, including my friends and nice teachers on my way of growing up; without them I would not be where I am today. Special thanks go to my friends at Caltech. They have kept me company, made my life colorful, and gave me many unforgettable memories. At the same time, I wish to acknowledge an incomplete list of my friends: Maolin Ci, Tong Chen, Cheng-hao (Simon) Chien, Na Hu, Na Li, Wendian Shi, Molei Tao, Huan Yuang, Sinan Zhao, Yu Zhao, etc.

Finally, I would like to give my deepest gratitude to my maternal grandparents Hexing Wang and Zhenquan Hao for their endless love and their cultivation for my childhood. They are traditional Chinese farmers, not well educated, but their simplicity and kindness have deeply influenced me. I am also indebted to my parents for their understanding and support. They have worked hard to provide me with my education. I dedicate this thesis to my family as an inadequate appreciation of everything that they have done for me.

Abstract

This dissertation is devoted to the study of randomness and noise in a number of information systems including computation systems, storage systems, and natural paradigms like molecular systems, where randomness plays important and distinct roles. Motivated by applications in engineering and science we address a number of theoretical research questions.

- In a computation system, randomness enables to perform tasks faster, simpler, or more space efficient. Hence, randomness is a useful computational resource, and the research question we address is: How to *efficiently extract randomness* from natural sources?

- In a molecular system such as a chemical reaction network or a gene regulatory network, randomness is inherent and serves as the key mechanism for producing the desired quantities of molecular species. A chemical reaction can be abstractly described as a probabilistic switch. Hence, given a set of probabilistic switches (with some fixed switching probabilities), the research question we address is: How to *synthesize a stochastic network* consisting of those switches that *computes a pre-specified probability distribution?*

- In an information storage system, like flash memories where information is represented by a relatively small number of electrons, randomness is a threat to data reliability. Hence, the research question we address is: How to *represent, write and read information in the presence of randomness (noise)*?

This dissertation is focusing on the foregoing key questions and describes novel contributions related to *randomness generation and extraction, stochastic system synthesis and coding for information storage.*

The dissertation is organized in four parts. In part I, we study the classical problem of efficient *generation of perfect random bits from an ideal source.* We first focus on the simple source model, namely, an i.i.d. source, and derive a universal scheme for transforming an arbitrary algorithm for binary sources to one that manages general source of a larger alphabet, and hence enable the application of existing algorithms to general sources. We then address the long-standing open problem related to Blum's beautiful algorithm (1986) for generating random bits from Markov chains and propose the first known optimal algorithm that generates random bits from an arbitrary Markov chain in expected linear operation time. Finally, we propose an optimal streaming algorithm for generation of random bits, it transforms an input stream into a stream of random bits. Compared to existing methods, our algorithm is currently the best choice for implementation in practical systems.

In part II, we study *randomness extraction from non-ideal sources.* Instead of generating perfect random bits, our goal is to generate random bits that are ϵ-close to perfect random bits. We show that linear transformations based on sparse random matrices are very powerful for extracting randomness from a variety of weak random sources; the simplicity of this method has high potential for enabling applications in high-speed random number generation. We then study the problem of extracting a prescribed number of random bits by reading the smallest possible number of symbols from imperfect stochastic processes. Although fixed-length extractors such as seeded extractors have been well studied, their information efficiency is far from optimal. We introduce the concept of *variable-length extractors* and prove that they preform closely to the optimal entropy limit.

In part III, motivated by DNA-based molecular systems, we discuss research problems related to *stochastic computation.* One fundamental question we study is related to the physical *synthesis of stochasticity.* In particular, we consider stochastic switching circuits as a simple generalization of traditional switching circuits, where deterministic switches are replaced by probabilistic switches. We study the robustness of stochastic switching circuits, and present several methods for synthesizing or approximating probabilities. We then propose a new model for stochastic computation called stochastic flow networks - those are directed graphs with incoming edges and outgoing edges where tokens enter through the input edges, travel stochastically in the network, and exit the net-

work through the output edges. We show that when each element has two outgoing edges and is unbiased, an arbitrary rational probability can be realized by a stochastic flow network of optimal size. In addition, we demonstrate that feedback greatly improves the expressibility of stochastic flow networks.

In part IV, we study *coding for information storage.* This topic is becoming increasingly important due to the introduction of new storage technologies such as flash and phase-change memories. We begin by considering the binary asymmetric channel and introduce the concept of nonuniform codes. Our main observation is that asymmetric errors are content dependent, however, in information storage applications, the reliability should be content independent; hence, we introduce the new concept of nonuniform codes, whose codewords can tolerate different numbers of asymmetric errors depending on their Hamming weights. To further increase the capacities of storage devices, we combine channel modulation with code construction - we introduce a simple and practical write/read scheme for nonvolatile memories, called balanced modulation. Finally, we propose a novel systematic error-correcting code for rank modulation where the errors are characterized by the Kendall τ-distance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Maybe the brain uses random elements; maybe the universe does too.

– *Things a Computer Scientist Rarely Talks About*, Donald E. Knuth (1999)

1.1 Randomness and Noise

"In World War II, Airplane bombers used mechanical computers to perform navigation and bomb trajectory calculations. Curiously, these computers (boxes filled with hundreds of gears and cogs) performed more accurately when flying on board the aircraft, and less well on ground. Engineers realized that the vibration from the aircraft reduced the error from sticky moving parts. Instead of moving in short jerks, they moved more continuously." This is one example from Principles of Digital Audio [91], showing that external randomness, namely noise, increases the accuracy of an information system.

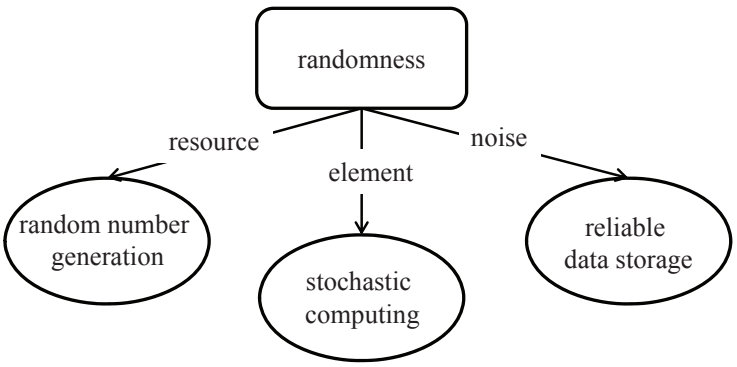


Figure 1.1. The roles of randomness in different information systems.

Randomness plays an important role in a number of information systems, including computation, communications and storage as well as in natural paradigms like molecular systems and social networks. Randomness makes it possible to perform tasks that are hard to complete deterministically, or making some tasks computationally faster, simpler, or more space efficient. Examples [86] of such tasks include generating prime numbers, polynomial factoring, permanent approximation and volume approximation. On the other hand, randomness can be treated as noise, which is a foe of reliable storage and fault-tolerant computing. Manipulating, utilizing and controlling this double-edged sword is the key motivation for this dissertation. We study randomness in a number of information systems; specifically, (1) the *generation* of randomness in computation systems, (2) the *synthesis* of randomness in molecular systems, and (3) the *elimination* of randomness (noise) in storage systems. In this dissertation we study and develop algorithms, approaches, and schemes to process and manage randomness in those information systems (see figure 1.1).

1.2 Randomness in Computation Systems

There are countless applications of randomness in computation systems, such as randomized algorithms [86] (like Monte Carlo method), network coding, compressive sensing, cryptography, machine learning, intelligent systems, and optimization. Most such applications expect to receive "truly random bits" as the input. Although pseudorandom number generation algorithms have been extensively studied [13, 47, 56, 124], they do not provide a sufficient level of security when applied in cryptography (see the cover article of IEEE spectrum, September 2011, entitled "Behind Intel's new random-number generator" [113]). For most applications, it cannot be proved that pseudorandom bits can perfectly simulate truly random bits. In addition, the process of pseudorandom number generation requires truly random bits as a seed, namely, it uses a sequence of truly random bits and stretches it to produce a long sequence bearing an appearance of randomness. This serves as the motivation for the study of generating or extracting truly random bits from natural physical sources. Examples of such sources include radioactive decay, quantum-effects in semiconductor devices, thermal noise, shot noise, avalanche noise in Zener diodes, clock drift, magnetic disk timing,

and radio noise.

The problem of random number generation dates back to von Neumann [128] in 1951 who first considered the problem of simulating an unbiased coin by using a biased coin with unknown probability. He observed that when one focuses on a pair of coin tosses, the events HT and TH have the same probability (H is for "head" and T is for "tail"); hence, HT produces the output symbol 0 and TH produces the output symbol 1. The other two possible events, namely, HH and TT, are ignored and they do not produce any output symbols. Generally, given an arbitrary biased coin or an arbitrary Markov chain, one can generate a sequence of truly random bits. Although this is a well studied area, a number of fundamental problems remain unanswered. In the first part of this dissertation, we will address some of those problems. Our contributions include a universal scheme for transforming an arbitrary algorithm for binary sources to manage the general source of an m-sided die (chapter 2), the first-known optimal algorithm that generates random bits from an arbitrary finite-state Markov chain (chapter 3), and an optimal algorithm that generates random-bit streams from an arbitrary biased coin (chapter 4).

The reality is that some (likely, most) physical sources are neither perfect biased coins nor perfect Markov chains. In this case, one cannot generate perfect random bits. Instead, people derive algorithms to generate a sequence that is arbitrarily close to perfect random bits, namely, it can be used to replace the sequence of perfect random bits in any randomized application such that the additional error probability of the application is upper bounded by a small constant ϵ. We call this process randomness extraction, and we call such an algorithm an extractor. For some types of random sources [63], like independent sources, bit-fixing sources, and small-space sources, one can derive an extractor deterministically to extract randomness, and we call it a seedless extractor. For a more general source, namely a k-source, in which each possible sequence has probability at most 2^{-k} of being generated, it was shown that it is impossible to derive a single function that extracts even a single random bit. This observation led to the introduction of seeded extractors, which use a small number of truly random bits as the seed (catalyst). When simulating a probabilistic algorithm, one can simply eliminate the requirement of truly random bits by enumerating all the possible strings

for the seed and taking a majority vote. We noticed that the existing contributions on randomness extractors and random number generators are distinct. We introduced the concept of "variable-length extractors" and created a conceptual bridge between them. Variable-length extractors can achieve efficiency close to Shannon's limit (chapter 6). In addition to efficiency, we also consider the simplicity of the extractors; hence, we study linear constructions of extractors, especially those based on sparse random matrices (chapter 5), which have potentially important applications in high-speed random number generators.

1.3 Randomness in Molecular Systems

Randomness is inherent in biology: Within a living cell, the number of molecules involved in a specific regulatory process is usually small. Hence, the analysis of the reactions can not be based on averages and is typically assuming that the collisions between molecules are random [38]. Another example is synapses in neural systems, where signals are generated and transmitted in a stochastic way [53]. As engineers, we treat biology as an integrated system that behaves randomly. For example, insects in flight tend to move about with random changes in direction [66], so that their trajectories are hard to predict by pursuing predators. Generally, randomness is pivotal to biology for increasing diversity, enhancing robustness, eluding enemies, and creating intelligence. Randomness is one of the most beautiful and mysterious parts of nature. As Donald E. Knuth said in 1999 [70]:

"*I tend to believe that recently proposed models of the brain, which are based on the idea of continuous dynamic evolution of symbolic signals instead of on processes that resemble today's computing machines, have good prospects for helping to explain the mysteries of consciousness. If so, a lot of* randomness *must be involved in that .... "*

Motivated by biology, we study the synthesis of systems that can process randomness and can be easily implemented by molecular reactions. A concrete question is how to compose existing probabilistic elements to produce any other target distribution. The study of this question is an initial step that potentially can lead to stochastic computing or molecular computing, where computation is performed on distributions of random variables.

We study the problem of stochastic system synthesis. Continuing Wilhelm and Bruck's work [134], we study the robustness of stochastic switching circuits and present a number of general methods for synthesizing or approximating desired probabilities (chapter 7). Then, we introduce a new framework, called stochastic flow networks, which is more computationally powerful than any of existing models for probability synthesis (chapter 8).

1.4 Randomness in Storage Systems

The domain of information storage is becoming increasingly important due to the rapid growth of global data, the development of new storage devices, and the emergence of new services like cloud computing. In this domain, the dissertation focuses on reliability and coding of nonvolatile memories including flash memories, which are currently the most widely used family of nonvolatile memories, as well as emerging nonvolatile memory technologies such as phase-change memories.

In storage systems, we treat randomness as noise. Reading information from physical devices (like memories) is a the dual process to extracting randomness, in the sense that it removes randomness (noise) from the source. Developing data protection schemes is important in nonvolatile memories, in which stored data can be lost due to many mechanisms, including cell-level drift (like charge leakage in Flash memories), heterogeneity, programming noise, write disturbance and read disturbance. These mechanisms make the errors in nonvolatile memories heterogeneous, asymmetric, time dependent, and unpredictable. Hence the development of simple, reliable, and efficient reading/writing schemes is a timely research challenge.

In the fourth part of this dissertation, we will adapt the design of error-correcting schemes to the requirement of data storage and the physical properties of storage devices and introduce some new types of codes and practical writing/reading schemes for storage systems; including, nonuniform codes for data storage (chapter 9), balanced modulation for nonvolatile memories (chapter 10), and systematic rank modulation codes (chapter 11).

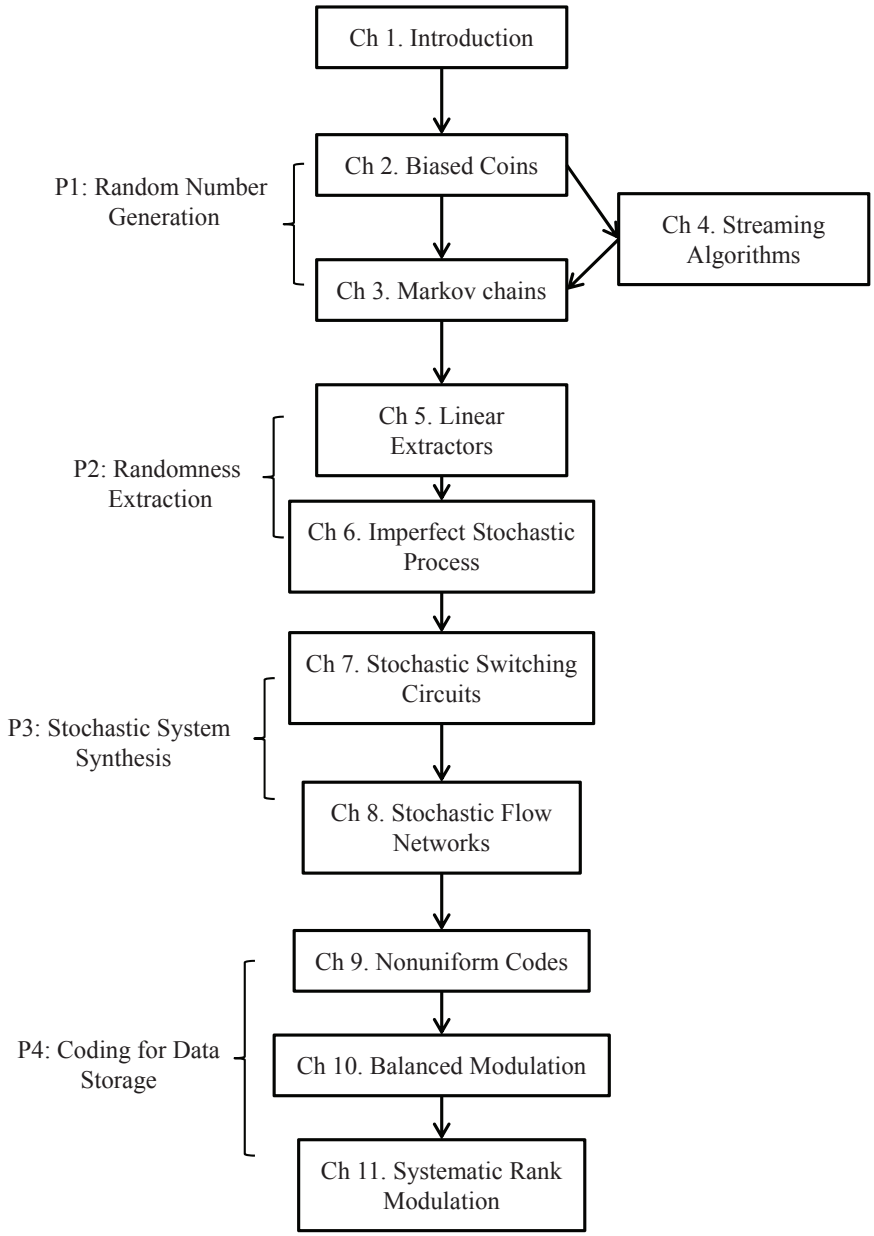


Figure 1.2. The structure of this thesis.

1.5 Structure and Contributions of the Thesis

The scope of this dissertation is to study the generation, extraction, synthesis and elimination of randomness in information systems, including computation systems, molecular systems and storage systems. The structure of this dissertation is shown in figure 1.2. The contribution of each chapter is listed bellows. All of the chapters can be read separately according to the readers' interests and

backgrounds.

Part I: Random Number Generation (Chapters 2, 3, 4)

Chapter 2 focuses on the problem of generating random bits from biased coins or biased dice and derives a universal scheme for transforming an arbitrary algorithm for 2-faced coins to generate random bits from the general source of an m-sided die, hence enabling the application of existing algorithms to general sources.

Chapter 3 studies the problem of efficiently generating random bits from Markov chains and provides the first known algorithm that generates unbiased random bits from an arbitrary finite Markov chain, operates in expected linear time and achieves the information-theoretic upper bound on efficiency.

Chapter 4 introduces an algorithm that generates random bit streams from biased coins, uses bounded space and runs in expected linear time. As the size of the allotted space increases, the algorithm approaches the information-theoretic upper bound on efficiency.

Part II: Randomness Extraction (Chapters 5, 6)

Chapter 5 studies linear transformations for randomness extraction and shows that sparse random matrices are very powerful for extracting randomness from many noisy sources, which are very attractive in the practical use of high-speed random number generators due to their simplicity.

Chapter 6 studies the problem of extracting a prescribed number of random bits by reading the smallest possible number of symbols from imperfect stochastic processes. A new class of extractors called variable-length extractors is introduced, they achieve efficiency near Shannon's (optimal) limit.

Part III: Stochastic System Synthesis (Chapters 7, 8)

Chapter 7 studies stochastic switching circuits, which are relay circuits that consist of stochastic switches called pswitches. It introduces new properties of stochastic switching circuits, including robustness, expressibility, and probability approximation.

Chapter 8 designs optimal-sized stochastic flow networks for "synthesizing" target distributions. It shows that when each splitter (basic probabilistic element) has probability 1/2, an arbitrary rational probability $\frac{a}{b}$ with $a \leq b \leq 2^n$ can be realized by a stochastic flow network of size n, and

its size is optimal.

Part IV: Coding for Data Storage (Chapters 9, 10, 11)

Chapter 9 introduces a new type of code called a nonuniform code, whose codewords can tolerate different numbers of asymmetric errors depending on their Hamming weights. The goal of nonuniform codes is to guarantee the reliability of every codeword while maximizing the code size for correcting asymmetric errors.

Chapter 10 presents a practical writing/reading scheme in nonvolatile memories, called balanced modulation, for minimizing the asymmetric component of errors. The main idea is to encode data using a balanced error-correcting code. When reading information from a block, it adjusts the reading threshold such that the resulting word is also balanced or approximately balanced. Balanced modulation has suboptimal performance for any cell-level distribution and it can be easily implemented in the current systems of nonvolatile memories.

Chapter 11 explores systematic error-correcting codes for rank modulation while considering the Kendall τ-distance. It presents $(k+2;k)$ systematic codes for correcting a single error, and proves that systematic codes for rank modulation can achieve the same capacity as general error-correcting codes.

Part I

Random Number Generation

Chapter 2

Random Number Generation from Biased Coins and Dice

This chapter focuses on the problem of generating random bits from biased coins or loaded dice and derives a universal scheme for transforming an arbitrary algorithm for 2-faced coins to generate random bits from the general source of an m-sided die, hence enabling the application of existing algorithms to general sources.

2.1 Introduction

In this chapter, we study the problem of random number generation from i.i.d. sources, which is the most fundamental and important source model. Many real sources can be well approximated by this model, and the algorithms developed based on this model can be further generalized in generating random bits from more sophisticated models, like Markov chains [138], or more generally, approximately stationary ergodic processes [143].

The problem of random number generation dates back to von Neumann [128] in 1951 who considered the problem of simulating an unbiased coin by using a biased coin with unknown probability. He observed that when one focuses on a pair of coin tosses, the events HT and TH have the same probability (H is for 'head' and T is for 'tail'); hence, HT produces the output symbol 1 and TH produces the output symbol 0. The other two possible events, namely, HH and TT, are ignored, namely, they do not produce any output symbols. More efficient algorithms for generating random bits from a biased coin were proposed by Hoeffding and Simons [54], Elias [33], Stout and War-

ren [109] and Peres [90]. Elias [33] was the first to devise an optimal procedure in terms of the information efficiency, namely, the expected number of unbiased random bits generated per coin toss is asymptotically equal to the entropy of the biased coin. In addition, Knuth and Yao [71] presented a simple procedure for generating sequences with arbitrary probability distributions from an unbiased coin (the probability of H and T is $\frac{1}{2}$). Han and Hoshi [52] generalized this approach and considered the case where the given coin has an arbitrary known bias.

In this chapter, we consider the problem of generating random bits from a loaded die as a natural generalization of generating random bits from a biased coin. There is some related work: In [30], Dijkstra considered the opposite question and showed how to use a biased coin to simulate a fair die. In [61], Juels et al. studied the problem of simulating random bits from loaded dice, and their algorithm can be treated as the national generalization of Elias's algorithm. However, for a number of known beautiful algorithms, like Peres's algorithm [90], we still do not know how to generalize them for larger alphabets (loaded dice).

In addition, we notice that most existing works for biased coins take a fixed number of coin tosses as the input and they generate a variable number of random bits. In some occasions, the opposite question seems more reasonable and useful: given a biased coin, how to generate a prescribed number of random bits with as a few as possible coin tosses? Hence, we want to create a function f that maps the sequences in a dictionary $\mathcal{D}$, whose lengthes may be different, to binary sequences of the same length. This dictionary $\mathcal{D}$ is complete and prefix-free. That means for any infinite sequence, it has exactly one prefix in the dictionary. To generate random bits, we read symbols from the source until the current input sequence matches one in the dictionary.

For completeness, in this chapter, we first present some of the existing algorithms that generate random bits from an arbitrary biased coin in section 2.2, including the von Neumann Scheme, Elias algorithm and Peres algorithm. Then in section 2.3, we present a universal scheme for transforming an arbitrary algorithm for 2-faced coins to generate random bits from the general source of an m-sided die, hence enabling the application of existing algorithms to general sources. In section 2.4, we study approaches of efficiently generating a required number of random bits from an arbitrary

biased coin and achieving the information-theoretic upper bound on efficiency. Finally, we provide the concluding remarks in section 2.5.

2.2 Existing Algorithms for Biased Coins

2.2.1 Von Neumann Scheme

In 1951, von Neumann [128] considered the problem of random number generation from biased coins and described a simple procedure for generating an independent unbiased binary sequence $z_1 z_2 ...$ from an input sequence $X = x_1 x_2 ....$ His original procedure is described as follows: For an input sequence, we divide all the bits into pairs $x_1 x_2, x_3 x_4, ...$ and apply the following mapping to each pair

$$\text{HT} \to 1, \quad \text{TH} \to 0, \quad \text{HH} \to \phi, \quad \text{TT} \to \phi,$$

where ϕ denotes the empty sequence. By concatenating the outputs of all the pairs, we can get a binary sequence, which is independent and unbiased. The von Neumann scheme is computationally (very) fast, however, its information efficiency is far from being optimal. Here, the information efficiency is defined by the expected number of random bits generated per input symbol. Let p_1, p_2 with $p_1 + p_2 = 1$ be the probabilities of getting H and T, then the probability for a pair of input bits to generate one output bit (not a ϕ) is $2p_1 p_2$, hence the information efficiency is $\frac{2p_1 p_2}{2} = p_1 p_2$, which is $\frac{1}{4}$ at $p_1 = p_2 = \frac{1}{2}$ and less elsewhere.

2.2.2 Elias Algorithm

In 1972, Elias [33] proposed an optimal (in terms of information efficiency) algorithm as a generalization of the von Neumann scheme.

Elias's method is based on the following idea: The possible 2^n binary input sequences of length n can be partitioned into classes such that all the sequences in the same class have the same number of H's and T's. Note that for every class, the members of the class have the same probability to be

generated. For example, let $n = 4$, we can divide the possible $2^n = 16$ input sequences into 5 classes:

$$S_0 = \{\text{HHHH}\},$$

$$S_1 = \{\text{HHHT}, \text{HHTH}, \text{HTHH}, \text{THHH}\},$$

$$S_2 = \{\text{HHTT}, \text{HTHT}, \text{HTTH}, \text{THHT}, \text{THTH}, \text{TTHH}\},$$

$$S_3 = \{\text{HTTT}, \text{THTT}, \text{TTHT}, \text{TTTH}\},$$

$$S_4 = \{\text{TTTT}\}.$$

Now, our goal is to assign a string of bits (the output) to each possible input sequence, such that any two possible output sequences Y and Y' with the same length (say k), have the same probability to be generated, which is $\frac{c_k}{2^k}$ for some $0 \leq c_k \leq 1$. The idea is that for any given class we partition the members of the class to sets of sizes that are a power of 2, for a set with 2^i members (for some i) we assign binary strings of length i. Note that when the class size is odd we have to exclude one member of this class. We now demonstrate the idea by continuing the example above.

In the example above, we cannot assign any bits to the sequence in S_0, so if the input sequence is HHHH, the output sequence should be ϕ (denoting the empty sequence). There are 4 sequences in S_1 and we assign the binary strings as follows:

$$\text{HHHT} \to 00, \quad \text{HHTH} \to 01,$$

$$\text{HTHH} \to 10, \quad \text{THHH} \to 11.$$

Similarly, for S_2, there are 6 sequences that can be divided into a set of 4 and a set of 2:

$$\text{HHTT} \to 00, \quad \text{HTHT} \to 01,$$

$$\text{HTTH} \to 10, \quad \text{THHT} \to 11,$$

$$\text{THTH} \to 0, \quad \text{TTHH} \to 1.$$

In general, for a class with W members that were not assigned yet, assign 2^j possible output binary sequences of length j to 2^j distinct unassigned members, where $2^j \leq W < 2^{j+1}$. Repeat the procedure above for the rest of the members that were not assigned. When a class has an odd number of members, there will be one and only one member assigned to ϕ.

Given a binary input sequence X of length n, using the method above, the output sequence can be written as a function of X, denoted by $\Psi_E(X)$, called the Elias function. In [99], Ryabko and Matchikina showed that the Elias function of an input sequence of length n (that is generated by a biased coin with two faces) is computable in $O(n \log^3 n \log\log(n))$ time.

2.2.3 Peres Algorithm

In 1992, Peres [90] demonstrated that iterating the original von Neumann scheme on the discarded information can asymptotically achieve optimal information efficiency. Let us define the function related to the von Neumann scheme as $\Psi_1 : \{\mathrm{H}, \mathrm{T}\}^* \rightarrow \{0,1\}^*$. Then the iterated procedures Ψ_v with $v \geq 2$ are defined inductively. Given an input sequence $x_1 x_2 ... x_{2m}$, let $i_1 < i_2 < ... < i_k$ denote all the integers $i \leq m$ for which $x_{2i} = x_{2i-1}$, then Ψ_v is defined as

$$\begin{aligned} & \Psi_v(x_1, x_2, ..., x_{2m}) \\ = \; & \Psi_1(x_1, x_2, ..., x_{2m}) * \Psi_{v-1}(x_1 \oplus x_2, ..., x_{2m-1} \oplus x_{2m}) \\ & * \Psi_{v-1}(x_{2i_1}, ..., x_{2i_k}). \end{aligned}$$

Note that on the righthand side of the equation above, the first term corresponds to the random bits generated with the von Neumann scheme, the second and third terms relate to the symmetric information discarded by the von Neumann scheme. For example, when the input sequence is $X = \mathrm{HHTHTT}$, the output sequence based on the von Neumann scheme is

$$\Psi_1(\mathrm{HHTHTT}) = 0.$$

But based on the Peres scheme, we have the output sequence

$$\Psi_v(\text{HHTHTT}) = \Psi_1(\text{HHTHTT}) * \Psi_{v-1}(\text{THT}) * \Psi_{v-1}(\text{HT}),$$

which is 001, longer than that generated by the von Neumann scheme.

Finally, we can define Ψ_v for sequences of odd length by

$$\Psi_v(x_1, x_2, ..., x_{2m+1}) = \Psi_v(x_1, x_2, ..., x_{2m}).$$

Surprisingly, this simple iterative procedure achieves the optimal information efficiency asymptotically. The computational complexity and memory requirements of this scheme are substantially smaller than those of the Elias scheme. However, the generalization of this scheme to the case of an m-sided die with $m > 2$ is still unknown.

2.2.4 Properties

Let us denote $\Psi : \{\text{H}, \text{T}\}^n \to \{0, 1\}^*$ as a scheme that generates independent unbiased sequences from any biased coins (with unknown probabilities). Such Ψ can be the von Neumann scheme, the Elias scheme, the Peres scheme, or any other scheme. Let X be a sequence of biased coin tosses of length n, then a property of Ψ is that for any $Y \in \{0, 1\}^*$ and $Y' \in \{0, 1\}^*$ with $|Y| = |Y'|$, we have

$$P[\Psi(X) = Y] = P[\Psi(X) = Y'],$$

i.e., two output sequences of equal length have equal probability.

This observation leads to the following property for Ψ. It says that given the numbers of H's and T's, the number of sequences yielding a binary sequence Y equals the number of sequences yielding Y' when Y and Y' have the same length. It further implies that given the condition of knowing the number of H's and T's in the input sequence, the output sequence of Ψ is still independent and unbiased. This property is due to the linear independence of probability functions of the sequences

with different numbers of H's and T's.

Lemma 2.1. [138] *Let S_{k_1,k_2} be the subset of $\{H, T\}^n$ consisting of all sequences with k_1 appearances of H and k_2 appearances of T such that $k_1 + k_2 = n$. Let B_Y denote the set $\{X|\Psi(X) = Y\}$. Then for any $Y \in \{0,1\}^*$ and $Y' \in \{0,1\}^*$ with $|Y| = |Y'|$, we have*

$$|S_{k_1,k_2} \bigcap B_Y| = |S_{k_1,k_2} \bigcap B_{Y'}|.$$

2.3 Generalization for Loaded Dice

In this section, we propose a universal scheme for generalizing all the existing algorithms for biased coins such that they can deal with loaded dice with more than two sides. There is some related work: In [30], Dijkstra considered the opposite question and showed how to use a biased coin to simulate a fair die. In [61], Juels et al. studied the problem of simulating random bits from loaded dice, and their algorithm can be treated as the generalization of Elias's algorithm. However, for a number of known beautiful algorithms, like Peres's algorithm, we still do not know how to generalize them for larger alphabets (loaded dice). We propose a universal scheme that is able to generalize all the existing algorithms, including Elias's algorithm and Peres's algorithm. Compared to the other generalizations, this scheme is universal and easier to implement, and it preserves the optimality of the original algorithm on information efficiency. The brief idea of this scheme is that given a loaded die, we can convert it into multiple binary sources and apply existing algorithms to these binary sources separately. This idea seems natural, but not obvious.

2.3.1 An Example

Let us start from a simple example: Assume we want to generate random bits from a sequence $X = 012112210$, which is produced by a 3-sided die. Now, we write each symbol (die roll) into a

binary representation of length two (H for 1 and T for 0), so

$$0 \to \mathrm{TT}, 1 \to \mathrm{TH}, 2 \to \mathrm{HT}.$$

Hence, X can be represented as

$$\mathrm{TT,TH,HT,TH,TH,HT,HT,TH,TT}.$$

Only collecting the first bits of all the symbols yields an independent binary sequence

$$X_\phi = \mathrm{TTHTTHHTT}.$$

Collecting the second bits following T, we get another independent binary sequence

$$X_\mathrm{T} = \mathrm{THHHHT}.$$

Note that although both X_ϕ and X_T are independent sequences individually, X_ϕ and X_T are correlated with each other, since the length of X_T is determined by the content of X_ϕ.

Let Ψ be any function that generates random bits from a fixed number of coin tosses, such as Elias's algorithm and Peres's algorithm. We see that both $\Psi(X_\phi)$ and $\Psi(X_\mathrm{T})$ are sequences of random bits. But we do not know whether $\Psi(X_\phi)$ and $\Psi(X_\mathrm{T})$ are independent of each other since X_ϕ and X_T are correlated. One of our main contributions is to show that concatenating them together, i.e.,

$$\Psi(X_\phi) + \Psi(X_\mathrm{T})$$

still yields a sequence of random bits.

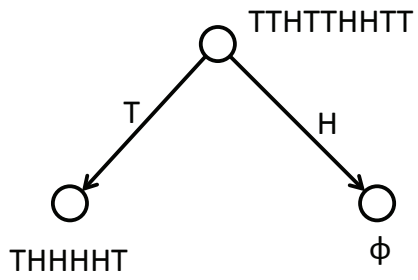


Figure 2.1. An instance of binarization tree.

2.3.2 A Universal Scheme

Generally, given a sequence of symbols generated from an m-sided die, written as

$$X = x_1 x_2 ... x_n \in \{0, 1, ..., m-1\}^n$$

with the number of states (sides) $m > 2$, we want to convert it into a group of binary sequences. To do this, we create a binary tree, called a binarization tree, in which each node is labeled with a binary sequence of H and T. See figure 2.1 as an instance of binarization tree for the above example. Given the binary representations of x_i for all $1 \leq i \leq n$, the path of each node in the tree indicates a prefix, and the binary sequence labeled at this node consists of all the bits (H or T) following the prefix in the binary representations of $x_1, x_2, ..., x_n$ (if it exists).

Given the number of sides m of a loaded die, the depth of the binarization tree is $b = \lceil \log_2 m \rceil - 1$. At the beginning, the binarization tree is a complete binary tree of depth b in which each node is labeled with an empty string, then we process all the input symbols $x_1, x_2, ..., x_n$ one by one. For the ith symbol, namely x_i, its binary representation is of length $b+1$. We add its first bit to the root node. If this bit is T, we add its second bit to the left child, otherwise we add its second bit to the right child ... repeating this process until all the $b+1$ bits of x_i are added along a path in the tree. Finally, we can get the binarization tree of X by processing all the symbols in X, i.e., $x_1, x_2, ..., x_n$.

Lemma 2.2. *Given the binarization tree of a sequence $X \in \{0, 1, ..., m-1\}^n$, we can reconstruct X uniquely.*

Proof. The construction of X from its binarization tree can be described as follows: At first, we read the first bit (H or T) from the root (once we read a bit, we remove it from the current sequence). If it is T, we read the first bit of its left child; if it is H, we read the first bit of its right child ... finally we reach a leaf, whose path indicates the binary representation of x_1. Repeating this procedure, we can continue to obtain $x_2, x_3, ..., x_n$. □

Let Υ_b denote the set consisting of all the binary sequences of length at most b, i.e.,

$$\Upsilon_b = \{\phi, \text{ T, H, TT, TH, HT, HH, ..., HHH...HH}\}.$$

Given $X \in \{0, 1, ..., m-1\}^n$, let X_γ denote the binary sequence labeled on a node corresponding to a prefix γ in the binarization tree, then we get a group of binary sequences

$$X_\phi, X_{\mathrm{T}}, X_{\mathrm{H}}, X_{\mathrm{TT}}, X_{\mathrm{TH}}, X_{\mathrm{HT}}, X_{\mathrm{HH}}, ...$$

For any function Ψ that generates random bits from a fixed number of coin tosses, we can generate random bits from X by calculating

$$\Psi(X_\phi) + \Psi(X_{\mathrm{T}}) + \Psi(X_{\mathrm{H}}) + \Psi(X_{\mathrm{TT}}) + \Psi(X_{\mathrm{TH}}) + ...,$$

where $A + B$ is the concatenation of A and B. We call this method as the generalized scheme of Ψ.

We show that the generalized scheme works for any binary algorithm Ψ such that it can generate random bits from an arbitrary m-sided die.

Theorem 2.3. *Let Ψ be any function that generates random bits from a fixed number of coin tosses. Given a sequence $X \in \{0, 1, ..., m-1\}^n$ with $m \geq 2$ generated from an m-sided die, the generalized scheme of Ψ generates an independent and unbiased sequence.*

The proof of this theorem will be given in the next subsection.

2.3.3 Proof of Theorem 2.3

Lemma 2.4. *Let $\{X_\gamma\}$ with $\gamma \in \Upsilon_b$ be the binary sequences labeled on the binarization tree of $X \in \{0, 1, ..., m-1\}^n$ as defined above. Assume X'_γ is a permutation of X_γ for all $\gamma \in \Upsilon_b$, then there exists exactly one sequence $X' \in \{0, 1, ..., m-1\}^n$ such that it yields a binarization tree that labels $\{X'_\gamma\}$ with $\gamma \in \Upsilon_b$.*

Proof. Based on $\{X'_\gamma\}$ with $\gamma \in \Upsilon_b$, we can construct the corresponding binarization tree and then create the sequence X' in the following way (if it exists). At first, we read the first bit (H or T) from the root (once we read a bit, we remove it from the current sequence). If it is T, we read the first bit of its left child; if it is H, we read the first bit of its right child ... finally we reach a leaf, whose path indicates the binary representation of x'_1. Repeating this procedure, we continue to obtain $x'_2, x'_3, ..., x'_n$. Hence, we are able to create the sequence $X' = x'_1 x'_2 ... x'_{n-1} x'_n$ if it exists.

It can be proved that the sequence X' can be successfully constructed if and only the following condition is satisfied: For any $\gamma \in \Upsilon_{b-1}$,

$$w_{\mathrm{T}}(X_\gamma) = |X_{\gamma\mathrm{T}}|, \quad w_{\mathrm{H}}(X_\gamma) = |X_{\gamma\mathrm{H}}|,$$

where $w_{\mathrm{T}}(X)$ counts the number of T's in X and $w_{\mathrm{H}}(X)$ counts the number of H's in X.

Obviously, the binary sequences $\{X_\gamma\}$ with $\gamma \in \Upsilon_b$ satisfy the above condition. Permuting them into $\{X'_\gamma\}$ with $\gamma \in \Upsilon_b$ does not violate this condition. Hence, we can always construct a sequence $X' \in \{0, 1, ..., m-1\}^n$, which yields $\{X'_\gamma\}$ with $\gamma \in \Upsilon_b$.

This completes the proof. □

Now, we divide all the possible input sequences in $\{0, 1, ..., m-1\}^n$ into classes. Two sequences $X, X' \in \{0, 1, ..., m-1\}^n$ are in the same class if and only if the binary sequences obtained from X and X' are permutations with each other, i.e., X'_γ is a permutation of X_γ for all $\gamma \in \Upsilon_b$. Here, we use $\mathbb{G}$ to denote the set consisting of all such classes.

Lemma 2.5. *All the sequences in a class $G \in \mathbb{G}$ have the same probability of being generated.*

Proof. Based on the probability distribution of each die roll $\{p_0, p_1, ..., p_{m-1}\}$, we can get a group of conditional probabilities, denoted as

$$q_{\mathrm{T}|\phi}, q_{\mathrm{H}|\phi}, q_{\mathrm{T}|\mathrm{T}}, q_{\mathrm{H}|\mathrm{T}}, q_{\mathrm{T}|\mathrm{H}}, q_{\mathrm{H}|\mathrm{H}}, q_{\mathrm{T}|\mathrm{TT}}, q_{\mathrm{H}|\mathrm{TT}}, \cdots,$$

where $q_{a|\gamma}$ is the conditional probability of generating a die roll x_i such that in its binary representation the bit following a prefix γ is a.

Note that $q_{0|\gamma} + q_{1|\gamma} = 1$ for all $\gamma \in \Upsilon_b$. For example, if $\{p_0, p_1, p_2\} = \{0.2, 0.3, 0.5\}$, then

$$q_{0|\phi} = 0.5, q_{0|0} = 0.4, q_{0|1} = 1.$$

It can be proved that the probability of generating a sequence $X \in \{0, 1, ..., m-1\}^n$ equals

$$\prod_{\gamma \in \Upsilon_b} q_{\mathrm{T}|\gamma}^{w_{\mathrm{T}}(X_\gamma)} q_{\mathrm{H}|\gamma}^{w_{\mathrm{H}}(X_\gamma)},$$

where $w_{\mathrm{T}}(X)$ counts the number of T's in X and $w_{\mathrm{H}}(X)$ counts the number of H's in X. This probability keeps unchanged when we permute X_γ to X'_γ for all $\gamma \in \Upsilon_b$.

This implies that all the elements in G have the same probability of being generated. □

Lemma 2.6. *Let Ψ be any function that generates random bits from a fixed number of coin tosses. Given $Z_\gamma, Z'_\gamma \in \{0,1\}^*$ for all $\gamma \in \Upsilon_b$, we define*

$$S = \{X | \forall \gamma \in \Upsilon_b, \Psi(X_\gamma) = Z_\gamma\},$$

$$S' = \{X | \forall \gamma \in \Upsilon_b, \Psi(X_\gamma) = Z'_\gamma\}.$$

If $|Z_\gamma| = |Z'_\gamma|$ for all $\gamma \in \Upsilon_b$, i.e., Z_γ and Z'_γ have the same length, then for all $G \in \mathbb{G}$,

$$|G \bigcap S| = |G \bigcap S'|,$$

i.e., $G \bigcap S$ and $G \bigcap S'$ have the same size.

Proof. We prove that for any $\theta \in \Upsilon_b$, if $Z_\gamma = Z'_\gamma$ for all $\gamma \neq \theta$ and $|Z_\theta| = |Z'_\theta|$, then

$$|G \bigcap S| = |G \bigcap S'|.$$

If this statement is true, we can obtain the conclusion in the lemma by replacing Z_γ with Z'_γ one by one for all $\gamma \in \Upsilon_b$.

In the class G, assume $|X_\theta| = n_\theta$. Let us define G_θ as the subset of $\{0,1\}^{n_\theta}$ consisting of all the permutations of X_θ. We also define

$$S_\theta = \{X_\theta | \Psi(X_\theta) = Z_\theta\},$$

$$S'_\theta = \{X_\theta | \Psi(X_\theta) = Z'_\theta\}.$$

According to lemma 2.1, if Ψ can generate random bits from an arbitrary biased coin, then

$$|G_\theta \bigcap S_\theta| = |G_\theta \bigcap S'_\theta|.$$

This implies that all the elements in $G_\theta \bigcap S_\theta$ and those in $G_\theta \bigcap S'_\theta$ are one-to-one mapping.

Based on this result, we are ready to show that the elements in $G \bigcap S$ and those in $G \bigcap S'$ are one-to-one mapping: For any sequence X in $G \bigcap S$, we get a series of binary sequences $\{X_\gamma\}$ with $\gamma \in \Upsilon_b$. Given Z'_θ with $|Z'_\theta| = |Z_\theta|$, we can find a (one-to-one) mapping of X_θ in $G_\theta \bigcap S'_\theta$, denoted by X'_θ. Here, X'_θ is a permutation of X_θ. According to lemma 2.4, there exists exactly one sequence $X' \in \{0, 1, ..., m-1\}^n$ such that it yields $\{X_\phi, X_{\mathrm{T}}, X_{\mathrm{H}}, ..., X'_\theta, ...\}$. Right now, we see that for any sequence X in $G \bigcap S$, we can always find its one-to-one mapping X' in $G \bigcap S'$, which implies that

$$|G \bigcap S| = |G \bigcap S'|.$$

This completes the proof. □

Based on the lemma above, we get theorem 2.3.

Theorem 2.3. *Let Ψ be any function that generates random bits from a fixed number of coin tosses. Given a sequence $X \in \{0, 1, ..., m-1\}^n$ with $m \geq 2$ generated from an m-sided die, the generalized scheme of Ψ generates an independent and unbiased sequence.*

Proof. In order to prove that the binary sequence generated is independent and unbiased, we show that for any two sequences $Y_1, Y_2 \in \{0,1\}^k$, they have the same probability to be generated. Hence, each binary sequence of length k can be generated with probability $\frac{c_k}{2^k}$ for some $0 \leq c_k \leq 1$.

First, we let $f : \{0, 1, ..., m-1\}^n \to \{0,1\}^*$ be the function of the generalized scheme of Ψ, then we write

$$P[f(X) = Y_1] = \sum_{G \in \mathbb{G}} P[f(X) = Y_1, X \in G].$$

According to lemma 2.5, all the elements in G have the same probability of being generated. Hence, we denote this probability as p_G, and the formula above can written as

$$P[f(X) = Y_1] = \sum_{G \in \mathbb{G}} p_G |\{X \in G, f(X) = Y_1\}|.$$

Let $Z_\gamma \in \{0,1\}^*$ be the sequence of bits generated from the node corresponding to γ for all $\gamma \in \Upsilon_b$, then $Y_1 = \sum_{\gamma \in \Upsilon_b} Z_\gamma$. We get that $P[f(X) = Y_1]$ equals

$$\sum_{G \in \mathbb{G}} \sum_{\{Z_\gamma : \gamma \in \Upsilon_b\}} p_G |\{X \in G, \forall \gamma \in \Upsilon_b, \Psi(X_\gamma) = Z_\gamma\}|$$

$$\times I_{\sum_{\gamma \in \Upsilon_b} Z_\gamma = Y_1},$$

where $I_{\sum_{\gamma \in \Upsilon_b} Z_\gamma = Y_1} = 1$ if and only if $\sum_{\gamma \in \Upsilon_b} Z_\gamma = Y_1$, otherwise it is zero.

Similarly, $P[f(X) = Y_2]$ equals

$$\sum_{G \in \mathbb{G}} \sum_{\{Z'_\gamma : \gamma \in \Upsilon_b\}} p_G |\{X \in G, \forall \gamma \in \Upsilon_b, \Psi(X_\gamma) = Z'_\gamma\}|$$

$$\times I_{\sum_{\gamma\in\Upsilon_b} Z'_\gamma = Y_2},$$

If $|Z'_\gamma| = |Z_\gamma|$ for all $\gamma \in \Upsilon_b$, then based on lemma 2.6, we can get

$$|\{X \in G, \forall\gamma \in \Upsilon_b, \Psi(X_\gamma) = Z_\gamma\}|$$

$$= |\{X \in G, \forall\gamma \in \Upsilon_b, \Psi(X_\gamma) = Z'_\gamma\}|.$$

Substituting it into the expressions of $P[f(X) = Y_1]$ and $P[f(X) = Y_2]$ shows

$$P[f(X) = Y_1] = P[f(X) = Y_2].$$

So we can conclude that for any binary sequences of the same length, they have the same probability of being generated. Furthermore, we can conclude that the bits generated are independent and unbiased.

This completes the proof. □

2.3.4 Optimality

In this subsection, we show that the universal scheme keeps the optimality of original algorithms, i.e., if the binary algorithm is asymptotically optimal, like Elias's algorithm or Peres's algorithm, its generalized version is also asymptotically optimal. Here, we say an algorithm is asymptotically optimal if and only if the number of random bits generated per input symbol is asymptotically equal to the entropy of an input symbol.

Theorem 2.7. *Given an m-sided die with probability distribution $\rho = (p_0, p_1, ..., p_{m-1})$, let n be the number of symbols (dice rolls) used in the generalized scheme of Ψ and let k be the number of random bits generated. If Ψ is asymptotically optimal, then the generalized scheme of Ψ is also asymptotically optimal, that means*

$$\lim_{n\to\infty} \frac{E[k]}{n} = H(\rho),$$

where

$$H(\rho) = H(p_0, p_1, ..., p_{m-1}) = \sum_{i=0}^{m-1} p_i \log_2 \frac{1}{p_i}$$

is the entropy of the m-sided die.

Proof. We prove this by induction. Using the same notations as above, we have the depth of the binarization tree $b = \lceil \log_2 m \rceil - 1$. If $b = 0$, i.e., $m \leq 2$, the algorithm is exactly Ψ. Hence, it is asymptotically optimal on efficiency. Now, assume that the conclusion holds for any integer $b - 1$, we show that it also holds for the integer b.

Since the length-$(b + 1)$ binary representations of $\{0, 1, ..., 2^b - 1\}$ start with 0, the probability for a symbol starting with 0 is

$$q_0 = \sum_{i=0}^{2^b-1} p_i.$$

In this case, the conditional probability distribution of these symbols is

$$\{\frac{p_0}{q_0}, \frac{p_1}{q_0}, ..., \frac{p_{2^b-1}}{q_0}\}.$$

Similarly, let

$$q_1 = \sum_{i=2^b}^{m} p_i,$$

then the conditional probability distribution of the symbols starting with 1 is

$$\{\frac{p_{2^b}}{q_1}, \frac{p_{2^b+1}}{q_1}, ..., \frac{p_{m-1}}{q_1}\}.$$

When n is large enough, the number of symbols starting with 0 approaches nq_0 and the number of symbols starting with 1 approaches nq_1. According to our assumption for $b-1$, the total number of random bits generated approaches

$$nH(q_0, q_1) + nq_0 H(\frac{p_0}{q_0}, \frac{p_1}{q_0}, ..., \frac{p_{2^b-1}}{q_0})$$

$$+nq_1 H(\frac{p_{2^b}}{q_1}, \frac{p_{2^b+1}}{q_1}, ..., \frac{p_{m-1}}{q_1}),$$

which equals

$$\begin{aligned} & nq_0 \log_2 \frac{1}{q_0} + nq_1 \log_2 \frac{1}{q_1} + nq_0 \sum_{i=0}^{2^b-1} \frac{p_i}{q_0} \log_2 \frac{q_0}{p_i} \\ & +nq_1 \sum_{i=2^b}^{m-1} \frac{p_i}{q_1} \log_2 \frac{q_1}{p_i} \\ = \; & n \sum_{i=0}^{m-1} p_i \log_2 \frac{1}{p_i} \\ = \; & nH(p_0, p_1, ..., p_{m-1}). \end{aligned}$$

This completes the proof. □

2.4 Efficient Generation of k Random Bits

2.4.1 Motivation

Most existing works on random bits generation from biased coins aim at maximizing the expected number of random bits generated from a fixed number of coin tosses. Falling into this category, Peres's scheme and Elias's scheme are asymptotically optimal for generating random bits. However, in these methods, the number of random bits generated is a random variable. In some occasions, we prefer to generate a prescribed number of random bits, hence it motivates us an opposite question: fixing the number of random bits to generate, i.e., k bits, how can we minimize the expected number of coin tosses? This question is equally important as the original one, since in many applications a prescribed number of random bits are required while the source is usually a stream of coin tosses instead of a sequence of fixed length. But the existing study on this question is very limited.

To generate k random bits, we are always able to make use of the existing schemes with fixed input length and variable output length like Peres's scheme or Elias's scheme. For example, we can keep reading n tosses (H or T) for several times and concatenate their outputs until the total number of random bits generated is slightly larger than k. However, if n is small, this approach is

less information efficient. If n is large, this approach may generate too many extra random bits, which can be treated as a waste. In this section, we propose an algorithm to generate exactly k random bits efficiently. It is motivated by the Elias's scheme. It can be proved that this algorithm is asymptotically optimal, namely, the expected number of coin tosses required per random bit generated is asymptotically equal to one over the entropy of the biased coin.

2.4.2 An Iterative Scheme

It is not easy to generate k random bits directly from a biased coin with very high information efficiency. Our approach of achieving this goal is to generate random bits iteratively – we first produce $m \leq k$ random bits, where m is a variable number that is equal to or close to k with very high probability. In next step, instead of trying to generate k random bits, we try to generate $k-m$ random bits ... we repeat this procedure until generating total k random bits.

How can we generate m random bits from a biased coin such that m is variable number that is equal to or very close to k? Our idea is to construct a group of disjoint prefix sets, denoted by $S_1, S_2, ..., S_w$, such that (1) all the sequences in a prefix set S_i with $1 \leq i \leq w$ have the same probability of being generated, and (2) $S = S_1 \bigcup S_2 \bigcup ... \bigcup S_w$ form a stopping set, namely, we can always get a sequence in S (or with probability almost 1) when keeping reading tosses from a biased coin. For example, we can let

$$\begin{aligned} S_1 &= \{\mathrm{HH}, \mathrm{HT}\}, \\ S_2 &= \{\mathrm{THH}, \mathrm{TTT}\}, \\ S_3 &= \{\mathrm{THT}, \mathrm{TTH}\}. \end{aligned}$$

Then $S = S_1 \bigcup S_2 \bigcup S_3$ forms a stopping set, which is complete and prefix-free.

In the scheme, we let all the sequences in S_i for all $1 \leq i \leq w$ have the same probability, i.e., S_i consists of sequences with the same number of H's and T's. We select criteria carefully such that $|S_i|$ is slightly larger than 2^k. Similarly as Elias's original scheme, we assign output binary

sequences to all the members in S_i for all $1 \leq i \leq w$. Let W be the number of members that were not assigned yet in a prefix set, then 2^j possible output binary sequences of length j are assigned to 2^j distinct unassigned members, where $j = k$ if $W \geq 2^k$ and $2^j \leq W < 2^{j+1}$ if $W < 2^k$. We repeat the procedure above for the rest of the members that were not assigned.

Theorem 2.8. *The above method generates m random bits for some m with $0 \leq m \leq k$.*

Proof. It is easy to see that the above method never generates a binary sequence longer than k. We only need to prove that for any binary sequences $Y, Y' \in \{0,1\}^m$, they have the same probability of being generated.

Let f denote the function corresponding to the above method. Then

$$P[f(X) = Y] = \sum_{i=1}^{w} P[X \in S_i] P[f(X) = Y | X \in S_i].$$

Given $X \in S_i$, we have $P[f(X) = Y | X \in S_i] = P[f(X) = Y' | X \in S_i]$, which supports our claim that any two binary sequences of the same length have the same probability of being generated. □

The next question is how to construct such prefix sets $S_1, S_2, ..., S_w$. Let us first consider the construction of their union, i.e., the stopping set S. Given a biased coin, we design an algorithm that reads coin tosses and stops the reading until it meets the first input sequence that satisfies some criterion. For instance, let k_1 be the number of H's and k_2 be the number of T's in the current input sequence, one possible choice is to read coin tosses until we get the first sequence such that $\binom{k_1 + k_2}{k_1} \geq 2^k$. Such an input sequence is a member in the stopping set S. However, this criterion is not the best one that we can have, since it will introduce too many iterations to generate k random bits. To reduce the number of iterations, we hope that the size of each prefix set, saying S_i, is slightly larger than 2^k. As a result, we use the following stopping set:

$$S = \{\text{the first sequence s.t. } \binom{k_1 + k_2}{k_1} \geq \frac{2^k (k_1 + k_2)}{\min(k_1, k_2)}\}.$$

Later, we will show that the selection of such a stopping set can make the number of iterations very

small.

Now we divide all the sequences in the stopping set S into different classes, i.e., the prefix sets $S_1, S_2, ..., S_w$, such that each prefix set consists of the sequences with the same number of H's and T's. Assume S_{k_1,k_2} is a nonempty prefix set that consists of sequences with k_1 H's and k_2 T's, then

$$S_{k_1,k_2} = G_{k_1,k_2} \bigcap S,$$

where G_{k_1,k_2} is the set consisting of all the sequences with k_1 H's and k_2 T's. According to the stopping set constructed above, we have

$$S_{k_1,k_2} = \{x \in G_{k_1,k_2} | \binom{k_1+k_2}{k_1} \geq \frac{2^k(k_1+k_2)}{\min(k_1,k_2)},$$

$$\binom{k_1+k_2-1}{k_1'} < \frac{2^k(k_1+k_2-1)}{\min(k_1',k_2')}\},$$

where k_1' is the number of H's in x without considering the last symbol and k_2' is the number of H's in x without considering the last symbol. So if the last symbol of x is H, then $k_1' = k_1 - 1, k_2' = k_2$; if the last symbol of x is T, then $k_1' = k_1, k_2' = k_2 - 1$. According to the expression of S_{k_1,k_2}, we see that the sequences in a prefix set are not prefixes of sequences in another prefix set. Furthermore, we can prove that the size of each prefix set is at least 2^k.

Lemma 2.9. *If $S_{k_1,k_2} \neq \phi$, then $|S_{k_1,k_2}| \geq 2^k$.*

Proof. Without loss of generality, we assume that $k_1 \leq k_2$, hence, $\binom{k_1+k_2}{k_1} \geq \frac{2^k(k_1+k_2)}{k_1}$. It also implies $k_1 \geq 1$. To prove $|S_{k_1,k_2}| \geq 2^k$, we show that S_{k_1,k_2} includes all the sequences $x \in G_{k_1,k_2}$ ending with H. If $x \in G_{k_1,k_2}$ ending with H does not belong to S_{k_1,k_2}, then

$$\binom{k_1+k_2-1}{k_1'} \geq \frac{2^k(k_1+k_2-1)}{k_1'}.$$

From which, we can get

$$\binom{k_1+k_2-1}{k_1} \geq \frac{2^k(k_1+k_2-1)}{k_1}.$$

It further implies that all the sequences $x \in G_{k_1,k_2}$ ending with T are also not members in S_{k_1,k_2}. So S_{k_1,k_2} is empty. It is a contradiction.

The number of sequences $x \in G_{k_1,k_2}$ ending with H is

$$\binom{k_1+k_2-1}{k_1-1} = \binom{k_1+k_2}{k_1}\frac{k_1}{k_1+k_2} \geq 2^k.$$

So the size of S_{k_1,k_2} is at least 2^k if $S_{k_1,k_2} \neq \phi$. This completes the proof. □

Based on the construction of prefix sets, we can get an algorithm Φ_k for generating m random bits with $0 \leq m \leq k$, described as follows.

Algorithm Φ_k

Input: A stream of biased coin tosses.

Output: m bits with $0 \leq m \leq k$.

(1) Reading coin tosses until there are k_1 H's and k_2 T's for some k_1 and k_2 such that

$$\binom{k_1+k_2}{k_1} \geq \frac{2^k(k_1+k_2)}{\min(k_1,k_2)}.$$

(2) Let X denote the current input sequence of coin tosses. If the last coin toss is H, we let $k_1' = k_1 - 1, k_2' = k_2$; otherwise, we let $k_1' = k_1, k_2' = k_2 - 1$. We remove this coin toss from X if

$$\binom{k_1+k_2-1}{k_1'} \geq \frac{2^k(k_1+k_2-1)}{\min(k_1',k_2')}.$$

(3) Let Ψ_E denote the Elias's function[1] for generating random bits from a fixed number of coin tosses. A fast computation of Ψ_E was provided by Ryabko and Matchikina in [99]. The output of the algorithm Ψ_k is $\Psi_E(X)$ or the last k bits of $\Psi_E(X)$ if $\Psi_E(X)$ is longer than k.

According to lemma 2.9, we can easily get the following conclusion.

[1]Here, an arbitrary algorithm for generating random bits from a fixed number of coin tosses works.

Corollary 2.10. *The algorithm Φ_k generates m random bits for some m with $0 \leq m \leq k$, and $m = k$ with probability at least $1/2$.*

Proof. The sequence generated by Φ_k is independent and unbiased. This conclusion is immediate from lemma 2.9. Assume that the input sequence $x \in S_i$ for some i with $1 \leq i \leq w$, then the probability of $m = k$ is

$$\frac{\lfloor \frac{|S_i|}{2^k} \rfloor 2^k}{|S_i|},$$

which is at least $1/2$ based on the fact that $|S_i| \geq 2^k$. Since this conclusion is true for all S_i with $1 \leq i \leq w$, we can claim that $m = k$ with probability at least $1/2$. □

Since the algorithm Φ_k generates m random bits for some m with $0 \leq m \leq k$ from an arbitrary biased coin, we are able to generate k bits iteratively: After generating m random bits, we apply the algorithm Φ_{k-m} for generating $k - m$ bits. Repeating this procedure, the total number of random bits generated will converge to k very quickly. We call this scheme as an iterative scheme for generating k random bits.

To generate k random bits, we do not want to iterate Φ_k too many times. Fortunately, in the following theorem, we show that in our scheme the expected number of iterations is upper bounded by a constant 2.

Theorem 2.11. *The expected number of iterations in the iterative scheme for generating k random bits is at most 2.*

Proof. According to corollary 2.10, Φ_k generates $m = k$ random bits with probability at least $1/2$. Hence, the scheme stops at each iteration with probability more than $1/2$. Following this fact, the result in the theorem is immediate. □

2.4.3 Optimality

In this subsection, we study the information efficiency of the iterative scheme and show that this scheme is asymptotically optimal.

Lemma 2.12. *Given a biased coin with probability p being H, let n be the number of coin tosses used by the algorithm Φ_k, then*

$$\lim_{k\to\infty} \frac{E[n]}{k} \leq \frac{1}{H(p)}.$$

Proof. We consider the probability of having an input sequence of length at least n, denote as P_n. In this case, we can write $n = k_1 + k_2$, where k_1 is the number of H's and k_2 is the number of T's. According to the construction of the stopping set,

$$\binom{n-1}{\min(k_1,k_2)-1} < 2^k \frac{n-1}{\min(k_1,k_2)-1}.$$

Or we can write it as

$$\binom{n-2}{\min(k_1,k_2)-2} < 2^k.$$

Hence, we get an upper bound for $\min(k_1, k_2)$, which is

$$t_n = \max\{i \in \{0,1,...,n\} | \binom{n-2}{i-2} < 2^k\}. \tag{2.1}$$

Note that if $\binom{n-2}{\frac{n}{2}-2} \geq 2^k$, then t_n is a nondecreasing function of n.

According to the symmetry of our criteria, we can get

$$P_n \leq \sum_{i=0}^{t_n} (p^i(1-p)^{n-i} + (1-p)^i p^{n-i}) \binom{n}{i}.$$

For convenience, we write

$$Q_n = \sum_{i=0}^{t_n} (p^i(1-p)^{n-i} + (1-p)^i p^{n-i}) \binom{n}{i},$$

then $P_n \leq Q_n$ and Q_n is also a nondecreasing function of n.

Now, we are ready to calculate the expected number of coin tosses required, which equals

$$\begin{aligned} E[n] &= \sum_{n=1}^{\infty}(P_n - P_{n+1})n = \sum_{n=1}^{\infty} P_n \\ &\leq \sum_{n=1}^{\frac{k}{H(p)}(1+\epsilon)} P_n + \sum_{n=\frac{k}{H(p)}(1+\epsilon)}^{\infty} Q_n + \sum_{n=2\frac{k}{H(p)}(1+\epsilon)}^{\infty} Q_n, \end{aligned} \tag{2.2}$$

where $\epsilon > 0$ is a small constant. In the rest, we study the upper bounds for all the three terms when n is large enough.

For the first term, we have

$$\sum_{n=1}^{\frac{k}{H(p)}(1+\epsilon)} P_n \leq \frac{k}{H(p)}(1+\epsilon). \tag{2.3}$$

Now let us consider the second term

$$\sum_{n=\frac{k}{H(p)}(1+\epsilon)}^{2\frac{k}{H(p)}(1+\epsilon)} Q_n \leq \frac{k}{H(p)}(1+\epsilon)Q_{\frac{k}{H(p)}(1+\epsilon)}.$$

Using the Stirling bounds on factorials yields

$$\lim_{n\to\infty} \frac{1}{n} \log_2 \binom{n}{\rho n} = H(\rho),$$

where H is the binary entropy function. Hence, following (2.1), we can get

$$\lim_{n\to\infty} H(\frac{t_n}{n}) = \lim_{n\to\infty} \frac{k}{n}.$$

When $n = \frac{k}{H(p)}(1+\epsilon)$, we can write

$$\lim_{n\to\infty} H(\frac{t_n}{n}) = \frac{H(p)}{1+\epsilon},$$

which implies that

$$\lim_{n\to\infty} \frac{t_n}{n} = p - \epsilon_1$$

for some $\epsilon_1 > 0$. So there exists an N_1 such that for $n > N_1$, $\frac{n_t}{n} \leq p - \epsilon_1/2$.

By the weak law for the binomial distribution, given any $\epsilon_2 > 0$ and $\delta > 0$, there is an N_2 such that for $n > N_2$, with probability at least $1 - \delta$ there are i H's among the n coin tosses such that $|\frac{i}{n} - p| \leq \epsilon_2$. Letting $\epsilon_2 = \epsilon_1/2$ and $n = \frac{k}{H(p)}(1+\epsilon)$ gives

$$Q_n \leq \delta,$$

for any $\delta > 0$ when $n > \max(N_1, N_2)$.

So for any $\delta > 0$, when k is large enough, we have

$$\sum_{n=\frac{k}{H(p)}(1+\epsilon)}^{2\frac{k}{H(p)}(1+\epsilon)} Q_n \leq \frac{k}{H(p)}(1+\epsilon)\delta. \tag{2.4}$$

To calculate the third term, we notice that Q_n decays very quickly as n increase when $n \geq 2\frac{k}{H(p)}(1+\epsilon)$. In this case,

$$\begin{aligned}
& \frac{Q_{n+1}}{Q_n} \\
= & \frac{\sum_{i=0}^{t_{n+1}} (p^i(1-p)^{n+1-i} + (1-p)^i p^{n+1-i}) \binom{n+1}{i}}{\sum_{i=0}^{t_n} (p^i(1-p)^{n-i} + (1-p)^i p^{n-i}) \binom{n}{i}} \\
\leq & \frac{\sum_{i=0}^{t_n} (p^i(1-p)^{n+1-i} + (1-p)^i p^{n+1-i}) \binom{n+1}{i}}{\sum_{i=0}^{t_n} (p^i(1-p)^{n-i} + (1-p)^i p^{n-i}) \binom{n}{i}} \\
\leq & \max_{i=0}^{t_n} \frac{(p^i(1-p)^{n+1-i} + (1-p)^i p^{n+1-i}) \binom{n+1}{i}}{(p^i(1-p)^{n-i} + (1-p)^i p^{n-i}) \binom{n}{i}} \\
\leq & (1-p) \max_{i=1}^{t_n} \frac{n+1}{n+1-t_n} \\
\leq & \frac{(1-p)n}{n-t_n}.
\end{aligned}$$

When $n \geq 2\frac{k}{H(p)}(1+\epsilon)$, we have

$$\lim_{n\to\infty} H(\frac{t_n}{n}) = \lim_{n\to\infty} \frac{k}{n} \leq \frac{H(p)}{2(1+\epsilon)}.$$

This implies that when n is large enough, $H(\frac{t_n}{n}) \leq \frac{H(p)}{2}$. Let us define a constant α such that $\alpha \leq \frac{1}{2}$ and $H(\alpha) = \frac{H(p)}{2}$. Then for all $n \geq 2\frac{k}{H(p)}(1+\epsilon)$, when k is large enough,

$$\frac{Q_{n+1}}{Q_n} \leq \frac{1-p}{1-\alpha} < 1.$$

Therefore, given any $\delta > 0$, when k is large enough, the value of the third term

$$\begin{aligned} \sum_{n=2\frac{k}{H(p)}(1+\epsilon)}^{\infty} Q_n &\leq Q_{2\frac{k}{H(p)}(1+\epsilon)} \sum_{i=0}^{\infty} (\frac{1-p}{1-\alpha})^i \\ &\leq Q_{\frac{k}{H(p)}(1+\epsilon)} \frac{1}{1-\frac{1-p}{1-\alpha}} \\ &\leq \frac{1-\alpha}{p-\alpha}\delta. \end{aligned} \tag{2.5}$$

Substituting (2.3), (2.4), and (2.5) into (2.2) yields that for any $\epsilon > 0$ and $\delta > 0$, if k is large enough, we have

$$E[n] \leq \frac{k}{H(p)}(1+\epsilon)(1+\delta) + \frac{1-\alpha}{p-\alpha}\delta,$$

with $\alpha < p$.

Then it is easy to get that

$$\lim_{k\to\infty} \frac{E[n]}{k} \leq \frac{1}{H(p)}.$$

This completes the proof. □

Theorem 2.13. *Given a biased coin with probability p being H, let n be the number of coin tosses required to generate k random bits in the iterative scheme, then*

$$\lim_{k\to\infty} \frac{E[n]}{k} = \frac{1}{H(p)}.$$

Proof. First, we prove that $\lim_{k\to\infty} \frac{E[n]}{k} \geq \frac{1}{H(p)}$. Let $X \in \{0,1\}^*$ be the input sequence, then

$$\lim_{k\to\infty} \frac{E[n]H(p)}{H(X)} = 1.$$

Shannon's theory tells us that it is impossible to extract more than $H(X)$ random bits from X, i.e., $H(X) \geq k$. So

$$\lim_{k\to\infty} \frac{E[n]}{k} \geq \frac{1}{H(p)}.$$

To get the conclusion in the theorem, we only need to show that

$$\lim_{k\to\infty} \frac{E[n]}{k} \leq \frac{1}{H(p)}.$$

To distinguish the n in this theorem and the one in the previous theorem, we use $n_{(k)}$ denote the number of coin tosses required to generate k random bits in the iterative scheme and let $n^{\Phi}_{(k)}$ denote the number of coin tosses required by Φ_k. Let p_m be the probability for Φ_k generating m random bits with $0 \leq m \leq k$. Then we have that

$$E[n_{(k)}] = E[n^{\Phi}_{(k)}] + \sum_{m=0}^{k} p_m E[n_{(k-m)}]. \tag{2.6}$$

According to the algorithm, $p_k \geq \frac{1}{2}$ and $E[n_{(k-m)}] \leq E[n_{(k)}]$. Substituting them into the equation above gives

$$E[n_{(k)}] \leq E[n^{\Phi}_{(k)}] + \frac{1}{2}E[n_{(k)}],$$

i.e., $E[n_{(k)}] \leq 2E[n^{\Phi}_{(k)}]$.

Now, we divide the second term in (2.6) into two parts such that

$$E[n_{(k)}] \leq E[n^{\Phi}_{(k)}] + \sum_{m=0}^{k-\epsilon k} p_m E[n_{(k-m)}] + \sum_{m=k-\epsilon k}^{k} p_m E[n_{(k-m)}],$$

for a constant $\epsilon > 0$. In which,

$$\sum_{m=0}^{k-\epsilon k} p_m E[n_{(k-m)}] \leq (\sum_{m=0}^{k-\epsilon k} p_m) 2E[n^{\Phi}_{(k)}],$$

$$\sum_{m=k-\epsilon k}^{k} p_m E[n_{(k-m)}] \leq 2E[n^{\Phi}_{(\epsilon k)}].$$

Hence

$$E[n_{(k)}] \leq E[n^{\Phi}_{(k)}] + (\sum_{m=0}^{k-\epsilon k} p_m) 2E[n^{\Phi}_{(k)}] + 2E[n^{\Phi}_{(\epsilon k)}]. \tag{2.7}$$

Given k, all the possible input sequences are divided into w prefix sets $S_1, S_2, ..., S_w$, where w can be an infinite number. Given an input sequence $X \in S_i$ for $1 \leq i \leq w$, we are considering the probability for Φ_k generating a sequence of length m.

In our algorithm, $|S_i| \geq 2^k$. Assume

$$|S_i| = \alpha_k 2^k + \alpha_{k-1} 2^{k-1} + ... + \alpha_0 2^0,$$

where $\alpha_k \geq 1$ and $0 \leq \alpha_0, \alpha_1, ..., \alpha_{k-1} \leq 1$. Given the condition $X \in S_i$, we have

$$\sum_{m=0}^{k-\epsilon k} p_m = \frac{\sum_{i=0}^{k-\epsilon k} \alpha_i 2^i}{\sum_{i=0}^{k} \alpha_i 2^i} \leq \frac{2^{k-\epsilon k+1}}{2^k + 2^{k-\epsilon k+1}} \leq \frac{2^{k-\epsilon k+1}}{2^k}.$$

So given any $\delta > 0$, when k is large enough, we have

$$\sum_{m=0}^{k-\epsilon k} p_m \leq \delta. \tag{2.8}$$

Although we reach this conclusion for $X \in S_i$, this conclusion holds for any S_i with $0 \leq i \leq w$. Hence, we are able to remove this constrain that $X \in S_i$.

According to the previous lemma, for any $\delta > 0$, when k is large enough, we have

$$\frac{E[n^{\Phi}_{(\epsilon k)}]}{\epsilon k} \leq \frac{1}{H(p)} + \delta, \tag{2.9}$$

$$\frac{E[n_{(k)}^{\Phi}]}{k} \leq \frac{1}{H(p)} + \delta. \tag{2.10}$$

Substituting (2.8), (2.9), and (2.10) into (2.7) gives us

$$E[n_{(k)}] \leq k(\frac{1}{H(p)} + \delta)(1 + 2\delta) + 2k\epsilon(\frac{1}{H(p)} + \delta).$$

From which, we obtain

$$\lim_{k\to\infty} \frac{E[n]}{k} = \lim_{k\to\infty} \frac{E[n_{(k)}]}{k} \leq \frac{1}{H(p)}.$$

This completes the proof. □

The theorem above shows that the iterative scheme is asymptotically optimal, i.e., the expected number of coin tosses for generating k random bits approaches the information theoretic bound by below when k becomes large.

2.5 Conclusion

In this chapter, we have presented a universal scheme that transforms an arbitrary algorithm for 2-faced coins to generate random bits from general m-sided dice, hence enabling the application of existing algorithms to general sources. Although a similar question has been studied before, as in [61], their solution can only be applied to a specified algorithm, i.e., Elias's algorithm.

The second contribution of this chapter is an efficient algorithm for generating a prescribed number of random bits from an arbitrary biased coin. In many applications, this is a natural way of considering the problem of random bits generation from biased coins, but it is not well studied in the literature. This problem is similar to the one studied in universal variable-to-fixed length codes, which are used to parse an infinite sequence into variable-length phases. Each phase is then encoded into a fixed number of bits. In [74], Lawrence devised a variable-to-fixed length code for the class of binary memoryless sources (biased coins), which is based on Pascal's triangle (so is our algorithm). Tjalkens and Willems [114] modified Lawrence's algorithm as a more natural and

simple implementation, and they showed that the rate of the resulting code converges asymptotically optimally fast to the source entropy. These universal variable-to-fixed length codes are probably capable to generate random bits asymptotically in some (week) sense, namely, the random bits generated in this way are not perfect, and they cannot satisfy the typical requirement based on statistical distance (widely used in computer science).

Chapter 3

Random Number Generation from Markov Chains

This chapter studies the problem of efficiently generating random bits from Markov chains and provides the first known algorithm that generates unbiased random bits from an arbitrary finite Markov chain, operates in expected linear time and achieves the information-theoretic upper bound on efficiency.[1]

3.1 Introduction

In this chapter, we study the problem of generating random bits from an arbitrary and unknown finite Markov chain (the transition matrix is unknown). The input to our problem is a sequence of symbols that represent a random trajectory through the states of the Markov chain; given this input sequence our algorithm generates an independent unbiased binary sequence called the output sequence. This problem was first studied by Samuelson [101]. His approach was to focus on a single state (ignoring the other states) treat the transitions out of this state as the input process, hence, reducing the problem of correlated sources to the problem of a single 'independent' random source; obviously, this method is not efficient. Elias [33] suggested to utilize the sequences related to all states: Producing an 'independent' output sequence from the transitions out of every state and then pasting (concatenating) the collection of output sequences to generate a long output sequence.

[1] Some of the results presented in this chapter have been previously published in [138] and [139].

However, neither Samuelson nor Elias proved that their methods work for arbitrary Markov chains, namely, they did not prove that the transitions out of each state are independent. In fact, Blum [14] probably realized it, as he mentioned that: (i) "Elias's algorithm is excellent, but certain difficulties arise in trying to use it (or the original von Neumann scheme) to generate bits in expected linear time from a Markov chain," and (ii) "Elias has suggested a way to use all the symbols produced by a MC (Markov Chain). His algorithm approaches the maximum possible efficiency for a one-state MC. For a multi-state MC, his algorithm produces arbitrarily long finite sequences. He does not, however, show how to paste these finite sequences together to produce *infinitely* long independent unbiased sequences." Blum [14] derived a beautiful algorithm to generate random bits from a degree-2 Markov chain *in expected linear time* by utilizing the von Neumann scheme for generating random bits from biased coin flips. While his approach can be extended to arbitrary out-degrees (the general Markov chain model used in this chapter), the information efficiency is still far from optimal due to the low information efficiency of the von Neumann scheme.

We generalize Blum's algorithm to arbitrary-degree finite Markov chains and combine it with existing methods for efficient generation of unbiased bits from biased coins, such as Elias's method. As a result, we provide the first known algorithm that generates unbiased random bits from arbitrary finite Markov chains, operates in expected linear time and achieves the information-theoretic upper bound on efficiency. Specifically, we propose an algorithm (that we call algorithm A), that is a simple modification of Elias's suggestion to generate random bits; it operates on finite sequences and its efficiency can asymptotically reach the information-theoretic upper bound for long input sequences. In addition, we propose a second algorithm, called algorithm B, that is a combination of Blum's and Elias's algorithms, it generates infinitely long sequences of random bits in expected linear time. One of our key ideas for generating random bits is that we explore equal-probability sequences of the same length. Hence, a natural question is: Can we improve the efficiency by utilizing as many as possible equal-probability sequences? We provide a positive answer to this question and describe algorithm C, that is the first known polynomial-time and optimal algorithm (it is optimal in terms of information efficiency for an arbitrary input length) for random bit generation from finite Markov

chains.

The remainder of this chapter is organized as follows. Section 3.2 introduces some existing works in generating random bits from Markov chains and discusses the challenges. Section 3.3 presents our main lemma that characterizes the exit sequences of Markov chains. Algorithm A is presented and analyzed in section 3.4, it is related to Elias's ideas for generating random bits from Markov chains. Algorithm B is presented in section 3.5, it is a generalization of Blum's algorithm. An optimal algorithm, called algorithm C, is described in section 3.6. Finally, section 3.7 provides numerical evaluations of our algorithms.

3.2 Preliminaries

3.2.1 Notations

For the convenience of descriptions, the following notations will be used in this chapter:

x_a	:	the ath element of X
$X[a]$	:	same as x_a, the ath element of X
$X[a:b]$	:	subsequence of X from the ath to bth element
X^a	:	$X[1:a]$
$X * Y$	:	the concatenation of X and Y,
		e.g., $s_1s_2 * s_2s_1 = s_1s_2s_2s_1$
$Y \equiv X$	:	Y is a permutation of X,
		e.g., $s_1s_2s_2s_3 \equiv s_3s_2s_2s_1$
$Y \doteq X$	:	Y is a permutation of X and $y_{\|Y\|} = x_{\|X\|}$
		namely the last element is fixed,
		e.g., $s_1s_2s_2s_3 \doteq s_2s_2s_1s_3$ where s_3 is fixed

3.2.2 Exit Sequences

Our goal is to efficiently generate random bits from a Markov chain with unknown transition probabilities. The model we study is that a Markov chain generates the sequence of states that it is visiting and this sequence of states is the input sequence to our algorithm for generating random bits. Specifically, we express an input sequence as $X = x_1x_2...x_N$ with $x_i \in \{s_1, s_2, ..., s_n\}$, where $\{s_1, s_2, ..., s_n\}$ indicate the states of a Markov chain.

One idea is that for a given Markov chain, we can treat each state, say s, as a coin and consider the 'next states' (the states the chain has transitioned to after being at state s) as the results of a coin toss. Namely, we can generate a collection of sequences $\pi(X) = [\pi_1(X), \pi_2(X), ..., \pi_n(X)]$, called exit sequences, where $\pi_i(X)$ is the sequence of states following s_i in X, namely,

$$\pi_i(X) = \{x_{j+1} | x_j = s_i, 1 \leq j < N\}.$$

For example, assume that the input sequence is

$$X = s_1s_4s_2s_1s_3s_2s_3s_1s_1s_2s_3s_4s_1.$$

If we consider the states following s_1 we get $\pi_1(X)$ as the set of states in boldface:

$$X = s_1\mathbf{s_4}s_2s_1\mathbf{s_3}s_2s_3s_1\mathbf{s_1}\mathbf{s_2}s_3s_4s_1.$$

Hence, the exit sequences are:

$$\begin{aligned} \pi_1(X) &= s_4s_3s_1s_2; \\ \pi_2(X) &= s_1s_3s_3; \\ \pi_3(X) &= s_2s_1s_4; \\ \pi_4(X) &= s_2s_1. \end{aligned}$$

Lemma 3.1 (Uniqueness)**.** *An input sequence X can be uniquely determined by x_1 and $\pi(X)$.*

Proof. Given x_1 and $\pi(X)$, according to the work of Blum in [14], $x_1x_2...x_N$ can uniquely be constructed in the following way: Initially, set the starting state as x_1. Inductively, if $x_i = s_k$, then set x_{i+1} as the first element in $\pi_k(X)$ and remove the first element of $\pi_k(X)$. Finally, we can uniquely generate the sequence $x_1x_2...x_N$. □

Lemma 3.2 (Equal-probability)**.** *Two input sequences $X = x_1x_2...x_N$ and $Y = y_1y_2...y_N$ with $x_1 = y_1$ have the same probability to be generated if $\pi_i(X) \equiv \pi_i(Y)$ for all $1 \leq i \leq n$.*

Proof. Note that the probability of generating X is

$$P[X] = P[x_1]P[x_2|x_1]...P[x_N|x_{N-1}]$$

and the probability of generating Y is

$$P[Y] = P[y_1]P[y_2|y_1]...P[y_N|y_{N-1}].$$

By permutating the terms in the expression above, it is not hard to get that $P[X] = P[Y]$ if $x_1 = y_1$ and $\pi_i(X) \equiv \pi_i(Y)$ for all $1 \leq i \leq n$. Basically, the exit sequences describe the edges that are used in the trajectory in the Markov chain. The edges in the trajectories that correspond to X and Y are identical, hence $P[X] = P[Y]$. □

3.2.3 Samulson and Elias's Methods

In [101], Samuelson considered a two-state Markov chain, and he pointed out that it may generate unbiased random bits by applying the von Neumann scheme to the exit sequence of state s_1. Later, in [33], in order to increase the efficiency, Elias has suggested a scheme that uses all the symbols produced by a Markov chain. His main idea was to create the final output sequence by concatenating the output sequences that correspond to $\pi_1(X), \pi_2(X), ...$ However, neither Samuelson nor Elias

Table 3.1. Probabilities of exit sequences

Input sequence	Probability	$\Psi(\pi_1(X))$	$\Psi(\pi_1(X)) * \Psi(\pi_2(X))$
$s_1s_1s_1s_1$	$(1-p_1)^3$	ϕ	ϕ
$s_1s_1s_1s_2$	$(1-p_1)^2p_1$	0	0
$s_1s_1s_2s_1$	$(1-p_1)p_1p_2$	0	0
$s_1s_1s_2s_2$	$(1-p_1)p_1(1-p_2)$	0	0
$s_1s_2s_1s_1$	$p_1p_2(1-p_1)$	1	1
$s_1s_2s_1s_2$	$p_1^2p_2$	ϕ	ϕ
$s_1s_2s_2s_1$	$p_1(1-p_2)p_2$	ϕ	1
$s_1s_2s_2s_2$	$p_1(1-p_2)^2$	ϕ	ϕ

proved that their methods produce random output sequences that are independent and unbiased. In fact, their proposed methods are not correct for some cases. To demonstrate it we consider: (1) $\Psi(\pi_1(X))$ as the final output. (2) $\Psi(\pi_1(X)) * \Psi(\pi_2(X)) * ...$ as the final output. For example, consider the two-state Markov chain in which $P[s_2|s_1] = p_1$ and $P[s_1|s_2] = p_2$, as shown in figure 3.1.

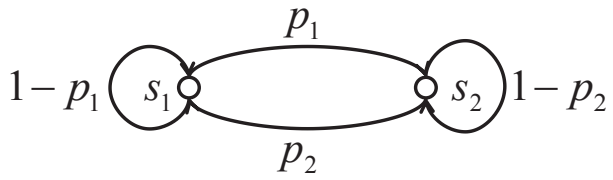


Figure 3.1. An example of Markov chain with two states.

Assume that an input sequence of length $N = 4$ is generated from this Markov chain and the starting state is s_1, then the probabilities of the possible input sequences and their corresponding output sequences are given in table 3.1. In the table we can see that the probabilities to produce 0 or 1 are different for some p_1 and p_2 in both methods, presented in columns 3 and 4, respectively.

3.2.4 Blum's Algorithm

In [14], Blum proposed a beautiful algorithm to generate an independent unbiased sequence of 0s and 1s from any Markov chain by extending the von Neumann's scheme. His algorithm can deal

with infinitely long sequences and uses only constant space and expected linear time. His algorithm can be described as follows.

Blum's Algorithm

Input: A sequence (or a stream) $x_1x_2...$ produced by a Markov chain, where $x_i \in \{s_1, s_2, ..., s_n\}$.

Output: A sequence (or a stream) Y of 0s and 1s.

Main Function:

$E_i = \phi$ (empty) for all $1 \leq i \leq n$.

$k_i = 1$ for all $1 \leq i \leq n$.

c : the index of current state, namely, $s_c = x_1$.

while next input symbol is s_j ($\neq$ **null**) **do**

$E_c = E_c s_j$ (Add s_j to E_c).

if $|E_j| \geq 2$ **then**

Output **1** if $E_j = s_u s_v$ with $u > v$;

Output **0** if $E_j = s_u s_v$ with $u < v$.

$E_j = \phi$.

$k_j = k_j + 1$.

end if

$c = j$.

end while

The beauty of this algorithm is its simplicity and elegance. It extends the original one-coin von Neumann scheme to generate an independent sequence from any Markov chain in expected linear time. Blum further demonstrated that the timing of announcing the random bits is crucial, namely, the order of the output bits matters.

3.3 Main Lemma

3.3.1 Description

The problem of generating random bits from an arbitrary Markov chain is challenging, as Blum said in [14]: "Elias's algorithm is excellent, but certain difficulties arise in trying to use it (or the original von Neumann scheme) to generate random bits in expected linear time from a Markov chain." It seems that the exit sequence of a state is independent since each exit of the state will not affect the other exits. However, this is not always true when the length of the input sequence is given, say N. Let us still consider the example of the two-state Markov chain in figure 3.1. Assume the starting state of this Markov chain is s_1, if $1 - p_1 > 0$, then with nonzero probability we have

$$\pi_1(X) = s_1 s_1 ... s_1,$$

whose length is $N - 1$. But it is impossible to have

$$\pi_1(X) = s_2 s_2 ... s_2$$

of length $N - 1$. That means $\pi_1(X)$ is not an independent sequence. The main reason is that although each exit of a state will not affect the other exits, it will affect the length of the exit sequence. In fact, $\pi_1(X)$ is an independent sequence if the length of $\pi_1(X)$ is given, instead of giving the length of X.

In this chapter, we consider this problem from another perspective. According to lemma 3.2, we know that permutating the exit sequences does not change the probability of a sequence, however, the permuted sequence has to correspond to a trajectory in the Markov chain. The reason for this contingency is that in some cases the permuted sequence does not correspond to a trajectory: Consider the following example,

$$X = s_1 s_4 s_2 s_1 s_3 s_2 s_3 s_1 s_1 s_2 s_3 s_4 s_1,$$

and

$$\pi(X) = [s_4 s_3 s_1 s_2, s_1 s_3 s_3, s_2 s_1 s_4, s_2 s_1].$$

If we permute the last exit sequence $s_2 s_1$ to $s_1 s_2$, we cannot get a new sequence such that its starting state is s_1 and its exit sequences are

$$[s_4 s_3 s_1 s_2, s_1 s_3 s_3, s_2 s_1 s_4, s_1 s_2].$$

This can be verified by attempting to construct the sequence using Blum's method (which is given in the proof of lemma 3.1). Notice that if we permute the first exit sequence $s_4 s_3 s_1 s_2$ into $s_1 s_2 s_3 s_4$, we *can* find such a new sequence, which is

$$Y = s_1 s_1 s_2 s_1 s_3 s_2 s_3 s_1 s_4 s_2 s_3 s_4 s_1.$$

This observation motivated us to study the characterization of exit sequences that are feasible in Markov chains (or finite state machines).

Definition 3.1 (Feasibility)**.** *Given a Markov chain, a starting state s_α and a collection of sequences $\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_n]$, we say that (s_α, Λ) is feasible if and only if there exists a sequence X that corresponds to a trajectory in the Markov chain such that $x_1 = s_\alpha$ and $\pi(X) = \Lambda$.*

Based on the definition of feasibility, we present the main technical lemma of the chapter. Repeating the notation from the beginning of the chapter, we say that a sequence Y is a tail-fixed permutation of X, denoted as $Y \doteq X$, if and only if (1) Y is a permutation of X, and (2) X and Y have the same last element, namely, $y_{|Y|} = x_{|X|}$.

Lemma 3.3 (Main lemma: feasibility and equivalence of exit sequences)**.** *Given a starting state s_α and two collections of sequences $\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_n]$ and $\Gamma = [\Gamma_1, \Gamma_2, ..., \Gamma_n]$ such that $\Lambda_i \doteq \Gamma_i$ (tail-fixed permutation) for all $1 \leq i \leq n$. Then (s_α, Λ) is feasible if and only if (s_α, Γ) is feasible.*

The proof of this main lemma will be given in the next section. According to the main lemma,

we have the following equivalent statement.

Lemma 3.4 (Feasible permutations of exit sequences)**.** *Given an input sequence $X = x_1x_2...x_N$ with $x_N = s_\chi$ that produced from a Markov chain. Assume that $[\Lambda_1, \Lambda_2, ..., \Lambda_n]$ is an arbitrary collection of exit sequences that corresponds to the exit sequences of X as follows:*

1. *Λ_i is a permutation ($\equiv$) of $\pi_i(X)$, for $i = \chi$.*

2. *Λ_i is a tail-fixed permutation ($\doteq$) of $\pi_i(X)$, for $i \neq \chi$.*

Then there exists a feasible sequence $X' = x'_1x'_2...x'_N$ such that $x'_1 = x_1$ and $\pi(X') = [\Lambda_1, \Lambda_2, ..., \Lambda_n]$. For this X', we have $x'_N = x_N$.

One might reason that lemma 3.4 is stronger than the main lemma (Lemma 3.3). However, we will show that these two lemmas are equivalent. It is obvious that if the statement in lemma 3.4 is true, then the main lemma is also true. Now we show that if the main lemma is true then the statement in lemma 3.4 is also true.

Proof. Given $X = x_1x_2...x_N$, let us add one more symbol s_{n+1} to the end of X (s_{n+1} is different from all the states in X), then we can get a new sequence $x_1x_2...x_Ns_{n+1}$, whose exit sequences are

$$[\pi_1(X), \pi_2(X), ..., \pi_\chi(X)s_{n+1}, ..., \pi_n(X), \phi].$$

According to the main lemma, we know that there exists another sequence $x'_1x'_2...x'_Nx'_{N+1}$ such that its exit sequences are

$$[\Lambda_1, \Lambda_2, ..., \Lambda_\chi s_{n+1}, ..., \Lambda_n, \phi],$$

and $x'_1 = x_1$. Definitely, the last symbol of this sequence is s_{n+1}, i.e., $x'_{N+1} = s_{n+1}$. As a result, we have $x'_N = s_\chi$.

Now, by removing the last element from $x'_1x'_2...x'_Nx'_{N+1}$, we can get a new sequence $x = x'_1x'_2...x'_N$ such that its exit sequences are

$$[\Lambda_1, \Lambda_2, ..., \Lambda_\chi, ..., \Lambda_n],$$

and $x'_1 = x_1$. We also have $x'_N = s_\chi$.

This completes the proof. □

We demonstrate the result above by considering the example at the beginning of this section. Let

$$X = s_1 s_4 s_2 s_1 s_3 s_2 s_3 s_1 s_1 s_2 s_3 s_4 s_1,$$

with $\chi = 1$ and its exit sequences are given by

$$[s_4 s_3 s_1 s_2, s_1 s_3 s_3, s_2 s_1 s_4, s_2 s_1].$$

After permutating all the exit sequences (for $i \neq 1$, we keep the last element of the ith sequence fixed), we get a new group of exit sequences,

$$[s_1 s_2 s_3 s_4, s_3 s_1 s_3, s_1 s_2 s_4, s_2 s_1].$$

Based on these new exit sequences, we can generate a new input sequence,

$$X' = s_1 s_1 s_2 s_3 s_1 s_3 s_2 s_1 s_4 s_2 s_3 s_4 s_1.$$

This accords with the statements above.

3.3.2 Proof of the Main Lemma

Lemma 4 (Main lemma: feasibility and equivalence of exit sequences). *Given a starting state s_α and two collections of sequences $\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_n]$ and $\Gamma = [\Gamma_1, \Gamma_2, ..., \Gamma_n]$ such that $\Lambda_i \doteq \Gamma_i$ (tail-fixed permutation) for all $1 \leq i \leq n$. Then (s_α, Λ) is feasible if and only if (s_α, Γ) is feasible.*

In the rest of the appendix we will prove the main lemma. To illustrate the claim in the lemma, we express s_α and Λ by a directed graph that has labels on the vertices and edges, we call this graph a *sequence graph*. For example, when $s_\alpha = s_1$ and $\Lambda = [s_4 s_3 s_1 s_2, s_1 s_3 s_3, s_2 s_1 s_4, s_2 s_1]$, we have the

directed graph in figure 3.2.

Let V denote the vertex set, then

$$V = \{s_0, s_1, s_2, ..., s_n\},$$

and the edge set is

$$E = \{(s_i, \Lambda_i[k])\} \bigcup \{(s_0, s_\alpha)\}.$$

For each edge $(s_i, \Lambda_i[k])$, the label of this edge is k. For the edge (s_0, s_α), the label is 1. Namely, the label set of the outgoing edges of each state is $\{1, 2, ...\}$.

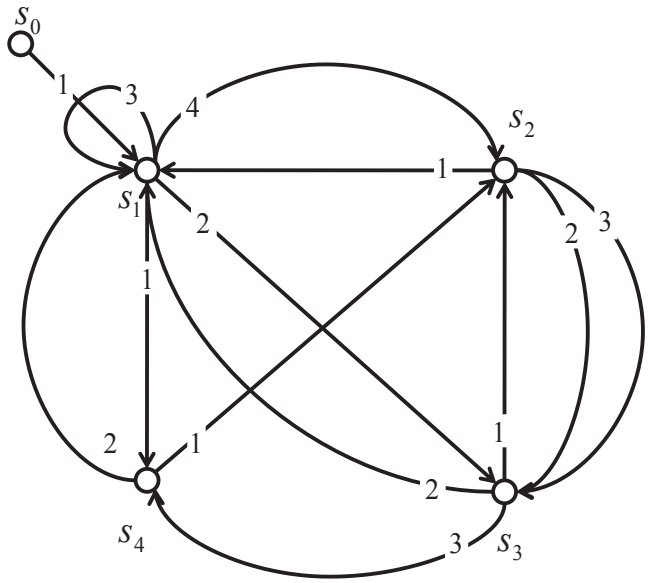


Figure 3.2. An example of a sequence graph G.

Given the labeling of the directed graph as defined above, we say that it contains a *complete walk* if there is a path in the graph that visits all the edges, without visiting an edge twice, in the following way: (1) Start from s_0. (2) At each vertex, we choose an unvisited edge with the minimal label to follow. Obviously, the labeling corresponding to (s_α, Λ) is a *complete walk* if and only if (s_α, Λ) is feasible. In this case, for short, we also say that (s_α, Λ) is a complete walk. Before continuing to prove the main lemma, we first give lemma 3.5 and lemma 3.6.

Lemma 3.5. *Assume* (s_α, Λ) *with* $\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_\chi, ..., \Lambda_n]$ *is a complete walk, which ends at state* s_χ*. Then* (s_α, Γ) *with* $\Gamma = [\Lambda_1, ..., \Gamma_\chi, ..., \Lambda_n]$ *is also a complete walk ending at* s_χ*, if* $\Lambda_\chi \equiv \Gamma_\chi$ *(permutation).*

Proof. (s_α, Λ) and (s_α, Γ) correspond to different labelings on the same directed graph G, denoted by L_1 and L_2. Since L_1 is a complete walk, it can travel all the edges in G one by one, denoted as

$$(s_{i_1}, s_{j_1}), (s_{i_2}, s_{j_2}), ..., (s_{i_N}, s_{j_N}),$$

where $s_{i_1} = s_0$ and $s_{j_N} = s_\chi$. We call $\{1, 2, ..., N\}$ as the indexes of the edges.

Based on L_2, let us have a walk on G starting from s_0 until there is no unvisited outgoing edges to select. In this walk, assume the following edges have been visited:

$$(s_{i_{w_1}}, s_{j_{w_1}}), (s_{i_{w_2}}, s_{j_{w_2}}), ..., (s_{i_{w_M}}, s_{j_{w_M}}),$$

where $w_1, w_2, ..., w_N$ are distinct indexes chosen from $\{1, 2, ..., N\}$ and $s_{i_{w_1}} = s_0$. In order to prove that L_2 is a complete walk, we need to show that (1) $s_{j_{w_M}} = s_\chi$ and (2) $M = N$.

First, let us prove that $s_{j_{w_M}} = s_\chi$. In G, let $N_i^{(out)}$ denote the number of outgoing edges of s_i and let $N_i^{(in)}$ denote the number of incoming edges of s_i, then we have that

$$\begin{cases} N_0^{(in)} = 0, N_0^{(out)} = 1, \\ N_\chi^{(in)} = N_\chi^{(out)} + 1, \\ N_i^{(in)} = N_i^{(out)} \text{ for } i \neq 0, i \neq \chi. \end{cases}$$

Based on these relations, we know that once we have a walk starting from s_0 in G, this walk will finally end at state s_χ. That is because we can always get out of s_i due to $N_i^{(in)} = N_i^{(out)}$ if $i \neq \chi, 0$.

Now, we prove that $M = N$. This can be proved by contradiction. Assume $M \neq N$, then we define

$$V = \{w_1, w_2, ..., w_M\},$$

$$\overline{V} = \{1, 2, ..., N\}/\{w_1, w_2, ..., w_M\},$$

where V corresponds to the visited edges based on L_2 and $\overline{V}$ corresponds to the unvisited edges based on L_2. Let $v = \min(\overline{V})$, then (s_{i_v}, s_{j_v}) is the unvisited edge with the minimal index. Let $l = i_v$, then (s_{i_v}, s_{j_v}) is an outgoing edge of s_l. Here $l \neq \chi$, because all the outgoing edges of s_χ have been visited. Assume the number of visited incoming edges of s_l is $M_l^{(in)}$ and the number of visited outgoing edges of s_l is $M_l^{(out)}$, then

$$M_l^{(in)} = M_l^{(out)},$$

see figure 3.3 as an example, in which the solid arrows indicate visited edges, and the dashed arrows indicate unvisited edges..

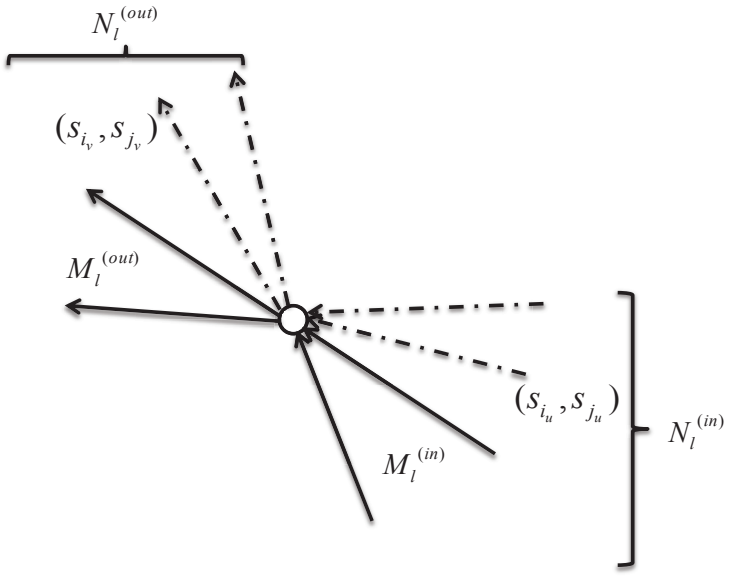


Figure 3.3. An illustration of the incoming and outgoing edges of s_l.

Note that the labels of the outgoing edges of s_l are the same for L_1 and L_2, since $l \neq \chi, 0$. Therefore, based on L_1, before visiting edge (s_{i_v}, s_{j_v}), there must be $M_l^{(out)}$ outgoing edges of s_l have been visited. As a result, based on L_1, there must be $M_l^{(out)} + 1 = M_l^{(in)} + 1$ incoming edges of s_l have been visited before visiting (s_{i_v}, s_{j_v}). Among all these $M_l^{(in)} + 1$ incoming edges, there exists at least one edge (s_{i_u}, s_{j_u}) such that $u \in \overline{V}$, since only $M_l^{(in)}$ incoming edges of s_l have been

visited based on L_2.

According to our assumption, both $u, v \in \overline{V}$ and v is the minimal one, so $u > v$. On the other hand, we know that (s_{i_u}, s_{j_u}) is visited before (s_{i_v}, s_{j_v}) based on L_1, so $u < v$. Here, the contradiction happens. Therefore, $M = N$.

This completes the proof. □

Here, let us give an example of the lemma above. We know that, when $s_\alpha = s_1$ and

$$\Lambda = [s_4 s_3 s_1 s_2, s_1 s_3 s_3, s_2 s_1 s_4, s_2 s_1],$$

(s_α, Λ) is feasible. The labeling on a directed graph corresponding to (s_α, Λ) is given in figure 3.2, which is a complete walk starting at state s_0 and ending at state s_1. The path of the walk is

$$s_0 s_1 s_4 s_2 s_1 s_3 s_2 s_3 s_1 s_1 s_2 s_3 s_4 s_1.$$

By permutating the labels of the outgoing edges of s_1, we can have the graph as shown in figure 3.4. The new labeling on G is also a complete walk ending at state s_1, and its path is

$$s_0 s_1 s_1 s_2 s_1 s_3 s_2 s_3 s_1 s_4 s_2 s_3 s_4 s_1.$$

Based on lemma 3.5, we have the following result

Lemma 3.6. *Given a starting state s_α and two collections of sequences $\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_k, ..., \Lambda_n]$ and $\Gamma = [\Lambda_1, ..., \Gamma_k, ..., \Lambda_n]$ such that $\Gamma_k \doteq \Lambda_k$ (tail-fixed permutation). Then (s_α, Λ) and (s_α, Γ) have the same feasibility.*

Proof. We prove that if (s_α, Λ) is feasible, then (s_α, Γ) is also feasible. If (s_α, Λ) is feasible, there exists a sequence X such that $s_\alpha = x_1$ and $\Lambda = \pi(X)$. Suppose its last element is $x_N = s_\chi$.

When $k = \chi$, according to lemma 3.5, we know that (s_α, Γ) is feasible.

When $k \neq \chi$, we assume that $\Lambda_k = \pi_k(X) = x_{k_1} x_{k_2} ... x_{k_w}$. We consider the subsequence

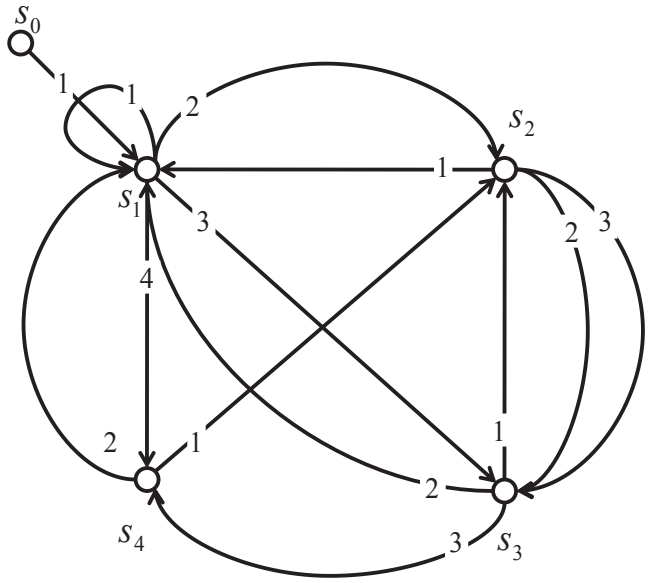


Figure 3.4. The sequence graph G with new labels.

$\overline{X} = x_1x_2...x_{k_w-1}$ of X. Then $\pi_k(\overline{X}) = \Lambda_k^{|\Lambda_k|-1}$ and the last element of $\overline{X}$ is s_k. According to lemma 3.5, we can get that there exists a sequence $x'_1x'_2...x'_{k_w-1}$ with $x'_1 = x_1$ and $x'_{k_w-1} = x_{k_w-1}$ such that

$$\pi(x'_1x'_2...x'_{k_w-1}) = [\pi_1(\overline{X}), ..., \Gamma_k^{|\Gamma_k|-1}, \pi_{k+1}(\overline{X}), ..., \pi_n(\overline{X})],$$

since $\Gamma_k^{|\Gamma_k|-1} \equiv \Lambda_k^{|\Lambda_k|-1}$.

Let $x'_{k_w}x'_{k_w+1}...x'_N = x_{k_w}x_{k_w+1}...x_N$, i.e., concatenating $x_{k_w}x_{k_w+1}...x_N$ to the end of $x'_1x'_2...x'_{k_w-1}$, we can generate a sequence $x'_1x'_2...x'_N$ such that its exit sequence of state s_k is

$$\Gamma_k^{|\Gamma_k|-1} * x_{k_w} = \Gamma_k,$$

and its exit sequence of state s_i with $i \neq k$ is $\Lambda_i = \pi_i(X)$.

So if (s_α, Λ) is feasible, then (s_α, Γ) is also feasible. Similarly, if (s_α, Γ) is feasible, then (s_α, Λ) is feasible. As a result, (s_α, Λ) and (s_α, Γ) have the same feasibility. □

According to the lemma above, we know that

$(s_\alpha, [\Lambda_1, \Lambda_2, ..., \Lambda_n])$ and $(s_\alpha, [\Gamma_1, \Lambda_2, ..., \Lambda_n])$ have the same feasibility,

$(s_\alpha, [\Gamma_1, \Lambda_2, ..., \Lambda_n])$ and $(s_\alpha, [\Gamma_1, \Gamma_2, ..., \Lambda_n])$ have the same feasibility,

... ,

$(s_\alpha, [\Gamma_1, \Gamma_2, ..., \Gamma_{n-1}, \Lambda_n])$ and $(s_\alpha, [\Gamma_1, \Gamma_2, ..., \Gamma_{n-1}, \Gamma_n])$ have the same feasibility.

So the statement in the main lemma is true.

3.4 Algorithm A: Modification of Elias's Suggestion

Elias suggested to generate random bits from an arbitrary Markov chain by concatenating the outputs of different exit sequences. In the above section, we showed that direct concatenation cannot always work. This motivates us to derive algorithm A, which is a simple modification of Elias's suggestion and is able to generate random bits from any Markov chain efficiently.

Algorithm A

Input: A sequence $X = x_1x_2...x_N$ produced by a Markov chain, where $x_i \in S = \{s_1, s_2, ..., s_n\}$.

Output: A sequence Y of $0'$s and $1'$s.

Main Function:

Suppose $x_N = s_\chi$.

for $i := 1$ to n **do**

if $i = \chi$ **then**

Output $\Psi(\pi_i(X))$.

else

Output $\Psi(\pi_i(X)^{|\pi_i(X)|-1})$

end if

end for

Comment: (1) $\Psi(X)$ can be any scheme that generates random bits from biased coins. For example, we can use the Elias function. (2) When $i = \chi$, we can also output $\Psi(\pi_i(X)^{|\pi_i(X)|-1})$ for simplicity, but the efficiency may be reduced a little.

The only difference between algorithm A and direct concatenation is that: Algorithm A ignores the last symbols of some exit sequences. Let us go back to the example of a two-state Markov chain with $P[s_2|s_1] = p_1$ and $P[s_1|s_2] = p_2$ in figure 3.1, which demonstrates that direct concatenation does not always work well. Here, still assuming that an input sequence with length $N = 4$ is generated from this Markov chain with starting state s_1, then the probability of each possible input sequence and its corresponding output sequence (based on algorithm A) are given by the following.

Input sequence	Probability	Output sequence
$s_1s_1s_1s_1$	$(1-p_1)^3$	ϕ
$s_1s_1s_1s_2$	$(1-p_1)^2p_1$	ϕ
$s_1s_1s_2s_1$	$(1-p_1)p_1p_2$	0
$s_1s_1s_2s_2$	$(1-p_1)p_1(1-p_2)$	ϕ
$s_1s_2s_1s_1$	$p_1p_2(1-p_1)$	1
$s_1s_2s_1s_2$	$p_1^2p_2$	ϕ
$s_1s_2s_2s_1$	$p_1(1-p_2)p_2$	ϕ
$s_1s_2s_2s_2$	$p_1(1-p_2)^2$	ϕ

We can see that when the input sequence length $N = 4$, a bit 0 and a bit 1 have the same probability of being generated and no longer sequences are generated. In this case, the output sequence is independent and unbiased.

In order to prove that all the sequences generated by algorithm A are independent and unbiased, we need to show that for any sequences Y and Y' of the same length, they have the same probability of being generated.

Theorem 3.7 (Algorithm A)**.** *Let the sequence generated by a Markov chain be used as input to algorithm A, then the output of algorithm A is an independent unbiased sequence.*

Proof. Let us first divide all the possible sequences in $\{s_1, s_2, ..., s_n\}^N$ into classes, and use G to denote the set of the classes. Two sequences X and X' are in the same class if and only if

1. $x'_1 = x_1$ and $x'_N = x_N = s_\chi$ for some χ.

2. If $i = \chi$, $\pi_i(X') \equiv \pi_i(X)$.

3. If $i \neq \chi$, $\pi_i(X') \doteq \pi_i(X)$.

Let us use Ψ_A to denote algorithm A. For $Y \in \{0,1\}^*$, let B_Y be the set of sequences X of length N such that $\Psi_A(X) = Y$. We show that for any $S \in G$, $|S \bigcap B_Y| = |S \bigcap B_{Y'}|$ whenever $|Y| = |Y'|$. If S is empty, this conclusion is trivial. In the following, we only consider the case that S is not empty.

Now, given a class S, if $i = \chi$ let us define S_i as the set consisting of all the permutations of $\pi_i(X)$ for $X \in S$, and if $i \neq \chi$ let us define S_i as the set consisting of all the permutations of $\pi_i(X)^{|\pi_i(X)|-1}$ for $X \in S$. For all $1 \leq i \leq n$ and $Y_i \in \{0,1\}^*$, we continue to define

$$S_i(Y_i) = \{\Lambda_i \in S_i | \Psi(\Lambda_i) = Y_i\},$$

which is the subset of S_i consisting of all sequences yielding Y_i. Based on lemma 2.1, we know that $|S_i(Y_i)| = |S_i(Y_i')|$ whenever $|Y_i| = |Y_i'|$. This implies that $|S_i(Y_i)|$ is a function of $|Y_i|$, which can be written as $M_i(|Y_i|)$.

For any partition of Y, namely $Y_1, Y_2, ..., Y_n$ such that $Y_1 * Y_2 * ... * Y_n = Y$, we have the following conclusion: $\forall \Lambda_1 \in S_1(Y_1), \Lambda_2 \in S_2(Y_2), ..., \Lambda_n \in S_n(Y_n)$, we can always find a sequence $X \in S \bigcap B_Y$ such that $\pi_i(X) = \Lambda_i$ for $i = \chi$ and $\pi_i(X)^{|\pi_i(X)|-1} = \Lambda_i$ for all $i \neq \chi$. This conclusion is immediate from lemma 3.4. As a result, we have

$$|S \bigcap B_Y| = \sum_{Y_1 * Y_2 * ... * Y_n = Y} \prod_{i=1}^{n} |S_i(Y_i)|.$$

Let $l_1, l_2, ..., l_n$ be a group of nonnegative integers partitioning $|Y|$, then the formula above can be rewritten as

$$|S \bigcap B_Y| = \sum_{l_1 + ... + l_n = |Y|} \prod_{i=1}^{n} M_i(l_i).$$

Similarly, we also have

$$|S \bigcap B_{Y'}| = \sum_{l_1+...+l_n=|Y'|} \prod_{i=1}^{n} M_i(l_i),$$

which tells us that $|S \bigcap B_Y| = |S \bigcap B_{Y'}|$ if $|Y| = |Y'|$.

Note that all the sequences in the same class S have the same probability of being generated. So when $|Y| = |Y'|$, the probability of generating Y is

$$\begin{aligned} & P[X \in B_Y] \\ = & \sum_{S \in G} P[S] \sum_{X \in S} P[X \in B_Y | X \in S] \\ = & \sum_{S \in G} P[S] \sum_{X \in S} \frac{|S \bigcap B_Y|}{|S|} \\ = & \sum_{S \in G} P[S] \sum_{X \in S} \frac{|S \bigcap B_{Y'}|}{|S|} \\ = & P[X \in B_{Y'}], \end{aligned}$$

which implies that output sequence is independent and unbiased. □

Theorem 3.8 (Efficiency)**.** *Let X be a sequence of length N generated by a Markov chain, which is used as input to algorithm A. Let Ψ in algorithm A be Elias's function. Suppose the length of its output sequence is M, then the limiting efficiency $\eta_N = \frac{E[M]}{N}$ as $N \to \infty$ realizes the upper bound $\frac{H(X)}{N}$.*

Proof. Here, the upper bound $\frac{H(X)}{N}$ is provided by Elias [33]. We can use the same argument in Elias's paper [33] to prove this theorem.

For all $1 \leq i \leq n$, let X_i denote the next state following s_i in the Markov chain. Then X_i is a random variable on $\{s_1, s_2, ..., s_n\}$ with distribution $\{p_{i1}, p_{i2}, ..., p_{in}\}$, where p_{ij} with $1 \leq i, j \leq n$ is the transition probability from state s_i to state s_j. The entropy of X_i is denoted as $H(X_i)$. Let $U = (u_1, u_2, \ldots, u_n)$ denote the stationary distribution of the Markov chain, then we have [27]

$$\lim_{N \to \infty} \frac{H(X)}{N} = \sum_{i=1}^{n} u_i H(X_i).$$

When $N \to \infty$, there exists an ϵ_N which $\to 0$, such that with probability $1 - \epsilon_N$, $|\pi_i(X)| > (u_i - \epsilon_N)N$ for all $1 \leq i \leq n$. Using algorithm A, with probability $1 - \epsilon_N$, the length M of the output sequence is bounded below by

$$\sum_{i=1}^{n}(1-\epsilon_N)(|\pi_i(X)|-1)\eta_i,$$

where η_i is the efficiency of the Ψ when the input is $\pi_i(X)$ or $\pi_i(X)^{|\pi_i(X)|-1}$. According to theorem 2 in Elias's paper [33], we know that as $|\pi_i(X)| \to \infty$, $\eta_i \to H(X_i)$. So with probability $1 - \epsilon_N$, the length M of the output sequence is bounded from below by

$$\sum_{i=1}^{N}(1-\epsilon_N)((u_i-\epsilon_N)N-1)(1-\epsilon_N)H(X_i).$$

Then we have

$$\begin{aligned} & \lim_{N\to\infty} \frac{E[M]}{N} \\ \geq & \lim_{N\to\infty} \frac{[\sum_{i=1}^{N}(1-\epsilon_N)^3((u_i-\epsilon_N)N-1)H(X_i)]}{N} \\ = & \lim_{N\to\infty} \frac{H(X)}{N}. \end{aligned}$$

At the same time, $\frac{E[M]}{N}$ is upper bounded by $\frac{H(X)}{N}$. So we can get

$$\lim_{N\to\infty} \frac{E[M]}{N} = \lim_{N\to\infty} \frac{H(X)}{N},$$

which completes the proof. □

Given an input sequence, it is efficient to generate independent unbiased sequences using algorithm A. However, it has some limitations: (1) The complete input sequence has to be stored. (2) For a long input sequence it is computationally intensive as it depends on the input length. (3) The method works for finite-length sequences and does not lend itself to stream processing. In order to address these limitations we propose two variants of algorithm A.

In the first variant of algorithm A, instead of applying Ψ directly to $\Lambda_i = \pi_i(X)$ for $i = \chi$ (or $\Lambda_i = \pi_i(X)^{|\pi_i(X)|-1}$ for $i \neq \chi$), we first split Λ_i into several segments with lengths $k_{i1}, k_{i2}, ...$, then apply Ψ to all of the segments separately. It can be proved that this variant of algorithm A can generate independent unbiased sequences from an arbitrary Markov chain, as long as $k_{i1}, k_{i2}, ...$ do not depend on the order of elements in each exit sequence. For example, we can split Λ_i into two segments of lengths $\lfloor \frac{|\Lambda_i|}{2} \rfloor$ and $\lceil \frac{|\Lambda_i|}{2} \rceil$, we can also split it into three segments of lengths $(a, a, |\Lambda_i|-2a)$ .... Generally, the shorter each segment is, the faster we can obtain the final output. But at the same time, we may have to sacrifice a little information efficiency.

The second variant of algorithm A is based on the following idea: for a given sequence from a Markov chain, we can split it into some shorter sequences such that they are independent of each other, therefore we can apply algorithm A to all of the sequences and then concatenate their output sequences together as the final one. In order to do this, given a sequence $X = x_1x_2...$, we can use $x_1 = s_\alpha$ as a special state to it. For example, in practice, we can set a constant k, if there exists a minimal integer i such that $x_i = s_\alpha$ and $i > k$, then we can split X into two sequences $x_1x_2...x_i$ and $x_ix_{i+1}...$ (note that both of the sequences have the element x_i). For the second sequence $x_ix_{i+1}...$, we can repeat the same procedure. Iteratively, we can split a sequence X into several sequences such that they are independent of each other. These sequences, with the exception of the last one, start and end with s_α, and their lengths are usually slightly longer than k.

3.5 Algorithm B: Generalization of Blum's Algorithm

In [14], Blum proposed a beautiful algorithm to generate an independent unbiased sequence of 0s and 1s from any Markov chain by extending the von Neumann's scheme. His algorithm can deal with infinitely long sequences and uses only constant space and expected linear time. The only drawback of his algorithm is that its efficiency is still far from the information-theoretic upper bound, due to the limitation (compared to the Elias algorithm) of the von Neumann's scheme. In this section, we generalize Blum's algorithm by replacing von Neumann's scheme with Elias's. As a result, we get algorithm B: It maintains some good properties of Blum's algorithm and its efficiency approaches

the information-theoretic upper bound.

Algorithm B

Input: A sequence (or a stream) $x_1x_2...$ produced by a Markov chain, where $x_i \in \{s_1, s_2, ..., s_n\}$.

Parameter: n positive integer functions (window size) $\varpi_i(k)$ with $k \geq 1$ for all $1 \leq i \leq n$.

Output: A sequence (or a stream) Y of 0s and 1s.

Main Function:

$E_i = \phi$ (empty) for all $1 \leq i \leq n$.

$k_i = 1$ for all $1 \leq i \leq n$.

c : the index of current state, namely, $s_c = x_1$.

while next input symbol is s_j ($\neq$ **null**) **do**

$E_c = E_c s_j$ (Add s_j to E_c).

if $|E_j| \geq \varpi_j(k_j)$ **then**

Output $\Psi(E_j)$.

$E_j = \phi$.

$k_j = k_j + 1$.

end if

$c = j$.

end while

In the algorithm above, we apply function Ψ on E_j to generate random bits if and only if the window for E_j is completely filled and the Markov chain is currently at state s_j.

For example, we set $\varpi_i(k) = 4$ for all $1 \leq i \leq n$ and for all $k \geq 1$ and assume that the input sequence is

$$X = s_1s_1s_1s_2s_2s_2s_1s_2s_2.$$

After reading the second to last (8th) symbol s_2, we have

$$E_1 = s_1 s_1 s_2 s_2, \quad E_2 = s_2 s_2 s_1.$$

In this case, $|E_1| \geq 4$ so the window for E_1 is full, but we do not apply Ψ to E_1 because the current state of the Markov chain is s_2, not s_1.

By reading the last (9th) symbol s_2, we get

$$E_1 = s_1 s_1 s_2 s_2, \quad E_2 = s_2 s_2 s_1 s_2.$$

Since the current state of the Markov chain is s_2 and $|E_2| \geq 4$, we produce $\Psi(E_2 = s_2 s_2 s_1 s_2)$ and reset E_2 as ϕ.

In the example above, treating X as input to algorithm B, we get the output sequence $\Psi(s_2 s_2 s_1 s_2)$. The algorithm does not output $\Psi(E_1 = s_1 s_1 s_2 s_2)$ until the Markov chain reaches state s_1 again. Timing is crucial!

Note that Blum's algorithm is a special case of algorithm B by setting the window size functions $\varpi_i(k) = 2$ for all $1 \leq i \leq n$ and $k \in \{1, 2, ...\}$. Namely, algorithm B is a generalization of Blum's algorithm, the key is that when we increase the windows sizes, we can apply more efficient schemes (compared to the von Neumann's scheme) for Ψ. Assume a sequence of symbols $X = x_1 x_2 ... x_N$ with $x_N = s_\chi$ have been read by the algorithm above, we want to show that for any N, the output sequence is always independent and unbiased. Unfortunately, Blum's proof for the case of $\varpi_i(k) = 2$ cannot be applied to our proposed scheme.

For all i with $1 \leq i \leq n$, we can write

$$\pi_i(X) = F_{i1} F_{i2} ... F_{im_i} E_i,$$

where F_{ij} with $1 \leq j \leq m_i$ are the segments used to generate outputs. For all i, j, we have

$$|F_{ij}| = \varpi_i(j),$$

and

$$\begin{cases} 0 \leq |E_i| < \varpi_i(m_i + 1) & \text{if } i = \chi, \\ 0 < |E_i| \leq \varpi_i(m_i + 1) & \text{otherwise.} \end{cases}$$

See figure 3.5 for simple illustration.

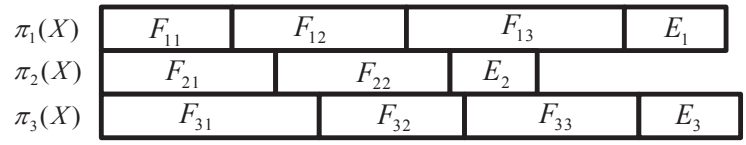


Figure 3.5. The simplified expressions for the exit sequences of X.

Theorem 3.9 (Algorithm B)**.** *Let the sequence generated by a Markov chain be used as input to algorithm B, then algorithm B generates an independent unbiased sequence of bits in expected linear time.*

Proof. In the following proof, we use the same idea as in the proof for algorithm A.

Let us first divide all the possible input sequences in $\{s_1, s_2, ..., s_n\}^N$ into classes, and use G to denote the set consisting of all the classes. Two sequences X and X' are in the same class if and only if

1. $x_1 = x'_1$ and $x_N = x'_N$.

2. For all i with $1 \leq i \leq n$,

$$\pi_i(X) = F_{i1}F_{i2}...F_{im_i}E_i,$$

$$\pi_i(X') = F'_{i1}F'_{i2}...F'_{im_i}E'_i,$$

where F_{ij} and F'_{ij} are the segments used to generate outputs.

3. For all i, j, $F_{ij} \equiv F'_{ij}$.

4. For all i, $E_i = E'_i$.

Let us use Ψ_B to denote algorithm B. For $Y \in \{0,1\}^*$, let B_Y be the set of sequences X of length N such that $\Psi_B(X) = Y$. We show that for any $S \in G$, $|S \bigcap B_Y| = |S \bigcap B_{Y'}|$ whenever $|Y| = |Y'|$. If S is empty, this conclusion is trivial. In the following, we only consider the case that S is not empty.

Now, given a class S, let us define S_{ij} as the set consisting of all the permutations of F_{ij} for $X \in S$. Given $Y_{ij} \in \{0,1\}^*$, we continue to define

$$S_{ij}(Y_{ij}) = \{\Lambda_{ij} \in S_{ij} | \Psi(\Lambda_{ij}) = Y_{ij}\}$$

for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$, which is the subset of S_{ij} consisting of all sequences yielding Y_{ij}. According to lemma 2.1, we know that $|S_{ij}(Y_{ij})| = |S_{ij}(Y'_{ij})|$ whenever $|Y_{ij}| = |Y'_{ij}|$. This implies that $|S_{ij}(Y_{ij})|$ is a function of $|Y_{ij}|$, which can be written as $M_{ij}(|Y_{ij}|)$.

Let $l_{11}, l_{12}, ..., l_{1m_1}, l_{21} ..., l_{nm_n}$ be nonnegative integers such that their sum is $|Y|$, we want to prove that

$$|S \bigcap B_Y| = \sum_{l_{11}+...+l_{nm_n}=|Y|} \prod_{i=1}^{n} \prod_{j=1}^{m_i} M_{ij}(l_{ij}).$$

The proof is by induction. Let $w = \sum_{i=1}^{n} m_i$. First, the conclusion holds for $w = 1$. Assume the conclusion holds for $w > 1$, we want to prove that the conclusion also holds for $w + 1$.

Note that for all $1 \leq i \leq n$, if $j_1 < j_2$, then F_{ij_1} generates an output before F_{ij_2} in algorithm B. So given an input sequence $X \in S$, the last segment that generates an output (the output can be an empty string) is F_{im_i} for some i with $1 \leq i \leq n$. Now, we show that this i is fixed for all the sequences in S, i.e., the position of the last segment generating an output keeps unchanged. To prove this, given a sequence $X \in S$, let us see the first a symbols of X, i.e., X^a, such that the last segment F_{im_i} generates an output just after reading x_a when the input sequence is X. Based on our algorithm, X^a has the following properties.

1. The last symbol $x_a = s_i$.

2. $\pi_i(X^a) = F_{i1}F_{i2}...F_{im_i}$.

3. $\pi_j(X^a) = F_{j1}F_{j2}...F_{jm_j}\widetilde{E}_j$ for $j \neq i$, where $|\widetilde{E}_j| > 0$.

Now, let us permute each segment of $F_{11}, F_{12}, ..., F_{nm_n}$ to $F'_{11}, F'_{12}, ..., F'_{nm_n}$, then we get another sequence $X' \in S$. According to lemma 3.4, if we consider the first a symbols of X', i.e., X'^a, it has the similar properties as X^a:

1. The last symbol $x'_a = s_i$.

2. $\pi_i(X'^a) = F'_{i1}F'_{i2}...F'_{im_i}$.

3. $\pi_j(X'^a) = F'_{j1}F'_{j2}...F'_{jm_j}\widetilde{E}_j$ for $j \neq i$, where $|\widetilde{E}_j| > 0$.

This implies that when the input sequence is X', F'_{im_i} generates an output just after reading x'_a and it is the last one. So we can conclude that for all the sequences in S, their last segments generating outputs are at the same position.

Let us fix the last segment F_{im_i} and assume F_{im_i} generates the last l_{im_i} bits of Y. We want to know how many sequences in $S \bigcap B_Y$ have F_{im_i} as their last segments that generate outputs. In order to get the answer, we concatenate F_{im_i} with E_i as the new E_i. As a result, we have $\sum_{i=1}^{n} m_i - 1 = w$ segments to generate the first $|Y| - l_{im_i}$ bits of Y. Based on our assumption, the number of such sequences will be

$$\sum_{l_{11}+...+l_{i(m_i-1)}+...=|Y|-l_{im_i}} \frac{1}{M_{im_i}(l_{im_i})} \prod_{k=1}^{n} \prod_{j=1}^{m_i} M_{kj}(l_{kj}),$$

where $l_{11}, ..., l_{i(m_i-1)}, l_{(i+1)1}..., l_{nm_n}$ are nonnegative integers. For each l_{im_i}, there are $M_{im_i}(l_{im_i})$ different choices for F_{im_i}. Therefore, $|S \bigcap B_Y|$ can be obtained by multiplying $M_{im_i}(l_{im_i})$ by the number above and summing them up over l_{im_i}. Namely, we can get the conclusion above.

According to this conclusion, we know that if $|Y| = |Y'|$, then $|S \bigcap B_Y| = |S \bigcap B_{Y'}|$. Using the same argument as in Theorem 3.7 we complete the proof of the theorem. □

Normally, the window size functions $\varpi_i(k)$ for $1 \leq i \leq n$ can be any positive integer functions. Here, we fix these window size functions as a constant, namely, ϖ. By increasing the value of ϖ, we can increase the efficiency of the scheme, but at the same time it may cost more storage space and need more waiting time. It is helpful to analyze the relationship between scheme efficiency and window size ϖ.

Theorem 3.10 (Efficiency)**.** *Let X be a sequence of length N generated by a Markov chain with transition matrix P, which is used as input to algorithm B with constant window size ϖ. Then as the length of the sequence goes to infinity, the limiting efficiency of algorithm B is*

$$\eta(\varpi) = \sum_{i=1}^{n} u_i \eta_i(\varpi),$$

where $U = (u_1, u_2, ..., u_n)$ is the stationary distribution of this Markov chain, and $\eta_i(\varpi)$ is the efficiency of Ψ when the input sequence of length ϖ is generated by a n-face coin with distribution $(p_{i1}, p_{i2}, ..., p_{in})$.

Proof. When $N \to \infty$, there exists an ϵ_N which $\to 0$, such that with probability $1-\epsilon_N$, $(u_i-\epsilon_N)N < |\pi_i(X)| < (u_i+\epsilon_N)N$ for all $1 \leq i \leq n$.

The efficiency of algorithm B can be written as $\eta(\varpi)$, which satisfies

$$\frac{\sum_{i=1}^{n} \lfloor \frac{|\pi_i(X)|-1}{\varpi} \rfloor \eta_i(\varpi)\varpi}{N} \leq \eta(\varpi) \leq \frac{\sum_{i=1}^{n} \lfloor \frac{|\pi_i(X)|}{\varpi} \rfloor \eta_i(\varpi)\varpi}{N}.$$

With probability $1-\epsilon_N$, we have

$$\frac{\sum_{i=1}^{n} (\frac{(u_i-\epsilon_N)N}{\varpi} - 1)\eta_i(\varpi)\varpi}{N} \leq \eta(\varpi) \leq \frac{\sum_{i=1}^{n} \frac{(u_i-\epsilon_N)N}{\varpi} \eta_i(\varpi)\varpi}{N}.$$

So when $N \to \infty$, we have that

$$\eta(\varpi) = \sum_{i=1}^{n} u_i \eta_i(\varpi).$$

This completes the proof. □

Let us define $\alpha(N) = \sum n_k 2^{n_k}$, where $\sum 2^{n_k}$ is the standard binary expansion of N. Assume Ψ is the Elias function, then

$$\eta_i(\varpi) = \frac{1}{\varpi} \sum_{k_1+...+k_n=\varpi} \alpha(\frac{\varpi!}{k_1!k_2!...k_n!}) p_{i1}^{k_1} p_{i2}^{k_2} ... p_{in}^{k_n}.$$

Based on this formula, we can numerically study the relationship between the limiting efficiency and the window size (see section 3.7). In fact, when the window size becomes large, the limiting efficiency ($n \to \infty$) approaches the information-theoretic upper bound.

3.6 Algorithm C: An Optimal Algorithm

Both algorithm A and algorithm B are asymptotically optimal, but when the length of the input sequence is finite they may not be optimal. In this section, we try to construct an optimal algorithm, called algorithm C, such that its information efficiency is maximized when the length of the input sequence is finite. Before presenting this algorithm, following the idea of Pae and Loui [88], we first discuss the equivalent condition for a function f to generate random bits from an arbitrary Markov chain, and then present the sufficient condition for f to be optimal.

Lemma 3.11 (Equivalent condition)**.** *Let $K = \{k_{ij}\}$ be an $n \times n$ nonnegative integer matrix with $\sum_{i=1}^{n} \sum_{j=1}^{n} k_{ij} = N - 1$. We define $S_{(\alpha,K)}$ as*

$$S_{(\alpha,K)} = \{X \in \{s_1, s_2, ..., s_n\}^N | k_j(\pi_i(X)) = k_{ij}, x_1 = s_\alpha\},$$

where $k_j(X)$ is the number of s_j in X. A function $f : \{s_1, s_2, ..., s_n\}^N \to \{0,1\}^*$ can generate random bits from an arbitrary Markov chain, if and only if for any (α, K) and two binary sequences Y and Y' with $|Y| = |Y'|$,

$$|S_{(\alpha,K)} \bigcap B_Y| = |S_{(\alpha,K)} \bigcap B_{Y'}|,$$

where $B_Y = \{X | X \in \{s_1, s_2, ..., s_n\}^N, f(X) = Y\}$ is the set of sequences of length N that yield Y.

Proof. It is easy to see that if $|S_{(\alpha,K)} \bigcap B_Y| = |S_{(\alpha,K)} \bigcap B_{Y'}|$ for all (α, K) and $|Y| = |Y'|$, then Y and Y' have the same probability to be generated. In this case, f can generate random bits from an arbitrary Markov chain. In the rest, we only need to prove the inverse claim.

If f can generate random bits from an arbitrary Markov chain, then $P[f(X) = Y] = P[f(X) = Y']$ for any two binary sequences Y and Y' of the same length. Here, let p_{ij} be the transition probability from state s_i to state s_j for all $1 \leq i, j \leq n$, we can write

$$P[f(X) = Y] = \sum_{\alpha,K \in G} |S_{(\alpha,K)} \bigcap B_Y| \phi_K(p_{11}, p_{12}, ..., p_{nn}) P(x_1 = s_\alpha),$$

where

$$G = \{K | k_{ij} \in \{0\} \bigcup \mathbb{Z}^+, \sum_{i,j} k_{ij} = N - 1\},$$

and

$$\phi_K(p_{11}, p_{12}, ..., p_{nn}) = \prod_{i=1}^{n} \prod_{j=1}^{n} p_{ij}^{k_{ij}}.$$

Similarly,

$$P[f(X) = Y'] = \sum_{\alpha,K \in G} |S_{(\alpha,K)} \bigcap B_{Y'}| \phi_K(p_{11}, p_{12}, ..., p_{nn}) P(x_1 = s_\alpha).$$

As a result,

$$\sum_{\alpha,K \in G} (|S_{(\alpha,K)} \bigcap B_{Y'}| - |S_{(\alpha,K)} \bigcap B_Y|) \phi_K(p_{11}, ..., p_{nn}) \times P(x_1 = s_\alpha) = 0.$$

Since $P(x_1 = s_\alpha)$ can be any value in $[0, 1]$, for all $1 \leq \alpha \leq n$ we have

$$\sum_{K \in G} (|S_{(\alpha,K)} \bigcap B_{Y'}| - |S_{(\alpha,K)} \bigcap B_Y|) \phi_K(p_{11}, ..., p_{nn}) = 0.$$

It can be proved that $\bigcup_{K \in G} \{\phi_K(p_{11}, p_{12}, ..., p_{nn})\}$ are linear independent in the vector space of

functions on the transition probabilities, namely

$$\{(p_{11}, p_{12}, ..., p_{nn}) | p_{ij} \in [0,1], \sum_{j=1}^{n} p_{ij} = 1\}.$$

Based on this fact, we can conclude that $|S_{(\alpha,K)} \bigcap B_Y| = |S_{(\alpha,K)} \bigcap B_{Y'}|$ for all (α, K) if $|Y| = |Y'|$.

□

Let us define $\alpha(N) = \sum n_k 2^{n_k}$, where $\sum 2^{n_k}$ is the standard binary expansion of N, then we have the sufficient condition for an optimal function .

Lemma 3.12 (Sufficient condition for an optimal function)**.** *Let f^* be a function that generates random bits from an arbitrary Markov chain with unknown transition probabilities. If for any α and any $n \times n$ nonnegative integer matrix K with $\sum_{i=1}^{n} \sum_{j=1}^{n} k_{ij} = N - 1$, the following equation is satisfied,*

$$\sum_{X \in S_{(\alpha,K)}} |f^*(X)| = \alpha(|S_{(\alpha,K)}|),$$

then f^* generates independent unbiased random bits with optimal information efficiency. Note that $|f^*(X)|$ is the length of $f^*(x)$ and $|S_{(\alpha,K)}|$ is the size of $S_{(\alpha,K)}$.

Proof. Let h denote an arbitrary function that is able to generate random bits from any Markov chain. According to lemma 2.9 in [88], we know that

$$\sum_{X \in S_{(\alpha,K)}} |h(X)| \leq \alpha(|S_{(\alpha,K)}|).$$

Then the average output length of h is

$$\begin{aligned} E(|h(X)|) &= \frac{1}{N} \sum_{(\alpha,K)} \sum_{X \in S_{(\alpha,K)}} |h(X)| \phi(K) P[x_1 = s_\alpha] \\ &\leq \frac{1}{N} \sum_{(\alpha,K)} \alpha(|S_{(\alpha,K)}|) \phi(K) P[x_1 = s_\alpha] \\ &= \frac{1}{N} \sum_{(\alpha,K)} \sum_{X \in S_{(\alpha,K)}} |f^*(X)| \phi(K) P[x_1 = s_\alpha] \\ &= E(|f^*(X)|). \end{aligned}$$

So f^* is the optimal one. This completes the proof. □

Here, we construct the following algorithm (Algorithm C) which satisfies all the conditions in lemma 3.11 and lemma 3.12. As a result, it can generate unbiased random bits from an arbitrary Markov chain with optimal information efficiency.

Algorithm C

Input: A sequence $X = x_1x_2...,x_N$ produced by a Markov chain, where $x_i \in S = \{s_1, s_2, ..., s_n\}$.

Output: A sequence Y of $0'$s and $1'$s.

Main Function:

1) Get the matrix $K = \{k_{ij}\}$ with

$$k_{ij} = k_j(\pi_i(X)).$$

2) Define $S(X)$ as

$$S(X) = \{X'|k_j(\pi_i(X')) = k_{ij} \forall i, j; x'_1 = x_1\},$$

then compute $|S(X)|$.

3) Compute the rank $r(X)$ of X in $S(X)$ with respect to a given order. The rank with respect to a lexicographic order will be given later.

4) According to $|S(X)|$ and $r(X)$, determine the output sequence. Let $\sum_k 2^{n_k}$ be the standard binary expansion of $|S(X)|$ with $n_1 > n_2 > ...$ and assume the starting value of $r(X)$ is 0. If $r(X) < 2^{n_1}$, the output is the n_1 digit binary representation of $r(x)$. If $\sum_{k=1}^{i} 2^{n_k} \leq r(x) < \sum_{k=1}^{i+1} 2^{n_k}$, the output is the n_{i+1} digit binary representation of $r(x)$.

Comment: The fast calculations of $|S(X)|$ and $r(x)$ will be given in the rest of this section.

In algorithm A, when we use Elias's function as Ψ, the limiting efficiency $\eta_N = \frac{E[M]}{N}$ (as $N \to \infty$) realizes the bound $\frac{H(X)}{N}$. Algorithm C is optimal, so it has the same or higher efficiency. Therefore, the limiting efficiency of algorithm C as $N \to \infty$ also realizes the bound $\frac{H(X)}{N}$.

In algorithm C, for an input sequence X with $x_N = s_\chi$, we can rank it with respect to the lexicographic order of $\theta(X)$ and $\sigma(X)$. Here, we define

$$\theta(X) = (\pi_1(X)_{|\pi_1(X)|}, \ldots, \pi_n(X)_{|\pi_n(X)|}),$$

which is the vector of the last symbols of $\pi_i(X)$ for $1 \leq i \leq n$. And $\sigma(X)$ is the complement of $\theta(X)$ in $\pi(X)$, namely,

$$\sigma(X) = (\pi_1(X)^{|\pi_1(X)|-1}, \ldots, \pi_n(X)^{|\pi_n(X)|-1}).$$

For example, when the input sequence is

$$X = s_1 s_4 s_2 s_1 s_3 s_2 s_3 s_1 s_1 s_2 s_3 s_4 s_1,$$

its exit sequences are

$$\pi(X) = [s_4 s_3 s_1 s_2, s_1 s_3 s_3, s_2 s_1 s_4, s_2 s_1].$$

Then for this input sequence X, we have that

$$\theta(X) = [s_2, s_3, s_4, s_1],$$

$$\sigma(X) = [s_4 s_2 s_1, s_1 s_3, s_2 s_1, s_2].$$

Based on the lexicographic order defined above, both $|S(X)|$ and $r(X)$ can be obtained using a brute-force search. However, this approach in not computationally efficient. Here, we describe an efficient algorithm for computing $|S(X)|$ and $r(X)$ when n is a small constant, such that algorithm C is computable in $O(N \log^3 N \log\log N)$ time. This method is inspired by the algorithm for computing the Elias function that is described in [99]. However, when n is not small, the complexity of computing $|S(X)|$ (or $r(x)$) has an exponential dependence on n, which will make this algorithm much slower in computation than the previous algorithms.

Lemma 3.13. *Let*

$$Z = (\prod_{i=1}^{n} \frac{(k_{i1} + k_{i2} + ... + k_{in})!}{k_{i1}!k_{i2}!...k_{in}!}),$$

and let $N = \sum_{i=1}^{n} \sum_{j=1}^{n} k_{ij}$*, then* Z *is computable in* $O(N \log^3 N \log\log N)$ *time (not related with* n*).*

Proof. It is known that given two numbers of length n bits, their multiplication or division is computable in $O(n \log n \log\log n)$ time based on Schönhage-Strassen algorithm [4]. We can calculate Z based on this fast multiplication.

For simplification, we denote $k_i = \sum_{j=1}^{n} k_{ij}$. Note that we can write Z as a multiplication of N terms, namely

$$\frac{k_1}{1}, \frac{k_1}{2}, ..., \frac{k_1}{k_{11}}, \frac{k_1}{1}, \frac{k_1}{2} ..., \frac{k_n}{k_{nn}},$$

which are denoted as

$$\rho_1^0, \rho_2^0, ..., \rho_{N-1}^0, \rho_N^0.$$

It is easy to see that the notation of every ρ_i^0 used $2 \log_2 N$ bits ($\log_2 N$ for the numerator and $\log N$ for the denominator). The total time to compute all of them is much less than $O(N \log^3 N \log\log N)$.

Based on these notations, we write Z as

$$Z = \rho_1^0 \rho_2^0 ... \rho_{N-1}^0 \rho_N^0.$$

Suppose that $\log_2 N$ is an integer. Otherwise, we can add trivial terms to the formula above to make $\log_2 N$ be an integer. In order to calculate Z quickly, the following calculations are performed:

$$\rho_i^s = \rho_{2i-1}^{s-1} \rho_{2i}^{s-1},$$

$$s = 1, 2, ..., \log_2 N; \quad i = 1, 2, ..., 2^{-s} N.$$

Then we are able to compute Z iteratively and finally get

$$Z = \rho_1^{\log_2 N}.$$

To calculate ρ_i^1 for $i = 1, 2, ..., N/2$, it takes $2(N/2)$ multiplications of numbers with length $\log_2 N$ bits. Similarly, to calculate ρ_i^s for $i = 1, 2, ..., N/2$, it takes $2(N/2^s)$ multiplications of numbers with length $2^s \log_2 N$ bits. So the time complexity of computing Z is

$$\sum_{s=1}^{\log_2 N} 2(N/2^s)O(2^s \log_2 N \log(2^s \log_2 N) \log\log(2^s \log_2 N)).$$

This value is not greater than

$$O(N \log^2 N \log(N \log N) \log\log(N \log N)),$$

which yields the result in the lemma. □

Lemma 3.14. *Let n be a small constant and N be the input length, then $|S(X)|$ in algorithm C is computable in $O(N \log^3 N \log\log N)$ time.*

Proof. The idea to compute $|S(X)|$ in algorithm C is that we can divide $S(X)$ into different classes, denoted by $S(X, \theta)$ for different θ such that

$$S(X, \theta) = \{X' | \forall i, j, k_j(\pi_i(X')) = k_{ij}, \theta(X') = \theta\},$$

where $k_{ij} = k_j(\pi_i(X))$ is the number of s_j's in $\pi_i(X)$ for all $1 \leq i, j \leq n$. $\theta(X)$ is the vector of the last symbols of $\pi(X)$ defined above. As a result, we have $|S(X)| = \sum_\theta |S(X, \theta)|$. Although it is not easy to calculate $|S(X)|$ directly, but it is much easier to compute $|S(X, \theta)|$ for a given θ.

For a given $\theta = (\theta_1, \theta_2, ..., \theta_n)$, we need first determine whether $S(X, \theta)$ is empty or not. In order to do this, we quickly construct a collection of exit sequences $\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_n]$ by moving the first θ_i in $\pi_i(X)$ to the end for all $1 \leq i \leq n$. According to the main lemma, we know that $S(X, \theta)$ is

empty if and only if $\pi_i(X)$ does not include θ_i for some i or (x_1, Λ) is not feasible.

If $S(X, \theta)$ is not empty, then (x_1, Λ) is feasible. In this case, based on the main lemma, we have

$$|S(X,\theta)| = \prod_{i=1}^{n} \frac{(k_{i1} + k_{i2} + ... + k_{in} - 1)!}{k_{i1}!...(k_{i\theta_i} - 1)!...k_{in}!}$$

$$= (\prod_{i=1}^{n} \frac{(k_{i1} + k_{i2} + ... + k_{in})!}{k_{i1}!k_{i2}!...k_{in}!})(\prod_{i=1}^{n} \frac{k_{i\theta_i}}{(k_{i1} + k_{i2} + ... + k_{in})}).$$

Here, we let

$$Z = (\prod_{i=1}^{n} \frac{(k_{i1} + k_{i2} + ... + k_{in})!}{k_{i1}!k_{i2}!...k_{in}!}).$$

Then we can get

$$|S(X)| = \sum_{\theta} |S(X,\theta)| = Z(\sum_{\theta} \prod_{i=1}^{n} \frac{k_{i\theta_i}}{(k_{i1} + k_{i2} + ... + k_{in})}).$$

According to lemma 3.13, Z is computable in $O(N \log^3 N \log\log N)$ time. So if n is a small constant, then $|S(X)|$ is also computable in $O(N \log^3 N \log\log N)$ time. However, when n is not small, we have to enumerate all the possible combinations for θ with $O(n^n)$ time, which is not computationally efficient. □

Lemma 3.15. *Let n be a small constant and N be the input length, then $r(X)$ in algorithm C is computable in $O(N \log^3 N \log\log N)$ time.*

Proof. Based on some calculations in the lemma above, we can try to obtain $r(X)$ when X is ranked with respect to the lexicographic order of $\theta(X)$ and $\sigma(X)$. Let $r(X, \theta(X))$ denote the rank of X in $S(X, \theta(X))$, then we have that

$$r(X) = \sum_{\theta < \theta(X)} |S(X,\theta)| + r(X, \theta(X)),$$

where $<$ is based on the lexicographic order. In the formula, when n is a small constant, $\sum_{\theta<\theta(X)} |S(X,\theta)|$

can be obtained in $O(N \log^3 N \log\log N)$ time by computing

$$Z \frac{\sum_{\theta<\theta(X):|S(X,\theta)|>0} \prod_{i=1}^{n} k_{i\theta_i}}{\prod_{i=1}^{n}(k_{i1}+k_{i2}+...+k_{in})},$$

where Z is defined in the last lemma and the second term can be calculated fast when n is a small constant. (However, n cannot be big, since the complexity of computing the second term has an exponential dependence on n.)

So far, we only need to compute $r(X, \theta(X))$, with respect to the lexicography order of $\sigma(X)$. Here, we write $\sigma(X)$ as the concatenation of a group of sequences, namely

$$\sigma(X) = \sigma_1(X) * \sigma_2(X) * ... * \sigma_n(X),$$

such that for all $1 \leq i \leq n$ $\sigma_i(X) = \pi_i(X)^{|\pi_i(X)|-1}$.

There are $M = (N-1)-n$ symbols in $\sigma(X)$. Let $r_i(X)$ be the number of sequences in $S(X, \theta(X))$ such that their first $M-i$ symbols are $\sigma(X)[1, M-i]$ and their $M-i+1$th symbols are smaller than $\sigma(X)[M-i+1]$. Then we can get that

$$r(X, \theta(X)) = \sum_{i=1}^{M} r_i(X).$$

Let us assume that $\sigma(X)[M-i+1] = s_{w_i}$ for some w_i, and it is the u_ith symbol in $\sigma_{v_i}(X)$. For simplicity, we denote $\sigma_{v_i}(X)[u_i, |\sigma_{v_i}(X)|]$ as ζ_i. For example, when $n = 3$ and $[\sigma_1(X), \sigma_2(X), \sigma_3(X)] = [s_1s_2, s_2s_3, s_1s_1s_1]$, we have

$$\zeta_1 = s_1, \zeta_2 = s_1s_1, \zeta_3 = s_1s_1s_1, \zeta_4 = s_3, \zeta_5 = s_2s_3, ...$$

To calculate $r_i(X)$, we can count all the sequences generated by permuting the symbols of $\zeta_i, \sigma_{v_i+1}(X), ..., \sigma_n(X)$ such that the $M-i+1$th symbol of the new sequence is smaller than s_{w_i}.

Then we can get

$$r_i(X) = \sum_{j<w_i} \frac{(|\zeta_i|-1)!}{k_1(\zeta_i)!...(k_j(\zeta_i)-1)!...k_n(\zeta_i)!} \prod_{i=v_i+1}^{n} \frac{|\sigma_i(X)|!}{k_1(\sigma_i(X))!k_2(\sigma_i(X))!...k_n(\sigma_i(X))!},$$

where $k_j(X)$ counts the number of s_i's in X.

Let us define the values

$$\rho_{i-1}^0 = \frac{|\zeta_i|}{k_{w_i}(\zeta_i)},$$

for all $1 \leq i \leq M$. In this expression, $k_{w_i}(\zeta_i)$ is the number of s_{w_i}'s in ζ_i, and s_{w_i} is the first symbol of ζ_i.

It is easy to show that for $1 \leq i \leq M$

$$\rho_{i-1}^0\rho_{i-2}^0....\rho_2^0\rho_1^0 = \frac{|\zeta_i|!}{k_1(\zeta_i)!...k_j(\zeta_i)!...k_n(\zeta_i)!} \prod_{i=v+1}^{n} \frac{|\sigma_i(X)|!}{k_1(\sigma_i(X))!k_2(\sigma_i(X))!...k_n(\sigma_i(X))!}.$$

If we also define the values

$$\lambda_i^0 = \frac{\sum_{j<w_i} k_j(\zeta_i)}{|\zeta_i|},$$

for all $1 \leq i \leq M$, then we have

$$r_i(X) = \lambda_i^0\rho_{i-1}^0\rho_{i-2}^0...\rho_1^0,$$

and

$$r(X, \theta(X)) = \sum_{i=1}^{M} \lambda_i^0\rho_{i-1}^0\rho_{i-2}^0...\rho_2^0\rho_1^0.$$

Suppose that $\log_2 M$ is an integer. Otherwise, we can add trivial terms to the formula above to make $\log_2 M$ an integer. In order to quickly calculate $r(X, \theta(X))$, the following calculations are performed for s from 1 to $\log_2 M$:

$$\rho_i^s = \rho_{2i}^{s-1}\rho_{2i-1}^{s-1}, \quad i = 1, 2, ..., 2^{-s}M - 1,$$

$$\lambda_i^s = \lambda_{2i-1}^{s-1} + \lambda_{2i}^{s-1}\rho_{2i-1}^{s-1}, \quad i = 1, 2, ..., 2^{-s}M.$$

By computing all ρ_i^s and λ_i^s for s from 1 to $\log_2 M$ iteratively, we can get that

$$r(X, \theta(X)) = \lambda_1^{\log_2 M}.$$

Now, we use the same idea in [99] to analyze the computational complexity. Note that every ρ_i^0 and λ_i^0 can be represented using $2\log_2 M$ bits ($\log_2 M$ for the numerator and $\log M$ for the denominator). And we can calculate all of them quickly. To calculate ρ_i^1 for $i = 1, 2, ..., M/2 - 1$, it takes at most $2(M/2)$ multiplications of numbers with length $\log_2 M$ bits. To calculate λ_i^1 for $i = 1, 2, ..., M/2$, it takes $3(M/2)$ multiplications of numbers with length $\log_2 M$ bits. That is because we can write λ_i^1 as $\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$ for some integers a, b, c, d with length $\log_2 M$ bits. Similarly, to calculate all ρ_i^s and λ_i^s for some s, it takes at most $5(M/2^s)$ multiplications of numbers with length $2^s \log_2 M$ bits. As a result, the time complexity of computing Z is

$$\sum_{s=1}^{\log_2 M} 5(M/2^s)O(2^s \log_2 M \log(2^s \log_2 M) \log\log(2^s \log_2 M)),$$

which is computable in $O(M \log^3 M \log\log M)$ time. As a result, for a small constant n, $r(X)$ is computable in $O(N \log^3 N \log\log N)$ time. □

Based on the discussion above, we know that algorithm C is computable in $O(N \log^3 N \log\log N)$ time when n is a small constant. However, when n is not a constant, this algorithm is not computationally efficient since its time complexity depends exponentially on n.

3.7 Numerical Results

In this section, we describe numerical results related to the implementations of algorithm A, algorithm B, and algorithm C. We use the Elias function for Ψ.

In the first experiment, we use the following randomly generated transition matrix for a Markov

Table 3.2. The probability of each possible output sequence and the expected output length

Output	Probability Algorithm A	Probability Algorithm B with $\varpi = 4$	Probability Algorithm C
Λ	0.0224191	0.1094849	0.0208336
0	0.0260692	0.0215901	0.0200917
1	0.0260692	0.0215901	0.0200917
00	0.0298179	0.1011625	0.0206147
10	0.0298179	0.1011625	0.0206147
01	0.0298179	0.1011625	0.0206147
11	0.0298179	0.1011625	0.0206147
000	0.0244406	0.0242258	0.0171941
100	0.0244406	0.0242258	0.0171941
...	...	...	...
011111	0.0018831	1.39E-5	0.0029596
111111	0.0018831	1.39E-5	0.0029596
0000000	1.305E-4		6.056E-4
1000000	1.305E-4		6.056E-4
...			...
0111111	1.305E-4		6.056E-4
1111111	1.305E-4		6.056E-4
00000000			1.44E-5
10000000			1.44E-5
...			...
01111111			1.44E-5
11111111			1.44E-5
Expected Length	3.829	2.494	4.355

chain with three states.

$$P = \begin{pmatrix} 0.300987 & 0.468876 & 0.230135 \\ 0.462996 & 0.480767 & 0.056236 \\ 0.42424 & 0.032404 & 0.543355 \end{pmatrix}$$

Consider a sequence of length 12 that is generated by the Markov chain defined above and assume that s_1 is the first state of this sequence. Namely, there are $3^{11} = 177147$ possible input sequences.

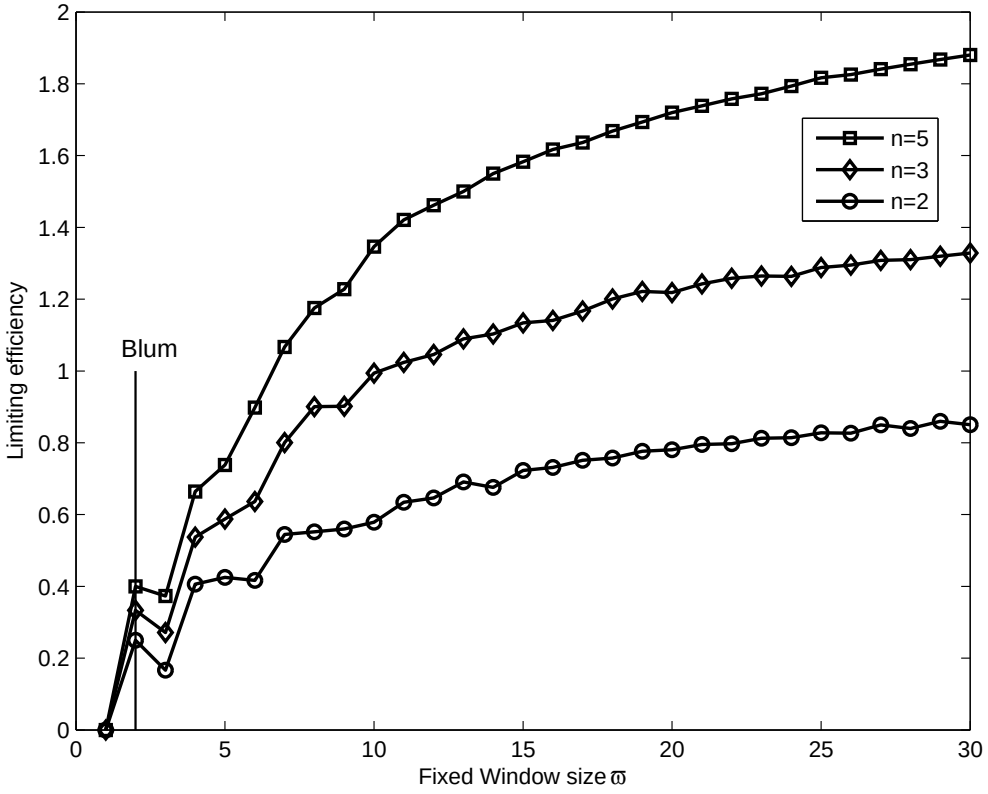


Figure 3.6. The limiting efficiency of algorithm B varies with the value of window size ϖ.

For each possible input sequence, we can compute its generating probability and the corresponding output sequences using our three algorithms. Table 3.2 presents the results of calculating the probabilities of all possible output sequences for the three algorithms. Note that the results show that indeed the outputs of the algorithms are independent unbiased sequences. Also, algorithm C has the highest information efficiency (it is optimal), and algorithm A has a higher information efficiency than algorithm B (with window size 4).

In the second calculation, we want to test the influence of window size ϖ (assume $\varpi_i(k) = \varpi$ for $1 \leq i \leq n$) on the efficiency of algorithm B. Since the efficiency depends on the transition matrix of the Markov chain we decided to evaluate of the efficiency related to the uniform transition matrix, namely all the entries are $\frac{1}{n}$, where n is the number of states. We assume that n is infinitely large. In this case, the stationary distribution of the Markov chain is $\{\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n}\}$. Figure 3.6 shows that

when $\varpi = 2$ (Blum's Algorithm), the limiting efficiencies for $n = (2, 3, 5)$ are $(\frac{1}{4}, \frac{1}{3}, \frac{2}{5})$, respectively. When $\varpi = 15$, their corresponding efficiencies are $(0.7228, 1.1342, 1.5827)$. So if the input sequence is long enough, by changing ϖ from 2 to 15, the efficiency can increase 189% for $n = 2$, 240% for $n = 3$ and 296% for $n = 4$. When ϖ is small, we can increase the efficiency of algorithm B significantly by increasing the window size ϖ. When ϖ becomes larger, the efficiency of algorithm B will converge to the information-theoretical upper bound, namely, $\log_2 n$. Note that 3 is not a good value for the window size in the algorithm. That is because the Elias function is not very efficient when the length of the input sequence is 3. Let us consider a biased coin with two states s_1, s_2. If the input sequence is $s_1s_1s_1$ or $s_2s_2s_2$, the Elias function will generate nothing. For all other cases, it has only 2/3 chance to generate one bit and 1/3 chance to generate nothing. As a result, the efficiency is even worse than the efficiency when the length of the input sequence equals 2.

3.8 Conclusion

We considered the classical problem of generating independent unbiased bits from an arbitrary Markov chain with unknown transition probabilities. Our main contribution is the first known algorithm with expected linear-time complexity that achieves the information-theoretic upper bound on efficiency.

In information theory, it was discovered that optimal source codes can be used as universal random bit generators from arbitrary stationary ergodic random sources [126] [51] (The Markov chains studied in this chapter are special cases of stationary ergodic sources). When the input sequence is generated from a stationary ergodic process and it is long enough, one can obtain an output sequence that behaves like truly random bits in the sense of normalized divergence. However, in some cases, the definition of normalized divergence is not strong enough. For example, suppose Y is a sequence of unbiased random bits in the sense of normalized divergence, and $1 * Y$ is Y with a 1 concatenated at the beginning. If the sequence Y is long enough, the sequence $1 * Y$ is a sequence of unbiased random bits in the sense of normalized divergence. However the sequence $1 * Y$ might not be useful in applications that are sensitive to the randomness of the first bit.

Chapter 4

Streaming Algorithms for Random Number Generation

This chapter introduces an algorithm that generates random bit streams from biased coins, uses bounded space and runs in expected linear time. As the size of the allotted space increases, the algorithm approaches the information-theoretic upper bound on efficiency.

4.1 Introduction

Von Neumann's algorithm (see chapter 2) is not optimal in generating random bits from a biased coin, in terms of the number of random bits that are generated. This problem was solved in [33,88,90,99]. Specifically, given a fixed number of biased coin tosses with unknown probability, it is well-known how to generate random bits with asymptotically optimal efficiency; that is, it is known how to generate random bits in a way such that the expected number of unbiased random bits generated per coin toss is asymptotically equal to the entropy of the biased coin. However, these solutions, including Elias's algorithm and Peres's algorithm, can generate random bits only after receiving the complete input sequence (or a fixed number of input bits), and the number of random bits generated is a random variable.

We consider the setup of generating a "stream" of random bits; that is, whenever random bits are required, the algorithm reads new coin tosses and generates random bits dynamically. Our new streaming algorithm is more efficient (in the number of input bits, memory and time) for producing

the required number of random bits and is a better choice for implementation in practical systems. We notice that von Neumann scheme is the one which is able to generate a stream of random bits, but its efficiency is far from optimal. Our goal is to modify this scheme such that it can achieve the information-theoretic upper bound on efficiency. Specifically, we would like to construct a function $f : \{\mathrm{H}, \mathrm{T}\}^* \to \{0, 1\}^*$ which satisfies the following conditions:

- f generates a stream. For any two sequences of coin tosses $x, y \in \{\mathrm{H}, \mathrm{T}\}^*$, $f(x)$ is a prefix of $f(xy)$.
- f generates random bits. Let $X_k \in \{0, 1\}^*$ be the sequence of coin tosses inducing k bits; that is, $|f(X_k)| \geq k$ but for any strict prefix X of X_k, $|f(X)| \leq k$. Then the first k bits of $f(X_k)$ are independent and unbiased.
- f has asymptotically optimal efficiency. That is, for any $k > 0$,

$$\frac{E[|X_k|]}{k} \to \frac{1}{H(p)}$$

  as $k \to \infty$, where $H(p)$ is the entropy of the biased coin [27].

We note that the von Neumann scheme uses only 3 states, i.e., a symbol in $\{\phi, \mathrm{H}, \mathrm{T}\}$, for storing state information. For example, the output bit is 1 if and only if the current state is H and the input symbol is T. In this case, the new state is ϕ. Similarly, the output bit is 0 if and only if the current state is T and the input symbol is H. In this case, the new state is ϕ. Our approach for generalizing von Neumann's scheme is by increasing the memory (or state) of our algorithm such that we do not lose information that might be useful for generating future random bits. We represent the state information as a binary tree, called status tree, in which each node is labeled by a symbol in $\{\phi, \mathrm{H}, \mathrm{T}, 0, 1\}$. When a source symbol (a coin toss) is received, we modify the status tree based on certain simple rules and generate random bits in a dynamic way. This is the key idea in our algorithm; we call this approach the random-stream algorithm. In some sense, the random-stream algorithm is the streaming version of Peres's algorithm. We show that this algorithm satisfies all

three conditions above, namely, it can generate a stream of random bits with asymptotically optimal efficiency. In practice, we can reduce the space size by limiting the depth of the status tree. We will demonstrate that as the depth of the status tree increases, the efficiency of the algorithm quickly converges to the information-theoretic upper bound.

The rest of the chapter is organized as follows. Section 4.2 presents our key result, the random-stream algorithm that generates random bit streams from arbitrary biased coins and achieves the information-theoretic upper bound on efficiency. In section 4.3, we generalize the random-stream algorithm to generate random bit streams from a source of a larger alphabet. An extension for Markov chains is provided in section 4.4, followed by the concluding remarks.

4.2 The Random-Stream Algorithm

4.2.1 Description

Many algorithms have been proposed for efficiently generating random bits from a fixed number of coins tosses, including Elias's algorithm and Peres's algorithm. However, in these algorithms, the input bits can be processed only after all of them have been received, and the number of random bits generated cannot be controlled. In this section, we focus on deriving a new algorithm, the *random-stream algorithm*, that generates a stream of random bits from an arbitrary biased-coin source and achieves the information-theoretic upper bound on efficiency. Given an application that requires random bits, the random-stream algorithm can generate random bits dynamically based on requests from the application.

While von Neumann's scheme can generate a stream of random bits from an arbitrary biased coin, its efficiency is far from being optimal. The main reason is that it uses minimal state information, recorded by a symbol of alphabet size three in $\{\phi, \mathrm{H}, \mathrm{T}\}$. The key idea in our algorithm is to create a binary tree for storing the state information, called a status tree. A node in the status tree stores a symbol in $\{\phi, \mathrm{H}, \mathrm{T}, 0, 1\}$. The following procedure shows how the status tree is created and is dynamically updated in response to arriving input bits. At the beginning, the tree has only a single

root node labeled as ϕ. When reading a coin toss from the source, we modify the status tree based on certain rules. For each node in the status tree, if it receives a message (H or T), we do operations on the node. Meanwhile, this node may pass some new messages to its children. Iteratively, we can process the status tree until no more messages are generated. Specifically, let u be a node in the tree. Assume the label of u is $x \in \{\phi, \mathrm{H}, \mathrm{T}, 1, 0\}$ and it receives a symbol $y \in \{\mathrm{H}, \mathrm{T}\}$ from its parent node (or from the source if u is the root node). Depending on the values of x and y, we do the following operations on node u.

1. When $x = \phi$, set $x = y$.

2. When $x = 1$ or 0, output x and set $x = y$.

3. When $x = \mathrm{H}$ or T, we first check whether u has children. If it does not have, we create two children with label ϕ for it. Let u_l and u_r denote the two children of u.

   - If $xy = \mathrm{HH}$, we set $x = \phi$, then pass a symbol T to u_l and a symbol H to u_r.
   - If $xy = \mathrm{TT}$, we set $x = \phi$, then pass a symbol T to u_l and a symbol T to u_r.
   - If $xy = \mathrm{HT}$, we set $x = 1$, then pass a symbol H to u_l.
   - If $xy = \mathrm{TH}$, we set $x = 0$, then pass a symbol H to u_l.

   We see that the node u passes a symbol $x + y \bmod 2$ to its left child and if $x = y$ it passes a symbol x to its right child.

Note that the timing is crucial that we output a node's label (when it is 1 or 0) only after it receives the next symbol from its parent or from the source. This is different from von Neumann's scheme where a 1 or a 0 is generated immediately without waiting for the next symbol. If we only consider the output of the root node in the status tree, then it is similar to von Neumann's scheme. And the other nodes correspond to the information discarded by von Neumann's scheme. In some sense, the random-stream algorithm can be treated as a "stream" version of Peres's algorithm. The following example is constructed for the purpose of demonstration.

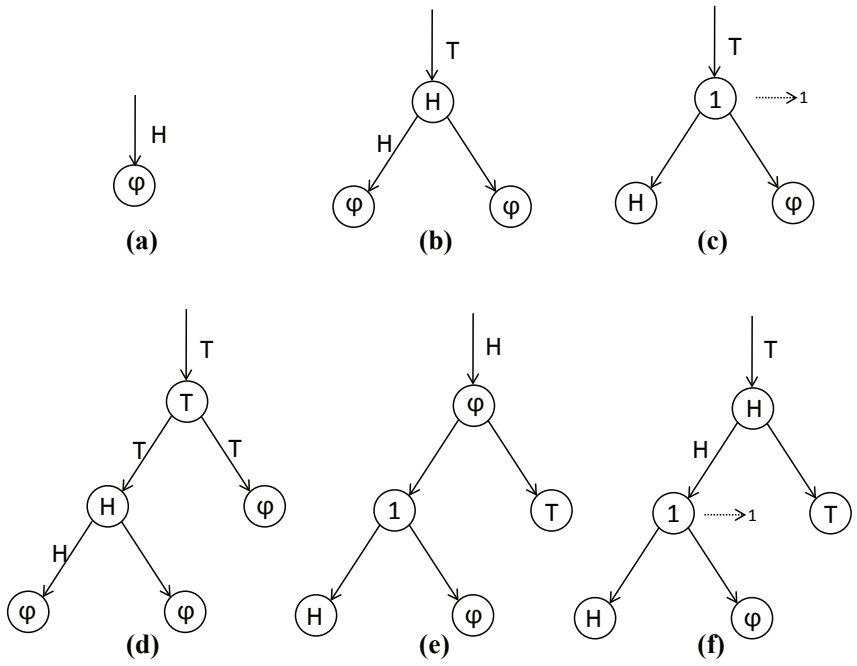


Figure 4.1. An instance for generating 2 random bits using the random-stream algorithm.

Example 4.1. *Assume we have a biased coin and our randomized application requires* 2 *random bits. Figure 4.1 illustrates how the random-stream algorithm works when the incoming stream is HTTTHT... In this figure, we can see the changes of the status tree and the messages (symbols) passed throughout the tree for each step. We see that the output stream is* 11...

Lemma 4.1. *Let X be the current input sequence and let $\mathcal{T}$ be the current status tree. Given $\mathcal{T}$ and the bits generated by each node in $\mathcal{T}$, we can reconstruct X uniquely.*

Proof. Let us prove this lemma by induction. If the maximum depth of the status tree is 0, it has only a single node. In this case, X is exactly the label on the single node. Hence the conclusion is trivial. Now we show that if the conclusion holds for all status trees with maximum depth at most k, then it also holds for all status trees with maximum depth $k+1$.

Given a status tree $\mathcal{T}$ with maximum depth $k+1$, we let $Y \in \{0,1\}^*$ denote the binary sequence generated by the root node, and $L, R \in \{\mathrm{H},\mathrm{T}\}^*$ are the sequences of symbols received by its left child and right child. If the label of the root node is in $\{0,1\}$, we add it to Y. According to the

random-stream algorithm, it is easy to get that

$$|L| = |Y| + |R|.$$

Based on our assumption, L, R can be constructed from the left and right subtrees and the bits generated by each node in the subtree since their depths are at most k. We show that once L, R, Y satisfy the equality above, the input sequence X can be uniquely constructed from L, R, Y and α, where α is the label of the root node. The procedure is as follows: Let us start from an empty string for X and read symbols from L sequentially. If a symbol read from L is H, we read a bit from Y. If this bit is 1 we add HT to X, otherwise we add TH to X. If a symbol read from L is T, we read a symbol (H or T) from R. If this symbol is H we add HH to X, otherwise we add TT to X.

After reading all the elements in L, R and Y, the length of the resulting input sequence is $2|L|$. Now, we add α to the resulting input sequence if $\alpha \in \{\mathrm{H}, \mathrm{T}\}$. This leads to the final sequence X, which is unique. □

Example 4.2. *Let us consider the status tree in figure 4.1(f). And we know that the root node generates* 1 *and the first node in the second level generates* 1*. We can have the following conclusions iteratively.*

- *In the third level, the symbols received by the node with label H are H, and the node with label ϕ does not receive any symbols.*
- *In the second level, the symbols received by the node with label* 1 *are HTH, and the symbols received by the node with label T are T.*
- *For the root node, the symbols received are HTTTHT, which accords with example 4.1.*

Let $f : \{\mathrm{H}, \mathrm{T}\}^* \to \{0, 1\}^*$ be the function of the random-stream algorithm. We show that this function satisfies all the three conditions described in the introduction. It is easy to see that the first condition holds, i.e., for any two sequences $x, y \in \{\mathrm{H}, \mathrm{T}\}^*$, $f(x)$ is a prefix of $f(xy)$, hence it generates streams. The following two theorems indicate that f also satisfies the other two conditions.

Theorem 4.2. *Given a source of biased coin with unknown probability, the random-stream algorithm generates a stream of random bits, i.e., for any $k > 0$, if we stop running the algorithm after generating k bits then these k bits are independent and unbiased.*

Let S_Y with $Y \in \{0,1\}^k$ denote the set consisting of all the binary sequences yielding Y. Here, we say that a binary sequence X *yields* Y if and only if $X[1 : |X| - 1]$ (the prefix of X with length $|X| - 1$) generates a sequence shorter than Y and X generates a sequence with Y as a prefix (including Y itself). To prove that the algorithm can generate random-bit streams, we show that for any distinct binary sequences $Y_1, Y_2 \in \{0,1\}^k$, the elements in S_{Y_1} and those in S_{Y_2} are one-to-one mapping. The detailed proof is given in subsection 4.2.2.

Theorem 4.3. *Given a biased coin with probability p being H, let n be the number of coin tosses required for generating k random bits in the random-stream algorithm, then*

$$\lim_{k\to\infty} \frac{E[n]}{k} = \frac{1}{H(p)}.$$

The proof of theorem 4.3 is based on the fact that the random-stream algorithm is as efficient as Peres's algorithm. The difference is that in Peres's algorithm the input length is fixed and the output length is variable. But in the random-stream algorithm the output length is fixed and the input length is variable. So the key of the proof is to connect these two cases. The detailed proof is given in subsection 4.2.3.

So far, we can conclude that the random-stream algorithm can generate a stream of random bits from an arbitrary biased coin with asymptotically optimal efficiency. However, the size of the binary tree increases as the number of input coin tosses increases. The longest path of the tree is the left-most path, in which each node passes one message to the next node when it receives two messages from its previous node. Hence, the maximum depth of the tree is $\log_2 n$ for n input bits. This linear increase in space is a practical challenge. Our observation is that we can control the size of the space by limiting the maximum depth of the tree – if a node's depth reaches a certain threshold, it will stop creating new leaves. We can prove that this method correctly generates a stream of

random bits from an arbitrary biased coin. We call this method the random-stream algorithm with maximum depth d.

Theorem 4.4. *Given a source of a biased coin with unknown probability, the random-stream algorithm with maximum depth d generates a stream of random bits, i.e., for any $k > 0$, if we stop running the algorithm after generating k bits then these k bits are independent and unbiased.*

The proof of theorem 4.4 is a simple modification of the proof of theorem 4.2, given in subsection 4.2.4. In order to save memory space, we need to reduce the efficiency. Fortunately, as the maximum depth increases, the efficiency of this method can quickly converge to the theoretical limit.

Example 4.3. *When the maximum depth of the tree is* 0 *(it has only the root node), then the algorithm is approximately von Neumann's scheme. The expected number of coin tosses required per random bit is*

$$\frac{1}{pq}$$

asymptotically, where $q = 1 - p$ and p is the probability for the biased coin being H.

Example 4.4. *When the maximum depth of the tree is* 1*, the expected number of coin tosses required per random bit is*

$$\frac{1}{pq + \frac{1}{2}(p^2+q^2)(2pq) + \frac{1}{2}(p^2+q^2)\frac{p^2q^2}{(p^2+q^2)^2}}$$

asymptotically, where $q = 1 - p$ and p is the probability for the biased coin being H.

Generally, if the maximum depth of the tree is d, then we can calculate the efficiency of the random-stream algorithm by iteration in the following way:

Theorem 4.5. *When the maximum depth of the tree is d and the probability of the biased coin is p of being H, the expected number of coin tosses required per random bit is*

$$\frac{1}{\rho_d(p)}$$

Table 4.1. The expected number of coin tosses required per random bit for different probability p and different maximum depths

maximum depth	p=0.1	p=0.2	p=0.3	p=0.4	p=0.5
0	11.1111	6.2500	4.7619	4.1667	4.0000
1	5.9263	3.4768	2.7040	2.3799	2.2857
2	4.2857	2.5816	2.0299	1.7990	1.7297
3	3.5102	2.1484	1.7061	1.5190	1.4629
4	3.0655	1.9023	1.5207	1.3596	1.3111
5	2.7876	1.7480	1.4047	1.2598	1.2165
7	2.4764	1.5745	1.2748	1.1485	1.1113
10	2.2732	1.4619	1.1910	1.0772	1.0441
15	2.1662	1.4033	1.1478	1.0408	1.0101
∞	2.1322	1.3852	1.1347	1.0299	1.0000

asymptotically, where $\rho_d(p)$ can be obtained by iterating

$$\rho_d(p) = pq + \frac{1}{2}\rho_{d-1}(p^2+q^2) + \frac{1}{2}(p^2+q^2)\rho_{d-1}(\frac{p^2}{p^2+q^2}) \tag{4.1}$$

with $q = 1-p$ and $\rho_0(p) = pq$.

Theorem 4.5 shows that the efficiency of a random-stream algorithm with maximum depth d can be easily calculated by iteration. One thing that we can claim is,

$$\lim_{d\to\infty} \rho_d(p) = H(p).$$

However, it is difficult to get an explicit expression for $\rho_d(p)$ when d is finite. As d increases, the convergence rate of $\rho_d(p)$ depends on the value of p. The following extreme case implies that $\rho_d(p)$ can converge to $H(p)$ very quickly.

Example 4.5. *Let us consider the case that $p = \frac{1}{2}$. According to equation (4.1), we have*

$$\rho_d(\frac{1}{2}) = \frac{1}{4} + \frac{1}{2}\rho_{d-1}(\frac{1}{2}) + \frac{1}{4}\rho_{d-1}(\frac{1}{2}),$$

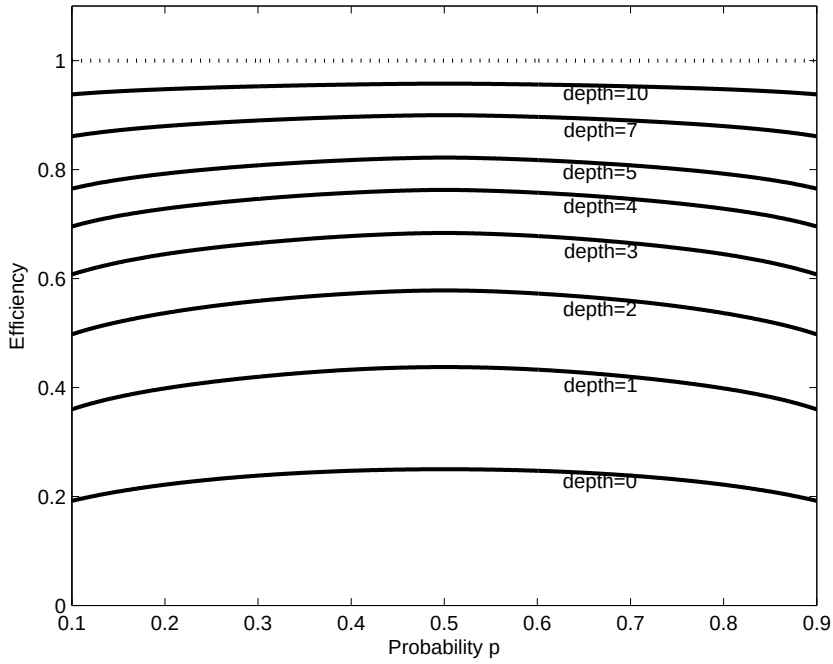


Figure 4.2. The efficiency for different probability p and different maximum depths.

where $\rho_0(\frac{1}{2}) = \frac{1}{4}$. *Based on this iterative relation, it can be obtained that*

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

So when $p = \frac{1}{2}$, $\rho_d(p)$ *can converge to* $H(p) = 1$ *very quickly as* d *increases.*

In table 4.1, we tabulate the expected number of coin tosses required per random bit in the random-stream algorithm with different maximum depths. We see that as the maximum depth increases, the efficiency of the random-stream algorithm approaches the theoretical limitation quickly. Let us consider the case of $p = 0.3$ as an example. If the maximum depth is 0, the random-stream algorithm is as efficient as von Neumann's scheme, which requires expected 4.76 coin tosses to generate one random bit. If the maximum depth is 7, it requires only expected 1.27 coin tosses to generate one random bit. That is very close to the theoretical limitation 1.13. However, the space cost of the algorithm has an exponential dependence on the maximum depth. That requires us to balance the efficiency and the space cost in real applications. Specifically, if we define efficiency as the ratio

Table 4.2. The expected time for processing a single input coin toss for different probability p and different maximum depths

maximum depth	p=0.1	p=0.2	p=0.3	p=0.4	p=0.5
0	1.0000	1.0000	1.0000	1.0000	1.0000
1	1.9100	1.8400	1.7900	1.7600	1.7500
2	2.7413	2.5524	2.4202	2.3398	2.3125
3	3.5079	3.1650	2.9275	2.7840	2.7344
4	4.2230	3.6996	3.3414	3.1256	3.0508
5	4.8968	4.1739	3.6838	3.3901	3.2881
7	6.1540	4.9940	4.2188	3.7587	3.5995
10	7.9002	6.0309	4.8001	4.0783	3.8311
15	10.6458	7.5383	5.5215	4.3539	3.9599

between the theoretical lower bound and the real value of the expected number of coin tosses, then figure 4.2 shows the relation between the efficiency and the maximum depth for different probability p.

Another property that we consider is the expected time for processing a single coin toss. Assume that it takes a single unit of time to process a message received at a node, then the expected time is exactly the expected number of messages that have been generated in the status tree (including the input coin toss itself). Table 4.2 shows the expected time for processing a single input bit when the input is infinitely long, implying the computational efficiency of the random-stream algorithm with limited depth. It can be proved that for an input generated by an arbitrary biased coin the expected time for processing a single coin toss is upper bounded by the maximum depth plus one (it is not a tight bound).

4.2.2 Proof of Theorem 4.2

In this subsection, we prove Theorem 4.2.

Lemma 4.6. *Let $\mathcal{T}$ be the status tree induced by $X_A \in \{H,T\}^*$, and let $k_1, k_2, ..., k_{|\mathcal{T}|}$ be the number of bits generated by the nodes in $\mathcal{T}$, where $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$. Then for any $y_i \in \{0,1\}^{k_i}$ with $1 \leq i \leq |\mathcal{T}|$, there exists an unique sequence $X_B \in \{H, T\}^*$ such that it induces the same status*

tree $\mathcal{T}$, and the bits generated by the ith node in $\mathcal{T}$ is y_i. For such a sequence X_B, it is a permutation of X_A with the same last element.

Proof. To prove this conclusion, we can apply the idea of Lemma 4.1. It is obviously that if the maximum depth of $\mathcal{T}$ is zero, then the conclusion is trivial. Assume that the conclusion holds for any status tree with maximum depth at most k, then we show that it also holds for any status tree with maximum depth $k+1$.

Given a status tree $\mathcal{T}$ with maximum depth $k+1$, we use $Y_A \in \{0,1\}^*$ to denote the binary sequence generated by the root node based on X_A, and L_A, R_A to denote the sequences of symbols received by its left child and right child. According to our assumption, by flipping the bits generated by the left subtree, we can construct an unique sequence $L_B \in \{\mathrm{H},\mathrm{T}\}^*$ uniquely such that L_B is a permutation of L_A with the same last element. Similarly, for the right subtree, we have $R_B \in \{\mathrm{H},\mathrm{T}\}^*$ uniquely such that R_B is a permutation of R_A with the same last element.

Assume that by flipping the bits generated by the root node, we get a binary sequence Y_B such that $|Y_B| = |Y_A|$ (If the label $\alpha \in \{0,1\}$, we add it to Y_A and Y_B), then

$$|L_B| = |Y_B| + |R_B|,$$

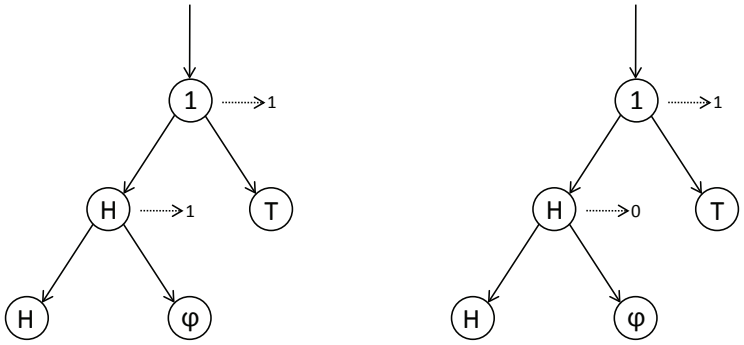
which implies that we can construct X_B from L_B, R_B, Y_B and the label α on the root node uniquely (according to the proof of the above lemma). Since the length of X_B is uniquely determined by $|L_B|$ and α, we can also conclude that X_A and X_B have the same length.

To see that X_B is a permutation of X_A, we show that X_B has the same number of H's as X_A. Given a sequence $X \in \{\mathrm{H},\mathrm{T}\}^*$, let $w_{\mathrm{H}}(X)$ denote the number of H's in X. It is not hard to see that

$$w_{\mathrm{H}}(X_A) = w_{\mathrm{H}}(L_A) + w_{\mathrm{H}}(R_A) + w_{\mathrm{H}}(\alpha),$$

$$w_{\mathrm{H}}(X_B) = w_{\mathrm{H}}(L_B) + w_{\mathrm{H}}(R_B) + w_{\mathrm{H}}(\alpha),$$

where $w_{\mathrm{H}}(L_A) = w_{\mathrm{H}}(L_B)$ and $w_{\mathrm{H}}(R_A) = w_{\mathrm{H}}(R_B)$ due to our assumption. Hence $w_{\mathrm{H}}(X_A) =$



(a) **(b)**

Figure 4.3. An example for demonstrating lemma 4.6, where the input sequence for (a) is HTTTHT, and the input sequence for (b) is TTHTHT.

$w_{\mathrm{H}}(X_B)$ and X_B is a permutation of X_A.

Finally, we would like to see that X_A and X_B have the same last element. That is because if $\alpha \in \{\mathrm{H}, \mathrm{T}\}$, then both X_A and X_B end with α. If $\alpha \in \{\phi, 0, 1\}$, the last element of X_B depends on the last element of L_B, the last element of R_B, and α. Our assumption gives that L_B has the same element as L_A and R_B has the same element as R_A. So we can conclude that X_A and X_B have the same last element. □

Example 4.6. *The status tree of a sequence HTTTHT is given by figure 4.3(a). If we flip the second bit* 1 *into* 0*, see figure 4.3(b), we can construct a sequence of coin tosses , which is TTHTHT.*

Now, we define an equivalence relation on $\{\mathrm{H}, \mathrm{T}\}^*$.

Definition 4.1. *Let $\mathcal{T}_A$ be the status tree of X_A and $\mathcal{T}_B$ be the status tree of X_B. Two sequences $X_A, X_B \in \{H, T\}^*$ are equivalent denoted by $X_A \equiv X_B$ if and only if $\mathcal{T}_A = T_B$, and for each pair of nodes (u, v) with $u \in T_A$ and $v \in T_B$ at the same position they generate the same number of bits.*

Let S_Y with $Y \in \{0, 1\}^k$ denote the set consisting of all the binary sequences yielding Y. Here, we say that a binary sequence X *yields* Y if and only if $X[1 : |X| - 1]$ generates a sequence shorter than Y and X generates a sequence with Y as a prefix (including Y itself). Namely, let f be the

function of the random-stream algorithm, them

$$|f(X[1:|X|-1])| < |Y|, \quad f(X) = Y\Delta \text{ with } \Delta \in \{0,1\}^*.$$

To prove that the algorithm can generate random-bit streams, we show that for any distinct binary sequences $Y_1, Y_2 \in \{0,1\}^k$, the elements in S_{Y_1} and those in S_{Y_2} are one-to-one mapping.

Lemma 4.7. *Let f be the function of the random-stream algorithm. For any distinct binary sequences $Y_1, Y_2 \in \{0,1\}^k$, if $X_A \in S_{Y_1}$, there are exactly one sequence $X_B \in S_{Y_2}$ such that*

- $X_B \equiv X_A$.
- $f(X_A) = Y_1\Delta$ *and* $f(X_B) = Y_2\Delta$ *for some binary sequence* $\Delta \in \{0,1\}^*$.

Proof. Let us prove this conclusion by induction. Here, we use X'_A to denote the prefix of X_A of length $|X_A| - 1$ and use β to denote the last symbol of X_A. So $X_A = X'_A\beta$.

When $k = 1$, if $X_A \in S_0$, we can write $f(X_A)$ as 0Δ for some $\Delta \in \{0,1\}^*$. In this case, we assume that the status tree of X'_A is $\mathcal{T}'_A$, and in which node u generates the first bit 0 when reading the symbol β. If we flip the label of u from 0 to 1, we get another status tree, denoted by $\mathcal{T}'_B$. Using the same argument as lemma 4.1, we are able to construct a sequence X'_B such that its status tree is $\mathcal{T}'_B$ and it does not generate any bits. Concatenating X'_B with β results in a new sequence X_B, i.e., $X_B = X'_B\beta$, such that $X_B \equiv X_A$ and $f(X_B) = 1\Delta$. Similarly, for any sequence X_B that yields 1, i.e., $X_B \in S_1$, if $f(X_B) = 1\Delta$, we can find a sequence $X_A \in S_0$ such that $X_A \equiv X_B$ and $f(X_A) = 0\Delta$. So we can say that the elements in S_0 and S_1 are one-to-one mapping.

Now we assume that all the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping for all $Y_1, Y_2 \in \{0,1\}^k$, then we show that this conclusion also holds for any $Y_1, Y_2 \in \{0,1\}^{k+1}$. Two cases need to be considered.

1) Y_1, Y_2 end with the same bit. Without loss of generality, we assume this bit is 0, then we can write $Y_1 = Y'_1 0$ and $Y_2 = Y'_2 0$. If $X_A \in S_{Y'_1}$, then we can write $f(X_A) = Y'_1\Delta'$ in which the first bit of Δ' is 0. According to our assumption, there exists a sequence $X_B \in S_{Y'_2}$ such that $X_B \equiv X_A$ and

$f(X_B) = Y_2'\Delta'$. In this case, if we write $f(X_A) = Y_1\Delta = Y_1'0\Delta$, then $f(X_B) = Y_2'\Delta' = Y_2'0\Delta = Y_2\Delta$. So such a sequence X_B satisfies our requirements.

If $X_A \notin S_{Y_1'}$, that means Y_1' has been generated before reading the symbol β. Let us consider a prefix of X_A, denote by $\overline{X_A}$, such that it yields Y_1'. In this case, $f(X_A') = Y_1'$ and we can write $X_A = \overline{X_A}Z$. According to our assumption, there exists exactly one sequence $\overline{X_B}$ such that $\overline{X_B} \equiv \overline{X_A}$ and $f(X_B') = Y_2'$. Since $\overline{X_A}$ and $\overline{X_B}$ induce the same status tree, if we construct a sequence $X_B = \overline{X_B}Z$, then $X_B \equiv X_A$ and X_B generates the same bits as X_A when reading symbols from Z. It is easy to see that such a sequence X_B satisfies our requirements.

Since this result is also true for the inverse case, if Y_1, Y_2 end with same bit the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping.

2) Let us consider the case that Y_1, Y_2 end with different bits. Without loss of generality, we assume that $Y_1 = Y_1'0$ and $Y_2 = Y_2'1$. According to the argument above, the elements in $S_{00\ldots00}$ and $S_{Y_1'0}$ are one-to-one mapping; and the elements in $S_{00..01}$ and $S_{Y_2'1}$ are one-to-one mapping. So our task is to prove that the elements in $S_{00..00}$ and $S_{00\ldots01}$ are one-to-one mapping. For any sequence $X_A \in S_{00\ldots00}$, let X_A' be its prefix of length $|X_A| - 1$. Here, X_A' generates only zeros whose length is at most k. Let $\mathcal{T}_A'$ denote the status tree of X_A' and let u be the node in $\mathcal{T}_A'$ that generates the $k+1th$ bit (zero) when reading the symbol β. Then we can construct a new sequence X_B' with status tree $\mathcal{T}_B'$ such that

- $\mathcal{T}_B'$ and $\mathcal{T}_A'$ are the same except the label of u is 0 and the label of the node at the same position in $\mathcal{T}_B'$ is 1.
- For each node u in $\mathcal{T}_A'$, let v be its corresponding node at the same position in $\mathcal{T}_B'$, then u and v generate the same bits.

The construction of X_B' follows the proof of lemma 4.6. If we construct a sequence $X_B = X_B'\beta$, it is not hard to show that X_B satisfies our requirements, i.e.,

- $X_B \equiv X_A$;
- X_B' generates less than $k+1$ bits, i.e., $|f(X_B')| \leq k$;

- If $f(X_A) = 0^k 0\Delta$, then $f(X_B) = 0^k 1\Delta$, where 0^k is for k zeros.

Also based on the inverse argument, we see that the elements in $S_{00..00}$ and $S_{00...01}$ are one-to-one mapping. So if Y_1, Y_2 end with different bits, the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping.

Finally, we can conclude that the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping for any $Y_1, Y_2 \in \{0,1\}^k$ with $k > 0$. □

Theorem 4.2. *Given a source of biased coin with unknown probability, the random-stream algorithm generates a stream of random bits, i.e., for any $k > 0$, if we stop running the algorithm after generating k bits then these k bits are independent and unbiased.*

Proof. According to lemma 4.7, for any $Y_1, Y_2 \in \{0,1\}^k$, the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping. If two sequences are one-to-one mapping, they have to be equivalent, which implies that their probabilities of being generated are the same. Hence, the probability of generating a sequence in S_{Y_1} is equal to that of generating a sequence in S_{Y_2}. It implies that Y_1 and Y_2 have the same probability of being generated for a fixed number k. Since this is true for any $Y_1, Y_2 \in \{0,1\}^k$, the probability of generating an arbitrary binary sequence $Y \in \{0,1\}^k$ is 2^{-k}. Finally, we have the statement in the theorem. □

4.2.3 Proof of Theorem 4.3

Lemma 4.8. *Given a stream of biased coin tosses, where the probability of generating H is p, we run the random-stream algorithm until the number of coin tosses reaches l. In this case, let m be the number of random bits generated, then for any $\epsilon, \delta > 0$, if l is large enough, we have that*

$$P[\frac{m - lH(p)}{lH(p)} < -\epsilon] < \delta,$$

where

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$$

is the entropy of the biased coin.

Proof. If we consider the case of fixed input length, then the random-stream algorithm is as efficient as Peres's algorithm asymptotically. Using the same proof given in [90] for Peres's algorithm, we can get

$$\lim_{l\to\infty} \frac{E[m]}{l} = H(p).$$

Given a sequence of coin tosses of length l, we want to prove that for any $\epsilon > 0$,

$$\lim_{l\to\infty} P[\frac{m - E[m]}{E[m]} < -\epsilon] = 0.$$

To prove this result, we assume that this limitation holds for $\epsilon = \epsilon_1$, i.e., for any $\delta > 0$, if l is large enough, then

$$P[\frac{m - E[m]}{E[m]} < -\epsilon_1] < \delta.$$

Under this assumption, we show that there always exists $\epsilon_2 < \epsilon_1$ such that the limitation also holds for $\epsilon = \epsilon_2$. Hence, the value of ϵ can be arbitrarily small.

In the random-stream algorithm, l is the number of symbols (coin tosses) received by the root. Let m_1 be the number of random bits generated by the root, $m_{(l)}$ be the number of random bits generated by its left subtree and $m_{(r)}$ be the number of random bits generated by its right subtree. Then it is easy to get

$$m = m_1 + m_{(l)} + m_{(r)}.$$

Since the m_1 random bits generated by the root node are independent, we can always make l large enough such that

$$P[\frac{m_1 - E[m_1]}{E[m_1]} < -\epsilon_1/2] < \delta/3.$$

At the same time, by making l large enough, we can make both $m_{(l)}$ and $m_{(r)}$ large enough such that (based on our assumption)

$$P[\frac{m_{(l)} - E[m_{(l)}]}{E[m_{(l)}]} < -\epsilon_1] < \delta/3$$

and

$$P[\frac{m_{(r)} - E[m_{(r)}]}{E[m_{(r)}]} < -\epsilon_1] < \delta/3.$$

Based on the three inequalities above, we can get

$$P[m - E[m] \leq -\epsilon_1(\frac{E[m_1]}{2} + E[m_{(l)}] + E[m_{(r)}])] < \delta.$$

If we set

$$\epsilon_2 = \epsilon_1 \frac{\frac{E[m_1]}{2} + E[m_{(l)}] + E[m_{(r)}]}{E[m_1 + m_{(l)} + m_{(r)}]},$$

then

$$P[\frac{m - E[m]}{E[m]} < -\epsilon_2] < \delta.$$

Given the probability p of the coin, when l is large,

$$E[m_1] = \Theta(E[m]), E[m_{(l)}] = \Theta(E[m]), E[m_{(r)}] = \Theta(E[m]),$$

which implies that $\epsilon_2 < \epsilon_1$.

So we can conclude that for any $\epsilon > 0, \delta > 0$, if l is large enough then

$$P[\frac{m - E[m]}{E[m]} < -\epsilon] < \delta.$$

And based on the fact that $E[m] \to lH(p)$, we get the result in the lemma. □

Theorem 4.3. *Given a biased coin with probability p being H, let n be the number of coin tosses required for generating k random bits in the random-stream algorithm, then*

$$\lim_{k\to\infty} \frac{E[n]}{k} = \frac{1}{H(p)}.$$

Proof. For any $\epsilon, \delta > 0$, we set $l = \frac{k}{H(p)}(1 + \epsilon)$, according to the conclusion of the previous lemma,

with probability at least $1-\delta$, the output length is at least k if the input length l is fixed and large enough. In another word, if the output length is k, which is fixed, then with probability at least $1-\delta$, the input length $n \leq l$.

So with probability less than δ, we require more than l coin tosses. The worst case is that we did not generate any bits for the first l coin tosses. In this case, we need to generate k more random bits. As a result, the expected number of coin tosses required is at most $l + E[n]$.

Based on the analysis above, we derive

$$E[n] \leq (1-\delta)l + (\delta)(l + E[n]),$$

then

$$E[n] \leq \frac{l}{1-\delta} = \frac{k}{H(p)} \frac{(1+\epsilon)}{(1-\delta)}.$$

Since ϵ, δ can be arbitrarily small when l (or k) is large enough

$$\lim_{k \to \infty} \frac{E[n]}{k} \leq \frac{1}{H(p)}.$$

Based on Shannon's theory [27], it is impossible to generate k random bits from a source with expected entropy less than k. Hence

$$\lim_{k \to \infty} E[nH(p)] \geq k,$$

i.e.,

$$\lim_{k \to \infty} \frac{E[n]}{k} \geq \frac{1}{H(p)}.$$

Finally, we get the conclusion in the theorem. This completes the proof. □

4.2.4 Proof of Theorem 4.4

The proof of theorem 4.4 is very similar as the proof of theorem 4.2. Let S_Y with $Y \in \{0,1\}^k$ denote the set consisting of all the binary sequences yielding Y in the random-stream algorithm with limited maximum depth. Then for any distinct binary sequences $Y_1, Y_2 \in \{0,1\}^k$, the elements in S_{Y_1} and those in S_{Y_2} are one-to-one mapping. Specifically, we can get the following lemma.

Lemma 4.9. *Let f be the function of the random-stream algorithm with maximum depth d. For any distinct binary sequences $Y_1, Y_2 \in \{0,1\}^k$, if $X_A \in S_{Y_1}$, there exists one sequence $X_B \in S_{Y_2}$ such that*

- $X_A \equiv X_B$.
- *Let $\mathcal{T}_A$ be the status tree of X_A and $\mathcal{T}_B$ be the status tree of X_B. For any node u with depth larger than d in $\mathcal{T}_A$, let v be its corresponding node in $\mathcal{T}_B$ at the same position, then u and v generate the same bits.*
- $f(X_A) = Y_1\Delta$ *and* $f(X_B) = Y_2\Delta$ *for some binary sequence* $\Delta \in \{0,1\}^*$.

Proof. The proof of this lemma is a simple modification of that for lemma 4.7, which is by induction. A simple sketch is given as follows.

First, similar as the proof for lemma 4.7, it can be proved that: when $k = 1$, for any sequence $X_A \in S_0$, there exists one sequence $X_B \in S_1$ such that X_A, X_B satisfy the conditions in the lemma, and vice versa. So we can say that the elements in S_0 and S_1 are one-to-one mapping. Then we assume that all the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping for all $Y_1, Y_2 \in \{0,1\}^k$, then we show that this conclusion also holds for any $Y_1, Y_2 \in \{0,1\}^{k+1}$. Two cases need to be considered.

1) Y_1, Y_2 end with the same bit. Without loss of generality, we assume this bit is 0, then we can write $Y_1 = Y_1'0$ and $Y_2 = Y_2'0$.

If $X_A \in S_{Y_1'}$, then according to our assumption, it is easy to prove the conclusion, i.e., there exists a sequence X_B satisfies the conditions.

If $X_A \notin S_{Y_1'}$, then we can write $X_A = \overline{X_A}Z$ and $\overline{X_A} \in S_{Y_1'}$. According to our assumption, for the sequence $\overline{X_A}$, we can find its mapping $\overline{X_B}$ such that (1) $\overline{X_A} \equiv \overline{X_B}$; (2) $\overline{X_A}, \overline{X_B}$ induce the

same status tree and their corresponding nodes with depth larger than d generate the same bits; and (3) $f(\overline{X_A}) = Y_1'$ and $f(\overline{X_B}) = Y_2'$. If we construct a sequence $X_B = X_B Z$, it will satisfy all the conditions in the lemma.

Since this result is also true for the inverse case, if Y_1, Y_2 end with same bit, the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping.

2) Y_1, Y_2 end with different bits. Without loss of generality, we assume that $Y_1 = Y_1'0$ and $Y_2 = Y_2'1$. According to the argument above, the elements in S_{0^k0} and S_{Y_1} are one-to-one mapping; and the elements in S_{0^k1} and S_{Y_2} are one-to-one mapping. So we only need to prove that the elements in S_{0^k0} and S_{0^k1} are one-to-one mapping. In this case, for any $X_A \in S_{0^{k-1}0}$, let $X_A = X_A'\beta$ with a single symbol β. Then X_A' generates only zeros whose length is at most k. Let $\mathcal{T}_A'$ denote the status tree of X_A' and let u be the node in $\mathcal{T}_A'$ that generates the $k+1$th bit (zero) when reading the symbol β. Note that the depth of u is at most d. In this case, we can construct a new sequence X_B' with status tree $\mathcal{T}_B'$ such that

- $\mathcal{T}_B'$ and $\mathcal{T}_A'$ are the same except the label of u is 0 and the label of the node at the same position in $\mathcal{T}_B'$ is 1.
- For each node u in $\mathcal{T}_A'$, let v be its corresponding node at the same position in $\mathcal{T}_B'$, then u and v generate the same bits.

Then we can prove that the sequence $X_B = X_B'\beta$ satisfies our all our conditions in the lemma. Also based on the inverse argument, we can claim that the elements in S_{0^k0} and S_{0^k1} are one-to-one mapping.

Finally, we can conclude that the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping for any $Y_1, Y_2 \in \{0,1\}^k$ with $k > 0$. □

From the above lemma, it is easy to get theorem 4.4.

Theorem 4.4. *Given a source of biased coin with unknown probability, the random-stream algorithm with maximum depth d generates a stream of random bits, i.e., for any $k > 0$, if we stop running the algorithm after generating k bits then these k bits are independent and unbiased.*

Proof. We can apply the same procedure of proving theorem 4.3. □

4.2.5 Proof of Theorem 4.5

Similar to the proof of Theorem 4.3, we first consider the case that the input length is fixed.

Lemma 4.10. *Given a stream of biased coin tosses, where the probability of generating H is p, we run the random-stream algorithm with maximum depth d until the number of coin tosses reaches l. In this case, let m be the number of random bits generated, then for any $\epsilon, \delta > 0$, if l is large enough, we have that*

$$P[\frac{m - l\rho_d(p)}{l\rho_d(p)} < -\epsilon] < \delta,$$

where $\rho_d(p)$ is given in (4.1).

Proof. Let $\rho_d(p)$ be the asymptotic expected number of random bits generated per coin toss when the random-stream algorithm has maximum depth d and the probability of the biased coin is p. Then

$$\lim_{l\to\infty} \frac{E[m]}{l} = \rho_d(p).$$

When the fixed input length l is large enough, the random-stream algorithm generates approximately $l\rho_d(p)$ random bits, which are generated by the root node, the left subtree (subtree rooted at root's left child) and the right subtree (subtree rooted at the root's right child). Considering the root node, it generates approximately lpq random bits with $q = 1 - p$. At the same time, the root node passes approximately $\frac{l}{2}$ messages (H or T) to its left child, where the messages are independent and the probability of H is $p^2 + q^2$; and the root node passes approximately $\frac{l}{2}(p^2 + q^2)$ messages (H or T) to its right child, where the messages are independent and the probability of H is $\frac{p^2}{p^2+q^2}$. As a result, according to the definition of ρ_d, the left subtree generates approximately $\frac{l}{2}\rho_{d-1}(p^2 + q^2)$ random bits, and the right subtree generates approximately $\frac{l}{2}(p^2 + q^2)\rho_{d-1}(\frac{p^2}{p^2+q^2})$ random bits. As $l \to \infty$, we have

$$\lim_{l\to\infty} \frac{l\rho_d(p)}{lpq + \frac{l}{2}\rho_{d-1}(p^2 + q^2) + \frac{l}{2}(p^2 + q^2)\rho_{d-1}(\frac{p^2}{p^2+q^2})} = 1.$$

It yields

$$\rho_d(p) = pq + \frac{1}{2}\rho_{d-1}(p^2+q^2) + \frac{1}{2}(p^2+q^2)\rho_{d-1}(\frac{p^2}{p^2+q^2}).$$

So we can calculate $\rho_d(p)$ by iteration. When $d = 0$, the status tree has the single root node, and it is easy to get $\rho_0(p) = pq$.

Then, following the proof of lemma 4.8, for any $\epsilon, \delta > 0$, if l is large enough, we have that

$$P[\frac{m - E[m]}{E[m]} < -\epsilon] < \delta.$$

So we can get the conclusion in the lemma. This completes the proof. □

From the above lemma, we can get theorem 4.5, that is,

Theorem 4.5. *When the maximum depth of the tree is d and the probability of the biased coin is p of being H, the expected number of coin tosses required per random bit is*

$$\frac{1}{\rho_d(p)}$$

asymptotically, where $\rho_d(p)$ can be obtained by iterating

$$\rho_d(p) = pq + \frac{1}{2}\rho_{d-1}(p^2+q^2) + \frac{1}{2}(p^2+q^2)\rho_{d-1}(\frac{p^2}{p^2+q^2})$$

with $q = 1 - p$ and $\rho_0(p) = pq$.

Proof. We can apply the same procedure of proving theorem 4.2 except we apply lemma 4.10 instead of lemma 4.8. □

4.3 Generalized Random-Stream Algorithm

4.3.1 Preliminary

In chapter 2, we introduced a universal scheme for transforming an arbitrary algorithm for generating random bits from a sequence of biased coin tosses to manage the general source of an m-sided die. This scheme works when the input is a sequence of fixed length; in this section, we study how to modify this scheme to generate random-bit streams from m-sided dice. For sake of completeness we describe the original scheme here.

The main idea of the scheme is to convert a sequence with alphabet larger than two, written as

$$X = x_1 x_2 ... x_n \in \{0, 1, ..., m-1\}^n,$$

into multiple binary sequences. To do this, we create a binary tree, called a binarization tree, in which each node is labeled with a binary sequence of H and T. Given the binary representations of x_i for all $1 \leq i \leq n$, the path of each node in the tree indicates a prefix, and the binary sequence labeled at this node consists of all the bits (H or T) following the prefix in the binary representations of $x_1, x_2, ..., x_n$ (if it exists). Fig. 2.1 is an instance of binarization tree when the input sequence is $X = 012112210$, produced by a 3-sided die. To see this, we write each symbol (die roll) into a binary representation of length two, hence X can be represented as

TT,TH,HT,TH,TH,HT,HT,TH,TT.

Only collecting the first bits of all the symbols yields an independent binary sequence

$$X_\phi = \text{TTHTTHHTT},$$

which is labeled on the root node; Collecting the second bits following T, we get another independent

binary sequence

$$X_{\mathrm{T}} = \mathrm{THHHHT},$$

which is labeled on the left child of the root node.

The universal scheme says that we can 'treat' each binary sequence labeled on the binarization tree as a sequence of biased coin tosses: Let Ψ be any algorithm that can generate random bits from an arbitrary biased coin, then applying Ψ to each of the sequences labeled on the binarization tree and concatenating their outputs together results in an independent and unbiased sequence, namely, a sequence of random bits.

Specifically, given the number of sides m of a loaded die, the depth of the binarization tree is $b = \lceil \log_2 m \rceil - 1$. Let Υ_b denote the set consisting of all the binary sequences of length at most b, i.e.,

$$\Upsilon_b = \{\phi,\ \mathrm{T},\ \mathrm{H},\ \mathrm{TT},\ \mathrm{TH},\ \mathrm{HT},\ \mathrm{HH},\ ...,\ \mathrm{HHH...HH}\}.$$

Given $X \in \{0, 1, ..., m-1\}^n$, let X_γ denote the binary sequence labeled on a node corresponding to a prefix γ in the binarization tree, then we get a group of binary sequences

$$X_\phi, X_{\mathrm{T}}, X_{\mathrm{H}}, X_{\mathrm{TT}}, X_{\mathrm{TH}}, X_{\mathrm{HT}}, X_{\mathrm{HH}}, ...$$

For any function Ψ that generates random bits from a fixed number of coin tosses, we can generate random bits from X by calculating

$$\Psi(X_\phi) + \Psi(X_{\mathrm{T}}) + \Psi(X_{\mathrm{H}}) + \Psi(X_{\mathrm{TT}}) + \Psi(X_{\mathrm{TH}}) + ...,$$

where $A + B$ is the concatenation of A and B.

So in the above example, the output of $X = 012112210$ is $\Psi(X_\phi) + \Psi(X_{\mathrm{T}})$, i.e.,

$$\Psi(\mathrm{TTHTTHHTT}) + \Psi(\mathrm{THHHHT}).$$

This conclusion is simple, but not obvious, since the binary sequences labeled on the same binarization tree are correlated with each other.

4.3.2 Generalized Random-Stream Algorithm

We want to generalize the random-stream algorithm to generate random-bit streams from an m-sided die. Using the similar idea as above, we convert the input stream into multiple binary streams, where each binary stream corresponds to a node in the binalization tree. We apply the random-stream algorithm to all these binary streams individually, and for each stream we create a status tree for storing state information. When we read a dice roll of m faces from the source, we pass all the $\log_2 m$ bits of its binary representation to $\lceil \log_2 m \rceil$ different streams that corresponds to a path in the binalization tree. Then we process all these $\lceil \log_2 m \rceil$ streams from top to bottom along that path. In this way, a single binary stream is produced. While each node in the binalization tree generates a random-bit stream, the resulting single stream is a mixture of these random-bit streams. But it is not obvious whether the resulting stream is a random-bit stream or not, since the values of the bits generated affect their orders.

The following example is constructed for demonstrating this algorithm.

Let us consider a stream of symbols generated from a 3-sided die,

012112210...

Instead of storing a binary sequence at each node in the binalization tree, we associate each node with a status tree corresponding to a random-stream algorithm. Here, we get two nontrivial binary streams

TTHTTHHTT..., THHHHT...

corresponding to prefix ϕ and T respectively, figure 4.4 demonstrates how the status trees change when we read symbols one by one. For instance, when the 4th symbol 1(TH) is read, it passes T to the root node (corresponding to the prefix ϕ) and passes H to the left child of the root node

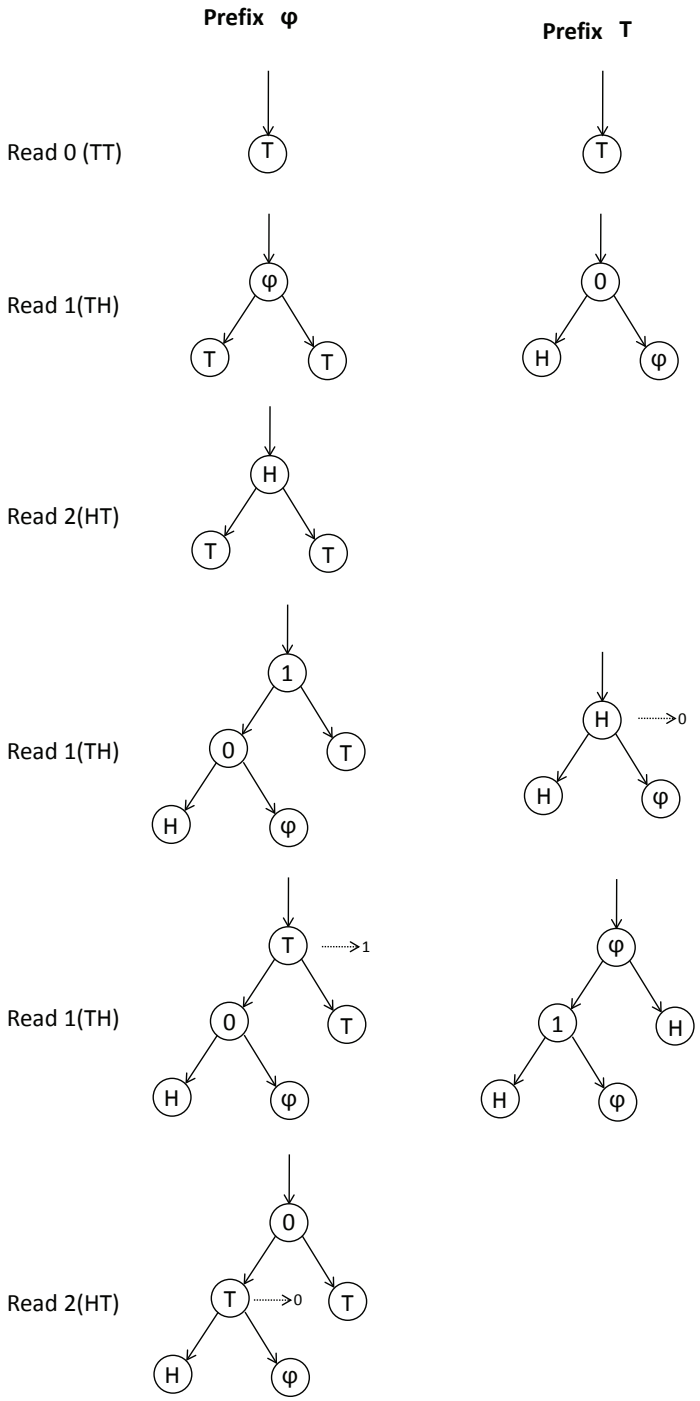


Figure 4.4. The changes of status trees in the generalized random-stream algorithm when the input stream is 012112210....

(corresponding to the prefix T) of the binalization tree. Based on the rules of the random-stream algorithm, we modify the status trees associated with these two nodes. During this process, a bit 0

is generated.

Finally, this scheme generates a stream of bits 010..., where the first bit is generated after reading the 4th symbol, the second bit is generated after reading the 5th symbol, ... We call this scheme as the generalized random-stream algorithm. As we expected, this algorithm can generate a stream of random bits from an arbitrary loaded die with $m \geq 2$ faces.

Theorem 4.11. *Given a loaded die with $m \geq 2$ faces, if we stop running the generalized random-stream algorithm after generating k bits, then these k bits are independent and unbiased.*

The proof of the above theorem is given in subsection 4.3.3.

Since the random-stream algorithm is as efficient as Peres's algorithm asymptotically, we can prove that the generalized random-stream algorithm is also asymptotically optimal.

Theorem 4.12. *Given an m-sided die with probability distribution $\rho = (p_0, p_1, ..., p_{m-1})$, let n be the number of symbols (dice rolls) used in the generalized random-stream algorithm and let k be the number of random bits generated, then*

$$\lim_{k\to\infty} \frac{E[n]}{k} = \frac{1}{H(p_0, p_1, ..., p_{m-1})},$$

where

$$H(p_0, p_1, ..., p_{m-1}) = \sum_{i=0}^{m-1} p_i \log_2 \frac{1}{p_i}$$

is the entropy of the m-sided die.

Proof. First, according to Shannon's theory, it is easy to get that

$$\lim_{k\to\infty} \frac{E[n]}{k} \geq \frac{1}{H(p_0, p_1, ..., p_{m-1})}.$$

Now, we let $n = \frac{k}{H(p_0,p_1,...,p_{m-1})}(1+\epsilon)$ with an arbitrary $\epsilon > 0$. Following the proof of theorem 2.7 in chapter 2, it can be shown that when k is large enough, the algorithm generates more than k random bits with probability at least $1 - \delta$ with any $\delta > 0$. Then using the same argument in

theorem 4.3, we can get

$$\lim_{k\to\infty}\frac{E[n]}{k}\leq\frac{1}{H(p_0,p_1,...,p_{m-1})}\frac{1+\epsilon}{1-\delta},$$

for any $\epsilon,\delta>0$.

Hence, we can get the conclusion in the theorem. □

Of source, we can limit the depths of all the status trees for saving space, with proof emitted. It can be seen that given a loaded die of m faces, the space usage is proportional to m and the expected computational time is proportional to $\log m$.

4.3.3 Proof of Theorem 4.11

Here, we want to prove that the generalized random-stream algorithm generates a stream of random bits from an arbitrary m-sided die. Similar as above, we let S_Y with $Y\in\{0,1\}^k$ denote the set consisting of all the sequences yielding Y. Here, we say that a sequence X yields Y if and only if $X[1:|X|-1]$ generates a sequence shorter than Y and X generates a sequence with Y as a prefix (including Y itself). We would like to show that the elements in S_{Y_1} and those in S_{Y_2} are one-to-one mapping if Y_1 and Y_2 have the same length.

Definition 4.2. *Two sequences $X_A, X_B\in\{0,1,...,m-1\}^*$ with $m>2$ are equivalent, denoted by $X_A\equiv X_B$, if and only $X_\gamma^A\equiv X_\gamma^B$ for all $\gamma\in\Upsilon_b$, where X_γ^A is the binary sequence labeled on a node corresponding to a prefix γ in the binalization tree induced by X_A, and the equivalence of X_γ^A and X_γ^B was given in definition 4.1.*

Lemma 4.13. *Let f be the function of the generalized random-stream algorithm, and let X_A be a sequence produced by an m-sided die. For any distinct sequences $Y_1,Y_2\in\{0,1\}^k$, if $X_A\in S_{Y_1}$, there are exactly one sequence $X_B\in S_{Y_2}$ such that*

- $X_B\equiv X_A$.
- $f(X_A)=Y_1\Delta$ *and* $f(X_B)=Y_2\Delta$ *for some binary sequence* Δ.

Proof. The idea of the proof is to combine the proof of lemma 4.7 with the result in lemma 2.4 in chapter 2.

Let us prove this conclusion by induction. Here, we use X'_A to denote the prefix of X_A of length $|X_A| - 1$ and use β to denote the last symbol of X_A. So $X_A = X'_A\beta$. X^A_γ is the binary sequence labeled on a node corresponding to a prefix γ in the binalization tree induced by X'_A, and the status tree of $X^{A\prime}_\gamma$ with $\gamma \in \Upsilon_b$ is denoted as $\mathcal{T}^A_\gamma$.

When $k = 1$, if $X_A \in S_0$, we can write $f(X_A)$ as 0Δ. In this case, let u in $\mathcal{T}^A_\theta$ with $\theta \in \Upsilon_b$ be the node that generates the first bit 0. If we flip the label of u from 0 to 1, we get another status tree $\mathcal{T}^B_\theta$. Using the same argument in lemma 4.6, we are able to construct a sequence X^B_θ such that its status tree is $\mathcal{T}^B_\theta$ and it does not generate any bits. Here, X^B_θ is a permutation of X^A_θ. From $X^A_\phi, X^A_T, ..., X^B_\theta, ...$, we can construct a sequence X'_B uniquely following the procedure in lemma 2.4 in chapter 2. Concatenating X'_B with β results in a new sequence X_B, i.e., $X_B = X'_B\beta$ such that $X_B \equiv X_A$ and $f(X_B) = 1\Delta$. Inversely, we can get the same result. It shows that the elements in S_0 and S_1 are one-to-one mapping.

Now we assume that the conclusion holds for all $Y_1, Y_2 \in \{0,1\}^k$, then we show that it also holds for any $Y_1, Y_2 \in \{0,1\}^{k+1}$. Two cases need to be considered.

1) Y_1, Y_2 end with the same bit. Without loss of generality, we assume that this bit is 0, then we can write $Y_1 = Y'_1 0$ and $Y_2 = Y'_2 0$. If X_A yields Y_1, based on our assumption, it is easy to see that there exists a sequence X_B satisfies our requirements. If X_A does not yield Y_1, that means Y'_1 has been generated before reading the symbol β. Let us consider a prefix of X_A, denote by $\overline{X_A}$, such that it yields Y'_1. In this case, $f(X'_A) = Y'_1$ and we can write $X_A = X'_A Z$. According to our assumption, there exists exactly one sequence $\overline{X_B}$ such that $\overline{X_B} \equiv \overline{X_A}$ and $f(X'_B) = Y'_2$. Since $\overline{X_A}$ and $\overline{X_B}$ lead to the same binalization tree (all the status trees at the same positions are the same), if we construct a sequence $X_B = \overline{X_B}Z$, then $X_B \equiv X_A$ and X_B generates the same bits as X_A when reading symbols from Z. It is easy to see that such a sequence X_B satisfies our requirements.

Since this result is also true for the inverse case, if $Y_1, Y_2 \in \{0,1\}^{k+1}$ end with the same bit, the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping.

2) Let us consider the case that Y_1, Y_2 end with different bits. Without loss of generality, we assume that $Y_1 = Y_1'0$ and $Y_2 = Y_2'1$. According to the argument above, the elements in $S_{00...00}$ and $S_{Y_1'0}$ are one-to-one mapping; the elements in $S_{00..01}$ and $S_{Y_2'1}$ are one-to-one mapping. So our task is to prove that the elements in $S_{00..00}$ and $S_{00...01}$ are one-to-one mapping. For any sequence $X_A \in S_{00...00}$, let X'_A be its prefix of length $|X_A| - 1$. Here, X'_A generates only zeros whose length is at most k. Let $\mathcal{T}_\theta^A$ denote one of the status trees such that $u \in T_\theta^A$ is the node that generates that $k+1th$ bit (zero) when reading the symbol β. Then we can construct a new sequence X'_B such that

- Let $\{X_\gamma^B\}$ with $\gamma \in \Upsilon_b$ be the binary sequences induced by X'_B, and let $\mathcal{T}_\gamma^B$ be the status tree of X_γ^B. The binalization trees of X'_A and X'_B are the same (all the status trees at the same positions are the same), except the label of u is 0 and the label of its corresponding node v in $\mathcal{T}_\theta^B$ is 1.

- Each node u in $\mathcal{T}_\gamma^B$ generates the same bits as its corresponding node v in $\mathcal{T}_\gamma^A$ for all $\gamma \in \Upsilon_b$.

The construction of X'_B follows the proof of lemma 4.1 and then lemma 2.4 in chapter 2. If we construct a sequence $X_B = X'_B\beta$, it is not hard to show that X_B satisfies our requirements, i.e.,

- $X_B \equiv X_A$;

- X'_B generates less than $k+1$ bits, i.e., $|f(X'_B)| \leq k$;

- If $f(X_A) = Y_1\Delta = Y_1'0\Delta$, then $f(X_B) = Y_2'1\Delta = Y_2\Delta$.

Also based on the inverse argument, we see that the elements in $S_{00..00}$ and $S_{00...01}$ are one-to-one mapping.

Finally, we can conclude that the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping for any $Y_1, Y_2 \in \{0,1\}^k$ with $k > 0$. □

Based on the above result and the argument for theorem 4.2, we can easily prove theorem 4.11.

Theorem 4.11. *Given a loaded die with $m \geq 2$ faces, if we stop running the generalized random-stream algorithm after generating k bits, then these k bits are independent and unbiased.*

4.4 Extension for Markov Chains

In this section, we study how to efficiently generate random-bit streams from Markov chains. The nonstream case was studied by Samuelson [101], Blum [14] and later generalized by Zhou and Bruck, see chapter 3. Here, using the techniques developed in chapter 3, and applying the techniques introduced in this chapter, we are able to generate random-bit streams from Markov chains. We present the algorithm briefly.

For a given Markov chain, it generates a stream of states, denoted by $x_1x_2x_3... \in \{s_1, s_2, ..., s_m\}^*$. We can treat each state, say s, as a die and consider the 'next states' (the states the chain has transitioned to after being at state s) as the results of a die roll, called the exit of s. For all $s \in \{s_1, s_2, ..., s_m\}$, if we only consider the exits of s, they form a stream. So we have total m streams corresponding to the exits of $s_1, s_2, ..., s_m$ respectively. For example, assume the input is

$$X = s_1s_4s_2s_1s_3s_2s_3s_1s_1s_2s_3s_4s_1...$$

If we consider the states following s_1, we get a stream as the set of states in boldface:

$$X = s_1\mathbf{s_4}s_2s_1\mathbf{s_3}s_2s_3s_1\mathbf{s_1}\mathbf{s_2}s_3s_4s_1...$$

Hence the four streams are

$$s_4s_3s_1s_2..., s_1s_3s_3..., s_2s_1s_4..., s_2s_1...$$

The generalized random-stream algorithm is applied to each stream separately for generating random-bit streams. Here, when we get an exit of a state s, we should not directly pass it to the generalized random-stream algorithm that corresponds to the state s. Instead, it waits until we get the next exit of the state s. In another word, we keep the current exit in pending. In the above example, after we read $s_1s_4s_2s_1s_3s_2s_3s_1s_1s_2s_3s_4s_1$, $s_4s_3s_1$ has been passed to the generalized random-stream algorithm corresponding to s_1, s_1s_3 has been passed to the generalized random-

stream algorithm corresponding to s_2,...the most recent exit of each state, namely s_2, s_3, s_4, s_1 are in pending. Finally, we mix all the bits generated from different streams based on their natural generating order. As a result, we get a stream of random bits from an arbitrary Markov chain, and it achieves the information-theoretic upper bound on efficiency.

Now, we call this algorithm the random-stream algorithm for Markov chains, and we describe it as follows.

Input: A stream $X = x_1x_2x_3...$ produced by a Markov chain, where $x_i \in S = \{s_1, s_2, ..., s_m\}$.

Output: A stream of $0'$s and $1'$s.

Main Function:

Let Φ_i be the generalized random-stream algorithm for the exits of s_i for $1 \leq i \leq m$, and θ_i be the pending exit of s_i for $1 \leq i \leq m$.

Set $\theta_i = \phi$ for $1 \leq i \leq m$.

for each symbol x_j read from the Markov chain **do**

if $x_{j-1} = s_i$ **then**

if $\theta_i \neq \phi$ **then**

Input θ_i to Φ_i for processing.

end if

Set $\theta_i = x_j$.

end if

end for

Theorem 4.14. *Given a source of a Markov chain with unknown transition probabilities, the random-stream algorithm for Markov chains generates a stream of random bits, i.e., for any $k > 0$, if we stop running the algorithm after generating k bits then these k bits are independent and unbiased.*

The proof of the above theorem is a simple extension of the proof for theorem 4.11. Let S_Y denote the set of input sequences that yield a binary sequence Y. Our main idea is still to prove that all the elements in S_{Y_1} and S_{Y_2} are one-to-one mapping for all $Y_1, Y_2 \in \{0,1\}^k$ with $k > 0$. The

detailed proof is a little complex, but it is not difficult; we only need to follow the proof of theorem 4.11 and combine it with the following result from chapter 3. Here, we omit the detailed proof.

Lemma 4.15. *Given an input sequence* $X = x_1x_2...x_N \in \{s_1, s_2, ..., s_m\}^N$ *that produced from a Markov chain, let* $\pi_i(X)$ *be the exit sequence of* s_i *(the symbols following* s_i*) for* $1 \leq i \leq m$*. Assume that* $[\Lambda_1, \Lambda_2, ..., \Lambda_n]$ *is an arbitrary collection of exit sequences such that* Λ_i *and* $\pi_i(X)$ *are permutations and they have the same last element for all* $1 \leq i \leq m$*. Then there exists a sequence* $X' = x'_1x'_2...x'_N \in \{s_1, s_2, ..., s_m\}^N$ *such that* $x'_1 = x_1$ *and* $\pi_i(X') = \Lambda_i$ *for all* $1 \leq i \leq m$*. For this* X'*, we have* $x'_N = x_N$*.*

4.5 Conclusion

In this chapter, we addressed the problem of generating random-bit streams from i.i.d. sources with unknown distributions. First, we considered the case of biased coins and derived a simple algorithm to generate random-bit streams. This algorithm achieves the information-theoretic upper bound on efficiency. We showed that this algorithm can be generalized to generate random-bit streams from an arbitrary m-sided die with $m > 2$, and its information efficiency is also asymptotically optimal. Furthermore, we demonstrated that by applying the (generalized) random-stream algorithm, we can generate random-bit streams from an arbitrary Markov chain very efficiently.

Part II

Randomness Extraction

Chapter 5

Linear Transformations for Extracting Randomness

This chapter studies linear transformations for randomness extraction and shows that sparse random matrices are very powerful for extracting randomness from many noisy sources, which are very attractive in the practical use of high-speed random number generators due to their simplicity.[1]

5.1 Introduction

Randomness plays an important role in many fields, including complexity theory, cryptography, information theory and optimization. There are many randomized algorithms that are faster, more space efficient or simpler than any known deterministic algorithms [86]; hence, how to generate random numbers becomes an essential question in computer science. Pseudo-random numbers have been studied, but they cannot perfectly simulate truly random bits or have security issues in some applications. These problems motivate people to extract random bits from natural sources directly. In this chapter, we study linear transformation for randomness extraction. This approach is attractive due to its computational simplicity and information efficiency. Specifically, given an input binary sequence X of length n generated from an imperfect source, we construct an $n \times m$ binary

[1] Some of the results presented in this chapter have been previously published in [140].

matrix M called a transformation matrix such that the output sequence

$$Y = XM$$

is very close to the uniform distribution on $\{0,1\}^m$. Statistical distance [105] is commonly used to measure the distance between two distributions in randomness extraction. We say $Y \in \{0,1\}^m$ is ϵ-close to the uniform distribution U_m on $\{0,1\}^m$ if and only if

$$\frac{1}{2} \sum_{y \in \{0,1\}^m} |P[Y = y] - 2^{-m}| \leq \epsilon, \tag{5.1}$$

where $\epsilon > 0$ can be arbitrarily small. This condition guarantees that in any probabilistic application, if we replace truly random bits with the sequence Y, the additional error probability caused by the replacement is at most ϵ.

The classical question in random number generation considers ideal sources, like biased coins or Markov chains, as described in chapters 2, 3, 4. Although it is known how to extract random bits optimally from biased coins or Markov chains, these models are too narrow to describe real sources that suffer noise and disturbance. During last two decades, research has been focused on a general source model called k-sources [149], in which each possible sequence has probability at most 2^{-k} of being generated. This model can cover a very wide range of natural random sources, but it was shown that it is impossible to derive a single function that extracts even a single bit of randomness from such a source. This observation led to the introduction of seeded extractors, which use a small number of truly random bits as the seed (catalyst). When simulating a probabilistic algorithm, one can simply eliminate the requirement of truly random bits by enumerating all possible strings for the seed and taking a majority vote. There are a variety of very efficient constructions of seeded extractors, summarized in [32, 87, 105]. Although seeded extractors are information-efficient and applicable to most natural sources, they are not computationally fast when simulating probabilistic algorithms. Recently, there is renewed interest in designing seedless extractors, called deterministic extractors. Several specific classes of sources have been studied, including independent sources, which

can be divided into several independent parts consisting of certain amounts of randomness [9,95–97]; bit-fixing sources, where some bits in a binary sequence are truly random and the remaining bits are fixed [23, 41, 64]; and samplable sources, where the source is generated by a process that has a bounded amount of computational resources like space [63, 116].

Unlike prior works on deterministic extractors, we take both simplicity and efficiency into consideration. Simplicity is certainly an important issue; for example, it motivates the use of von Neumann's scheme [128] in Intel's random number generator (RNG) [62] rather than some other more sophisticated extractors. However, von Neumann's scheme is far from optimal in its efficiency, and it only works for ideal biased coins. Recently, in order to support future generations of hardware security in systems operating at ultrafast bit rates, many high-speed random number generators based on chaotic semiconductor lasers have been developed [118]. They can generate random bits at rates as high as $12.5 - 400$ Gbit/s [5, 65, 98]; hence, the simplicity of post-processing is becoming more important. These challenges motivate us to develop extractors that can extract randomness from natural sources in a manner that reaches the theoretical upper bound on efficiency without compromising simplicity. In particular, we focus on linear constructions; that is, we apply linear transformations for randomness extraction.

Our main contribution is to show that linear transformations based on sparse random matrices are asymptotically optimal for extracting randomness from independent sources and bit-fixing sources, and they are efficient (although not necessarily optimal) for extracting randomness from hidden Markov sources. We further show that these conclusions hold if we apply any invertible linear mapping on the sources. In fact, many natural sources for the purpose of high-speed random number generation are qualified to fit one of the above models or their mixture, making the construction based on sparse random matrices very attractive in practical use. The resulting extractors are not seeded extractors, which consume truly random bits whenever extracting randomness. They are, in some sense, probabilistic constructions of deterministic extractors. In addition, we explore explicit constructions of transformation matrices. We show that the generator matrices of primitive BCH codes are good choices, but linear transformations based on such matrices require more

computational time due to their high densities.

The remainder of this chapter is organized as follows. In section 5.2 we give an intuitive overview of linear transformations for randomness extraction and present some general properties. In section 5.3, we introduce the source models to be addressed in this chapter and briefly describe our main results. The detailed discussions for each source model, including independent sources, hidden Markov sources, bit-fixing sources and linear-subspace sources, are given in section 5.4, section 5.5, section 5.6 and section 5.7, respectively. In section 5.8, we briefly describe implementation issues followed by concluding remarks in section 5.9.

5.2 Linear Transformations

Let us start from a simple and fundamental question in random number generation: given a set of coin tosses $x_1, x_2, ..., x_n$ with $P[x_i = 1] \in [\frac{1}{2} - \delta, \frac{1}{2} + \delta]$, how can we simulate a single coin toss such that is as unbiased as possible? This question has been well studied and it is known that binary sum operation is optimal among all the methods, i.e., we generate a bit z which is

$$z = x_1 + x_2 + ... + x_n \bmod 2.$$

The following lemma shows that binary sum operation can decrease the bias of the resulting coin toss exponentially.

Lemma 5.1. [73] *Let $x_1, x_2, ..., x_n$ be n independent bits and the bias of x_i is δ_i, namely,*

$$\delta_i = |P[x_i = 1] - \frac{1}{2}|$$

for $1 \leq i \leq n$, then the bias of $z = x_1 + x_2 + ... + x_n \bmod 2$ is upper bounded by

$$\frac{\prod_{i=1}^{n}(2\delta_i)}{2}.$$

A generalization of the above question is that: given n independent bits, how do we generate $m < n$ random bits such that their statistical distance to the truly random bits is as small as possible? One way is to divide all the n independent bits into m nonoverlap groups, denoted by $S_1, S_2, ..., S_m$, such that $\bigcup_{i=1}^{m} S_i = \{x_1, x_2, ..., x_n\}$. For $1 \leq i \leq m$, the ith output bit, denoted by y_i, is produced by summing up the bits in S_i and modulo two. However, this method is not very efficient. By allowing overlaps between different groups, the efficiency can be significantly improved. In this case, although we have sacrificed a little independence of the output bits, but the bias of each bit has been reduced a lot. An equivalent way of presenting this method is to use a binary matrix, denoted by M, such that $M_{ij} = 1$ if and only if $x_i \in S_j$, otherwise, $M_{ij} = 0$. As a result, the output of this method is $Y = XM$ for a given input sequence X. This is an intuitive understanding why linear transformations can be used in random extraction from weak random sources, in particular, from independent sources.

In this chapter, we study linear transformations for extracting randomness from a few types of random sources. Given a source $X \in \{0,1\}^n$, we design a transformation matrix M such that the output $Y = XM$ is arbitrarily close to truly random bits. Here, we use the statistical distance between Y and the uniform distribution over $\{0,1\}^m$ to measure the goodness of the output sequence Y, defined by

$$\rho(Y) = \frac{1}{2} \sum_{y \in \{0,1\}^m} |P[Y = y] - 2^{-m}|. \tag{5.2}$$

It indicates the maximum error probability introduced by replacing truly random bits with the sequence Y in any randomized algorithm.

Given a random source X and a matrix M, the following lemma shows an upper bound of $\rho(XM)$.

Lemma 5.2. *Let $X = x_1x_2...x_n$ be a binary sequence generated from an arbitrary random source and let M be an $n \times m$ binary matrix with $m \leq n$. Then given $Y = XM$, we have*

$$\rho(Y) \leq \sum_{u \in \{0,1\}^m, u \neq 0} |P_X[XMu^T = 1] - \frac{1}{2}|.$$

Proof. Similar as the idea in [73], for all $y \in \{0,1\}^m$, we define function h as $h(y) = P(Y = y)$. For this function, its Fourier transform is denoted by F_h, then

$$\forall y \in \{0,1\}^m, h(y) = 2^{-m} \sum_{u \in \{0,1\}^m} F_h(u)(-1)^{y \cdot u},$$

and

$$\forall u \in \{0,1\}^m, F_h(u) = \sum_{y \in \{0,1\}^m} h(y)(-1)^{y \cdot u}.$$

When $u = 0$, we have

$$|F_h(u)| = \sum_{y \in \{0,1\}^m} h(y) = 1.$$

When $u \neq 0$, we have

$$\begin{aligned} |F_h(u)| &= |\sum_{y \in \{0,1\}^m} h(y)(-1)^{y \cdot u}| \\ &= |\sum_{y \cdot u = 0} h(y) - \sum_{y \cdot u = 1} h(y)| \\ &= |1 - 2\sum_{y \cdot u = 1} h(y)| \\ &= 2|P[XMu^T = 1] - \frac{1}{2}|. \end{aligned} \tag{5.3}$$

Substituting (5.3) into (5.2) leads to

$$\begin{aligned} \rho(Y) &= \frac{1}{2} \sum_{y \in \{0,1\}^m} |2^{-m} \sum_{u \in \{0,1\}^m} F_h(u)(-1)^{y \cdot u} - 2^{-m}| \\ &\leq \frac{1}{2} \sum_{y \in \{0,1\}^m} 2^{-m} \sum_{u \neq 0} |F_h(u)| \\ &= \frac{1}{2} \sum_{u \neq 0} |F_h(u)| \\ &\leq \sum_{u \neq 0} |P[XMu^T = 1] - \frac{1}{2}|. \end{aligned} \tag{5.4}$$

This completes the proof. □

There are some related works focusing on the constructions of linear transformations for the

purpose of randomness extraction. In [72], Lacharme studied linear correctors, and his goal is to generate a random sequence Y of length m such that

$$\max_{y\in\{0,1\}^m} |P[Y=y]-2^{-m}| \le \epsilon$$

for a specified small constant ϵ. At almost the same time as our work, in [1], Abbe uses polar codes to construct deterministic extractors. His idea is that given an independent sequence X and let $X' = XG_n$ with $G_n = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\bigoplus \log_2 n}$, then a subset of components in X' are roughly i.i.d. uniform and the remaining components are roughly deterministic. It was proved that this approach can generate a random sequence Y of length m and with entropy at least $m(1-\epsilon)$. In both of the works above, the random bits generated are 'weaker' than the requirement of statistical distance. For instance, let Y be a random sequence of length m, and assume $P[Y=y]$ with $y \in \{0,1\}^m$ is either $2^{-(m-1)}$ or 0. In this case, as $m \to \infty$, we have

$$\max_{y\in\{0,1\}^m} |P[Y=y]-2^{-m}| \to 0;$$

$$1-\frac{H(Y)}{m} = \frac{1}{m} \to 0.$$

That means this sequence Y satisfies the requirement of randomness in both of the works. But if we consider the statistical distance of Y to the uniform distribution on $\{0,1\}^m$, it is

$$\rho(Y) = \frac{1}{2}\sum_{y\in\{0,1\}^m} |P[Y=y]-2^{-m}| = \frac{1}{4}.$$

That does not satisfy our requirement of randomness in the sense of statistical distance. From this point, we generate random bits with higher requirement on quality than the above works.

In the rest of this chapter, we investigate those random sources on $\{0,1\}^n$ such that by applying linear transformations we can get a random sequence Y with $\rho(Y) \to 0$ as $n \to \infty$.

5.3 Source Models and Main Results

In this section, we introduce a few types of random sources including independent sources, hidden Markov sources, bit-fixing sources, and linear-subspace sources, and we summarize our main results for each type of sources. Two constructions of linear transformations will be presented and analyzed. The first construction is based on sparse random matrices. We say a random matrix with each entry being one with probability p is sparse if and only if p is small and $p = w(\frac{\log n}{n})$ that means $p > \frac{k}{\log n} n$ for any fixed $k > 0$ when the source length $n \to \infty$. The second construction is explicit – it is based on the generator matrices of linear codes with binomial weight distributions. The drawback of this construction is that it requires more computations than the first one.

Given a source X, let $H_{\min}(X)$ denote its min-entropy, defined by

$$H_{\min}(X) = \min_{x \in \{0,1\}^n} \log_2 \frac{1}{P[X = x]}. \tag{5.5}$$

For many sources, such as independent sources and bit-fixing sources, the number of randomness that can be extracted using deterministic extractors is upper bounded by the min-entropy of the source asymptotically. Note that this is not always true for some special sources when the input sequence is infinitely long. For example, we consider a source on $\{0,1\}^n$ such that there is one assignment with probability $2^{-\frac{n}{2}}$ and all the other assignments have probability either 2^{-n} or 0. For this source, its min-entropy is $\frac{n}{2}$, but as $n \to \infty$, this source itself is arbitrarily close to the uniform distribution on $\{0,1\}^n$.

5.3.1 Independent Sources

Independent sources, where the bits generated are independent of each other, have been studied by Santha and Vazirani [102], Varirani [123], P. Lacharme [72], etc. We consider a general model of independent sources, namely, let $X = x_1 x_2 ... x_n \in \{0,1\}^n$ be a binary sequence generated from such a source, then $x_1, x_2, ..., x_n$ are independent of each other, and all their probabilities are unknown and may be different. We assume that this source contains a certain amount of randomness, i.e., its

min-entropy $H_{\min}(X)$ is known.

Theorem 5.3. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be an independent sequence and let M be an $n \times m$ binary random matrix in which each entry is 1 with probability $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{H_{\min}(X)} < 1$, as $n \to \infty$, $\rho(Y)$ converges to 0 in probability, i.e.,*

$$\rho(Y) \quad \xrightarrow{p} \quad 0.$$

It shows that linear transformations based on sparse random matrices are asymptotically optimal for extracting randomness from independent sources. To consider explicit constructions, we focus on a type of independent sources $X = x_1x_2...x_n \in \{0,1\}^n$ such that the probability of x_i for all $1 \leq i \leq n$ is slightly unpredictable, i.e.,

$$p_i = P[x_i = 1] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}],$$

with a constant e. For such a source, it is possible to have min-entropy $n \log_2 \frac{2}{1+e}$. The following result shows that we can have an explicit construction that can extract as many as $n \log_2 \frac{2}{1+e}$ random bits from X asymptotically.

Theorem 5.4. *Let C be a linear code with dimension m and codeword length n. Assume its weight distribution is binomial and its generator matrix is G. Let $X = x_1x_2...x_n \in \{0,1\}^n$ be an independent source such that $P[x_i = 1] \in [\frac{1}{2} - e/2, \frac{1}{2} + e/2]$ for all $1 \leq i \leq n$, and let $Y = XG^T$. If $\frac{m}{n \log_2 \frac{2}{1+e}} < 1$, as $n \to \infty$, we have*

$$\rho(Y) \to 0.$$

This result shows that if we can construct a linear code with binomial weight distribution, it can extract as many as $n \log_2 \frac{2}{1+e}$ random bits asymptotically. It is known that primitive BCH codes have approximately binomial weight distribution. Hence, they are good candidates for extracting randomness from independent sources with bounded bias.

5.3.2 Hidden Markov Sources

A more-useful but less-studied model is a hidden Markov source. It is a good description of many natural sources for the purpose of high-speed random number generation, such as those based on thermal noise or clock drift. Given a binary sequence $X = x_1x_2...x_n \in \{0,1\}^n$ produced by such a source, we let θ_i be the complete information about the system at time i with $1 \leq i \leq n$. Examples of this system information include the value of the noise signal, the temperature, the environmental effects, the bit generated at time i, etc. So the bit generated at time i, i.e., x_i, is just a function of θ_i. We say that this source has the hidden Markov property if and only if for all $1 < i \leq n$,

$$P[x_i|\theta_{i-1}, x_{i-1}, x_{i-2}, ..., x_1] = P[x_i|\theta_{i-1}].$$

That means the bit generated at time i only depends on the complete system information at time $i-1$.

To analyze the performance of linear transformations on hidden Markov sources, we assume that the external noise of the sources is bounded, hence, we assume that for any three time points $1 \leq i_1 < i_2 < i_3 < n$,

$$P[x_{i_2} = 1|\theta_{i_1}, \theta_{i_3}] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}] \tag{5.6}$$

with a constant e.

Theorem 5.5. *Let $X = x_1x_2...x_n$ be a binary sequence generated from a hidden Markov source described above. Let M be an $n \times m$ binary random matrix in which the probability of each entry being 1 is $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{n \log_2 \frac{2}{1+\sqrt{e}}} < 1$, as n becomes large enough, we have that $\rho(Y)$ converges to 0 in probability, i.e.,*

$$\rho(Y) \quad \xrightarrow{p} \quad 0.$$

The following theorem implies that we can also use the generator matrices of primitive BCH codes for extracting randomness from hidden Markov sources, due to their approximately binomial

weight distributions.

Theorem 5.6. *Let C be a linear binary code with dimension m and codeword length n. Assume its weight distribution is binomial and its generator matrix is G. Let $X = x_1x_2...x_n$ be a binary sequence generated from a hidden Markov source described above, and let $Y = XG^T$. If $\frac{m}{n \log_2 \frac{2}{1+\sqrt{e}}} < 1$, as $n \to \infty$, we have*

$$\rho(Y) \to 0.$$

Although our constructions of linear transformations are not able to extract randomness optimally from hidden Markov sources, they have good capabilities of tolerating local correlations. The gap between their information efficiency and the optimality is reasonable small for hidden Markov sources, especially considering their constructive simplicity and the fact that most of physical sources for high-speed random number generation are roughly independent and with a very small amount of correlations.

5.3.3 Bit-Fixing Sources

Bit-fixing sources were first studied by Cohen and Wigderson [23]. In an oblivious bit-fixing source X of length n, k bits in X are unbiased and independent, and the remaining $n-k$ bits are fixed. We also have nonoblivious bit-fixing sources, in which the remaining $n-k$ bits linearly depend on the k independent and unbiased bits. Such sources were originally studied in the context of collective coin flipping [11]. Here, we say a bit-fixing source for the general nonoblivious case.

Theorem 5.7. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be a bit-fixing source in which k bits are unbiased and independent. Let M be an $n \times m$ binary random matrix in which the probability for each entry being 1 is $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{k} < 1$, as n becomes large enough, we have that $\rho(Y) = 0$ with almost probability 1, i.e.,*

$$P_M[\rho(Y) = 0] \to 1.$$

So sparse random matrices are asymptotically optimal to extract randomness from bit-fixing sources. Unfortunately, for bit-fixing sources, it is possible to find an efficient and explicit construction of linear transformations.

5.3.4 Linear-Subspace Sources

We generalize the sources described above in the following way: Assume $X \in \{0,1\}^n$ is a raw sequence that can be written as ZA, where $Z \in \{0,1\}^k$ with $k < n$ is an independent sequence or a hidden Markov sequence, and A is an $k \times n$ unknown matrix with full rank, i.e., it is an invertible matrix. Instances of such sources include sparse images studied in compressive sensing. We call such sources as linear-subspace sources, namely, they are obtained by mapping simpler sources into a subspace of higher dimensions. We demonstrate that linear transforms based on sparse random matrices can work on linear-subspace sources, and any linear invertible operation on the sources does not affect the asymptotic performance. Specifically, we have the following theorem.

Theorem 5.8. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be a source such that $X = ZA$ in which Z is an independent sequence and A is an unknown $k \times n$ full-rank matrix. Let M be an $n \times m$ random matrix such that each entry of M is 1 with probability $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{H_{\min}(X)} < 1$, as $n \to \infty$, $\rho(Y)$ converges to 0 in probability, i.e.,*

$$\rho(Y) \quad \xrightarrow{p} \quad 0.$$

A similar result holds if Z is a hidden Markov sequence. In this case, we only need to replace $H_{\min}(X)$ with $k \log_2 \frac{1}{1+\sqrt{e}}$, where k is the length of Z and e is defined in equation (5.6).

5.3.5 Comments

Compared to k-sources, the models that we study in this chapter are more specific. Perhaps, they are not perfect to describe some sources like users' operating behaviors or English articles. But for most natural sources that are used for building high-speed random number generators, they are very

good descriptions. Based on these models, we can explore simpler and more practical algorithms than those designed for general k-sources. In the following sections, we will present our technical results in detail for different types of sources respectively.

5.4 Independent Sources

In this section, we study a general independent source $X = x_1x_2...x_n \in \{0,1\}^n$, in which all the bits $x_1, x_2, ..., x_n$ are independent of each other and the probability of x_i with $1 \leq i \leq n$ can be arbitrary value, i.e., $p_i \in [0,1]$. We can consider this source as a biased coin with the existence of external adversaries.

Lemma 5.9. *Given a deterministic extractor $f : \{0,1\}^n \to \{0,1\}^m$, as $n \to \infty$, we have $\rho(f(X)) \to 0$ for an arbitrary independent source X only if*

$$\frac{m}{H_{\min}(X)} \leq 1,$$

where $H_{\min}(X)$ is the min-entropy of X.

Proof. To prove this theorem, we only need to consider a source $X = x_1x_2...x_n \in \{0,1\}^n$ such that

$$P[x_i = 1] = \frac{1}{2}, \forall 1 \leq i \leq H_{\min}(X),$$

and

$$P[x_i = 1] = 0, \forall H_{\min}(X) < i \leq n.$$

From such a source X, if $m > H_{min}(X)$, it is easy to see that $\rho(f(X)) > 0$ for all $n > 0$. □

Let us first consider a simple random matrix in which each entry is 1 or 0 with probability $1/2$ that we call a uniform random matrix. Given an independent input sequence $X \in \{0,1\}^n$ and an $n \times m$ uniform random matrix M, let $Y = MX \in \{0,1\}^m$ be the output sequence. The following lemma provides the upper bound of $E[\rho(Y)]$.

Lemma 5.10. *Let $X = x_1x_2...x_n$ be an independent sequence and M be an $n \times m$ uniform random matrix. Then given $Y = XM$, we have*

$$E_M[\rho(Y)] \leq 2^{m-H_{\min(X)-1}}.$$

Proof. Let p_i denote the probability of x_i and let δ_i be the bias of x_i, then $\delta_i = |p_i - \frac{1}{2}|$.

According to lemma 5.1, when $u \neq 0$, we have

$$|P_X[XMu^T = 1] - \frac{1}{2}| \leq \frac{\prod_{i=1}^{n}(2\delta_i)^{(Mu^T)_i}}{2}, \tag{5.7}$$

where $(Mu^T)_i$ is the ith element of the vector Mu^T.

Substituting (5.7) into lemma 5.2 yields

$$\rho(Y) \quad \leq \quad \frac{1}{2}\sum_{u\neq 0}\prod_{i=1}^{n}(2\delta_i)^{(Mu^T)_i}. \tag{5.8}$$

Now, we calculate the expectation of $\rho(Y)$, which is

$$\begin{aligned} E_M[\rho(Y)] \quad &\leq \quad \frac{1}{2}E_M[\sum_{u\neq 0}\prod_{i=1}^{n}(2\delta_i)^{(Mu^T)_i}] \\ &= \quad \frac{1}{2}\sum_{u\neq 0}\sum_{v\in\{0,1\}^n} P_M[Mu^T = v^T]\prod_{i=1}^{n}(2\delta_i)^{v_i}. \end{aligned} \tag{5.9}$$

Since M is a uniform random matrix (each entry is either 0 or 1 with probability 1/2), if $u \neq 0$, Mu^T is a random vector of length n in which each element is 0 or 1 with probability 1/2. So for any $u \neq 0$,

$$P_M[Mu^T = v^T] = 2^{-n}.$$

As a result,

$$\begin{aligned} E_M[\rho(Y)] &\leq 2^{m-n-1} \sum_{v\in\{0,1\}^n} \prod_{i=1}^{n} (2\delta_i)^{v_i} \\ &= 2^{m-1} \prod_{i=1}^{n} (\frac{1}{2} + \delta_i). \end{aligned}$$

For the independent sequence X, its min-entropy can be written as

$$\begin{aligned} H_{\min}(X) &= \log_2 \frac{1}{\prod_{i=1}^{n} \max(p_i, 1-p_i)} \\ &= \log_2 \frac{1}{\prod_{i=1}^{n} (\frac{1}{2} + \delta_i)}. \end{aligned}$$

So

$$E_M[\rho(Y)] \leq 2^{m-H_{\min}(X)-1}.$$

This completes the proof. □

Example 5.1. *Let us consider an independent source* $X = x_1x_2...x_{512} \in \{0,1\}^{512}$ *in which*

$$p_i \in [\frac{1}{2} - \frac{i}{1024}, \frac{1}{2} + \frac{i}{1024}]$$

for all $1 \leq i \leq 512$.

For this source, its min-entropy is

$$H_{\min}(X) \geq -\sum_{i=1}^{512} \log_2(\frac{1}{2} + \frac{i}{1024}) = 226.16.$$

If we use a 512×180 *random matrix in which each entry is* 0 *or* 1 *with probability* $1/2$*, then according to the above lemma,*

$$E[\rho(Y)] \leq 2^{-47.16} \leq 6.4 \times 10^{-15}.$$

That means that the output sequence is very close to the uniform distribution in the sense of statistical distance.

When n is large enough, we have the following corollary, showing that uniform random matrices are capable to extract as many as $H_{\min}(X)$ random bits from an independent source X asymptotically with almost probability one. Since $H_{\min}(X)$ is the theoretical upper bound, such an extractor is asymptotically optimal on efficiency.

Corollary 5.11. *Let $X \in \{0,1\}^n$ be an independent sequence and let M be an $n \times m$ uniform random matrix. Assume $Y = XM$. If $\frac{m}{H_{\min}(X)} < 1$, as $n \to \infty$, $\rho(Y)$ converges to 0 in probability, i.e.,*

$$\rho(Y) \quad \xrightarrow{p} \quad 0.$$

The above corollary shows that when the length of the input sequence n is large, we can extract random bits very efficiently from an independent source by simply constructing a uniform random matrix. We need to distinguish this method from those of seeded extractors that use some additional random bits whenever extracting randomness. In our method, the matrix is randomly generated but the extraction itself is still deterministic, that means we can use the same matrix to extract randomness for any number of times without reconstructing it. From this point, our method is a 'probabilistic construction of deterministic extractors'.

Although linear transformations based on uniform random matrices are very efficient for extracting randomness from independent sources, they are not computationally fast due to the high density. It is natural to ask whether it is possible to decrease the density of 1s in the matrices without affecting the performance too much. Motivated by this question, we study a sparse random matrix M in which each entry is 1 with probability $p = w(\frac{\log n}{n}) \ll \frac{1}{2}$, where $p = w(\frac{\log n}{n})$ means that $p > \frac{k \log n}{n}$ for any fixed k when $n \to \infty$. Surprisingly, such a sparse matrix has almost the same performance as that of a uniform random matrix, namely, it can extract as many as $H_{\min}(X)$ random bits when the input sequence is long enough.

Lemma 5.12. *Let* $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$ *and let*

$$f_n(p) = \sum_{j=1}^{\frac{\log \frac{1}{\epsilon}}{2p}} \binom{m}{j} (\frac{1}{2}(1 + (1-2p)^j))^n$$

with $\epsilon > 0$ *and* $m = \Theta(n)$. *As* $n \to \infty$, *we have*

$$f_n(p) \to 0.$$

Proof. Since $m = \Theta(n)$, we can write $m = cn$ with a constant c.

Let us introduce a function $F(j)$, defined by

$$\begin{aligned} F(j) &= m^j 2^{-n} (1 + (1-2p)^j)^n \\ &= c^j n^j 2^{-n} (1 + (1-2p)^j)^n. \end{aligned}$$

Then

$$f_n(p) \leq \sum_{j=1}^{\frac{\log \frac{1}{\epsilon}}{2p}} F(j).$$

First, if $p = \frac{1}{2}$, as $n \to \infty$, we have

$$\begin{aligned} f_n(p) &\leq \sum_{j=1}^{\frac{\log \frac{1}{\epsilon}}{2p}} c^j n^j 2^{-n} \\ &\leq \frac{\log \frac{1}{\epsilon}}{2p} 2^{log_2(cn) \frac{\log \frac{1}{\epsilon}}{2p}} 2^{-n} \\ &\leq \frac{\log \frac{1}{\epsilon}}{2p} 2^{\frac{2n \log_2(cn)}{w(\log n)} \log \frac{1}{\epsilon} - n} \\ &= \frac{\log \frac{1}{\epsilon}}{2p} 2^{-\Theta(n)} \\ &\to 0. \end{aligned}$$

If $p < \frac{1}{2}$, we show that $F(j)$ decreases as j increases for $1 \leq j \leq \frac{\log \frac{1}{\epsilon}}{2p}$ when n is large enough.

To see this, we show that its derivative $F'(j) < 0$ when for $n \to \infty$.

$$\begin{aligned} F'(j) &= c^j n^j \log(cn) 2^{-n}(1+(1-2p)^j)^n \\ &\quad + c^j n^j 2^{-n} n (1+(1-2p)^j)^{n-1}(1-2p)^j \log(1-2p) \\ &\leq c^j n^j 2^{-n} n \log(cn) 2^{-n}(1+(1-2p)^j)^n [1 + \frac{(1-2p)^j \log(1-2p) n}{2\log(cn)}]. \end{aligned}$$

So we only need to prove that

$$1 + \frac{(1-2p)^j \log(1-2p) n}{2\log(cn)} < 0$$

for $n \to \infty$.

Since $p \leq \alpha < \frac{1}{2}$ for a constant α, we have

$$(1-2p)^{-\frac{1}{2p}} \leq \beta = (1-2\alpha)^{-\frac{1}{2\alpha}},$$

where β is a constant.

We can also have

$$\log(1-2p) \leq -2p.$$

Hence,

$$\begin{aligned} & 1 + \frac{(1-2p)^j \log(1-2p) n}{2\log(cn)} \\ \leq\ & 1 + \frac{(1-2p)^{\frac{\log\frac{1}{\epsilon}}{2p}} \log(1-2p) n}{2\log(cn)} \\ \leq\ & 1 - \frac{\beta^{\log\epsilon} 2pn}{2\log(cn)} \\ =\ & 1 - \frac{\beta^{\log\epsilon} 2w(\frac{\log n}{n})}{2\log(cn)} \\ <\ & 0. \end{aligned}$$

So when $p < \frac{1}{2}$ and $n \to \infty$, $F(j)$ decreases as j increases for $1 \leq j \leq \frac{\log\frac{1}{\epsilon}}{2p}$. As a result, when n

is large enough, we have

$$\begin{aligned} f_n(p) &\leq \sum_{j=1}^{\frac{\log\frac{1}{\epsilon}}{2p}} F(j) \\ &\leq \frac{\log\frac{1}{\epsilon}}{2p} F(1) \\ &\leq \frac{\log\frac{1}{\epsilon}}{2p} cn(1-p)^n \\ &\leq (cn)^2(1-p)^n. \end{aligned}$$

Since

$$\begin{aligned} \log f_n(p) &\leq 2\log c + 2\log n + n\log(1-p) \\ &\leq 2\log c + 2\log n - \frac{np}{2} \\ &\to -\infty, \end{aligned}$$

we can conclude that

$$f_n(p) \to 0$$

as $n \to \infty$.

This completes the proof. □

Based on the above lemma, we get the following theorem.

Theorem 5.3. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be an independent sequence and let M be an $n \times m$ binary random matrix in which each entry is 1 with probability $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{H_{\min}(X)} < 1$, as $n \to \infty$, $\rho(Y)$ converges to 0 in probability, i.e.,*

$$\rho(Y) \xrightarrow{p} 0.$$

Proof. Let us use the same denotations as above. From equation (5.9) we have

$$E_M[\rho(Y)] \leq \frac{1}{2}\sum_{u\neq 0}\sum_{v\in\{0,1\}^n} P_M[Mu^T = v^T]\prod_{i=1}^{n}(2\delta_i)^{v_i}.$$

Since M is a random matrix in which each entry is 1 with probability p, for a fixed vector $u \neq 0$ with $\|u\| = j$, Mu^T is a random vector where all the entries are independent and each entry is 1 with probability p_j. Here, according to lemma 5.1, we have

$$p_j \in [\frac{1}{2}(1-(1-2p)^j), \frac{1}{2}(1+(1-2p)^j)].$$

There are totally $\binom{m}{j}$ vectors for u with $\|u\| = j$, hence, we get

$$\begin{aligned} E_M[\rho(Y)] &\leq \frac{1}{2}\sum_{j=1}^{m}\binom{m}{j}\sum_{v\in\{0,1\}^n}(\frac{1}{2}(1+(1-2p)^j))^n\prod_{i=1}^{n}(2\delta_i)^{v_i} \\ &= \frac{1}{2}\sum_{j=1}^{m}\binom{m}{j}(1+(1-2p)^j)^n\prod_{i=1}^{n}(\frac{1}{2}+\delta_i). \end{aligned}$$

Now, we divide the upper bound of $E_M[\rho(Y)]$ into two terms. To do this, we let

$$\gamma_1 = \sum_{j=1}^{\frac{\log\frac{1}{\epsilon}}{2p}}\binom{m}{j}(1+(1-2p)^j)^n\prod_{i=1}^{n}(\frac{1}{2}+\delta_i),$$

$$\gamma_2 = \sum_{j=\frac{\log\frac{1}{\epsilon}}{2p}}^{m}\binom{m}{j}(1+(1-2p)^j)^n\prod_{i=1}^{n}(\frac{1}{2}+\delta_i),$$

where ϵ can be arbitrarily small, then

$$E_M[\rho(Y)] \leq \frac{\gamma_1}{2} + \frac{\gamma_2}{2}.$$

According to lemma 5.12, we can get that as $n \to \infty$, if $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$, then $\gamma_1 \to 0$. So we

only need to consider the second term, that is

$$\gamma_2 \leq \sum_{j=\frac{\log\frac{1}{\epsilon}}{2p}}^{m} \binom{m}{j} (1+(1-2p)^{\frac{\log\frac{1}{\epsilon}}{2p}})^n \prod_{i=1}^{n} (\frac{1}{2}+\delta_i).$$

Since $(1-2p)^{-\frac{1}{2p}} \geq e$, we can get

$$(1-2p)^{\frac{\log\frac{1}{\epsilon}}{2p}} \leq \epsilon.$$

As a result,

$$\begin{aligned} \gamma_2 &\leq \sum_{j=\frac{\log\frac{1}{\epsilon}}{2p}}^{m} \binom{m}{j} (1+\epsilon)^n \prod_{i=1}^{n} (\frac{1}{2}+\delta_i) \\ &\leq 2^m (1+\epsilon)^n \prod_{i=1}^{n} (\frac{1}{2}+\delta_i) \\ &\leq 2^{m-n\log_2(1+\epsilon)-H_{\min}(X)}. \end{aligned}$$

Since ϵ can be arbitrary small, if $\frac{m}{H_{\min}(X)} < 1$, as $n \to \infty$, it has

$$\gamma_2 \to 0.$$

We can conclude that if $\frac{m}{H_{\min}(X)} < 1$, $E_M[\rho(Y)]$ can be arbitrarily small as $n \to \infty$. It implies that $\rho(Y) \xrightarrow{p} 0$ as $n \to \infty$.

This completes the proof. □

For practical use, we can set some constraints on each column of the sparse random matrices. For example, we can let the number of ones in each column be a constant k. We may also use pseudo-random bits instead of truly random bits. In coding theory, many good codes are constructed based on randomly generated matrices. Such examples include LDPC (low-density parity-check) codes, network coding and compressive sensing. While these codes have very good performances, efficient decoding algorithms are needed to recover the original messages. Compared to those applications,

randomness extraction is a one-way process that we do not need to reconstruct input sequences (we also cannot do this due to the entropy loss). This feature makes linear transformations based on random matrices very attractive in the applications of randomness extraction.

In the rest of this section, we study deterministic approaches for constructing linear transformations. Here, we focus on a type of independent sources that have been studied in [72, 102, 123], and we call them independent sources with bounded bias. Let $X = x_1x_2...x_n \in \{0,1\}^n$ be an independent sequence generated from such a source, then the probability of x_i for all $1 \leq i \leq n$ is slightly unpredictable, namely,

$$p_i = P[x_i = 1] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}]$$

for a constant e with $0 < e < 1$.

The following theorem shows that if the weight distribution of a linear code is binomial, then the transpose of its generator matrix is a good candidate for extracting randomness from independent sources with bounded bias.

Theorem 5.4. *Let C be a linear code with dimension m and codeword length n. Assume its weight distribution is binomial and its generator matrix is G. Let $X = x_1x_2...x_n \in \{0,1\}^n$ be an independent source such that $P[x_i = 1] \in [\frac{1}{2} - e/2, \frac{1}{2} + e/2]$ for all $1 \leq i \leq n$, and let $Y = XG^T$. If $\frac{m}{n \log_2 \frac{2}{1+e}} < 1$, as $n \to \infty$, we have*

$$\rho(Y) \to 0.$$

Proof. Following equation (5.8) in the proof of theorem 5.10, we get

$$\begin{aligned} \rho(Y) &\leq \frac{1}{2} \sum_{u \neq 0} e^{w((uG)^T)} \\ &= \frac{1}{2} \sum_{i=1}^{n} 2^m \frac{\binom{n}{i}}{2^n} e^i \\ &\leq 2^{m-n-1}(1+e)^n. \end{aligned}$$

Then it is easy to see that if $\frac{m}{n \log_2 \frac{2}{1+e}} < 1$, as $n \to \infty$, we have

$$\rho(Y) \to 0.$$

This completes the proof. □

According to the theorem above, as n becomes large enough, we can extract as many as $n \log_2(\frac{2}{1+e})$ random bits based on the generator matrix of a linear code with binomial weight distribution. Note that the min-entropy of the source is possible to be

$$H_{\min}(X) = n \log_2(\frac{2}{1+e}),$$

which can be achieved when $p_i = \frac{1}{2} + \frac{e}{2}$ for all $1 \leq i \leq n$. Hence, this construction is as efficient as that based on random matrices, both asymptotically optimal.

It turns out that the generator matrices of primitive BCH codes are good candidates. For a primitive BCH code of length $2^k - 1$, it is known that the weight distribution of the code is approximately binomial, see theorem 21 and 23 in [78]. Namely, the number b_i of codewords of weight i is

$$b_i = a \binom{2^k - 1}{i} (1 + E_i),$$

where a is a constant, and the error term E_i tends to zero as k grows.

We see that for the uniform random matrices (with each entry being 0 or 1 with probability 1/2), their weight distributions are binomial in expectation; for sparse random matrices and primitive binary BCH codes, their weight distributions are approximately binomial. Binomial weight distribution is one of important features for 'good' matrices, based on which one can extract randomness efficiently from independent sources.

5.5 Hidden Markov Sources

A generalized model of an independent source are a hidden Markov source. Given a hidden Markov source $X = x_1x_2...x_n \in \{0,1\}^n$, let θ_i be the complete information about the system at time i with $1 \leq i \leq n$. Examples of this system information include the value of the noise signal, the temperature, the environmental effects, the bit generated at time i, etc. So the bit generated at time i, i.e., x_i, is just a function of θ_i. We say that a source has hidden Markov property if and only if for all $1 < i \leq n$,

$$P[x_i|\theta_{i-1}, x_{i-1}, x_{i-2}, ..., x_1] = P[x_i|\theta_{i-1}].$$

That means the bit generated at time i only depends on the complete system information at time $i-1$. Apparently, such sources are good descriptions of many natural sources for the purpose of high-speed random number generation, like those based on thermal noise, avalanche noise, etc.

Example 5.2. *Let us consider a weak random source based on thermal noise. By sampling the noise signal, we get a time sequence of real numbers:*

$$y_1y_2...y_n \in \mathcal{R}^n.$$

For this time sequence it has Markov property, i.e.,

$$P[y_i|y_{i-1}, ..., y_1] = P[y_i|y_{i-1}].$$

By comparing the value at each time with a fixed threshold, we get a binary sequence as the source

$$X = x_1x_2...x_n \in \{0,1\}^n,$$

such that $x_i = sign(y_i - a)$ with a constant a for all $1 \leq i \leq n$.

To analyze the performance of linear transformations on hidden Markov sources, we assume that the external noise of the sources is bounded, hence, we assume that for any three time points

$1 \leq i_1 < i_2 < i_3 < n$,

$$P[x_{i_2} = 1 | \theta_{i_1}, \theta_{i_3}] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}]$$

for a constant e.

Lemma 5.13. *Let $X = x_1 x_2 ... x_n$ be a binary sequence generated from a hidden Markov source described above. Let $z = x_{i_1} + ... + x_{i_t}$ mod 2 for $1 \leq i_1 < i_2 < ... < i_t \leq n$ with some t, then we have*

$$|P[z = 1] - \frac{1}{2}| \leq \frac{e^{(t-1)/2}}{2}. \tag{5.10}$$

Proof.

$$\begin{aligned} |P[z = 1] - \frac{1}{2}| &= |\sum_{\theta_{i_1}, \theta_{i_3}, ...} P[\theta_{i_1}, \theta_{i_3}, ...] P[z = 1 | \theta_{i_1}, \theta_{i_3}, ...] - \frac{1}{2}| \\ &\leq \sum_{\theta_{i_1}, \theta_{i_3}, ...} P[\theta_{i_1}, \theta_{i_3}, ...] |P[z = 1 | \theta_{i_1}, \theta_{i_3}, ...] - \frac{1}{2}| \\ &\leq \max_{\theta_{i_1}, \theta_{i_3}, ...} |P[x_{i_2} + x_{i_4} + ... | \theta_{i_1}, \theta_{i_3}, ...] - \frac{1}{2}|. \end{aligned}$$

Given $\theta_{i_1}, \theta_{i_3}, ...$, we have $x_{i_2}, x_{i_4}, ...$ independent of each other. So the conclusion is immediate following the statement of lemma 5.1. □

For some hidden Markov sources, the constraint e is not so strict. It is possible that there exists a group of $\theta_{i_1}, \theta_{i_3}, ...$ such that

$$|P[z = 1 | \theta_{i_1}, \theta_{i_3}, ...] - \frac{1}{2}| > \frac{e^{(t-1)/2}}{2}.$$

In this case, we may find a typical set S such that

$$P[(\theta_{i_1}, \theta_{i_3}, ...) \in S] \to 1,$$

as the sequence becomes long enough, and in this typical set,

$$|P[z=1|(\theta_{i_1},\theta_{i_3},...)\in S]-\frac{1}{2}|\leq\frac{e^{(t-1)/2}}{2}.$$

In this case, we can write

$$|P[z=1]-\frac{1}{2}|\leq P[(\theta_{i_1},\theta_{i_3},...)\notin S]+\max_{(\theta_{i_1},\theta_{i_3},...)\in S}|P[z|\theta_{i_1},\theta_{i_3},...]-\frac{1}{2}|,$$

where the first term on the righthand side is ignorable.

Note that equation (5.10) can be rewritten as

$$|P[z=1]-\frac{1}{2}|\leq\frac{(\sqrt{e})^t}{2\sqrt{e}},$$

which is very similar to the result in lemma 5.1. If we ignore the constant term $\sqrt{e}$, the only difference between them is replacing e by $\sqrt{e}$. Based on this observation as well as the results in section 5.4 for independent sources, we can obtain the following results for hidden Markov sources.

Lemma 5.14. *Let $X=x_1x_2...x_n$ be a binary sequence generated from a hidden Markov source described above. Let M be an $n\times m$ random matrix such that each entry of M is 0 or 1 with probability $\frac{1}{2}$. Then given $Y=XM$, we have*

$$E_M[\rho(Y)]\leq\frac{2^{m-n-1}}{\sqrt{e}}(1+\sqrt{e})^n.$$

So with a uniform random matrix, one can extract as many as $n\log_2\frac{2}{1+\sqrt{e}}$ random bits from a hidden Markov source. And this conclusion is also true for sparse random matrices, given by the following theorem.

Theorem 5.5. *Let $X=x_1x_2...x_n$ be a binary sequence generated from a hidden Markov source described above. Let M be an $n\times m$ binary random matrix in which the probability of each entry being 1 is $p=w(\frac{\log n}{n})\leq\frac{1}{2}$. Assume $Y=XM$. If $\frac{m}{n\log_2\frac{2}{1+\sqrt{e}}}<1$, as n becomes large enough, we*

have that $\rho(Y)$ converges to 0 in probability, i.e.,

$$\rho(Y) \quad \xrightarrow{p} \quad 0.$$

Proof. The proof follows the same idea for the proof of theorem 5.3. □

Theorem 5.6. *Let C be a linear binary code with dimension m and codeword length n. Assume its weight distribution is binomial and its generator matrix is G. Let $X = x_1x_2...x_n$ be a binary sequence generated from a hidden Markov source described above, and let $Y = XG^T$. If $\frac{m}{n \log_2 \frac{2}{1+\sqrt{e}}} < 1$, as $n \to \infty$, we have*

$$\rho(Y) \to 0.$$

Proof. The proof follows the same idea for the proof of theorem 5.4. □

These theorems show that when n is large enough, we can extract as many as $n \log_2 \frac{2}{1+\sqrt{e}}$ random bits from the a hidden Markov source using linear transformations.

Let us consider an order-1 Markov source as a special instance. Assume that $X = x_1x_2...x_n$ is a binary sequence generated from this source such that each bit $x_i \in \{0, 1\}$ only depends on its previous one bit, namely,

$$P[x_i = 1|x_{i-1}] \in [\frac{1}{2} - \varepsilon/2, \frac{1}{2} + \varepsilon/2]$$

for a constant ε. Note that the transition probabilities are slightly unpredictable.

We first show that such a source can be treated as a (hidden) Markov source such that for any $1 \leq i_{j-1} \leq i_j \leq i_{j+1} \leq n$,

$$|P[x_{i_j}|x_{i_{j-1}}, x_{i_{j+1}}] - \frac{1}{2}| \leq \frac{e}{2}$$

for a constant e.

According to the definition, we have

$$\begin{aligned} & |P[x_{i_j}|x_{i_{j-1}}] - \frac{1}{2}| \\ = & |\sum_{x_{i_{j-1}+1},\ldots,x_{i_j-1}} P[x_{i_j}|x_{i_j-1}]...P[x_{i_{j-1}+1}|x_{i_{j-1}}] - \frac{1}{2}| \\ \leq & \sum_{x_{i_{j-1}+1},\ldots,x_{i_j-1}} P[x_{i_j-1}|x_{i_j-2}]...P[x_{i_{j-1}+1}|x_{i_{j-1}}]|P[x_{i_j}|x_{i_j-1}] - \frac{1}{2}| \\ \leq & \frac{\varepsilon}{2}. \end{aligned}$$

As a result,

$$\begin{aligned} & |P[x_{i_j}|x_{i_{j-1}}, x_{i_{j+1}}] - \frac{1}{2}| \\ \leq & |\frac{P[x_{i_{j-1}}]P[x_{i_j}|x_{i_{j-1}}]P[x_{i_{j+1}}|x_{i_j}]}{\sum_{x_{i_j}} P[x_{i_{j-1}}]P[x_{i_j}|x_{i_{j-1}}]P[x_{i_{j+1}}|x_{i_j}]} - \frac{1}{2}| \\ \leq & |\frac{(\frac{1}{2}+\frac{\varepsilon}{2})^2}{(\frac{1}{2}+\frac{\varepsilon}{2})^2 + (\frac{1}{2}-\frac{\varepsilon}{2})^2} - \frac{1}{2}| \\ = & \frac{\varepsilon}{1+\varepsilon^2}. \end{aligned}$$

Then, by setting $e = \frac{2\varepsilon}{1+\varepsilon^2}$, we can get

$$|P[x_{i_j}|x_{i_{j-1}}, x_{i_{j+1}}] - \frac{1}{2}| \leq \frac{e}{2}$$

for all $1 \leq i_{j-1} \leq i_j \leq i_{j+1} \leq n$.

According to the above theorems, with linear transformations, we can extract as many as $n \log_2(\frac{2}{1+\sqrt{\frac{2\varepsilon}{1+\varepsilon^2}}})$ random bits from the above source asymptotically. In this case,

$$n \log_2(\frac{2}{1+\sqrt{\frac{2\varepsilon}{1+\varepsilon^2}}}) \leq \min_X H_{\min}(X) = n \log_2(\frac{2}{1+\varepsilon}).$$

That means the linear transformations are not optimal for extracting randomness from order-1 Markov sources. It is true for most hidden Markov sources. But we need to see that linear transformations have good capabilities of tolerating local correlations. The gap between their information

efficiency and the optimality is reasonable small for hidden Markov sources, especially considering their constructive simplicity. In high-speed random number generation, the physical sources usually have relatively good quality, namely, the bits are roughly independent (with a very small amount of correlations). In this case, Linear transformation are very efficient in extracting randomness.

5.6 Bit-Fixing Sources

In this section, we consider another type of weak random sources, called bit-fixing sources, first studied by Cohen and Wigderson [23]. In an oblivious bit-fixing source X of length n, k bits in X are unbiased and independent, and the remaining $n-k$ bits are fixed. The positions of the k bits are unknown. In fact, oblivious bit-fixing sources is a special type of independent sources that we studied in the previous sections, where all the bits in the source are independent of each other, among them, k bits have probability $1/2$ and the other $n-k$ bits have probability either 0 or 1. So our conclusions about the application of sparse random matrices on independent sources still can work here.

Another type of bit-fixing sources are nonoblivious. Unlike the oblivious case, in nonoblivious bit-fixing sources, the remaining $n-k$ bits are linearly determined by the k independent and unbiased bits. Such sources were originally studied in the context of collective coin flipping [11].

Generally, we can describe a (nonoblivious) bit-fixing source in the following way: Let $Z \in \{0,1\}^k$ be an independent and unbiased sequence, the source $X \in \{0,1\}^n$ can be written as $X = ZA$, where A is an unknown $k \times n$ binary matrix such that there are k columns in A that form an identity matrix.

Example 5.3. *One example of such a matrix A is*

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

If we consider the columns $2, 4, 3$*, then they form an identity matrix.*

Given a bit-fixing source with k independent and unbiased bits, one cannot extract more than k random bits that are arbitrarily close to truly random bits. That's because the entropy of the output sequence must be upper bounded by the entropy of the input sequence, which is k.

Lemma 5.15. *Let* $X = x_1x_2...x_n \in \{0,1\}^n$ *be a bit-fixing source in which* k *bits are unbiased and independent. Let* M *be an* $n \times m$ *uniform random matrix such that each entry of* M *is* 0 *or* 1 *with probability* $\frac{1}{2}$*. Given* $Y = XM$*, then we have*

$$P_M[\rho(Y) \neq 0] \leq 2^{m-k}.$$

Proof. For a bit-fixing source $X \in \{0,1\}^n$, we can write it as $X = ZA$, where $Z \in \{0,1\}^k$ is an independent and unbiased sequence. Hence,

$$Y = XM = ZAM = ZB,$$

in which $B = AM$ is an $k \times m$ matrix.

We see that all the columns of B are independent of each other because the ith column of B only depends on the ith column of M for all $1 \leq i \leq m$. Furthermore, it can be proved that each column of B is a vector in which all the elements are independent of each other and each element is 0 or 1 with probability 1/2. To see this, we consider an entry in B, which is $B_{ij} = \sum_k A_{ik}M_{kj}$. Given this i, according to the definition of A, we can always find a column r such that only the element in the ith row is 1 and all the other elements in this column are 0s. So we can write

$$B_{ij} = M_{ir} + \sum_{k \neq r} A_{ik}M_{kj},$$

$$B_{i'j} = \sum_{k \neq r} A_{i'k}M_{kj}, \text{ for } i' \neq i,$$

where M_{ir} is an unbiased random bit independent of M_{kj} with $k \neq r$. In this case, B_{ij} is independent

of $B_{i'j}$ with $i' \neq i$. Hence, we can conclude that B is a random matrix in which each entry is 0 or 1 with probability 1/2.

According to lemma 5.2, we get that $\rho(Y) = 0$ if and only if ZBu^T is an unbiased random bit for all $u \neq 0$.

Hence,

$$\begin{aligned} P_M[\rho(Y) \neq 0] &\leq \sum_{u \neq 0} P_M[ZBu^T \text{ is fixed }] \\ &= \sum_{u \neq 0} P_B[Bu^T = 0], \end{aligned} \tag{5.11}$$

where Bu^T is a random vector with each element being 0 or 1 with probability 1/2 for all $u \neq 0$. So

$$P_B[Bu^T = 0] = 2^{-k}.$$

Finally, we can get that

$$P_M[\rho(Y) \neq 0] \leq \sum_{u \neq 0} 2^{-k} \leq 2^{m-k}.$$

This completes the proof. □

According to the above lemma, by using a uniform random matrix with $m - k \leq 0$, we can generate an independent and unbiased sequence from a bit-fixing source with almost probability 1. In the following theorem, we show that sparse random matrices can also work for bit-fixing sources.

Theorem 5.7. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be a bit-fixing source in which k bits are unbiased and independent. Let M be an $n \times m$ binary random matrix in which the probability for each entry being 1 is $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{k} < 1$, as n becomes large enough, we have that $\rho(Y) = 0$ with almost probability 1, i.e.,*

$$P_M[\rho(Y) = 0] \to 1.$$

Proof. According to equation (5.11), we have

$$P_M[\rho(Y) \neq 0] \quad = \quad \sum_{u \neq 0} P_M[AMu^T = 0].$$

When $u \neq 0$, Mu^T is a random vector in which all the elements are independent of each other. Let $|u| = j$, then according to lemma 5.1, the probability for each element in Mu^T being 1 is

$$p_j \in [\frac{1}{2}(1 - (1-2p)^j), \frac{1}{2}(1 + (1-2p)^j)].$$

Let $v^T = AMu^T$ and use v_i^T denote its ith element, then

$$P_M[v^T = 0] = \prod_{i=1}^{k} P[v_i^T = 0 | v_1^T = 0, ..., v_{i-1}^T = 0].$$

According to the constraint on A, we know that there exists a column that is $[0, ..., 0, 1, 0, ..., 0]^T$, in which only the entry in the ith row is 1. Without loss of generality, we assume that this column is the rth column. Then we can write

$$v_i^T = (Mu^T)_r + \sum_{t \neq r, t=1}^{n} a_{it}(Mu^T)_t,$$

where $(Mu^T)_r$ is 1 with probability $p_j \in [\frac{1}{2}(1 - (1-2p)^j), \frac{1}{2}(1 + (1-2p)^j)]$, and it is independent of $v_1^T, v_2^T, ...$ Hence,

$$\begin{aligned}
& P_M[v_i^T = 0 | v_1^T = 0, ..., v_{i-1}^T = 0] \\
= \quad & \sum_{a=0}^{1} P_M[(Mu^T)_r = a] P_M[\sum_{t \neq r, t=1}^{n} a_{it}(Mu^T)_t = a | v_1^T = 0, ..., v_{i-1}^T = 0] \\
\leq \quad & \max_{a=0}^{1} P_M[(Mu^T)_r = a] \\
= \quad & \frac{1}{2}(1 + (1-2p)^j).
\end{aligned}$$

So when $|u| = j$, we can get

$$P_M[AMu^T = 0] \leq (\frac{1}{2}(1 + (1-2p)^j)^k.$$

As a result,

$$P_M[\rho(Y) \neq 0] \leq \sum_{j=1}^{m} \binom{m}{j} (\frac{1}{2}(1 + (1-2p)^j))^k.$$

Let us divide it into two parts,

$$\gamma_1 = \sum_{j=1}^{\frac{\log \frac{1}{\epsilon}}{2p}} \binom{m}{j} (\frac{1}{2}(1 + (1-2p)^j))^k,$$

$$\gamma_2 = \sum_{j=\frac{\log \frac{1}{\epsilon}}{2p}}^{m} \binom{m}{j} (\frac{1}{2}(1 + (1-2p)^j))^k,$$

where ϵ is arbitrary small. Then

$$P_M[\rho(Y) \neq 0] \leq \gamma_1 + \gamma_2.$$

According to lemma 5.12, we can get that the first part $\gamma_1 \to 0$ as $n \to 0$.

For the second part γ_2, it is easy to show that for any $\epsilon > 0$, when n (or k) is large enough

$$\begin{aligned} \gamma_2 &= \sum_{j=\frac{\log \frac{1}{\epsilon}}{2p}}^{m} \binom{m}{j} (\frac{1}{2}(1 + (1-2p)^j))^k \\ &\leq \sum_{j=\frac{\log \frac{1}{\epsilon}}{p}}^{m} \binom{m}{j} (\frac{1}{2}(1 + \epsilon))^k \\ &\leq 2^{m-k}(1+\epsilon)^k. \end{aligned}$$

As a result, if $m - k \log \frac{2}{1+\epsilon} \ll 0$ for an arbitrary ϵ, then $P_M[\rho(Y) \neq 0]$ can be very small. Therefore, we get the conclusion in the theorem.

This completes the proof. □

We see that sparse random matrices are asymptotically optimal for extracting randomness from

bit-fixing sources. Now a question is whether we can find an explicit construction of linear transformations for extracting randomness efficiently from any bit-fixing source specified by n and k. Unfortunately, the answer is negative. The reason is that in order to extract independent random bits, it requires XMu^T to be an unbiased random bit for all $u \neq 0$ (See the proof above). So $\|Mu^T\| > n-k$ for all $u \neq 0$, otherwise we are able to find a bit-fixing source X such that XMu^T is fixed. Such a bit-fixing source can be constructed as follows: Assume $X = x_1x_2...x_n$, if $(Mu^T)_i = 1$ we set x_i as an unbiased random bit, otherwise we set $x_i = 0$ being fixed. It further implies that if we have a linear code with generator matrix M^T, then its minimum distance should be more than $n-k$. But for such a matrix, its efficiency ($\frac{m}{n}$) is usually very low. For example, when $k = \frac{n}{2}$, we have to find a linear code with minimum distance more than $\frac{n}{2}$. In this case, the dimension of the code, i.e., m, is much smaller than k, implying a low efficiency in randomness extraction.

5.7 Linear-Subspace Sources

In this previous section, we studied a bit-fixing source $X \in \{0,1\}^n$, which can be written as ZA, where $Z \in \{0,1\}^k$ is an independent and unbiased sequence and A is an unknown $k \times n$ matrix that embeds an identity matrix. Actually, we can generalize the model of bit-fixing sources in two directions. First, the matrix A can be generalized to any full-rank matrix. Second, the sequence Z is not necessary being independent and unbiased. Instead, it can be any random source described in this chapter, like an independent source or a hidden Markov source. The new generalized source X can be treated as a mapping of another source Z into a linear subspace of higher dimensions, so we call it a linear-subspace source. The rows of the matrix A, which are independent of each other, form the basis of the linear subspace. Linear-subspace sources are good descriptions of many natural sources, like sparse images studied in compressive sensing.

First, let us consider the case that the matrix A is an arbitrary unknown full rank matrix and Z is still an independent and unbiased sequence.

Lemma 5.16. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be a source such that $X = ZA$ in which Z is an*

independent and unbiased sequence, and A is an unknown $k \times n$ full-rank matrix. Let M be an $n \times m$ random matrix such that each entry of M is 1 with probability $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{k} < 1$, as n becomes large enough, we have $\rho(Y) = 0$ with almost probability 1, i.e.,

$$P_M[\rho(Y) = 0] \to 1.$$

Proof. In the proof of theorem 5.7, we have

$$P_M[\rho(Y) \neq 0] = \sum_{u \neq 0} P_M[AMu^T = 0].$$

If the matrix A has full rank, than we can write

$$A = UR,$$

where $det(U) \neq 0$ and R is in row echelon form. We see that RZ is a nonoblivious bit-fixing source.

Since $det(U) \neq 0$, $AMu^T = 0$ is equivalent to $RMu^T = 0$. Therefore,

$$P_M[\rho(Y) \neq 0] = \sum_{u \neq 0} P_M[RMu^T = 0].$$

Based on the proof of theorem 5.7, we can get the conclusion in the lemma.

This completes the proof. □

Furthermore, we generalize the sequence Z to a general independent source in which the probability of each bit is unknown and the min-entropy of the source is $H_{\min}(Z)$.

Theorem 5.8. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be a source such that $X = ZA$ in which Z is an independent sequence and A is an unknown $k \times n$ full-rank matrix. Let M be an $n \times m$ random matrix such that each entry of M is 1 with probability $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If*

$\frac{m}{H_{\min}(X)} < 1$, *as* $n \to \infty$, $\rho(Y)$ *converges to* 0 *in probability, i.e.,*

$$\rho(Y) \quad \xrightarrow{p} \quad 0.$$

Proof. Let δ_i be the bias of z_i in Z for all $1 \leq i \leq k$. According to equation (5.9), we can get

$$E_M[\rho(Y)] \leq \frac{1}{2} \sum_{u \neq 0} \sum_{v \in \{0,1\}^k} P_M[AMu^T = v^T] \prod_{i=1}^{k} (2\delta_i)^{v_i}.$$

When $\|u\| = j$, Mu^T is an independent sequence in which each bit is one with probability

$$p_j \in [\frac{1}{2}(1 - (1-2p)^j), \frac{1}{2}(1 + (1-2p)^j)].$$

In theorem 5.16, we have proved that

$$P_M[AMu^T = 0] \leq (\frac{1}{2}(1 + (1-2p)^j)^k.$$

Using a same idea, if $A = UR$ with $det(U) \neq 0$ and R in row echelon form, we can write

$$\begin{aligned}
& P_M[AMu^T = v^T] \\
= \quad & P_M[RMu^T = U^{-1}v^T] \\
= \quad & \prod_{i=1}^{k} P_M[(RMu^T)_i = (U^{-1}v^T)_i | (RMu^T)_{i-1} = (U^{-1}v^T)_{i-1}, ...] \\
\leq \quad & (\frac{1}{2}(1 + (1-2p)^j)^k
\end{aligned}$$

for all $v^T \in \{0,1\}^k$.

Hence

$$E_M[\rho(Y)] \leq \frac{1}{2} \sum_{j=1}^{m} \binom{m}{j} (\frac{1}{2}(1 + (1-2p)^j)^k \prod_{i=1}^{k} (1 + 2\delta_i).$$

In the next step, following the proof of theorem 5.3, we can get that if $\frac{m}{H_{\min}(Z)} < 1$, as $n \to \infty$,

$$E_M[\rho(Y)] \to 0.$$

It is equivalent to $\rho(Y) \xrightarrow{p} 0$.

Since $H_{\min}(Z) = H_{\min}(X)$, we can get the conclusion in the theorem.

This completes the proof. □

A similar result holds if Z is a hidden Markov sequence. In this case, we have the following theorem.

Theorem 5.17. *Let $X = x_1x_2...x_n \in \{0,1\}^n$ be a source such that $X = ZA$ in which $Z \in \{0,1\}^k$ is a hidden Markov sequence described in section 5.5, and A is an unknown $k \times n$ full-rank matrix. Let M be an $n \times m$ random matrix such that each entry of M is 1 with probability $p = w(\frac{\log n}{n}) \leq \frac{1}{2}$. Assume $Y = XM$. If $\frac{m}{k \log_2 \frac{2}{1+\sqrt{e}}} < 1$, as $n \to \infty$, $\rho(Y)$ converges to 0 in probability, i.e.,*

$$\rho(Y) \xrightarrow{p} 0.$$

From the above theorems, we see that by multiplying an invertible matrix to a given source does not affect the extracting capability of sparse random matrices.

5.8 Implementation for High-Speed Applications

In this section, we discuss the implementation of linear transformations in high-speed random number generators, where the physical sources usually provide a stream rather than a sequence of finite length. To generate random bits, we can apply a linear transformation to the incoming stream based on block by block, namely, we divide the incoming stream into blocks and generate random bits from each block separately. Such an operation can be finished by software or hardware like FPGAs [34, 148].

Another way is that we process each bit when it arrives. In this case, let $M = \{m_{ij}\}$ be an $n \times m$

Table 5.1. Asymptotical efficiencies of linear transformations for extracting randomness from different sources

Source $X = x_1x_2...x_n$	Sparse Random Matrices	Generator Matrices
Independent Sources	$H_{\min}(X)$	$n \log_2 \frac{2}{1+e}$ if $P[x_i = 1] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}]$
Hidden Markov Sources	$n \log_2 \frac{2}{1+\sqrt{e}}$ if $P[x_{i_2} = 1 \vert \theta_{i_1}, \theta_{i_3}] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}]$	$n \log_2 \frac{2}{1+\sqrt{e}}$ if $P[x_{i_2} = 1 \vert \theta_{i_1}, \theta_{i_3}] \in [\frac{1}{2} - \frac{e}{2}, \frac{1}{2} + \frac{e}{2}]$
Bit-Fixing Sources	$H_{\min}(X)$	NA
Linear-Subspace Sources	$H_{\min}(X)$ if $X = AZ$ with A full-rank and Z independent	NA

matrix (such as a sparse random matrix) for processing the incoming stream and let $V \in \{0,1\}^m$ denote a vector that stores m bits. The vector V is updated dynamically in response of the incoming bits. When the ith bit of the stream, denoted by x_i, arrives we do the following operation on V,

$$V \to V + x_i M_{1+(i \bmod n)},$$

where M_j is the jth row in the matrix M. Specifically, we can write the vector V at time i as $V[i]$ and denote its jth element as $V_j[i]$. To generate (almost) random bits, we output the bits in V sequentially and cyclically with a lower rate than that of the incoming stream. Namely, we generate an output stream $Y = y_1y_2...$ such that

$$y_i = V_{1+(i \bmod m)}[n + \lfloor \frac{ni}{m} \rfloor].$$

So the rate of the output stream is $\frac{m}{n}$ of the incoming stream. In this method, the expected computational time for processing a single incoming bit is proportional to the number of ones in M over n. According to our results of sparse random matrices, it can be as low as $(\log n)^\alpha$ with any $\alpha > 1$ asymptotically. So this method is computationally very efficient, and the working load is well balanced.

5.9 Conclusion

In this chapter, we demonstrated the power of linear transformations in randomness extraction from a few types of weak random sources, including independent sources, hidden Markov sources, bit-fixing sources, and linear-subspace sources, as summarized in table 5.1. Compared to the existing methods, the constructions of linear transformations are much simpler, and they can be easily implemented using FPGAs; these properties make methods based on linear transformations very practical. To reduce the hardware/computational complexity, we prefer sparse matrices rather than high-density matrices, and we proved that sparse random matrices can work as well as uniform random matrices. Explicit constructions of efficient sparse matrices remain a topic for future research.

Chapter 6

Extracting Randomness from Imperfect Stochastic Processes

This chapter studies the problem of extracting a prescribed number of random bits by reading the smallest possible number of symbols from imperfect stochastic processes. A new class of extractors called variable-length extractors is introduced, they achieve efficiency near Shannon's (optimal) limit.[1]

6.1 Introduction

We study the problem of extracting a prescribed number of random bits by reading the smallest possible number of symbols from imperfect stochastic processes. For perfect stochastic processes, including processes with known accurate distributions or perfect biased coins, this problem has been well studied. It dates back to von Neumann [9] who considered the problem of generating random bits from a biased coin with unknown probability. Recently, in [142], we improved von Neumann's scheme and introduced an algorithm that generates 'random bit streams' from biased coins, uses bounded space and runs in expected linear time. This algorithm can generate a prescribed number of random bits with an asymptotically optimal efficiency. On the other hand, efficient algorithms have also been developed for extracting randomness from any known stochastic process (whose

[1] Some of the results presented in this chapter have been previously published in [143]; Thanks are due to Professor Chris Umans for helpful discussions.

distribution is given). In [71], Knuth and Yao presented a simple procedure for generating sequences with arbitrary probability distributions from an unbiased coin (the probability of H and T is $\frac{1}{2}$). In [3], Abrahams considered a source of biased coin whose distribution is an integer power of a noninteger. Han and Hoshi [52] studied the general problem and proposed an interval algorithm that generates a prescribed number of random bits from any known stochastic process and achieves the information-theoretic upper bound on efficiency. However, in practice, sources of stochastic processes have inherent correlations and are affected by measurement's noise, hence, they are not perfect. Existing algorithms for extracting randomness from perfect stochastic processes cannot work for imperfect stochastic processes, where uncertainty exists.

To extract randomness from an imperfect stochastic process, one approach is to apply a seeded or seedless extractor to a sequence generated by the process that contains a sufficient amount of randomness, and we call this approach as a fixed-length extractor for stochastic processes since all the possible input sequences have the same fixed length. Efficient constructions of seeded or seedless extractors have been extensively studied in last two decades, and it shows that the number of random bits extracted by them can approach the source's min-entropy asymptotically [32, 63, 87, 95, 105]. Although fixed-length extractors can generate random bits with good quality from imperfect stochastic processes, their efficiencies are not close to the optimality. Here, we define the *efficiency* of an extractor for stochastic processes as the asymptotic ratio between the number of extracted random bits and the entropy of its input sequence (the entropy of its input sequence is proportional to the expected input length if the stochastic process is stationary ergodic), which is upper bounded by 1 since the process of extracting randomness does not increase entropy. Based on this definition, we can conclude that the efficiency of a fixed-length extractor is upper bounded by the ratio between the min-entropy and the entropy of the input sequence, which is usually several times smaller than 1. So fixed-length extractors are not very efficient in extracting randomness from stochastic processes. The intuition is that, in order to minimize the expected number of symbols read from an imperfect stochastic process, the length of the input sequence should be adaptive, not being fixed.

The concept of min-entropy and entropy are defined as follows.

Definition 6.1. *Given a random source X on $\{0,1\}^n$, the* min-entropy *of X is defined as*

$$H_{\min}(X) = \min_{x \in \{0,1\}^n} \log \frac{1}{P[X = x]}.$$

The entropy *of X is defined as*

$$H(X) = \sum_{x \in \{0,1\}^n} P[X = x] \log \frac{1}{P[X = x]}.$$

The following example is constructed for comparing entropy with min-entropy for a simple random variable.

Example 6.1. *Let X be a random variable such that $P[X = 0] = 0.9$ and $P[X = 1] = 0.1$, then $H_{\min}(X) = 0.152$ and $H(X) = 0.469$. In this case, the entropy of X is about three times its min-entropy.* □

In this chapter, we focus on the notion and constructions of variable-length extractors (short for variable-to-fixed length extractors), namely, extractors with variable input length and fixed output length. (Note that the interval algorithm proposed by Han and Hoshi [52] and the streaming algorithm proposed by us [142] are special cases of variable-length extractors). Our goal is to extract a prescribed number of random bits in the sense of statistical distance while minimizing the expected input cost, measured by the entropy of the input sequence (whose length is variable). To make this precise, we let $d(\mathcal{R}, \mathcal{M})$ be the difference between two known stochastic processes $\mathcal{R}$ and $\mathcal{M}$, defined by

$$d(\mathcal{R}, \mathcal{M}) = \limsup_{n \to \infty} \max_{x \in \{0,1\}^n} \frac{\log_2 \frac{P_{\mathcal{R}}(x)}{P_{\mathcal{M}}(x)}}{\log_2 \frac{1}{P_{\mathcal{M}}(x)}},$$

where $P_{\mathcal{R}}(x)$ is the probability of generating x from $\mathcal{R}$ when the sequence length is $|x|$, and $P_{\mathcal{M}}(x)$ is the probability of generating x from $\mathcal{M}$ when the sequence length is $|x|$.

A few models of imperfect stochastic processes are introduced and investigated, including,

- Let $\mathcal{M}$ be a known stochastic process, we consider an arbitrary stochastic process $\mathcal{R}$ such that $d(\mathcal{R}, \mathcal{M}) \leq \beta$ for a constant β.

- We consider $\mathcal{R}$ as an arbitrary stochastic process such that $\min_{\mathcal{M} \in \mathcal{G}_{s.e.}} d(\mathcal{R}, \mathcal{M}) \leq \beta$ for a constant β, where $\mathcal{G}_{s.e.}$ denotes the set consisting of all stationary ergodic processes.

Generally, given a real slight-unpredictable source $\mathcal{R}$, it is not easy to estimate the exact value of $d(\mathcal{R}, \mathcal{M})$ for a stochastic process M. But its upper bound, i.e., β, can be easily obtained. The parameter β describes how unpredictable the real source $\mathcal{R}$ is, so we call it the *uncertainty* of $\mathcal{R}$. We prove that it is impossible to construct an extractor that achieves efficiency strictly larger than $1-\beta$ for all the possible sources $\mathcal{R}$ with uncertainty β. Then we introduce several constructions of variable-length extractors, and show that their efficiencies can reach $\eta \geq 1-\beta$; that is, the constructions are asymptotically optimal. The proposed variable-length extractors have two benefits: (i) they are generalizations of algorithms for perfect sources to address general imperfect sources; and (ii) they bridge the gap between min-entropy and entropy on efficiency.

The following example is constructed to compare the performances of a variable-length extractor and a fixed-length extractor when extracting randomness from a slightly-unpredictable independent process.

Example 6.2. *Consider an independent process $x_1x_2x_3...$ such that $P[x_i = 1] \in [0.9, 0.91]$, then it can be obtained that $\beta \leq 0.0315$. For this source, a variable-length extractor can generate random bits with efficiency at least $1-\beta = 0.9685$ that is very close to the upper bound 1. In comparison, fixed-length extractors can only reach the efficiency at most 0.3117.*

The remainder of this chapter is organized as follows. Section 6.2 presents background and related results. In section 6.3, we demonstrate that one cannot construct a variable-length extractor with efficiency strictly larger than $1-\beta$ when the source has uncertainty β. Then we focus on the seeded constructions of variable-length extractors, namely, we use a small number of additional truly random bits as the seed (catalyst). Three different constructions are provided and analyzed in section 6.4, section 6.5 and section 6.6 separately. All these constructions have efficiencies lower bounded

by $1-\beta$, implying their optimality. Finally, we discuss seedless constructions of variable-length extractors for some types of random sources in section 6.7, followed by the concluding remarks.

6.2 Preliminaries

6.2.1 Statistical Distance

Statistical Distance is used in computer science to measure the difference between two distributions. Let X and Y be two random sequences with range $\{0,1\}^m$, then the statistical distance between X and Y is defined as

$$\|X-Y\| = \max_{T:\{0,1\}^m \to \{0,1\}} |P[T(X)=1] - P[T(Y)=1]|$$

over a boolean function T. We say that X and Y are ϵ-close if $\|X-Y\| \leq \epsilon$. According to this definition, we can also write

$$\|X-Y\| = \frac{1}{2} \sum_{x \in \{0,1\}^m} |P[X=x] - P[Y=x]| \leq \epsilon.$$

It is equivalent to the former expression.

Let U_m denote the uniform distribution on $\{0,1\}^m$. In order to let a sequence Y to be able to take place of the truly random bits in a randomized application, we let Y be ϵ-close to U_m, where ϵ is small enough. In this case, the extra probability error introduced by this replacement is at most ϵ. In this chapter, we want to extract m almost-random bits such that they form a sequence ϵ-close to the uniform distribution U_m on $\{0,1\}^m$ with specified small $\epsilon > 0$, i.e.,

$$\|Y - U_m\| \leq \epsilon.$$

6.2.2 Seeded Extractors

In 1990, Zuckerman introduced a general model of weak random sources, called k-sources, namely whose min-entropy is at least k [149]. It was shown that given a source on $\{0,1\}^n$ with min-entropy $k < n$, it is impossible to devise a single function that extracts even one bit of randomness. This observation led to the introduction of seeded extractors, which use a small number of additional truly random bits as the seed (catalyst). When simulating a probabilistic algorithm, one can simply eliminate the requirement of truly random bits by enumerating all possible strings for the seed and taking a majority vote on the final results. There are a variety of very efficient constructions of seeded extractors, summarized in [32, 87, 105]. Mathematically, a seeded extractor is a function,

$$E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m,$$

such that for every distribution X on $\{0,1\}^n$ with $H_{\min}(X) \geq k$, the distribution $E(X, U_d)$ is ϵ-close to the uniform distribution U_m. Here, d is the seed length, and we call such an extractor as a (k, ϵ) extractor. There are a lot of works focusing on efficient constructions of seeded extractors. A standard application of the probabilistic method [93] shows that there exists a seeded extractor which can extract asymptotically $H_{\min}(X)$ random bits with $\log(n - H_{\min}(X))$ additional truly random bits. Recently, Guruswami, Umans and Vadhan [50] provided an explicit construction of seeded extractors, whose efficiency is very close to the bound obtained based on the probabilistic method. Their main result is described as follows:

Lemma 6.1. [50] *For every constant $\alpha > 0$, and all positive integers n, k and all $\epsilon > 0$, there is an explicit construction of a (k, ϵ) extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $d \leq \log n + O(\log(k/\epsilon))$ and $m \geq (1-\alpha)k$.*

The above result implies that given any source $X \in \{0,1\}^n$ with min-entropy k, if $\geq (1+\alpha)m$ with $\alpha > 0$, we can always construct a seeded extractor to generates a random sequence $Y \in \{0,1\}^m$ that is ϵ-close to the uniform distribution. In this case, the seed length $d \leq \log n + O(\log(k/\epsilon))$ depends on the input length n and the parameter ϵ.

6.2.3 Seedless Extractors

In the last decade, the concept of seedless (deterministic) extractors has attracted renewed interests, motivated by the reduction of the computational complexity for simulating probabilistic algorithms as well as some requirements in cryptography [31]. Several specific classes of sources have been studied, including independent sources, which can be divided into several independent parts containing certain amount of randomness [9, 95, 97]; bit-fixing sources, where some bits in a binary sequence are truly random and the remaining bits are fixed [23, 41, 64]; samplable sources, where the source is generated by a process that has a bounded amount of computational resources like space [63, 116]. For example, suppose that we have multiple independent sources with the same length n. It is known how to extract from two sources when the min-entropy in each is $\geq 0.5n$ [97] or slightly less than $0.5n$ [15], how to extract from $O(1/\gamma)$ sources if the min-entropy in each is at least n^{γ} [94]. All these constructions have exponentially small error, and they are able to extract $\Theta(k)$ random bits.

Both seeded extractors and seedless extractors described above have fixed input length, fixed seed length ($d = 0$ for seedless extractors) and fixed output length. So we call them fixed-length extractors. To apply fixed-length extractors in extracting randomness from a stochastic process, it needs to first read a sequence of fixed length, whose min-entropy is strictly larger than the number of random bits that we need to generate. Fixed-length extractors can generate random bits of good quality from imperfect stochastic processes, but they usually consume more incoming symbols than what are necessarily required. To increase information efficiency, we let the length of input sequences be adaptive, hence, we have the concept of 'variable-length extractors'.

6.2.4 Variable-Length Extractors

A variable-length extractor is an extractor with variable input length and fixed output length. When applying a variable-length extractor to a stochastic process, it reads incoming symbols one by one until the whole incoming sequence meets certain criterion, then it maps the incoming sequence into a binary sequence of fixed length as the output. Depending on the sources, the construction may require a small number of additional truly random bits as the seed. Hence, we have seeded

variable-length extractors and seedless variable-length extractors.

A seeded variable-length extractor is a function,

$$V_E : S_p \times \{0,1\}^d \to \{0,1\}^m,$$

such that given a real source $\mathcal{R}$, the output sequence is ϵ-close to the uniform distribution U_m. Here, S_p is the set consisting of all possible input sequences, called the input set. It is complete and prefix-free. The input sequence is compete, that means, any infinite sequence has a prefix in the set; so when reading symbols from any source, we can always meet a sequence in the set. Then we stop reading and map this sequence into a binary sequence of length m. Being prefix-free is not very necessary; it ensures that all the sequences in S_p are possible to read.

A general procedure of extracting randomness by using variable-length extractors can be divided into three steps:

1. Determining an input set S_p such that its min-entropy based on the real source $\mathcal{R}$ is at least k, namely,

   $$\min_{x \in S_p} \log_2 \frac{1}{P_{\mathcal{R}}(x)} \leq k,$$

   where $k \geq (1+\alpha)m$ for any $\alpha > 0$.

2. We construct an injective function

   $$V : S_p \to \{0,1\}^n,$$

   to map the sequences in S_p into binary sequences of length m. We read symbols from the source $\mathcal{R}$ one by one until the current incoming sequence matches one in S_p. This incoming sequence is then mapped to a binary sequence of length n based on function V. As a result, we get a random sequence Z with length n and min-entropy k (since V is injective).

3. Since $k = (1+\alpha)$ with an $\alpha > 0$, according to lemma 6.1, we can always find a seeded extractor,

$$E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

that can extract m almost-random bits from a source with min-entropy k. By applying this seeded extractor E to the sequence Z, we get a random sequence of length m that is ϵ-close to the uniform distribution U_m. Here, the seed length $d \leq \log n + O(\log(k/\epsilon))$.

We can see that the construction of a variable-length extractor is a cascade of a function V and a seeded extractor E, i.e.,

$$V_E = E \bigotimes V.$$

Note that our requirement is to extract a sequence of m almost-random bits that is ϵ-close to the uniform distribution U_m. The key of constructing variable-length extractors is to find the input set S_p with min-entropy k, even the distribution of the real source $\mathcal{R}$ is slightly unpredictable, such that the expected length of the sequences in S_p is minimized. For stationary ergodic processes, minimizing the expected length is equivalent to minimizing the entropy of the sequences in S_p asymptotically (this will be discussed in this section).

For some specific types of sources, including independent sources and samplable sources, by applying the ideas in [95] and [63] we can remove the requirement of truly random bits without degrading the asymptotic performance. As a result, we have seedless variable-length extractors. For example, if the source $\mathcal{R}$ is an independent process, we can first apply the method in [95] to extract d almost-random bits from the first $\Theta(\log \frac{m}{\epsilon})$ bits, and then use them as the seed of a seeded variable-length extractor to extract randomness from the rest of the process. The detailed discussions will be given in section 6.7.

6.3 Efficiency and Uncertainty

6.3.1 Efficiency

To consider the performance of an extractor, we define its *efficiency* as the asymptotical ratio between the output length and the total entropy of all its inputs. So the efficiency of an extractor can be written as

$$\eta = \lim_{m\to\infty} \frac{m}{H_{\mathcal{R}}(X_m) + d},$$

such that the output sequence is ϵ-close to the uniform distribution U_m on $\{0,1\}^m$, where ϵ is small, d is the seed length, m is the output length, and $H_{\mathcal{R}}(X_m)$ is the entropy of the input sequence X_m with range on S_p. In our constructions, $d \leq \log n + O(\log(m/\epsilon))$, which is ignorable compared to $H_{\mathcal{R}}(X_m)$ when $m \to \infty$. Hence, we can write

$$\eta = \lim_{m\to\infty} \frac{m}{H_{\mathcal{R}}(X_m)}.$$

In the definition, we use the entropy of the input sequence rather than the expected input length, because the source that we considered may not be stationary ergodic. It needs to mention that, in seeded constructions, the value of d is also an important parameter although it is much smaller than m. The problem of minimizing the seed length d can be studied separately from minimizing the entropy of the input sequence, and it will be addressed in this chapter.

First, we demonstrate that if a distribution is ϵ-close to the uniform distribution U_m, then the entropy of this distribution is asymptotically m for any $\epsilon < 1$.

Lemma 6.2. *Let X be a random sequence on $\{0,1\}^m$ that is ϵ-close to the uniform distribution U_m, then*

$$m - \log_2 \frac{1}{1-\epsilon} \leq H(X) \leq m.$$

Proof. Since there are totally 2^m possible assignments for X, it is easy to get $H(X) \leq m$. So we

only need to prove that

$$H(X) \geq m - \log_2 \frac{1}{1-\epsilon}.$$

Let $p(x)$ denote $P[X = x]$ for $x \in \{0,1\}^m$. Since X is ϵ-close to the uniform distribution U_m, we have

$$\frac{1}{2} \sum_{x \in \{0,1\}^m} \|p(x) - 2^{-m}\| \leq \epsilon.$$

Then the lower bound of $H(X)$ can be written as

$$\min_p \sum_{x \in \{0,1\}^m} p(x) \log_2 \frac{1}{p(x)}$$

subject to

$$p(x) \geq 0, \forall x \in \{0,1\}^m;$$

$$\sum_{x \in \{0,1\}^m} p(x) = 1;$$

$$\sum_{x \in \{0,1\}^m} \|p(x) - 2^{-m}\| \leq 2\epsilon.$$

Obviously, the optimal solution of the above problem happens at

$$\sum_{x \in \{0,1\}^m} \|p(x) - 2^{-m}\| = 2\epsilon.$$

To solve the problem based on Lagrange Multipliers, we let

$$\lambda(p) = \sum_{x \in \{0,1\}^m} p(x) \log_2 \frac{1}{p(x)} + \lambda_1 (\sum_{x \in \{0,1\}^m} p(x) - 1)$$

$$+\lambda_2 (\sum_{x \in \{0,1\}^m} \|p(x) - 2^{-m}\| - 2\epsilon).$$

If $p(x) \geq 0$ with $x \in \{0,1\}^m$ is a solution of the above question, then

$$\frac{\partial \lambda}{\partial (p(x))} = 0,$$

i.e.,

$$\begin{cases} \frac{\ln p(x)+1}{\ln 2} + \lambda_1 + \lambda_2 = 0 & \text{if } 2^{-m} \le p(x) \le 1, \\ \frac{\ln p(x)+1}{\ln 2} + \lambda_1 - \lambda_2 = 0 & \text{if } 0 \le p(x) \le 2^{-m}. \end{cases}$$

So there exists two constants a and b with $0 \le a \le 2^{-m} \le b \le 1$, such that,

$$\begin{cases} p(x) = a & \text{if } 2^{-m} \le p(x) \le 1, \\ p(x) = b & \text{if } 0 \le p(x) \le 2^{-m}. \end{cases}$$

Assume that there are t assignments of x with $p(x) = a$, then there are $2^m - t$ assignments of x with $p(x) = b$. Hence, the problem is converted to the one over a, b, t, i.e.,

$$\min_{a,b,t} ta \log \frac{1}{a} + (2^m - t)b \log \frac{1}{b},$$

subject to

$$0 \le t \le 2^m;$$

$$ta + (2^m - t)b = 1; \tag{6.1}$$

$$t(2^{-m} - a) + (2^m - t)(b - 2^{-m}) = 2\epsilon. \tag{6.2}$$

From equation (6.1) and (6.2), we get

$$a = 2^{-m} - \frac{\epsilon}{t}, \quad b = 2^{-m} + \frac{\epsilon}{2^m - t}.$$

So the question is finding the optimal t that minimizes

$$-t(2^{-m} - \frac{\epsilon}{t}) \log_2(2^{-m} - \frac{\epsilon}{t}) - (2^m - t)(2^{-m} + \frac{\epsilon}{2^m - t}) \log_2(2^{-m} + \frac{\epsilon}{2^m - t}),$$

subject to

$$0 \le t \le \frac{\epsilon}{2^{-m}}.$$

The optimal solution is $t^* = \frac{\epsilon}{2^{-m}}$. In this case, the entropy of X is

$$H(X) = \log(2^m - t) = m - \log_2 \frac{1}{1-\epsilon},$$

which is the lower bound.

This completes the proof. □

In the following lemma, we show that for any extractor, its efficiency is upper bounded by 1. The reason is that the amount of information, i.e., entropy, does not increase during the process of randomness extraction.

Lemma 6.3. *For any extractor with seed length d and output length m, if $d = o(m)$, its efficiency $\eta \leq 1$.*

Proof. We consider fixed-length extractors as a special case of variable-length extractors, and consider seedless extractors as a special case of seeded extractors when $d = 0$. So our proof only focus on seeded variable-length extractors.

A main observation is that for any extractor, the entropy of its output sequence is bounded by the entropy of the input sequence plus the entropy of the seed, since the process of extracting randomness cannot create new randomness.

For the output sequence, denoted by Y, it is ϵ-close to the uniform distribution U_m. According to Lemma 6.2, its entropy is

$$H_{\mathcal{R}}(Y) \geq m - \log_2 \frac{1}{1-\epsilon}.$$

The total entropy of the inputs is $H_{\mathcal{R}}(X_m) + d$. Hence,

$$H_{\mathcal{R}}(Y) \leq H_{\mathcal{R}}(X_m) + d.$$

As a result, the efficiency of the extractor is

$$\eta = \lim_{m\to\infty} \frac{m}{H_{\mathcal{R}}(X_m)} = \lim_{m\to\infty} \frac{H_{\mathcal{R}}(Y)}{H_{\mathcal{R}}(X_m)+d} \leq 1.$$

This completes the proof. □

If $\mathcal{R}$ is a stationary ergodic process, we define its entropy rate as

$$h(\mathcal{R}) = \lim_{l\to\infty} \frac{H(X^l)}{l},$$

where X^l is a random sequence of length l generated from the source $\mathcal{R}$. In this case, the entropy of the input sequence on S_p is proportional to the expected input length.

Lemma 6.4. *Given a stationary ergodic source $\mathcal{R}$, let X_m be the input sequence of a variable-length extractor that has an output length m. Then*

$$\lim_{m\to\infty} \frac{H_{\mathcal{R}}(X_m)}{E_{\mathcal{R}}[|X_m|]} = h(\mathcal{R}),$$

where $E_{\mathcal{R}}[|X_m|]$ is the expected input length.

Proof. X_m is a random sequence from S_p based on the distribution of $\mathcal{R}$. Let l_1 be the minimum length of the sequences in S_p, as $m \to \infty$, $l_1 \to \infty$. Now, we define

$$l_i = l_1 + (i-1)\log l_1 \text{ for all } i \geq 1.$$

Based on them, we divide all the sequences in S_p into subsets

$$S_i = \{x | x \in S_p, l_i \leq |x| \leq l_{i+1} - 1\}$$

for $i \geq 1$.

Let $p_i = P_{\mathcal{R}}(X_m \in S_i)$, then

$$H_{\mathcal{R}}(X_m) \quad \geq \quad \sum_i [(\sum_{j>i} p_j) H_{\mathcal{R}}(X_{l_{i-1}+1}^{l_i} | X_1^{l_{i-1}}, |X_m| \geq l_i)],$$

where $l_0 = 0$, $\sum_{j>i} p_j$ is the probability that $|X_m| \geq l_i$, and X_a^b is a sequence of X_m from the ath element to the bth element.

Since X_m is generated from a stationary ergodic process, and $l_i - l_{i-1} \to \infty$ as $m \to \infty$, we can get

$$H_{\mathcal{R}}(X_{l_{i-1}+1}^{l_i} | X_1^{l_{i-1}}, |X_m| \geq l_i) \to (l_i - l_{i-1}) h(\mathcal{R}).$$

As a result, as $l_1 \to \infty$, we have

$$\begin{aligned} H_{\mathcal{R}}(X_m) &\geq (1-\epsilon) \sum_i (\sum_{j>i} p_j)(l_i - l_{i-1}) h(\mathcal{R}) \\ &= (1-\epsilon) \sum_i p_i l_i h(\mathcal{R}), \end{aligned}$$

for an arbitrary $\epsilon > 0$.

Also considering the other direction, we can get that as $l_1 \to \infty$,

$$\begin{aligned} H_{\mathcal{R}}(X_m) &\leq (1+\epsilon) \sum_i p_i l_{i+1} h(\mathcal{R}) \\ &= (1+\epsilon) \sum_i p_i (l_i + \log l_1) h(\mathcal{R}), \end{aligned}$$

for an arbitrary $\epsilon > 0$.

For the expected input length, i.e., $E_{\mathcal{R}}[|X_m|]$, it is easy to show that

$$\sum_i p_i l_i \leq E_{\mathcal{R}}[|X_m|] \leq \sum_i p_i l_{i+1} = \sum_i p_i (l_i + \log l_1).$$

So as $m \to \infty$, i.e., $l_1 \to \infty$, it yields

$$\lim_{m\to\infty} \frac{H_{\mathcal{R}}(X_m)}{E_{\mathcal{R}}[|X_m|]} = \lim_{m\to\infty} \frac{\sum_i p_i l_i h(\mathcal{R})}{\sum_i p_i l_i}$$

$$= h(\mathcal{R}).$$

This completes the proof. □

6.3.2 Sources and Uncertainty

Given a source $\mathcal{R}$, if its distribution is known, we say that this source is a known stochastic process, and its uncertainty is 0. In this chapter, we mainly focus on those imperfect processes whose distributions are slightly unpredictable due to many factors like the existence of external adversaries.

First, given two known stochastic processes $\mathcal{R}$ and $\mathcal{M}$, we let $d(\mathcal{R}, \mathcal{M})$ be the difference between $\mathcal{R}$ and $\mathcal{M}$. Here, we define $d(\mathcal{R}, \mathcal{M})$ as

$$d(\mathcal{R}, \mathcal{M}) = \limsup_{n\to\infty} \max_{x\in\{0,1\}^n} \frac{\log_2 \frac{P_{\mathcal{R}}(x)}{P_{\mathcal{M}}(x)}}{\log_2 \frac{1}{P_{\mathcal{M}}(x)}},$$

where $P_{\mathcal{R}}(x)$ is the probability of generating x from $\mathcal{R}$ when the sequence length is $|x|$, and $P_{\mathcal{M}}(x)$ is the probability of generating x from $\mathcal{M}$ when the sequence length is $|x|$. Although there are some existing ways such as normalized Kullback-Leibler divergence to measure the difference between two sources, with them it is not easy to estimate the uncertainty of a source and it is not easy to analyze the performances of constructed variable-length extractors.

In the rest of this chapter, we investigate a few models of unpredictable sources. Most natural source can be well described in those ways.

1. The source $\mathcal{R}$ is an arbitrary stochastic process such that $d(\mathcal{R}, \mathcal{M}) \leq \beta$ for a constant $\beta \in [0, 1]$ and a known stochastic process $\mathcal{M}$.

2. $\mathcal{R}$ is an arbitrary stochastic process such that there exists a stationary ergodic process $\mathcal{M}$ (whose distribution is unknown) and $d(\mathcal{R}, \mathcal{M}) \leq \beta$; that is, $\min_{\mathcal{M}\in\mathcal{G}_{s.e.}} d(\mathcal{R}, \mathcal{M}) \leq \beta$, where $\mathcal{G}_{s.e.}$ denotes the set consisting of all stationary ergodic processes.

In both the models, we call β as the *uncertainty* of the source $\mathcal{R}$. In the real world, β can be easily estimated without knowing the distribution of the processes. It just reflects how unpredictable

the real source $\mathcal{R}$ is.

To construct variable-length extractors, we only care about the possible input sequences, namely, those in S_p. Hence, for the case of finite length, $d_p(\mathcal{R}, \mathcal{M})$ is a more important parameter for us, defined by

$$d_p(\mathcal{R}, \mathcal{M}) = \max_{x \in S_p} \frac{\log_2 \frac{P_{\mathcal{R}}(x)}{P_{\mathcal{M}}(x)}}{\log_2 \frac{1}{P_{\mathcal{M}}(x)}},$$

As the number of required random bits m increases, $d_p(\mathcal{R}, \mathcal{M})$ quickly converge to $d(\mathcal{R}, \mathcal{M})$. And we can write

$$d_p(\mathcal{R}, \mathcal{M}) = d(\mathcal{R}, \mathcal{M}) + \epsilon_p$$

for a very small constant ϵ_p. As $m \to \infty$, $\epsilon_p \to 0$. In this case, the upper bound of $d_p(\mathcal{R}, \mathcal{M})$ or $\min_{\mathcal{M} \in \mathcal{G}_{s.e.}} d_p(\mathcal{R}, \mathcal{M})$ is

$$\beta_p = \beta + \epsilon_p.$$

Example 6.3. *Let $x_1 x_2 ... \in \{0,1\}^*$ be a sequence generated from an independent source $\mathcal{R}$ such that*

$$\forall i \geq 1, P[x_i = 1] \in [0.8, 0.82].$$

If we let $\mathcal{M}$ be a biased coin with probability 0.8132*, then*

$$\beta = \max_{possible\ \mathcal{R}} d(\mathcal{R}, \mathcal{M}) = \max\left(\frac{\log_2 \frac{0.2}{0.1868}}{\log_2 \frac{1}{0.1868}}, \frac{\log_2 \frac{0.82}{0.8132}}{\log_2 \frac{1}{0.8132}}\right) = 0.0405.$$

□

According to our definition, $d(\mathcal{M}, \mathcal{R}) \leq \beta$ if and only if

$$P_{\mathcal{R}}(x) \leq P_{\mathcal{M}}(x)^{1-\beta}$$

for all $x \in \{0,1\}^\infty$ with $|x| \to \infty$. This is a condition that is very easy to be satisfied by many natural stochastic processes for a small β.

Lemma 6.5. *If $d(\mathcal{R}, \mathcal{M}) \to 0$, we have*

$$P_{\mathcal{R}}(x) \to P_{\mathcal{M}}(x)$$

for all $x \in \{0,1\}^*$.

6.3.3 Efficiency and Uncertainty

In this subsection, we investigate the relation between the efficiency and uncertainty. We show that given a stochastic process $\mathcal{R}$ with uncertainty β, as described in the previous subsection, one cannot construct a variable-length extractor with efficiency strictly larger than $1-\beta$ for all the possibilities of $\mathcal{R}$.

Let us first consider a simple example: let X be a random sequence with the uniform distribution on $\{0,1\}^n$ and let Y be an arbitrary random sequence on $\{0,1\}^n$ such that

$$\frac{\log_2 \frac{P[Y=x]}{P[X=x]}}{\log_2 \frac{1}{P[X=x]}} \leq \beta, \forall x \in \{0,1\}^n.$$

Now, we show that from the source Y, one cannot construct an extractor with efficiency strictly larger than $1-\beta$. To see this, we consider an extractor f with output length m, and a source Y with

$$P[Y=y] \in \{0, 2^{-n(1-\beta)}\}, \forall y \in \{0,1\}^n.$$

For this a source Y, its entropy is $H(Y) = n(1-\beta)$. In order to make sure the output sequence of f, denoted by Z, is ϵ-close to U_m, it has

$$\lim_{m\to\infty} \frac{m}{n(1-\beta)} \leq \lim_{m\to\infty} \frac{H(Z) + \log_2 \frac{1}{1-\epsilon}}{H(Y)} \leq 1.$$

So we cannot generate more than $n(1-\beta)$ random bits asymptotically. In this case, if we apply the

seeded extractor f to the random sequence X, which is a possibility of Y, then the efficiency is

$$\eta = \lim_{m\to\infty} \frac{m}{H(X)} = \lim_{m\to\infty} \frac{m}{n} \leq 1 - \beta.$$

So there does not exist a seeded extractor that can extract randomness from an arbitrary Y and its efficiency is strictly larger than $1 - \beta$. Here, β is the uncertainty of the source.

Theorem 6.6. *Let $\mathcal{M}$ be a known stochastic process, and $\mathcal{R}$ be an arbitrary stochastic process such that $d(\mathcal{R}, \mathcal{M}) \leq \beta$, then one cannot construct a variable-length extractor whose efficiency is strictly larger than $1 - \beta$ for all possible $\mathcal{R}$.*

Proof. Let f be a variable-length extractor whose input sequence is a random sequence X_m on S_p and its output sequence is a random sequence Y on $\{0,1\}^m$. Assume that as $m \to \infty$, f can extract from an arbitrary $\mathcal{R}$ such that the output sequence Y is ϵ-close to U_m.

Let $h = H_{\mathcal{M}}(X_m)$ be the entropy of the input sequence based on the distribution of $\mathcal{M}$, then we want to show that there exists a process $\mathcal{R}$ such that $d(\mathcal{R}, \mathcal{M}) \leq \beta$ and $H_{\mathcal{R}}(X_m) \leq h(1 - \beta)$ as $m \to \infty$.

To find such a process $\mathcal{R}$, we order all the elements in S_p as $x_1, x_2, x_3, ...$ such that

$$P_{\mathcal{M}}(x_1) \geq P_{\mathcal{M}}(x_2) \geq P_{\mathcal{M}}(x_3) \geq ...$$

Then we divide all these elements into groups,

$$\{x_1, x_2, ..., x_{i_1}\}, \{x_{i_1+1}, x_{i_1+2}, ..., x_{i_2}\}, ...$$

such that the total probability of the elements in each group is almost the probability of its first element to the power of $1 - \beta$, i.e.,

$$0 \leq P_{\mathcal{M}}(x_{i_j+1})^{1-\beta} - \sum_{k=i_j+1}^{i_{j+1}} P_{\mathcal{M}}(x_k) \leq P_{\mathcal{M}}(x_{i_j+1}), \text{ for } j \geq 0,$$

where $i_0 = 0$.

Let $A = \{x_1, x_{i_1+1}, x_{i_2+1}, ...\}$ be the set consisting of the first elements of all the groups. Now, we consider a possibility of $\mathcal{R}$ in the following way: for all $x \in \{x_1, x_{i_1+1}, x_{i_2+1}, ...\}$, its probability is

$$P_{\mathcal{R}}(x) = \sum_{k=i_j+1}^{i_{j+1}} P_{\mathcal{M}}(x_k), \text{ if } x = x_{i_j+1};$$

For all $x \in S_p/A = S_p/\{x_1, x_{i_1+1}, x_{i_2+1}, ...\}$, its probability is

$$P_{\mathcal{R}}(x) = 0.$$

For this source $\mathcal{R}$, the entropy of the input sequence is

$$H_{\mathcal{R}}(X_m) = \sum_{x \in S_p} P_{\mathcal{R}}(x) \log_2 \frac{1}{P_{\mathcal{R}}(x)}.$$

As $m \to \infty$, we have

$$\begin{aligned}
& H_{\mathcal{R}}(X_m) \\
= & \sum_{x \in A} P_{\mathcal{R}}(x) \log_2 \frac{1}{P_{\mathcal{R}}(x)} \\
\to & (1-\beta) \sum_{x \in A} P_{\mathcal{R}}(x) \log_2 \frac{1}{P_{\mathcal{M}}(x)} \\
= & (1-\beta) \sum_{j \geq 0} \sum_{k=i_j+1}^{i_{j+1}} P_{\mathcal{M}}(x_k) \log_2 \frac{1}{P_{\mathcal{M}}(x_{i_j+1})} \\
\leq & (1-\beta) \sum_{j \geq 0} \sum_{k=i_j+1}^{i_{j+1}} P_{\mathcal{M}}(x_k) \log_2 \frac{1}{P_{\mathcal{M}}(x_k)} \\
= & (1-\beta) H_{\mathcal{M}}(X_m) \\
= & (1-\beta) h.
\end{aligned}$$

According to lemma 6.2, as $m \to \infty$, $\frac{m}{H_{\mathcal{R}}(Y)} \to 1$. Furthermore, we can get

$$\lim_{m \to \infty} \frac{H_{\mathcal{R}}(Y)}{H_{\mathcal{R}}(X_m)} \leq 1,$$

it implies that

$$\lim_{m\to\infty} \frac{m}{(1-\beta)h} \leq 1,$$

otherwise, the output sequence cannot be ϵ-close to the uniform distribution U_m.

If we apply the extractor f to the source $\mathcal{M}$, which is also a possibility for $\mathcal{R}$, then its efficiency is

$$\eta = \lim_{m\to\infty} \frac{m}{h} \leq 1-\beta.$$

So it is impossible to construct a variable-length extractor with efficiency strictly larger than $1-\beta$ for all the possibilities of the source $\mathcal{R}$. This completes the proof. □

With the same proof, we can also get the following theorem.

Theorem 6.7. *Let $\mathcal{R}$ be an arbitrary stochastic process such that $d(\mathcal{R},\mathcal{M}) \leq \beta$ for a stationary ergodic process $\mathcal{M}$ with unknown distribution, , then one cannot construct a variable-length extractor whose efficiency is strictly larger than $1-\beta$ for all possible $\mathcal{R}$.*

The above theorems show that one cannot construct an extractor whose efficiency is strictly larger than $1-\beta$ for all the possible source $\mathcal{R}$. Here, β is an important parameter that measures the uncertainty of a real source $\mathcal{R}$, either to a known process or to the nearest stationary ergodic process. In the next a few sections, we will present a few constructions for efficiently extracting randomness from the sources described in this section. We show that their efficiency η satisfies

$$1-\beta \leq \eta \leq 1.$$

That means the bound $1-\beta$ is actually achievable and the constructions proposed in this chapter are asymptotically optimal on efficiency.

6.4 Construction I: Approximated by Known Processes

In this section, we consider those sources which can be approximated by a known stochastic process $\mathcal{M}$, namely, an arbitrary process $\mathcal{R}$ with $d(\mathcal{R}, \mathcal{M}) \leq \beta$ for a known process $\mathcal{M}$. We say that a stochastic process $\mathcal{M}$ is known if its distribution is given, i.e., $P_{\mathcal{M}}(x)$ can be easily calculated for any $x \in \{0,1\}^*$. Note that this process $\mathcal{M}$ is not necessary to be stationary or ergodic. For instance, $\mathcal{M}$ can be an independent process $z_1 z_2 ... \in \{0,1\}^*$ such that

$$\forall i \geq 1, P_{\mathcal{M}}(z_i = 1) = \frac{1 + sin(i/10)}{2}.$$

6.4.1 Construction

Our goal is to extract randomness from an imperfect random source $\mathcal{R}$. The problem is that we do not know the exact distribution of $\mathcal{R}$, but we know that it can be approximated by a known process $\mathcal{M}$. So we can use the distribution of $\mathcal{M}$ to estimate the distribution of $\mathcal{R}$. As a result, we have the following procedure to extract m almost-random bits.

The idea of the procedure is first producing a random sequence of length n and min-entropy $k = m(1+\alpha)$ with $\alpha > 0$, from which we can further obtain a sequence ϵ-close to the uniform distribution U_m by applying a (k, ϵ) seeded extractor. According to the results of seeded extractors, this constant $\alpha > 0$ can be arbitrarily small.

Construction 6.1. *Assume the real source $\mathcal{R}$ is an arbitrary stochastic process such that $d(\mathcal{R}, \mathcal{M}) \leq \beta$ for a known process $\mathcal{M}$. Then we extract m almost-random bits from $\mathcal{R}$ based on the following procedure.*

1. *Read input bits one by one from $\mathcal{R}$ until we get an input sequence $x \in \{0,1\}^*$ such that*

$$\log_2 \frac{1}{P_{\mathcal{M}}(x)} \geq \frac{k}{1 - \beta_p},$$

where $\beta_p = \beta + \epsilon_p$ with $\epsilon_p > 0$ and $k = m(1+\alpha)$ with $\alpha > 0$. The small constant ϵ_p has value

depending on the input set S_p; as $m \to \infty$, $\epsilon_p \to 0$. The constant α can be arbitrarily small.

2. *Let n be the maximum length of all the possible input sequences, then*

$$n = \arg\min_l \{l \in \mathbb{N} | \forall y \in \{0,1\}^l, \log_2 \frac{1}{P_{\mathcal{M}}(y)} \geq \frac{k}{1-\beta_p}\}.$$

If $|x| < n$, we extend the length of x to n by adding $n - |x|$ trivial zeros at the end. Since x is randomly generated, from the above procedure we get a random sequence Z of length n. And it can be proved that this random sequence has min-entropy k.

3. *Applying a (k, ϵ) extractor to Z yields a binary sequence of length m that is ϵ-close to the uniform distribution U_m.* □

The following example is provided for comparing this construction with fixed-length constructions.

Example 6.4. *Let $\mathcal{M}$ be a biased coin with probability 0.8 (of being 1). If $\frac{k}{1-\beta_p} = 2$, then we can get the input set*

$$S_p = \{0, 10, 110, 1110, 11110, 111110, 1111110, 1111111\}.$$

In this case, the expected input length is strictly smaller than 7. For fixed-length constructions, to get a random sequence with min-entropy at least 2, we have to read 7 input bits independent of the context. It is less efficient than the former method. □

Theorem 6.8. *Construction 6.1 generates a random sequence of length m that is ϵ-close to U_m.*

Proof. We only need to prove that given a source $\mathcal{R}$ and a model $\mathcal{M}$ with $d_p(\mathcal{R}, \mathcal{M}) \leq \beta_p$, it yields a random sequence Z with min-entropy at least k.

According to the definition of $d_p(\mathcal{R}, \mathcal{M})$, for all $x \in S_p$,

$$\frac{\log_2 \frac{P_{\mathcal{R}}(x)}{P_{\mathcal{M}}(x)}}{\log_2 \frac{1}{P_{\mathcal{M}}(x)}} \leq \beta_p.$$

Based on the construction, for all $x \in S_p$

$$\log_2 \frac{1}{P_{\mathcal{M}}(x)} \geq \frac{k}{1-\beta_p}.$$

The two inequalities above yield that

$$\log_2 \frac{1}{P_{\mathcal{R}}(x)} \geq k,$$

for all $x \in S_p$.

Since the second step, i.e., adding trivial zeros, does not reduce the min-entropy of S_p. As a result, we get a random sequence Z of length n and with min-entropy at least k.

Since $k = m(1+\alpha)$ with $\alpha > 0$, according to lemma 6.1, we can construct a seeded extractor that applies to the sequence Z and generates a binary sequence ϵ-close to the uniform distribution U_m.

This completes the proof. □

6.4.2 Efficiency Analysis

Now, we study the efficiency of construction 6.1. According to our definition, given a construction, its efficiency is

$$\eta = \lim_{m\to\infty} \frac{m}{H_{\mathcal{R}}(X_m)}.$$

Theorem 6.9. *Given a real source $\mathcal{R}$ and a known process $\mathcal{M}$ such that $d(\mathcal{R},\mathcal{M}) \leq \beta$, then the efficiency of construction 6.1 is*

$$1-\beta \leq \eta \leq 1.$$

Proof. Since η is always upper bounded by 1, we only need to show that $\eta \geq 1-\beta$.

According to lemma 6.1, as $m \to \infty$, we have

$$\lim_{m\to\infty} \frac{k}{m} = 1.$$

Now, let us consider the number of elements in S_p, i.e., $|S_p|$. To calculate $|S_p|$, we let

$$S'_p = \{x[1 : |x| - 1] | x \in S_p\},$$

where $x[1 : |x| - 1]$ is the prefix of x of length $|x| - 1$, then for all $y \in S'_p$,

$$\log_2 \frac{1}{P_{\mathcal{M}}(y)} \leq \frac{k}{1 - \beta_p}.$$

Hence,

$$\log_2 |S'_p| \leq \frac{k}{1 - \beta_p}.$$

It is easy to see that $|S_p| \leq 2|S'_p|$, so

$$\log_2 |S_p| \leq \frac{k}{1 - \beta_p} + 1.$$

Let X_m be the input sequence, then

$$\lim_{k\to\infty} \frac{H_{\mathcal{R}}(X_m)}{k} \leq \lim_{k\to\infty} \frac{\log_2 |S_p|}{k}$$

$$\leq \lim_{k\to\infty} \frac{1}{1 - \beta_p} = \frac{1}{1 - \beta}.$$

Finally, it yields

$$\eta = \lim_{m\to\infty} \frac{m}{H_{\mathcal{R}}(X_m)} \geq 1 - \beta.$$

This completes the proof. □

We see that the efficiency of the above construction is between $1 - \beta$ and 1. As shown in

theorem 6.6, the gap β, introduced by the uncertainty of the real source $\mathcal{R}$, cannot be smaller. Our construction is asymptotically optimal in the sense that we cannot find a variable-length extractor with efficiency definitely larger than $1-\beta$.

Corollary 6.10. *Given a real source $\mathcal{R}$ and a known process $\mathcal{M}$ such that $d(\mathcal{R},\mathcal{M}) \leq \beta$, then as $\beta \to 0$, the efficiency of construction 6.1 is*

$$\eta \to 1.$$

In this case, the efficiency of the construction can achieve Shannon's limit.

If $\mathcal{R}$ is a stationary ergodic process, we can also get the following result.

Corollary 6.11. *Given a stationary ergodic process$\mathcal{R}$ and a known process $\mathcal{M}$ such that $d(\mathcal{R},\mathcal{M}) \leq \beta$, for the expected input length of construction 6.1, we have*

$$\frac{1}{h(\mathcal{R})} \leq \lim_{m\to\infty} \frac{E[|X_m|]}{m} \leq \frac{1}{(1-\beta)h(\mathcal{R})},$$

where $h(\mathcal{R})$ is the entropy rate of the source $\mathcal{R}$.

Proof. This conclusion is immediate following lemma 6.4 and theorem 6.9. □

6.5 Construction II: Approximately Biased Coins

In this section, we use a general ideal model such as a biased coin or a Markov chain to approximate the real source $\mathcal{R}$. Here, we do not care about the specific parameters of the ideal model. The reason is, in some cases, the source $\mathcal{R}$ is very close to an ideal source but we cannot (or do not want to) estimate the parameters accurately. As a result, we introduce a construction by exploring the characters of biased coins or Markov chains. For simplicity, we only discuss the case that the ideal model is a biased coin, and the same idea can be generalized when the ideal model is a Markov chain. Specifically, let $\mathcal{G}_{b.c.}$ denote the set consisting of all the models of biased coins with different

probabilities, and we consider $\mathcal{R}$ as an arbitrary stochastic process such that

$$\min_{\mathcal{M}\in\mathcal{G}_{b.c.}} d(\mathcal{R},\mathcal{M}) \leq \beta.$$

6.5.1 Construction

The idea of the construction is similar as construction 6.1, i.e., we first produce a random sequence of length n and with min-entropy $k = m(1+\alpha)$ for $\alpha > 0$, from which we can further obtain a sequence ϵ-close to the uniform distribution U_m by applying a (k,ϵ) seeded extractor.

Construction 6.2. *Assume the real source $\mathcal{R}$ is an arbitrary stochastic process such that*

$$\min_{\mathcal{M}\in\mathcal{G}_{b.c.}} d(\mathcal{R},\mathcal{M}) \leq \beta$$

for a constant β. Then we extract m almost-random bits from $\mathcal{R}$ based on the following procedure.

1. *Read input bits one by one from $\mathcal{R}$ until we get an input sequence $x \in \{0,1\}^*$ such that*

$$\log_2 \binom{k_0+k_1}{\max(1,\min(k_0,k_1))} \geq \frac{k}{1-\beta_p},$$

where k_0 is the number of zeros in x, k_1 is the number of ones in x, $\beta_p = \beta + \epsilon_p$ with $\epsilon_p > 0$ and $k = m(1+\alpha)$ with $\alpha > 0$. The small constant ϵ_p has value depending on the input set S_p; as $m \to \infty$, $\epsilon_p \to 0$. The constant α can be arbitrarily small.

2. *Since the input sequence x can be very long, we map it into a sequence z of fixed length n such that*

$$z = [I_{(k_0\geq k_1)}, \min(k_0,k_1), r(x)],$$

where $I_{(k_0\geq k_1)} = 1$ if and only if $k_0 \geq k_1$, and $r(x)$ is the rank of x among all the permutations of x with respect to the lexicographic order. Since x is randomly generated, the above procedure leads us to a random sequence Z of length n.

3. Applying a (k, ϵ) extractor to Z yields a random sequence of length m that is ϵ-close to U_m.$\square$

To see that the construction above works, we need to show that the random sequence Z obtained after the second step has min-entropy at least k, and its length n is well bounded.

Lemma 6.12. *Given a source $\mathcal{R}$ with $\min_{\mathcal{M}\in\mathcal{G}_{b.c.}} d(\mathcal{R}, \mathcal{M}) \leq \beta$, construction 6.2 yields a random sequence Z with length*

$$n \leq 1 + \lceil \log_2(\frac{k}{1-\beta_p} + 1)\rceil + \lceil \frac{2k}{1-\beta_p}\rceil.$$

Proof. 1) $I_{(k_0\geq k_1)}$ can be represented as 1 bit.

2) Without loss of generality, we assume $k_0 \leq k_1$. According to our construction,

$$\log_2 \binom{k_0 + k_1 - 1}{k_0 - 1} < \frac{k}{1-\beta_p} \text{ for } k_0 > 1,$$

and

$$\log_2 \binom{k_1}{1} < \frac{k}{1-\beta_p} \text{ for } k_0 = 0 \text{ or } k_0 = 1.$$

Then

$$\begin{aligned} k_0 - 1 &\leq \log_2 \binom{2k_0 - 1}{k_0 - 1} \\ &\leq \log_2 \binom{k_0 + k_1 - 1}{k_0 - 1} \\ &< \frac{k}{1-\beta_p}. \end{aligned}$$

So $\min(k_0, k_1)$ can be represented as $\lceil \log_2(\frac{k}{1-\beta_p} + 1)\rceil$ bits.

3) Let us consider the number of permutations of x, denoted by $N(x)$. If $k_0 > 1$, then

$$\begin{aligned} \log_2 N(x) &= \log_2 \binom{k_0 + k_1}{k_0} \\ &\leq \log_2 \binom{k_0 + k_1 - 1}{k_0 - 1} + \log_2 \frac{k_0 + k_1}{k_0} \\ &\leq \frac{k}{1-\beta_p} + \log_2 \frac{k_0 + k_1}{k_0}. \end{aligned}$$

If $k_0 = 1$, then

$$\log_2 N(x) \leq \log_2 \binom{k_1}{1} + \log_2 \frac{k_1 + 1}{k_1}.$$

If $k_0 = 0$, then

$$\log_2 N(x) = 0.$$

Based on the analysis above, we can get

$$\log_2 N(x) \leq \frac{2k}{1 - \beta_p}.$$

Hence, $r(x)$ can be represented as $\lceil \frac{2k}{1-\beta_p} \rceil$ bits.

This completes the proof. □

Let $\mathbf{1}^a$ denote the all-one vector of length a, then we get the following result.

Theorem 6.13. *Construction 6.2 generates a random sequence of length m that is ϵ-close to U_m if $P_{\mathcal{R}}(\mathbf{1}^a) \leq 2^{-k}, P_{\mathcal{R}}(\mathbf{0}^a) \leq 2^{-k}$ for $a = 2^{\lfloor \frac{k}{1-\beta_p} \rfloor}$.*

Proof. Since the mapping in the second step is injective, it will not affect the min-entropy; we only need to prove that the input sequence has min-entropy k, i.e.,

$$\log_2 \frac{1}{P_{\mathcal{R}}(x)} \geq k, \forall x \in S_p,$$

where S_p is the set consisting of all the possible input sequences.

It is not hard to see that if $\min(k_0, k_1) \geq 1$,

$$P_{\mathcal{M}}(x) \leq \frac{1}{\binom{k_0 + k_1}{k_0}},$$

which leads to

$$\log_2 \frac{1}{P_{\mathcal{M}}(x)} \geq \frac{k}{1 - \beta_p}.$$

Furthermore, based on the definition of $d_p(\mathcal{R}, \mathcal{M})$, we can get if $\min(k_0, k_1) \geq 1$,

$$\log_2 \frac{1}{P_{\mathcal{R}}(x)} \geq k.$$

If $\min(k_0, k_1) = 0$, according to the condition in the lemma, we can also have the same result.

Since $k = m(1 + \alpha)$ with $\alpha > 0$, according to lemma 6.1, we can construct a seeded extractor that applies to the sequence Z and generates a binary sequence ϵ-close to the uniform distribution U_m.

This completes the proof. □

Actually, the idea above can be easily generalized if $\mathcal{M}$ is a Markov chain that best approximates the real source $\mathcal{R}$. The idea follows the main lemma in chapter 3 that shows how to generate random bits with optimal efficiency from an arbitrary Markov chain.

6.5.2 Efficiency Analysis

For the efficiency of the construction, we can get the same bounds as construction 6.1.

Theorem 6.14. *Given an arbitrary source $\mathcal{R}$ such that*

$$\min_{\mathcal{M} \in \mathcal{G}_{b.c.}} d(\mathcal{R}, \mathcal{M}) \leq \beta,$$

if there exists a model $\mathcal{M} \in \mathcal{G}_{b.c.}$ with probability $p \leq \frac{1}{2}$ of being 1 or 0 and

$$p > \sqrt{d(\mathcal{R}, \mathcal{M}) \log_2 \frac{1}{p} \frac{\ln 2}{2}},$$

then the efficiency of construction 6.2 is

$$1 - \beta \leq \eta \leq 1.$$

Proof. Let N_{k_0,k_1} denote the number of input sequences with k_0 zeros and k_1 ones in S_p, and let

p_{k_0,k_1} be the probability based on $\mathcal{R}$ of generating such a sequence. Let us define

$$A = \{(k_0, k_1) | N_{k_0,k_1} > 0\},$$

then we can get

$$H_{\mathcal{R}}(X_m) \quad \leq \quad H(\{p_{k_0,k_1} | (k_0, k_1) \in A\}) + \sum_{(k_0,k_1)\in A} p_{k_0,k_1} \log_2 N_{k_0,k_1}.$$

According to the proof in the above theorem, $\min(k_0, k_1) \leq \frac{k}{1-\beta_p} + 1$. So there are totally at most $2(\frac{k}{1-\beta_p} + 1)$ available pairs of (k_0, k_1). Hence

$$H(\{p_{k_0,k_1} | (k_0, k_1) \in A\}) \leq \log_2(2 + (\frac{k}{1-\beta_p} + 1)) = o(k).$$

Now, we write $n = k_0 + k_1$. According to our method, if $\min(k_0, k_1) \geq 1$,

$$\binom{k_0 + k_1}{\min(k_0, k_1)} \geq 2^{\frac{k}{1-\beta_p}},$$

$$\binom{k_0 + k_1 - 1}{\min(k_0, k_1) - 1} < 2^{\frac{k}{1-\beta_p}}.$$

Hence, given n, we get an upper bound for $\min(k_0, k_1)$, which is

$$t_n = \max\{i \in \{0, 1, ..., n\} | \binom{n-1}{i-1} < 2^{\frac{k}{1-\beta_p}}\}. \tag{6.3}$$

Note that if $\binom{n-1}{\frac{n}{2}-1} \geq 2^{\frac{k}{1-\beta_p}}$, then t_n is a nondecreasing function of n. Using the Stirling bounds on factorials yields

$$\lim_{n\to\infty} \frac{1}{n} \log_2 \binom{n}{\rho n} = H(\rho),$$

where H is the binary entropy function. Hence, following (6.3), we can get

$$\lim_{n\to\infty} H(\frac{t_n}{n}) = \lim_{n\to\infty} \frac{k}{(1-\beta_p)n}. \tag{6.4}$$

Let P_n denote the probability of having an input sequence of length at least n based on the distribution of $\mathcal{R}$. In this case, P_n is a nonincreasing function of n. Let Q_n denote the probability of having an input sequence of length at least n based on the distribution of $\mathcal{M} \in \mathcal{G}_{b.c.}$ whose probability is $p \leq \frac{1}{2}$. Since for all binary sequence $x \in \{0,1\}^n$,

$$\log_2 \frac{1}{P_{\mathcal{M}}(x)} \leq n \log_2 \frac{1}{p},$$

we can get

$$\log_2 \frac{P_{\mathcal{R}(x)}}{P_{\mathcal{M}}(x)} \leq dn \log_2 \frac{1}{p},$$

where $d = d_p(\mathcal{R}, \mathcal{M})$.

Since $P_n = \sum_{x \in S} P_{\mathcal{R}}(x)$ and $Q_n = \sum_{x \in S} P_{\mathcal{M}}(x)$ for some $S \subset \{0,1\}^n$, it is not hard to prove that

$$\log_2 \frac{P_n}{Q_n} \leq dn \log_2 \frac{1}{p}. \tag{6.5}$$

According to Hoeffding's inequality, we can get

$$\begin{aligned} Q_n &\leq 2P[k_1 \leq t_n] \\ &\leq 2P[\frac{k_1}{n} - p \leq \frac{t_n}{n} - p] \\ &\leq 2e^{-2n(p-\frac{t_n}{n})^2}. \end{aligned}$$

Hence

$$P_n \leq 2^{-dn \log_2 p} Q_n \leq 2e^{-\log_2 p \ln 2 \cdot dn - 2n(p-\frac{t_n}{n})^2}. \tag{6.6}$$

From this inequality, we see that $P_n \to 0$ as $n \to 0$ if

$$-d \log_2 p \ln 2 - 2(p - \frac{t_n}{n})^2 < 0. \tag{6.7}$$

Based on (6.4) and (6.7), we can get that $P_n \to 0$ as $n \to 0$ if

$$\frac{n}{k} \geq \frac{1}{(1-\beta_p)H(p-\sqrt{d\log_2 \frac{1}{p}\frac{\ln 2}{2}})}.$$

Now, let $a = \frac{1+\epsilon}{(1-\beta_p)H(p-\sqrt{d\log_2 \frac{1}{p}\frac{\ln 2}{2}})}$ with $\epsilon > 0$, we can write

$$\begin{aligned} H_{\mathcal{R}}(X_m) &\leq o(k) + \sum_{k_0,k_1:k_0+k_1\geq ak} p_{k_0,k_1}\log_2 N_{k_0,k_1} \\ &\quad + \sum_{k_0,k_1:k_0+k_1<ak} p_{k_0,k_1}\log_2 N_{k_0,k_1}. \end{aligned}$$

According to our analysis, if $k_0 + k_1 \geq ak$, as $k \to \infty$,

$$P_n = \sum_{k_0,k_1:k_0+k_1\geq ak} p_{k_0,k_1} \to 0$$

and $\log_2 N_{k_0,k_1} \leq 2\frac{k}{1-\beta_p}$. If $k_0 + k_1 \leq ak$, then

$$\log_2 N_{k_0,k_1} \leq \frac{k}{1-\beta_p} + \log_2 \frac{k_0+k_1}{\min(k_0,k_1)} \leq \frac{k}{1-\beta_p} + o(k).$$

As a result, we can get

$$\begin{aligned} H_{\mathcal{R}}(X_m) &\leq o(k) + o(1)\frac{2k}{1-\beta_p} + (\frac{k}{1-\beta_p} + o(k)) \\ &\leq \frac{k}{1-\beta_p} + o(k). \end{aligned}$$

So

$$\lim_{k\to\infty} \frac{k}{H_{\mathcal{R}}(X_m)} \geq 1-\beta.$$

Furthermore, based on the fact that $\lim_{m\to\infty}\frac{k}{m} = 1$, we can get $\eta \geq 1-\beta$. It is known that $\eta \leq 1$, so it concludes the theorem. □

Similar to construction 6.1, this construction is also asymptotically optimal in the sense that

we cannot find a variable-length extractor with efficiency definitely larger than $1-\beta$, as shown in theorem 6.6.

Corollary 6.15. *Given an arbitrary source $\mathcal{R}$ such that*

$$\min_{\mathcal{M}\in\mathcal{G}_{b.c.}} d(\mathcal{R},\mathcal{M}) \leq \beta,$$

then as $\beta \to 0$, the efficiency of construction 6.2 is

$$\eta \to 1.$$

It is easy to see that as $\beta \to 0$, construction 6.2 reaches the Shannon's limit on efficiency. If $\mathcal{R}$ is a stationary ergodic process, we can also get the following corollary.

Corollary 6.16. *Given an arbitrary stationary ergodic source $\mathcal{R}$ such that*

$$\min_{\mathcal{M}\in\mathcal{G}_{b.c.}} d(\mathcal{R},\mathcal{M}) \leq \beta,$$

if there exists a model $\mathcal{M} \in \mathcal{G}_{b.c.}$ with probability $p \leq \frac{1}{2}$ of being 1 or 0 and

$$p > \sqrt{d(\mathcal{R},\mathcal{M}) \log_2 \frac{1}{p} \frac{\ln 2}{2}},$$

then for the expected input length of construction 6.2, we have

$$\frac{1}{h(\mathcal{R})} \leq \lim_{m\to\infty} \frac{E[|X_m|]}{m} \leq \frac{1}{(1-\beta)h(\mathcal{R})},$$

where $h(\mathcal{R})$ is the entropy rate of $\mathcal{R}$.

6.6 Construction III: Approximately Stationary Ergodic Processes

In this section, we consider imperfect sources that are approximately stationary and ergodic. Here, we let $\mathcal{R}$ be an arbitrary stochastic process such that $d(\mathcal{R}, \mathcal{M}) \leq \beta$ for a stationary ergodic process $\mathcal{M}$. For these sources, universal data compression can be used to 'purify' input sequences, i.e., shortening their lengths while maintaining their entropies. In [126], Visweswariah, Kulkarni and Verdú showed that optimal variable-length source codes asymptotically achieve optimal variable-length random bits generation in the sense of normalized divergence. Although their work only focused on ideal stationary ergodic processes and generates 'weaker' random bits, it motivates us to combine universal compression with fixed-length extractors for efficiently generating random bits from noisy stochastic processes. In this section, we will first introduce Lempel-Ziv code and then present its application in constructing variable-length extractors.

6.6.1 Construction

Lempel-Ziv code is a universal data compression scheme introduced by Ziv and Lempel [136], which is simple to implement and can achieve the asymptotically optimal rate for stationary ergodic sources. The idea of Lempel-Ziv code is to parse the source sequence into strings that have not appeared so far, as demonstrated by the following example.

Example 6.5. *Assume the input is* 010111001110000..., *then we parse it as strings*

$$0, 1, 01, 11, 00, 111, 000, ...$$

where each string is the shortest string that never appear before. That means all its prefixes have occurred earlier.

Let $c(n)$ be the number of strings obtained by parsing a sequence of length n. For each string, we describe its location with $\log c(n)$ bits. Given a string of length l, it can described by (1) the location

of its prefix of length $l-1$, and (2) its last bit. Hence, the code for the above sequence is

$$(000,0),(000,1),(001,1),(010,1),(001,0),(100,1),(101,0),...$$

where the first number in each pair indicates the prefix location and the second number is the last bit of the string.

□

Typically, Lempel-Ziv is applied to an input sequence of fixed length. Here, we are interested in Lempel-Ziv code with fixed output length and variable input length. As a result, we can apply a single fixed-length extractor to the output of Lempel-Ziv code for extracting randomness. In our algorithm, we read raw bits one by one from an imperfect source until the length of the output of a Lempel-Ziv code reaches a certain length. In another word, the number of strings after parsing is a predetermined number c. For example, if the source is 1011010100010... and $c = 4$, then after reading 6 bits, we can parse them into $1, 0, 11, 01$. Now, we get an output sequence $(000,1),(000,0),(001,1),(010,1)$, which can be used as the input of a fixed-length extractor. We call this Lempel-Ziv code as a variable-length Lempel-Ziv code.

Let Z be a random sequence obtained based on variable-length Lempel-Ziv code such that its length is

$$|Z| = (\log c + 1)c,$$

for a predetermined c. Then Z is very close to truly random bits in the term of min-entropy if the source $\mathcal{R}$ is stationary ergodic. As a result, we have the following construction for variable-length extractors.

Construction 6.3. *Assume the real source is $\mathcal{R}$ and there exists a stationary ergodic process $\mathcal{M}$ such that $d(\mathcal{R}, M) \leq \beta$. Then we extract m almost random bits from $\mathcal{R}$ based on the following procedure.*

1. *Read input bits one by one based on the variable-length Lempel-Ziv code until we get an output*

sequence Z whose length reaches

$$n = \frac{k}{1-\beta_p}(1+\varepsilon),$$

where $\varepsilon > 0$ is a small constant indicating the performance gap between the case of finite-length and that of infinite-length for Lempel-Ziv code; as $m \to \infty$, we have $\varepsilon \to 0$. Similar as above, $\beta_p = \beta + \epsilon_p$ with $\epsilon_p > 0$ and $k = m(1+\alpha)$ with $\alpha > 0$. The small constant ϵ_p has value depending on the input set S_p; as $m \to \infty$, $\epsilon_p \to 0$. The constant α can be arbitrarily small. Then we get a random sequence Z of length n and with min-entropy k.

2. *Applying a (k, ϵ) extractor to Z yields a random sequence of length m that is ϵ-close to U_m.*□

We show that the min-entropy of Z is at least k as $m \to \infty$. If m is not very large, by adjusting the parameter ε, we can make the min-entropy of Z be at least k. So we can continue to apply an efficient fixed-length extractor to 'purify' the resulting sequence. Finally, we can get m random bits that satisfy our requirements on quality in the sense of statistical distance.

Theorem 6.17. *When $m \to \infty$, construction 6.3 generates a random sequence of length m that is ϵ-close to U_m.*

Proof. Let x be an input sequence. According to theorem 12.10.1 in [27], for the stationary ergodic process $\mathcal{M}$, we can get

$$\frac{1}{|x|}\log_2 \frac{1}{P_{\mathcal{M}}(x)} \geq \frac{c}{|x|}\log_2 c - \frac{c}{|x|}H(U,V),$$

where

$$\frac{c}{|x|}H(U,V) \to 0 \text{ as } |x| \to 0.$$

As a result, if $k = \Theta(n)$,

$$\begin{aligned}
\lim_{k\to\infty} \frac{1}{k} \log_2 \frac{1}{P_{\mathcal{R}}(x)} &\geq \lim_{k\to\infty} (1-\beta_p) \frac{1}{k} \log_2 \frac{1}{P_{\mathcal{M}}(x)} \\
&\geq \lim_{k\to\infty} \frac{(1-\beta_p) c \log_2 c}{k} \\
&= \lim_{k\to\infty} \frac{(1-\beta_p) n}{k} \\
&= \lim_{k\to\infty} 1+\varepsilon \\
&= 1.
\end{aligned}$$

Finally, we can get that

$$\lim_{k\to\infty} \frac{H_{\min}(Z)}{k} = \lim_{k\to\infty} \frac{H_{\min}(X_m)}{k} \geq 1.$$

This implies that as $m \to \infty$, i.e., $k \to \infty$, the min-entropy of Z is at least k.

Since $k = m(1+\alpha)$ for an $\alpha > 0$, we can continue to apply a (k, ϵ) extractor to extract m almost-random bits from Z. □

6.6.2 Efficiency Analysis

Now, we study the efficiency of the construction based on variable-length Lempel-Ziv codes.

Theorem 6.18. *Given a real source $\mathcal{R}$ such that there exists a stationary ergodic process $\mathcal{M}$ with $d(\mathcal{R}, \mathcal{M}) \leq \beta$, then the efficiency of construction 6.3 is*

$$1 - \beta \leq \eta \leq 1.$$

Proof. Similar as above, we only need to prove that $\eta \geq 1 - \beta$.

Since there are at most $n = 2^{c(\log_2 c + 1)}$ distinct input sequences, their entropy

$$H_{\mathcal{R}}(X_m) \leq c(\log_2 c + 1) = n.$$

According to the proof in theorem 6.17, we have that the random sequence Z has min-entropy at least k, and it satisfies

$$\lim_{m\to\infty}\frac{n}{k}=\frac{1}{1-\beta}.$$

Based on the construction of seeded extractors, we can also get

$$\lim_{m\to\infty}\frac{m}{k}=1.$$

As a result,

$$\eta=\lim_{m\to\infty}\frac{m}{H_{\mathcal{R}}(X_m)}\geq 1-\beta.$$

This completes the proof. □

Although construction 6.3 has the same efficiency as the other constructions, when m is not large, it is less efficient than the other constructions because the Lempel-Ziv code does not always have the best performance when the input sequence is not long. But its advantage is that it can manage more general sources without accurate estimations. In the above theorem, the gap β represents how far the source $\mathcal{R}$ is from being stationary ergodic. In general, the efficiency loss introduced by the uncertainty of sources is a part that cannot be avoid.

Corollary 6.19. *Given a real source $\mathcal{R}$ such that there exists a stationary ergodic model $\mathcal{M}$ with $d(\mathcal{R},\mathcal{M})\leq\beta$, then as $\beta\to 0$, the efficiency of construction 6.3 is*

$$\eta\to 1.$$

It shows that as $\beta\to 0$, construction 6.3 reaches the Shannon's limit on efficiency.

Corollary 6.20. *Given a stationary ergodic source $\mathcal{R}$ (assume we do not know that it is stationary ergodic), for the expected input length of construction 6.3, we have*

$$\frac{1}{h(\mathcal{R})}\leq\lim_{m\to\infty}\frac{E[|X_m|]}{m}\leq\frac{1}{(1-\beta)h(\mathcal{R})},$$

where $h(\mathcal{R})$ is the entropy rate of $\mathcal{R}$.

6.7 Seedless Constructions

To simulate seeded constructions of variable-length extractors in randomized applications, we have to enumerate all possible assignments of the seed, hence, the computational complexity will be increased significantly. In real applications, we prefer seedless constructions rather than seeded constructions. It motivates us to study the seedless constructions of variable-length extractors in this section.

6.7.1 An Independent Source

Let us first consider a simple independent source described in the introduction. This type of sources have been widely studied in seedless constructions of fixed-length extractors.

Example 6.6. *Let $x_1x_2... \in \{0,1\}^*$ be an independent sequence generated from a source $\mathcal{R}$ such that*

$$P[x_i = 1] \in [0.9, 0.91] \quad \forall i \geq i.$$

□

We see that the existing methods for generating random bits from ideal sources (like biased coins or Markov chains) cannot be applied here, since the probability of each bit is slightly unpredictable. Some seedless extractors have been developed for extracting randomness from such sources. In particular, there exists seedless extractors which are able to extract as many as $H_{\min}(X)$ random bits from a independent random sequence X asymptotically. In order to extract m random bits in the above example, it needs to read $\frac{m}{\log_2 \frac{1}{0.91}}$ input bits as $m \to \infty$. In this case, the entropy of the input sequence is in

$$[H(0.9)\frac{m}{\log_2 \frac{1}{0.91}}, H(0.91)\frac{m}{\log_2 \frac{1}{0.91}}].$$

From which, we can get the efficiency of an optimal fixed-length extractor, which is

$$\eta_{fixed} \in [0.2901, 0.3117],$$

i.e., about only 0.3 of the input entropy is used for generating random bits, which is far from optimal

In the above example, we let $\mathcal{M}$ be a biased coin model with probability $p = 0.9072$. In this case,

$$\beta \leq d(\mathcal{R}, \mathcal{M}) = 0.0315.$$

According to the constructions in the previous sections, there exists seeded variable-length extractors such that their efficiencies are

$$\eta_{variable} \in [1 - \beta, 1] \subseteq [0.9685, 1],$$

which are near Shannon's limit.

Based on the fact that the source is independent, we can eliminate the requirement of truly random bits as the seed, hence, we have seedless variable-length extractors. To construct a seedless variable-length extractor, we first apply a seedless fixed-length extractor E_1 (which may not be very efficient) to extract a random sequence of length d from input bits. Using this random sequence as the seed, we continue to apply a seeded variable-length extractors E_2 to extract m almost-random bits from extra input bits. So seedless variable-length extractors can be constructed as cascades of seedless fixed-length extractors and seeded variable-length extractors. Since the input length of E_1 is much shorter (it is ignorable) than the input length of E_2, the efficiency of the resulting seedless extractor, i.e., $E = E_2 \bigotimes E_1$, is dominated by the efficiency of E_2. So the efficiency of the seedless extractor E is in $[0.9685, 1]$, which is very close to the optimality.

This example demonstrates a simple construction of seedless variable-length extractors for independent sources, and it shows the significant performance gain of variable-length extractors compared to fixed-length extractors.

6.7.2 Generalized Sources

Here we consider a generalization of independent processes. Given a system, we use λ_i denote the complete system status at time i. For example, in a system that generates thermal noise, the system status can include the value of the noise signal, the temperature, the environmental effects, etc. Usually, the evolution of such a system has a Markov property, namely,

$$P[\lambda_{i+1}, \lambda_{i+2}, ...|\lambda_i, \lambda_{i-1}, ..., \lambda_1] = P[\lambda_{i+1}, \lambda_{i+2}, ...|\lambda_i],$$

for all $i \geq 1$. Let $X = x_1x_2... \in \{0,1\}^n$ be the binary sequence generated from this system, then for any $1 < k < n-1$,

$$P[X_1^{k-1}, X_{k+1}^n|\lambda_k] = P[X_1^{k-1}|\lambda_k]P[X_{k+1}^n|\lambda_k], \tag{6.8}$$

where $X_a^b = x_a x_{a+1}...x_b$. In some sense, the source X that we consider is a hidden Markov process, but the number of hidden states can be infinite (λ_i can be discrete or continuous).

Example 6.7. *One example of the above sources is the one studied in [63], called a space s source. A space s source is basically a source generated by a width 2^s branching program. At each step, the state of the process generating the source is in one of 2^s states, and the bit generated is a function of the current state. Unlike perfect Markov chains, the transition probabilities can be different at each step. In this example, the system status λ_i is the content of space s at time i, that is, one of the 2^s states, and $x_i \in \{0,1\}$ is a function of λ_i.*

Space s sources are very general in that most other classes of sources that have been considered previously can be computed with a small amount of space [63]. The model that we consider, as described by (6.8), is a natural generalization of space s sources. This model has a very nice feature: from such a source, we can get a group of sequences conditionally independent of each other. Namely, given system statues at some time points

$$[\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(\gamma)}] = [\lambda_a, \lambda_{2a}, ..., \lambda_{\gamma a}],$$

the subsequences

$$[X^{(1)}, X^{(2)}, ..., X^{(\gamma)}, X^{(\gamma+1)}] = [X_1^{a-1}, X_{a+1}^{2a-1}, ..., X_{(\gamma-1)a+1}^{\gamma a-1}, X_{\gamma a+1}^{\infty}]$$

are conditionally independent of each other. Based on this condition, we have the following seedless construction of variable-length extractors.

Construction 6.4. *Given a source $\mathcal{R}$ described by (6.8), we can construct a seedless variable-length extractor E in the following way:*

1. *Suppose that*

$$H_{\min}(X^{(i)}|\lambda^{(i)}, \lambda^{(i+1)}) \geq k_d, \forall 0 \leq i \leq \gamma.$$

   We construct a γ-source extractor [95] $E_1 : [\{0,1\}^{a-1}]^{\gamma} \to \{0,1\}^d$ such that if each source has min-entropy k_d, it can extract d almost-random bits which are ϵ_1-close to the uniform distribution on $\{0,1\}^d$.

2. *We construct a seeded variable-length extractor $E_2 : S_p \times \{0,1\}^d \to \{0,1\}^m$ such that with condition on $\lambda^{(\gamma)}$, it can extract m almost-random bits from $X^{(\gamma+1)}$ and these m almost-random bits are ϵ_2-close to the uniform distribution on $\{0,1\}^m$ if the seed is truly random.*

3. *The seedless variable-length extractor E is a cascade of E_1 and E_2: Let*

$$D = E_1(X^{(1)}, X^{(2)}, ..., X^{(\gamma)}),$$

   then we apply D as the seed of E_2 to generate m almost-random bits from $X^{(\gamma+1)}$; that is,

$$E(X) = E_2(X^{(\gamma+1)}, E_1(X^{(1)}, X^{(2)}, ..., X^{(\gamma)})).$$

For this construction, we have the following theorems.

Theorem 6.21. *In construction 6.4, the m almost-random bits generated by the seedless variable-*

length extractor E are $(\epsilon_1 + \epsilon_2)$-close to the uniform distribution on $\{0,1\}^m$.

Proof. According to the construction, we can let the parameter $a = |X^{(i)}| + 1$ with $1 \leq i \leq \gamma$ be large enough, so given $\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(\gamma)}$,

$$\|D - U_d\| \leq \epsilon_1.$$

Let X_m be the input sequence of E_2 that read from $X^{(\gamma+1)}$, then given $\lambda^{(\gamma)}$, we have

$$\|E_2(X_m, U_d) - U_m\| \leq \epsilon_2.$$

From the two inequalities above, given $\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(\gamma)}$, we have

$$\|E_2(X_m, D) - U_m\| \leq \epsilon_1 + \epsilon_2.$$

Since it is true for any assignments of $\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(\gamma)}$, we can get

$$\begin{aligned} \|E_2(X_m, D) - U_m\| &= \sum_{\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(\gamma)}} P[\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(\gamma)}](\epsilon_1 + \epsilon_2) \\ &\leq \epsilon_1 + \epsilon_2. \end{aligned}$$

Hence, the m almost-random bits extracted by E is also $(\epsilon_1 + \epsilon_2)$-close to U_m. □

In the following theorem, we show that the seedless variable-length extractor E has the efficiency as the seeded variable-length extractor E_2.

Theorem 6.22. *In construction 6.4, suppose that*

$$H_{\min}(X^{(i)}|\lambda^{(i)}, \lambda^{(i+1)}) = \Theta(|X^{(i)}|), \forall 0 \leq i \leq \gamma.$$

Let η_E denote the efficiency of the resulting seedless variable-length extractor E, and let η_{E_2} denote

the efficiency of the E_2, then

$$\eta_E = \eta_{E_2}.$$

Proof. According to the construction of E_1, we can get that

$$d = \Theta(a),$$

where $a = |X^{(i)}| + 1$ for $1 \leq i \leq \gamma$.

If ϵ_2 is a constant, then

$$d = O(\log m) = o(m).$$

As a result,

$$\lim_{m\to\infty} \frac{a\gamma}{m} = 0.$$

Let H denote the entropy of the input sequence of E_2, then $\eta_{E_2} = \lim_{m\to\infty} \frac{m}{H}$, and

$$\lim_{m\to\infty} \frac{m}{a\gamma + H} \leq \eta_E \leq \lim_{m\to\infty} \frac{m}{H}.$$

Hence, $\eta_E = \eta_{E_2}$. □

The theorem above shows that the efficiency of seedless variable-length extractors can be very close to optimality. For many sources, such as biased coins with noise, or Markov chains with noise, the existing algorithms for ideal sources (e.g., perfect biased coins or perfect Markov chains) cannot generate high-quality random bits from them. At the same time, the traditional approaches of fixed-length extractors are not very efficient. The gap between their efficiency and the optimality is determined by the bias of the source. Seedless variable-length extractors take the advantages of both, as a result, they can approach the information-theoretic upper bound on efficiency while being capable to combat noise in the sources.

6.8 Conclusion and Discussion

In this chapter, we introduced the concept of the variable-length extractors, namely, those extractors with variable input length and fixed output length. Variable-length extractors are generalizations of the existing algorithms for ideal sources to manage general stochastic processes. They are also improvements of traditional fixed-length extractors to fill the gap between min-entropy and entropy of the source on efficiency. The key idea of constructing variable-length extractors is to approximate the source using a simple model, which is a known process, a biased coin, or a stationary ergodic process. Depending on the model selected, we proposed and analyzed three seeded constructions of variable-length extractors. Their efficiency is lower bounded by $1-\beta$ and upper bounded by 1 (optimality), where $\beta(0 \leq \beta \leq 1)$ indicate the uncertainty of the real source. We also show that our constructions are asymptotically optimal, in the sense that one cannot find a construction whose efficiency is always strictly larger than $1-\beta$. In addition, we demonstrated how to construct seedless variable-length extractors by cascading seeded variable-length extractors with seedless fixed-length extractors. They can work for many (but not all) natural sources such as those based on noise signals.

There are certain connections between variable-length extractors and a whole family of variable-to-fixed length source codes [74, 84, 103, 114, 115, 117, 127]. With a variable-to-fixed length code, an infinite sequence is parsed into variable-length phases, chosen from some finite set $\mathcal{D}$ of phases. Each phase is then coded into a binary sequence of fixed length m. The set $\mathcal{D}$ of phases is complete, i.e., every infinite sequence has a prefix in $\mathcal{D}$. The key of constructing a good variable-to-fixed length source code is to find the best set $\mathcal{D}$ that consists of at last 2^m prefix-free phases and maximizes the expected phase length. As comparison, the key of constructing a variable-length extractor is to find the best input set S_p that consists of sequences with probability at most 2^{-k} for each and minimizes the expected sequence length. Although their goals are different, some common ideas can be used to construct both the phase set $\mathcal{D}$ and the input set S_p. For example, in [127], Visweswariah et al. defined the phase set $\mathcal{D}$ by $x^* \in \mathcal{D}$ if $P(x^*) \leq c$ and no prefix of x^* satisfies this property. The same idea is applied in our construction I. In [74, 114], the phase set

$\mathcal{D}$ is determined by the number of ones and zeros in the phase, so is our construction II. In some sense, an optimal variable-to-fixed length code can result in a fixed-length binary sequence whose min-entropy is close to its length. However, variable-to-fixed length source codes do not always work well in constructing variable-length extractors, because (1) the designing criteria are different and they may degrade the performance; (2) variable-to-fixed length source codes take both encoding and decoding in consideration, hence, they are more complex in computation than what we require (decoding is not necessary) for constructing variable-length extractors; and (3) the sources that we considered for variable-length extractors are unpredictable, which are more general than the ones considered in variable-to-fixed length source codes.

Part III

Stochastic System Synthesis

Chapter 7

Synthesis of Stochastic Switching Circuits

This chapter studies stochastic switching circuits, which are relay circuits that consist of stochastic switches called pswitches. It introduces new properties of stochastic switching circuits, including robustness, expressibility, and probability approximation.[1]

7.1 Introduction

In his master's thesis of 1938, Claude Shannon demonstrated how Boolean algebra can be used to synthesize and simplify relay circuits, establishing the foundation of modern digital circuit design [106]. Later, deterministic switches were replaced with probabilistic switches to make stochastic switching circuits, which were studied in [134]. There are a few features of stochastic switching circuits that make them very similar to neural systems. First, randomness is inherent in neural systems and it may play a crucial role in thinking and reasoning. Switching (and relaying) technique provides us a natural way of manipulating this randomness. Second, in a switching system, each switch can be treated as either a memory element or a control element for computing. This might enable creating an intelligent system where storage and computing are highly integrated. In this chapter, we study stochastic switching circuits from a basic starting point with focusing on probability synthesis. We consider two-terminal stochastic switching circuits, where each probabilistic switch, or *pswitch*, is

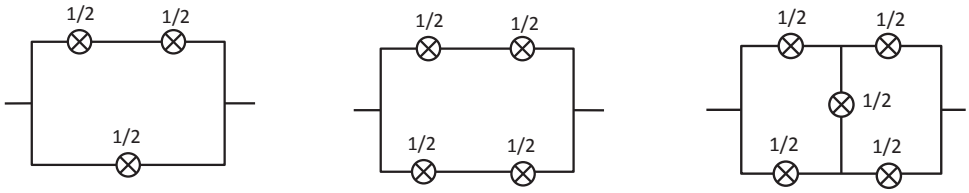
[1] Some of the results presented in this chapter have been previously published in [75] and [141].

closed with some probability chosen from a finite set of rational numbers, called a *pswitch set*. By selecting pswitches with different probabilities and composing them in appropriate ways, we can realize a variety of different closure probabilities.

Formally, for a two-terminal stochastic switching circuit C, the probabilities of pswitches are taken from a fixed pswitch set S, and all these pswitches are open or closed independently. We use $P(C)$ to denote the probability that the two terminals of C are connected, and call $P(C)$ the *closure probability* of C. Given a pswitch set S, a probability x can be *realized* if and only if there exists a circuit C such that $x = P(C)$. Based on the ways of composing pswitches, we have series-parallel (sp) circuits and non-series-parallel (non-sp) circuits. An sp circuit consists of either a single pswitch or two sp circuits connected in series or parallel, see the circuit in figure 7.1(a) and 7.1(b) as examples. The circuit in figure 7.1(c) is a non-sp circuit. A special type of sp circuits is called simple-series-parallel (ssp) circuits. An ssp circuit is either a single pswitch, or is built by taking an ssp circuit and adding another pswitch in either series or parallel. For example, the circuit in figure 7.1(a) is an ssp circuit but the one in figure 7.1(b) is not.

In this chapter, we first study the robustness of different stochastic switching circuits in the presence of small error perturbations. We assume that the probabilities of individual pswitches are taken from a fixed pswitch set with a given error allowance of ϵ; that is, the error probabilities of the pswitches are bounded by ϵ. We show that ssp circuits are robust to small error perturbations, but the error probability of a general sp circuit may be amplified by adding additional pswitches. These results might help us understand why local errors do not accumulate in a natural system, and how to enhance the robustness of a system when designing a circuit.

Next, we study the problem of synthesizing desired probabilities with stochastic switching circuits. We mainly focus on ssp circuits due to their robustness against small error perturbations. Two main questions are addressed: (1) *Expressibility*: Given the pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, ..., \frac{q-1}{q}\}$, where q is an integer, what kind of probabilities can be realized using stochastic switching circuits? And how many pswitches are sufficient to realize them? (2) *Approximation*: Given an arbitrary pswitch set S, how can we construct a stochastic switching circuit using as a few as possible pswitches, to



(a) ssp circuit. $P(C) = \frac{5}{8}$. (b) sp circuit, non-ssp. $P(C) = \frac{7}{16}$. (c) non-sp circuit. $P(C) = \frac{1}{2}$.

Figure 7.1. Examples of ssp, sp, and non-sp circuits.

get a good approximation of the desired probabilities?

The study of probability synthesis based on stochastic switching circuits has widespread applications. Recently, people found that DNA molecules can be constructed that closely approximate the dynamic behavior of arbitrary systems of coupled chemical reactions [108], which leads to the field of molecular computing [25]. In such systems, the quantities of molecules involved in a reaction are often surprisingly small, and the exact sequence of reactions is determined by chance [38]. Stochastic switching circuits provide a simple and powerful tool to manipulate stochasticity in molecular systems. Comparing with combinational logic circuits, stochastic switching circuits are easier to implement using molecular reactions. Another type of applications is probabilistic electrical systems without sophisticated computing components. In such systems, stochastic switching circuits have many advantages in generating desired probabilities, including its constructive simplicity, robustness, and low cost.

The remainder of this chapter is organized as follows: Section 7.2 describes related work and introduces some existing results on stochastic switching circuits. In section 7.3, we analyze the robustness of different kinds of stochastic switching circuits. Then we discuss the expressibility of stochastic switching circuits in section 7.4 and probability approximation in section 7.5, followed by the conclusion in section 7.6.

7.2 Related Works and Preliminaries

There are a number of studies related to the problem of generating desired distributions from the algorithmic perspective. This problem dates back to von Neumann [128], who considered of simulating an unbiased coin using a biased coin with unknown probability. Later, Elias [33] improved this algorithm such that the expected number of unbiased random bits generated per coin toss is asymptotically equal to the entropy of the biased coin. On the other hand, people have considered the case that the probability distribution of the tossed coin is known. Knuth and Yao [71] have given a procedure to generate an arbitrary probability using an unbiased coin. Han and Hoshi [52] have demonstrated how to generate an arbitrary probability using a general M-sided biased coin. All these works aim to efficiently convert one distribution to another. However, they require computing models and may not be applicable for some simple or distributed electrical/molecular systems.

There are a number of studies focusing on synthesizing a simple physical device to generate desired probabilities. Gill [44] [45] discussed the problem of generating rational probabilities using a sequential state machine. Motivated by neural computation, Jeavons et al. provided an algorithm to generate binary sequences with probability $\frac{a}{q^n}$ from a set of stochastic binary sequences with probabilities in $\{\frac{1}{q}, \frac{2}{q}, ..., \frac{q-1}{q}\}$ [57]. Their method can be implemented using the concept of linear feedback shift registers. Recently, inspired by PCMOS technology [22], Qian et al. considered the synthesis of decimal probabilities using combinational logic [92]. They have considered three different scenarios, depending on whether the given probabilities can be duplicated, and whether there is freedom to choose the probabilities. In contact to the foregoing contributions, we consider the properties and probability synthesis of stochastic switching circuits. Our approach is orthogonal and complementary to that of Qian and Riedel, which is based on combinational logic. Generally, each switching circuit can be equivalently expressed by a combinational logic circuit. All the constructive methods of stochastic switching circuits in this chapter can be directly applied to probabilistic combinational logic circuits.

In the rest of this section, we introduce the original work that started the study on stochastic switching circuits (Wilhelm and Bruck [134]). Similar to resistor circuits [77], connecting one termi-

nal of a switching circuit C_1 (where $P(C_1) = p_1$) to one terminal of a circuit C_2 (where $P(C_2) = p_2$) places them in series. The resulting circuit is closed if and only if both of C_1 and C_2 are closed, so the probability of the resulting circuit is

$$p_{\text{series}} = p_1 \cdot p_2.$$

Connecting both terminals of C_1 and C_2 together places the circuits in parallel. The resulting circuit is closed if and only if either C_1 or C_2 is closed, so the probability of the resulting circuit is

$$p_{\text{parallel}} = 1 - (1 - p_1)(1 - p_2) = p_1 + p_2 - p_1 p_2.$$

Based on these rules, we can calculate the probability of any given ssp or sp circuit. For example, the probability of the circuit in figure 7.1(a) is

$$p_{(a)} = \left(\frac{1}{2} \cdot \frac{1}{2}\right) + \frac{1}{2} - \left(\frac{1}{2} \cdot \frac{1}{2}\right) \frac{1}{2} = \frac{5}{8},$$

and the probability of the circuit in figure 7.1(b) is

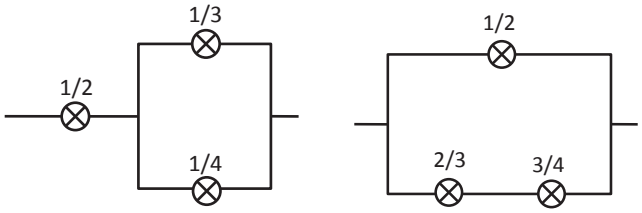
$$p_{(b)} = \left(\frac{1}{2} \cdot \frac{1}{2}\right) + \left(\frac{1}{2} \cdot \frac{1}{2}\right) - \left(\frac{1}{2} \cdot \frac{1}{2}\right) \left(\frac{1}{2} \cdot \frac{1}{2}\right) = \frac{7}{16}.$$

Let us consider the non-sp circuit in figure 7.1(c). In this circuit, we call the pswitch in the middle a 'bridge'. If the bridge is closed, the circuit has a closure probability of $\frac{9}{16}$. If the bridge is open, the circuit has a closure probability of $\frac{7}{16}$. Since the bridge is closed with probability $\frac{1}{2}$, the overall probability of the circuit is

$$p_{(c)} = \frac{1}{2} \cdot \frac{9}{16} + \frac{1}{2} \cdot \frac{7}{16} = \frac{1}{2}.$$

An important and interesting question is that if S is uniform, i.e., $S = \{\frac{1}{q}, \frac{2}{q}, ..., \frac{q-1}{q}\}$ for some q, what kind of probabilities can be realized using stochastic switching circuits? In [134], Wilhelm

and Bruck proposed an optimal algorithm (called B-Algorithm) to realize all rational probabilities of the form $\frac{a}{2^n}$ with $0 < a < 2^n$, using an ssp circuit when $S = \{\frac{1}{2}\}$. In their algorithm, at most n pswitches are used, which is optimal. They also proved that given the pswitch set $S = \{\frac{1}{3}, \frac{2}{3}\}$, all rational probabilities $\frac{a}{3^n}$ with $0 < a < 3^n$ can be realized by an ssp circuit with at most n pswitches; given the pswitch set $S = \{\frac{1}{4}, \frac{2}{4}, \frac{3}{4}\}$, all rational probabilities $\frac{a}{4^n}$ with $0 < a < 4^n$ can be realized by an ssp circuit with at most $2n - 1$ pswitches.



(a) Initial circuit. $P = \frac{1}{4}$. (b) The dual. $P = \frac{3}{4}$.

Figure 7.2. A circuit and its dual.

Wilhelm and Bruck also demonstrated the concept of *duality* in sp circuits. The dual of a single pswitch of probability p appearing in series is the corresponding pswitch of probability $1-p$ appearing in parallel. Similarly, the dual of a pswitch of probability p appearing in parallel is a pswitch of probability $1 - p$ appearing in series. For example, in figure 7.2, the circuit in (b) is the dual of the circuit in (a), and vice versa. It can be proved that dual circuits satisfy the following relation:

Theorem 7.1 (Duality Theorem [134])**.** *For a stochastic series-parallel circuit C and its dual $\overline{C}$, we have*

$$P(C) + P(\overline{C}) = 1,$$

where $P(C)$ is the probability of circuit C and $P(\overline{C})$ is the probability of circuit $\overline{C}$.

7.3 Robustness

In this section, we analyze the robustness of different kinds of stochastic switching circuits, where the probabilities of individual pswitches are taken from a fixed pswitch set, but given an error

allowance of ϵ; i.e., the error probabilities of the pswitches are bounded by ϵ. For a stochastic circuit with multiple pswitches, the *error probability* of the circuit is the absolute difference between the probability that the circuit is closed when error probabilities of pswitches are included, and the probability that the circuit is closed when error probabilities are omitted. We show that ssp circuits are robust to small error perturbations, but the error probability of a general sp circuit may be amplified with additional pswitches.

7.3.1 Robustness of ssp Circuits

Here, we analyze the susceptibility of ssp circuits to small error perturbations in individual pswitches. Based on our assumption, instead of assigning a pswitch a probability of p, the pswitch may be assigned a probability between $p - \epsilon$ and $p + \epsilon$, where ϵ is a fixed error allowance.

Theorem 7.2 (Robustness of ssp circuits)**.** *Given a pswitch set S, if the error probability of each pswitch is bounded by ϵ, then the total error probability of an ssp circuit is bounded by*

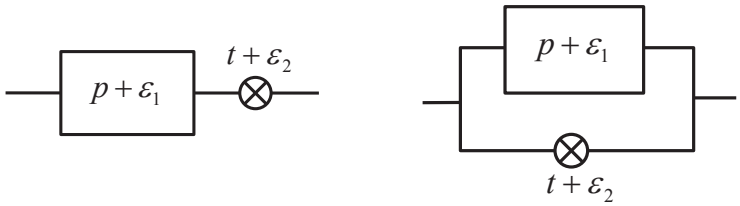
$$\frac{\epsilon}{\min\{\min\{S\}, 1 - \max\{S\}\}}.$$

Proof. We induct on the number of pswitches. If we have just one pswitch, the result is trivial. Suppose the result holds for n pswitches, and note that for an ssp circuit with $n+1$ pswitches, the last pswitch will either be added in series or in parallel with the first n pswitches. By the induction hypothesis, the circuit constructed from the first n pswitches has probability $p + \epsilon_1$ of being closed, where ϵ_1 is the error probability introduced by the first n pswitches and $|\epsilon_1| \leq \frac{\epsilon}{\min\{\min\{S\}, 1-\max\{S\}\}}$. The $(n+1)$st pswitch has probability $t + \epsilon_2$ of being closed, where $t \in S$ and $|\epsilon_2| \leq \epsilon$.

If the $(n+1)$st pswitch is added in series, see figure 7.3(a), then the new circuit (with errors) has probability

$$(p + \epsilon_1)(t + \epsilon_2) = tp + \epsilon_2(p + \epsilon_1) + t\epsilon_1$$

of being closed. Without considering the error probability of each pswitch, the probability of the new circuit is tp. Hence, the overall error probability of the circuit is $e_1 = \epsilon_2(p + \epsilon_1) + t\epsilon_1$. By the



(a) The last pswitch is added in series.

(b) The last pswitch is added in parallel.

Figure 7.3. Robustness of ssp circuits.

triangle inequality and the induction hypothesis,

$$\begin{aligned} |e_1| &\leq |\epsilon_2||(p+\epsilon_1)| + t|\epsilon_1| \leq |\epsilon_2| + t|\epsilon_1| \\ &\leq \left(\frac{t}{\min\{\min\{S\}, 1-\max\{S\}\}} + 1\right)\epsilon \\ &\leq \frac{\min\{\min\{S\}, 1-\max\{S\}\} + \max\{S\}}{\min\{\min\{S\}, 1-\max\{S\}\}} \cdot \epsilon \\ &\leq \frac{\epsilon}{\min\{\min\{S\}, 1-\max\{S\}\}}, \end{aligned}$$

completing the induction.

Similarly, if the $(n+1)$st pswitch is added in parallel, see figure 7.3(b), then the new circuit (with errors) has probability

$$(p+\epsilon_1) + (t+\epsilon_2) - (p+\epsilon_1)(t+\epsilon_2) = (p+t-tp) + \epsilon_1(1-t) + \epsilon_2(1-p-\epsilon_1)$$

of being closed. Without considering the error probability of each pswitch, the probability that the circuit is closed is $p+t-tp$. Hence, the overall error probability of the circuit with $n+1$ pswitches is $e_2 = \epsilon_1(1-t) + \epsilon_2(1-p-\epsilon_1)$. Again using the induction hypothesis and the triangle inequality,

we have

$$\begin{aligned} |e_2| &\leq |\epsilon_2||(1-p-\epsilon_1)| + (1-t)|\epsilon_1| \leq |\epsilon_2| + (1-t)|\epsilon_1| \\ &\leq \left(\frac{1-t}{\min\{\min\{S\}, 1-\max\{S\}\}} + 1\right)\epsilon \\ &\leq \frac{\min\{\min\{S\}, 1-\max\{S\}\} + 1 - \min\{S\}}{\min\{\min\{S\}, 1-\max\{S\}\}} \cdot \epsilon \\ &\leq \frac{\epsilon}{\min\{\min\{S\}, 1-\max\{S\}\}}. \end{aligned}$$

This completes the proof. □

The theorem above implies that ssp circuits are robust to small error perturbations: no matter how big the circuit is, the error probability of an ssp circuit will be well bounded by a constant times ϵ. Let us consider a case that $S = \{\frac{1}{2}\}$. In this case, the overall error probability of any ssp circuit is bounded by 2ϵ if each pswitch is given an error allowance of ϵ.

7.3.2 Robustness of sp Circuits

We have proved that for a given pswitch set S, the overall error probability of an ssp circuit is well bounded. We want to know whether this property holds for all sp circuits. Unfortunately, we show that as the number of pswitches increases, the overall error probability of an sp circuit may also increase. In this subsection, we will give the upper bound and lower bound for the error probabilities of sp circuits.

Theorem 7.3 (Lower bound for sp circuits)**.** *Given a pswitch set S, if the error probability of each pswitch is ϵ (where $\epsilon \to 0$), then there exists an sp circuit of size n with overall error probability $O(\log n)\epsilon$.*

Proof. Suppose $p \in S$, and without loss of generality, assume $\epsilon > 0$. We construct an sp circuit as shown in figure 7.4, by connecting $a+1$ strings of pswitches in parallel. Among these strings, we have a strings of b pswitches and one string of $n - ab$ pswitches, and all pswitches have probability

p. Now, we let a and b satisfy the following relation:

$$a = \left\lceil \frac{n}{b} \right\rceil - 1, a = \left\lfloor (\frac{1}{p})^b \right\rfloor .$$

Without considering pswitch errors, the probability of the circuit is

$$p_1 = 1 - (1 - p^b)^a (1 - p^{n-ab}).$$

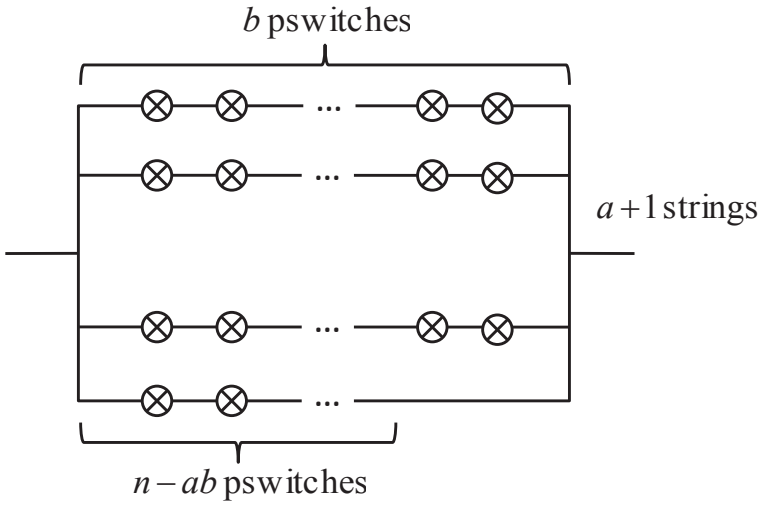


Figure 7.4. The construction of an sp circuit.

Suppose we introduce an error of ϵ to each pswitch, such that the probability of each pswitch is $p + \epsilon$ (assume $\epsilon > 0$). Then the probability of the circuit is

$$p_2(\epsilon) = 1 - (1 - (p + \epsilon)^b)^a (1 - (p + \epsilon)^{n-ab}),$$

where $p_2(0) = p_1$.

Assuming n is large enough, we have the following error probability for the circuit:

$$\begin{aligned}
e_1 &= p_2(\epsilon) - p_1 \\
&\simeq p_2'(\epsilon)\epsilon \\
&\simeq -[(1-(p+\epsilon)^b)^a(1-(p+\epsilon)^{n-ab})]'\epsilon \\
&\simeq -[e^{-a(p+\epsilon)^b}(1-(p+\epsilon)^{n-ab})]'\epsilon \\
&\simeq e^{-a(p+\epsilon)^b}ab(p+\epsilon)^{b-1}(1-(p+\epsilon)^{n-ab})\epsilon \\
&\quad +e^{-a(p+\epsilon)^b}(n-ab)(p+\epsilon)^{n-ab-1}\epsilon \\
&\simeq [e^{-ap^b}abp^{b-1}(1-p^{n-ab}) + e^{-ap^b}(n-ab)p^{n-ab-1}]\epsilon \\
&\simeq [e^{-1}\frac{b}{p}(1-p^{n-ab}) + e^{-1}(n-ab)p^{n-ab-1}]\epsilon.
\end{aligned}$$

So when n is large enough, we have

$$e^{-1}\frac{1-p}{p}b\epsilon \le |e_1| \le e^{-1}\frac{1}{p}b\epsilon.$$

Since $b\lfloor(\frac{1}{p})^b\rfloor < n \le b(\lfloor(\frac{1}{p})^b\rfloor + 1)$ for large n, we have

$$b \sim \frac{\log n}{\log\frac{1}{p}} - \frac{\log\log n}{\log\frac{1}{p}} + \frac{\log\log\frac{1}{p}}{\log\frac{1}{p}} \sim \frac{\log n}{\log\frac{1}{p}}.$$

Finally, we have $|e_1| \sim O(\log n)\epsilon$, completing the proof. □

In the following theorem, we will give the upper bound for the error probabilities of sp circuits.

Theorem 7.4 (Upper bound for sp circuits)**.** *Given an sp circuit with n pswitches taken from a finite pswitch set S, if each pswitch has error probability bounded by ϵ, then the total error probability of the circuit is bounded by $c\sqrt{n}\epsilon$, where $c = \max_{t\in S}\frac{1}{\sqrt{t(1-t)}}$ is a constant.*

Proof. Assume x is a pswitch in a stochastic circuit C, and the actual probability of x is $t_x + \epsilon_x$, where ϵ_x is the error part such that $|\epsilon_x| \le \epsilon$. Let $P(C|x=1)$ denote the probability of circuit C

when x is closed, and let $P(C|x=0)$ denote the probability of C when x is open.

Without considering the error probability of x, the probability of circuit C can be written as

$$P_x(C) = t_x P(C|x=1) + (1-t_x)P(C|x=0).$$

Considering the error part of x, we have

$$P(C) = (t_x+\epsilon_x)P(C|x=1) + (1-t_x-\epsilon_x)P(C|x=0).$$

In order to prove the theorem, we define a term called the *error contribution*. In a circuit C, the error contribution of pswitch x is defined as

$$e_x(C) = |P(C) - P_x(C)| = \epsilon_x |P(C|x=1) - P(C|x=0)|$$

$$\leq \epsilon(P(C|x=1) - P(C|x=0)).$$

In the rest of the proof, we have two steps.

(1) In the first step, we show that given an sp circuit with size n, there exists at least one pswitch such that its error contribution is bounded by $\frac{c\sqrt{(1-P)P}}{\sqrt{n}}\epsilon$, where P is the probability of the sp circuit and $c = \max_{t\in S} \frac{1}{\sqrt{t(1-t)}}$.

We induct on the number of pswitches. If the circuit has only one pswitch, the result is trivial. Suppose the result holds for k pswitches for all $k < n$. We need to prove that the result holds for any sp circuit C with n pswitches.

Suppose circuit C is constructed by connecting two sp circuits C_1 and C_2 in series, where C_1 has n_1 pswitches and probability P_1, and C_2 has n_2 pswitches and probability P_2. Note that $n_1+n_2 = n$ and $n_1 < n, n_2 < n$.

By the induction hypothesis, circuit C_1 contains a pswitch x_1 with error contribution

$$e_{x_1}(C_1) \leq \frac{c\sqrt{(1-P_1)P_1}}{\sqrt{n_1}}\epsilon.$$

In circuit C, the error contribution of pswitch x_1 is

$$e_{x_1}(C) = |P(C) - P_{x_1}(C)| = P_2|P(C_1) - P_{x_1}(C_1)| = P_2 e_{x_1}(C_1).$$

Similarly, in the circuit C_2, there exists a pswitch x_2 such that the error contribution of x_2 is

$$e_{x_2}(C_2) \leq \frac{c\sqrt{(1-P_2)P_2}}{\sqrt{n_2}}\epsilon,$$

and the error contribution of x_2 to circuit C is

$$e_{x_2}(C) = P_1 e_{x_2}(C_2).$$

Since the circuit C is constructed by connecting circuits C_1 and C_2 in series, the probability of circuit C is $P = P_1P_2$. Thus, we only need to prove that either $e_{x_1}(C)$ or $e_{x_2}(C)$ is bounded by

$$\frac{c\sqrt{(1-P_1P_2)P_1P_2}}{\sqrt{n_1+n_2}}\epsilon,$$

This can be proved by contradiction as follows.

Assume both $e_{x_1}(C)$ and $e_{x_2}(C)$ are larger than $\frac{c\sqrt{(1-P_1P_2)P_1P_2}}{\sqrt{n_1+n_2}}\epsilon$. Then we have

$$P_2\frac{c\sqrt{(1-P_1)P_1}}{\sqrt{n_1}} > \frac{c\sqrt{(1-P_1P_2)P_1P_2}}{\sqrt{n_1+n_2}}$$

and

$$P_1\frac{c\sqrt{(1-P_2)P_2}}{\sqrt{n_2}} > \frac{c\sqrt{(1-P_1P_2)P_1P_2}}{\sqrt{n_1+n_2}},$$

which can be simplified as

$$\frac{n_1}{n_1+n_2} < \frac{(1-P_1)P_2}{1-P_1P_2}$$

and

$$\frac{n_2}{n_1+n_2} < \frac{(1-P_2)P_1}{1-P_1P_2}.$$

Adding the two inequalities yields

$$P_1 + P_2 - 1 - P_1P_2 = -(1-P_1)(1-P_2) > 0,$$

which is a contradiction. So we conclude that at least one of $e_{x_1}(C)$ and $e_{x_2}(C)$ is bounded by $\frac{c\sqrt{(1-P_1P_2)P_1P_2}}{\sqrt{n_1+n_2}}\epsilon$ when C is constructed by connecting two sp circuits in series. If the circuit C is constructed by connecting two sp circuits in parallel, using a similar argument, we can get the same conclusion.

Finally, we get that given an sp circuit with size n, there exists at least one pswitch such that its error contribution is bounded by $\frac{c\sqrt{(1-P)P}}{\sqrt{n}}\epsilon$.

(2) In the second step, we prove the theorem based on the result above.

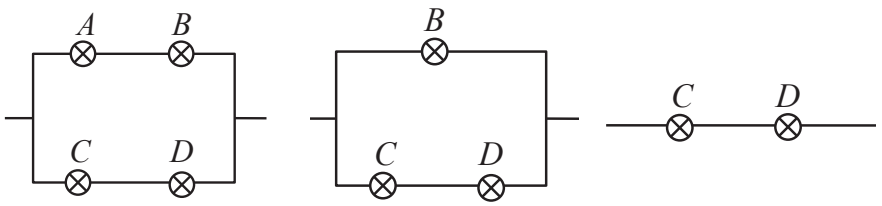
We again induct on the number of pswitches. If we have less than three pswitches, the result is trivial. Suppose the result holds for any sp circuit with $n \geq 2$ pswitches; we want to prove that the result also holds for any circuit with $n+1$ pswitches.

Based on the result in the first step, we know that given an sp circuit C with $n+1$ pswitches, there exists a pswitch x with error contribution bounded by $\frac{c}{2\sqrt{n+1}}\epsilon$.

By keeping pswitch x closed, we obtain an sp circuit D_1 with at most n pswitches. Please see figure 7.5(a)(b) as an example. Without considering pswitch errors, D_1 is closed with probability p_1; considering all pswitch errors, D_1 is closed with probability q_1. According to our assumption, we have

$$e_1 = |q_1 - p_1| \leq c\sqrt{n}\epsilon.$$

By keeping pswitch x open, we obtain an sp circuit D_2 with at most n pswitches. Please see



(a) Circuit C. (b) D_1, A closed. (c) D_2, A open.

Figure 7.5. An illustration of keeping a pswitch A closed or open in an sp circuit C.

figure 7.5(a)(c) as an example. Without considering pswitch errors, D_2 is closed with probability p_2; considering all pswitch errors, D_2 is closed with probability q_2. According to our assumption, we have

$$e_2 = |q_2 - p_2| \leq c\sqrt{n}\epsilon.$$

For the initial sp circuit C with $n+1$ pswitches, without considering pswitch errors, the overall probability of the circuit is given by

$$t_x p_1 + (1 - t_x)p_2,$$

where t_x is the probability of pswitch x.

Considering all pswitch errors, the overall probability of the circuit is

$$(t_x + \epsilon_x)q_1 + (1 - t_x - \epsilon_x)q_2.$$

We know that the error contribution of pswitch x to the circuit C is

$$e_x(C) = \epsilon_x|q_2 - q_1| \leq \frac{c}{2\sqrt{n+1}}\epsilon.$$

Then by the triangle inequality, we can get the error probability of the circuit C:

$$\begin{aligned} e &= |(t_x+\epsilon_x)q_1 + (1-t_x-\epsilon_x)q_2 - (t_x p_1 + (1-t_x)p_2)| \\ &\leq t_x|q_1-p_1| + (1-t_x)|q_2-p_2| + \epsilon_x|q_2-q_1| \\ &\leq \frac{c\sqrt{n(n+1)}+\frac{c}{2}}{\sqrt{n+1}}\epsilon \\ &\leq c\frac{(n+\frac{1}{2})+\frac{1}{2}}{\sqrt{n+1}}\epsilon \\ &= c\sqrt{n+1}\epsilon. \end{aligned}$$

This finishes the induction. □

7.3.3 Robustness of Non-sp Circuits

Here we extend our discussion to the case of general stochastic switching circuits. We have the following theorem, which clearly holds for sp and ssp circuits:

Theorem 7.5 (Upper bound for general circuits)**.** *Given a general stochastic switching circuit with n pswitches taken from a finite pswitch set S, if each pswitch has error probability bounded by ϵ, then the total probability of the circuit is bounded by $n\epsilon$.*

Proof. We first index all the pswitches in the circuit C as $x_1, x_2, ..., x_n$, see figure 7.6 as an example.

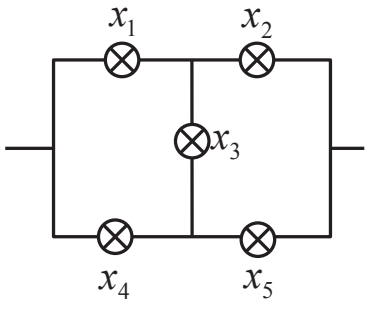


Figure 7.6. An example of a general stochastic switching circuit.

Let $t_i+\epsilon_i$ be the probability that x_i is closed, where ϵ_i is the error part such that $|\epsilon_i| \leq \epsilon$. Let $P^{(k)}$ denote the probability that C is closed when we only take into account the errors of $x_1, x_2, ..., x_k$,

i.e.,

$$P^{(k)} = P(t_1 + \epsilon_1, \ldots, t_k + \epsilon_k, t_{k+1}, \ldots, t_n),$$

where $P(a_1, a_2, ..., a_n)$ indicates the probability of C if x_i is closed with probability a_i for all $1 \leq i \leq n$.

The overall error probability of the circuit C can then be written as

$$\begin{aligned} e &= P^{(n)} - P(0) \\ &= (P^{(n)} - P^{(n-1)}) + (P^{(n-1)} - P^{(n-2)}) + \cdots + (P^{(1)} - P^{(0)}). \end{aligned}$$

Now, we prove that $|P^{(k)} - P^{(k-1)}| \leq \epsilon$ for all $1 \leq k \leq n$

$$\begin{aligned} & |P^{(k)} - P^{(k-1)}| \\ = \; & |P(t_1 + \epsilon_1, \ldots, t_k + \epsilon_k, t_{k+1}, \ldots, t_n) - P(t_1 + \epsilon_1, \ldots, t_{k-1} + \epsilon_{k-1}, t_k, \ldots, t_n)| \\ = \; & |(t_k + \epsilon_k)P(t_1 + \epsilon_1, \ldots, 1, t_{k+1}, \ldots, t_n) \\ & +(1 - t_k - \epsilon_k)P(t_1 + \epsilon_1, \ldots, 0, t_{k+1}, \ldots, t_n) \\ & -t_k P(t_1 + \epsilon_1, \ldots, t_{k-1} + \epsilon_{k-1}, 1, \ldots, t_n) \\ & -(1 - t_k)P(t_1 + \epsilon_1, \ldots, t_{k-1} + \epsilon_{k-1}, 0, \ldots, t_n)| \\ = \; & |\epsilon_k[P(t_1 + \epsilon_1, \ldots, 1, t_{k+1}, \ldots, t_n) - P(t_1 + \epsilon_1, \ldots, 0, t_{k+1}, \ldots, t_n)]| \\ \leq \; & \epsilon. \end{aligned}$$

Therefore, we have

$$e \leq \sum_{k=1}^{n} |P^{(k)} - P^{(k-1)}| \leq n\epsilon,$$

as we wanted. □

Note that in most of cases, the actual error probability of a circuit is much smaller than $n\epsilon$ when n is large. However, $n\epsilon$ is still achievable in the following case: by placing n pswitches with

Table 7.1. The expressibility of stochastic switching circuits

$S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$	Can all $\frac{a}{q^n}$ be realized?	upper bound of circuit size
q is even	yes, ssp circuit	$\lceil \log_2 q \rceil (n-1)+1$
q is an odd multiple of 3	yes, ssp circuit	$\lceil \log_3 q \rceil (n-1)+1$
q is a prime number larger than 3	no, not by sp circuits	–
other values of q	open problem	–

probability $p-\epsilon$ in series, where $\epsilon \to \infty$, we can get a circuit whose probability is

$$(p-\epsilon)^n \approx p^n - np^{n-1}\epsilon.$$

Without considering the errors, the probability of the circuit is p^n, so the overall error is

$$n \cdot p^{n-1}\epsilon.$$

Choosing p sufficiently close to 1, we can make the error probability of the circuit arbitrarily close to $n\epsilon$.

7.4 Expressibility

In the previous section, we showed that ssp circuits are robust against noise. This property is important in natural systems and useful in engineering system design, because the local error of a system should not be amplified. In this section, we consider another property of stochastic switching circuits, called expressibility. Namely, given a pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, ..., \frac{q-1}{q}\}$ for some integer q, the questions we ask are: What kinds of probabilities can be realized using stochastic switching circuits (or only ssp circuits)? How many pswitches are sufficient? Wilhelm and Bruck [134] proved that if $q = 2$ or $q = 3$, all rational $\frac{a}{q^n}$, with $0 < a < q^n$, can be realized by an ssp circuit with at most n pswitches, which is optimal. They also showed that if $q = 4$, all rational $\frac{a}{q^n}$, with $0 < a < q^n$, can be realized using at most $2n-1$ pswitches. In this section we generalize these results:

1. If q is an even number, all rational $\frac{a}{q^n}$, with $0 < a < q^n$, can be realized by an ssp circuit with

at most $\lceil \log_2 q \rceil (n-1) + 1$ pswitches (Theorem 7.8).

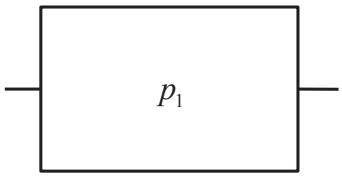
2. If q is odd and a multiple of 3, all rational $\frac{a}{q^n}$, with $0 < a < q^n$, can be realized by an ssp circuit with at most $\lceil \log_3 q \rceil (n-1) + 1$ pswitches (Theorem 7.9).

3. However, if q is a prime number greater than 3, there exists at least one rational $\frac{a}{q^n}$, with $0 < a < q^n$, that cannot be realized using an sp circuit (Theorem 7.12).

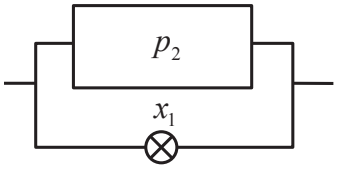
Table 7.1 summarizes these results. We see that when $q = 2, 3$, or 4, our results agree with the results in [134].

7.4.1 Backward Algorithms

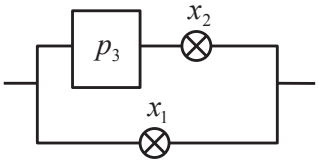
As mentioned in [134], switching circuits may be synthesized using forward algorithms, where circuits are built by adding pswitches sequentially, or backward algorithms, where circuits are built starting from the "outermost" pswitch.



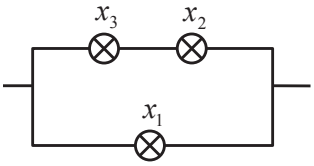
(a) Step 1: We assume that the desired probability is p_1.



(b) Step 2: Insert x_1 in parallel as the last pswitch. Now we try to realize p_2 such that $p_2 + x_1 - p_2 x_1 = p_1$.



(c) Step 3: Insert x_2 in series as the last pswitch. Now we try to realize p_3 such that $p_3 x_2 = p_2$.



(d) Last step: Replace p_3 with a single pswitch x_3.

Figure 7.7. An example of the backward algorithm.

Figure 7.7 gives a simple demonstration of a backward algorithm. Assume that the desired probability is p_1 and we plan to insert three pswitches, namely x_1, x_2, x_3 in backward direction. If

$x_1 \leq p_1$, then x_1 has to be inserted in parallel. If $x_1 > p_1$, then x_1 has to be inserted in series. After the insertion, we can try to realize the inner box with probability p_2 such that $p_2 + x_1 - p_2 x_1 = p_1$. This process is continued recursively until for some m, p_m can be realized with a single pswitch. Generally, in backward algorithms, we use x_k to denote the kth pswitch inserted in the backward direction, and use p_k to denote the probability that we want to realize with pswitches $x_k, x_{k+1}, x_{k+2}, ...$

Backward algorithms have significant advantages over forward algorithms for probability synthesis. In a forward algorithm, if we want to add one pswitch, we have $2|S|$ choices, since each pswitch may be added in either series or parallel. But in a backward algorithm, if we want to insert one pswitch, we have only $|S|$ choices. That is because the insertion (series or parallel) of a pswitch x_k simply depends on the comparison of x_k and p_k. Therefore, backward algorithms can significantly reduce the search space, hence are more efficient than forward algorithms. In this chapter, most of the circuit constructions are based on backward algorithms.

7.4.2 Multiples of 2 or 3

We consider the case that $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$ and q is a multiple of 2 or 3. We show that based on a backward algorithm, all rational $\frac{a}{q^n}$, with $0 < a < q^n$, can be realized using a bounded number of pswitches. Before describing the details, we introduce a characteristic function called d for a given probability $\frac{b}{q^w}$, that is

$$d\left(\frac{b}{q^w}\right) = \frac{q^{w-1}}{\gcd(b, q^{w-1})}.$$

Note that the value of d is unchanged when both b and q^w are multiplied by the same constant. From the definition of the characteristic function d, we see that for any rational $\frac{a}{q^n}$ with $0 < a < q^n$, d is a positive integer. In each iteration of the algorithm, we hope to reduce $d(p_k)$ such that it can reach 1 after a certain number of iterations. If $d = 1$, this means the desired probability can be realized using a single pswitch and the construction is done. During this process, we keep each successive probability p_k in the form of $\frac{b}{q^w}$, since only this kind of probabilities can be realized with the pswitch set S. Now, we describe the algorithm as follows.

Algorithm 7.6 (Backward algorithm to realize $p_1 = \frac{a}{q^n}$ with $0 < a < q^n$ and pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$).

1. *Set $k = 1$, starting with an empty circuit.*

2. *Let*

$$h(x_k, p_k) = \begin{cases} \frac{p_k}{x_k} & \text{if } x_k > p_k \text{ (series),} \\ \frac{p_k - x_k}{1 - x_k} & \text{if } x_k < p_k \text{ (parallel).} \end{cases}$$

We find the optimal $x_k \in S$ that minimizes $d(p_{k+1})$ with $p_{k+1} = h(x_k, p_k)$. If $p_{k+1} = \frac{b}{q^w}$, then

$$d(p_{k+1}) = d\left(\frac{b}{q^w}\right) = \frac{q^{w-1}}{\gcd(b, q^{w-1})}.$$

3. *Insert pswitch x_k to the circuit. If $x_k > p_k$, the pswitch is inserted in series; otherwise, it is inserted in parallel. Then we set*

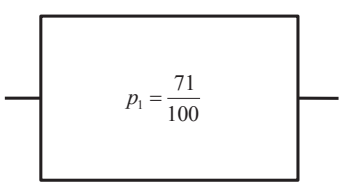
$$p_{k+1} = h(x_k, p_k).$$

4. *Let $k = k + 1$.*

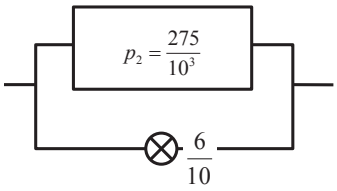
5. *Repeat steps 2–4 until p_k can be realized using a single pswitch. Then insert p_k into the circuit.*

In algorithm 7.6, the characteristic function $d(p_k)$ strictly decreases as k increases, until it reaches 1. Finally, p_k can be replaced by a single pswitch and the construction is done. Figure 7.8 gives an example of a circuit realized by this algorithm. At the beginning, we have $p_1 = \frac{71}{10^2}$, with $d(p_1) = 10$. Then we add the "best" pswitch to minimize $d(p_2)$, where the optimal pswitch is $\frac{6}{10}$. Since $\frac{6}{10} < \frac{71}{100}$, we insert the pswitch in parallel, making $d(p_2) = 4$. Repeating this process, we have

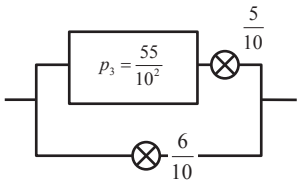
$$p_1 = \frac{71}{10^2},\ p_2 = \frac{275}{10^3},\ p_3 = \frac{55}{10^2},\ p_4 = \frac{1}{10},$$



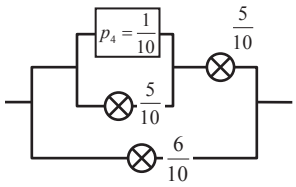
(a) Step 1 with $d(p_1) = 10$.



(b) Step 2 with $d(p_2) = 4$.



(c) Step 3 with $d(p_3) = 2$.



(d) Step 4 with $d(p_4) = 1$.

Figure 7.8. The procedure to realize $\frac{71}{100}$ for a given pswitch set $S = \{\frac{1}{10}, \frac{2}{10}, \ldots, \frac{9}{10}\}$.

with corresponding characteristic functions

$$d(p_1) = 10,\ d(p_2) = 4,\ d(p_3) = 2,\ d(p_4) = 1.$$

In the following theorem, we show that if q is a multiple of 2 or 3, then algorithm 7.6 realizes any rational $\frac{a}{q^n}$ with $0 < a < q^n$.

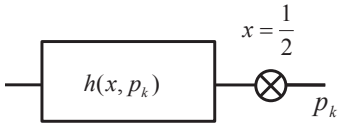
Theorem 7.7. *Given a pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$, if q is a multiple of 2 or 3, then algorithm 7.6 realizes any rational $\frac{a}{q^n}$ with $0 < a < q^n$, using an ssp circuit with a finite number of pswitches.*

Proof. The characteristic function $d(p_1)$ of the initial probability p_1 is bounded by q^{n-1}. We only need to prove that there exists an integer m such that $d(p_m) = 1$, i.e., p_m can be realized by a single pswitch. Hence the desired probability p_1 can be realized by an ssp circuit with m pswitches. It is enough to show that the characteristic function $d(p_k)$ decreases as k increases.

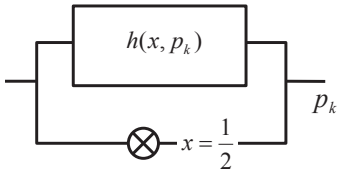
First, we consider the case where q is even. We will show that for any $p_k = \frac{b}{q^w}$, there exists $x \in S$ such that $d(h(x, p_k)) < d(p_k)$. See figure 7.9, depending on the values of p_k and $d(p_k)$, we have four different cases of inserting a pswitch x such that $d(h(x, p_k)) < d(p_k)$.

1. If $d(p_k)$ is even and $p_k < \frac{1}{2}$, let $x = \frac{1}{2}$ and insert the pswitch in series.

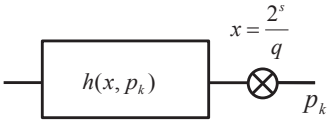
2. If $d(p_k)$ is even and $p_k > \frac{1}{2}$, let $x = \frac{1}{2}$ and insert the pswitch in parallel.



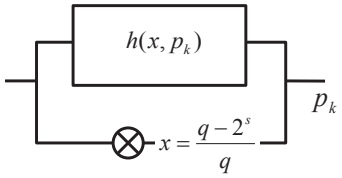
(a) $d(p_k)$ is even and $p_k < \frac{1}{2}$.



(b) $d(p_k)$ is even and $p_k > \frac{1}{2}$.



(c) $d(p_k)$ is odd and $p_k < \frac{1}{2}$.



(d) $d(p_k)$ is odd and $p_k > \frac{1}{2}$.

Figure 7.9. When q is even, the way to add a pswitch $x \in S$ such that $d(h(x, p_k)) < d(p_k)$.

3. If $d(p_k)$ is odd and $p_k < \frac{1}{2}$, let $x = \frac{2^s}{q}$ with $s = \lfloor \log_2 q \rfloor$ and insert the pswitch in series.

4. If $d(p_k)$ is odd and $p_k > \frac{1}{2}$, let $x = \frac{q - 2^s}{q}$ with $s = \lfloor \log_2 q \rfloor$ and insert the pswitch in parallel.

By checking all the cases to insert a pswitch, it is straightforward to see that when $d(p_k)$ is even, $d(h(x, p_k)) \leq \frac{1}{2} d(p_k)$, and when $d(p_k)$ is odd, $d(h(x, p_k)) \leq \frac{2^s d(p_k)}{\gcd(q, 2^s d(p_k))} < d(p_k)$. Since x_k is optimal in each step of algorithm 7.6, we have

$$d(p_{k+1}) = d(h(x_k, p_k)) \leq d(h(x, p_k)) < d(p_k).$$

Finally, we can conclude that when q is even, there exists an integer m such that $d(p_m) = 1$. Consequently, p_1 can be realized with at most m pswitches.

Similarly, when q is odd and a multiple of 3, if $p_k = \frac{b}{q^w}$, we can always insert a pswitch $x \in S$ such that $d(h(x, p_k)) < d(p_k)$, as follows:

1. If $d(p_k) \bmod 3 = 0$ and $p_k \leq \frac{1}{3}$, let $x = \frac{1}{3}$, and insert the pswitch in series.

2. If $d(p_k) \bmod 3 = 0$ and $\frac{1}{3} < p_k \leq \frac{2}{3}$ with even b, let $x = \frac{2}{3}$, and insert the pswitch in series.

3. If $d(p_k) \bmod 3 = 0$ and $\frac{1}{3} < p_k \leq \frac{2}{3}$ with odd b, let $x = \frac{2}{3}$, and insert the pswitch in parallel.

4. If $d(p_k) \bmod 3 = 0$ and $p_k > \frac{2}{3}$, let $x = \frac{2}{3}$, and insert the pswitch in parallel.

5. If $d(p_k) \bmod 3 \neq 0$ and $p_k \leq \frac{1}{3}$, let $x = \frac{3^s}{q}$ with $s = \lfloor \log_3 q \rfloor$, and insert the pswitch in series.

6. If $d(p_k) \bmod 3 \neq 0$ and $\frac{1}{3} < p_k \leq \frac{2}{3}$ with even b, let $x = \frac{2 \cdot 3^s}{q}$ with $s = \lfloor \log_3 q \rfloor$, and insert the pswitch in series.

7. If $d(p_k) \bmod 3 \neq 0$ and $\frac{1}{3} < p_k \leq \frac{2}{3}$ with odd b, let $x = \frac{q - 2 \cdot 3^s}{q}$ with $s = \lfloor \log_3 q \rfloor$, and insert the pswitch in parallel.

8. If $d(p_k) \bmod 3 \neq 0$ and $p_k > \frac{2}{3}$, let $x = \frac{q - 3^s}{q}$ with $s = \lfloor \log_3 q \rfloor$, and insert the pswitch in parallel.

Finally, we can conclude that p_1 can be realized with a finite number of pswitches when q is odd and a multiple of 3. □

For each value $q \in \{2, 3, 4, 6, 8, 9, 10\}$, we enumerate all rational numbers with optimal size $n \in (3, 4, 5)$. Here, we say that a desired probability is realized with optimal size if it cannot be realized with fewer pswitches. As a comparison, we use algorithm 7.6 to realize these rational numbers again. Figure 7.10 presents the average number of pswitches required using algorithm 7.6 when the optimal size is n. It is shown that when q is a multiple of 2 or 3, algorithm 7.6 can construct circuits with almost optimal size.

The next theorem gives an upper bound for the size of the circuits when q is even.

Theorem 7.8 (Upper bound of circuit size when q is even)**.** *Suppose q is even. Given a pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$, any rational $\frac{a}{q^n}$ with $0 < a < q^n$ can be realized by an ssp circuit, using at most $\lceil \log_2 q \rceil (n-1) + 1$ pswitches.*

Proof. In order to achieve this upper bound, we use a modified version of algorithm 7.6. Instead of inserting the optimal pswitch x_k, we insert the pswitch x described in figure 7.9 as the kth pswitch. The resulting characteristic function has the following properties:

(1) $d(p_k)$ decreases as k increases, and when $d(p_m) = 1$ for some m, the procedure stops.

(2) If $d(p_k)$ is even, then $d(p_{k+1})$ is a factor of $\frac{d(p_k)}{2}$.

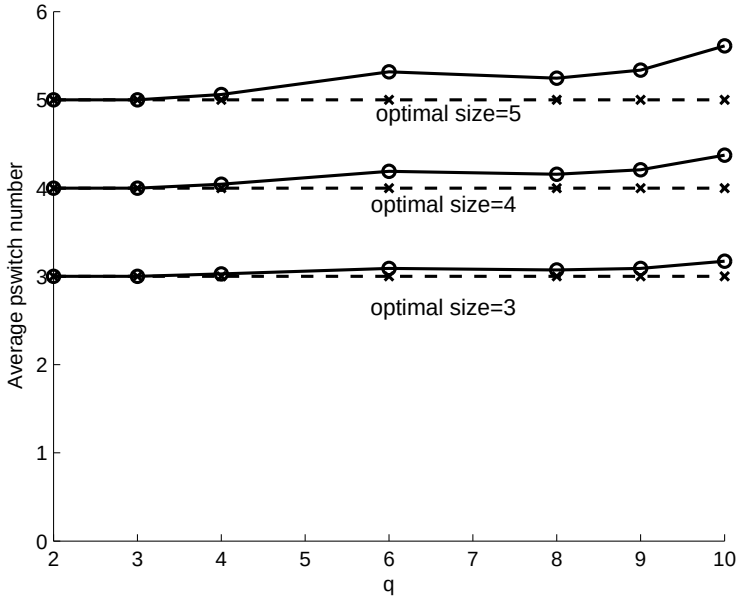


Figure 7.10. For each q, the average number of pswitches used in algorithm 7.6 to realize the rational probabilities when their optimal size is n.

(3) If $d(p_k)$ is odd, then $d(p_{k+1})$ is a factor of $\frac{2^s d(p_k)}{\gcd(q, 2^s d(p_k))}$.

We define

$$N = \min\{k | k \in (1, 2, 3, ...), d(p_k) = 1\},$$

then N is the number of required pswitches. We only need to prove that $N \leq \lceil \log_2 q \rceil (n-1) + 1$. Since q is even, we can write $q = 2^c$ or $q = 2^c t$, where $t > 1$ is odd.

Let us first consider the case of $q = 2^c$. At the beginning, $d(p_1)$ is a factor of q^{n-1}, so according to property (2), we can get

$$N \leq c(n-1) + 1 = \lceil \log_2 q \rceil (n-1) + 1.$$

In the case of $q = 2^c t$, let us define a set M as

$$M = \{k | k > 0, d(p_k) \text{ is odd}\},$$

and let M_i be the ith smallest element in M. According to properties (2) and (3) and the fact that

$d(p_1)$ is a factor of q^{n-1}, we see that $d(p_{M_i})$ is a factor of q^{n-i}. Therefore, there exits a minimal k, with $k \leq n$, such that $d(p_{M_k}) = 1$. Then $N = M_k$.

Based on properties (2) and (3), we also see that

$$M_1 \leq c(n-1) + 1,$$

and

$$M_{i+1} - M_i \leq s - c.$$

Therefore,

$$\begin{aligned} N &\leq \sum_{i=1}^{n-1}(M_{i+1} - M_i) + M_1 \leq s(n-1) + 1 \\ &= \lceil \log_2 q \rceil (n-1) + 1. \end{aligned}$$

This completes the proof. □

Using the similar methods, we can prove the following theorems as well when q is a multiple of 3 or 6. Note that theorem 7.8 also applies to the case that q is a multiple of 6, but theorem 7.10 provides a tighter upper bound.

Theorem 7.9 (Upper bound of circuit size when q is odd and a multiple of 3)**.** *Given a pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$, if q is odd and a multiple of 3, then any rational $\frac{a}{q^n}$ with $0 < a < q^n$ can be realized using an ssp circuit with at most $\lceil \log_3 q \rceil (n-1) + 1$ pswitches.*

Theorem 7.10 (Upper bound of circuit size when q is a multiple of 6)**.** *Given a pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$, if q is multiple of 6, all rational $\frac{a}{q^n}$ with $0 < a < q^n$ can be realized by an ssp circuit*

with at most N pswitches, where

$$N \leq \begin{cases} (2s)(n-1)+1 & (\textit{if } 6^s = q), \\ (2s+1)(n-1)+1 & (\textit{if } \frac{q}{2} \leq 6^s < q), \\ (2s+2)(n-1)+1 & (\textit{if } \frac{q}{3} \leq 6^s < \frac{q}{2}), \\ (2s+3)(n-1)+1 & (\textit{if } \frac{q}{6} < 6^s \leq \frac{q}{3}). \end{cases}$$

7.4.3 Prime Number Larger Than 3

We proved that if q is a multiple of 2 or 3, all rational $\frac{a}{q^n}$ can be realized with a finite number of pswitches. We want to know whether this result also holds if q is an arbitrary number greater than 2. Unfortunately, the answer is negative.

Lemma 7.11. *Suppose q is a prime number. Given a pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$, if a rational $\frac{a}{q^n}$ cannot be realized by an sp circuit with n pswitches, then it cannot be realized using an sp circuit with any number of pswitches.*

Proof. Assume there exits a rational $\frac{a}{q^n}$ which cannot be realized by an sp circuit with n pswitches, but can be realized with at least $l > n$ pswitches. Further, suppose that this l is minimal for all rationals with denominator q^k. Under these assumptions, we will prove that there exists a rational $\frac{a'}{q^{n'}}$ which cannot be realized with n' pswitches but can be realized with l' pswitches such that $l' < l$. This conclusion contradicts the assumption that l is minimal.

According to the definition of sp circuits, we know that $\frac{a}{q^n}$ can be realized by connecting two sp circuits C_1 and C_2 in series or in parallel. Assume C_1 consists of l_1 pswitches and is closed with probability $\frac{b_1}{q^{l_1}}$, and C_2 consists of l_2 pswitches and is closed with probability $\frac{b_2}{q^{l_2}}$, where $l_1 + l_2 = l$.

If C_1 and C_2 are connected in series, we can get

$$\frac{b_1}{q^{l_1}} \cdot \frac{b_2}{q^{l_2}} = \frac{a}{q^n}.$$

Therefore, $b_1 b_2 = a q^{l-n}$, where $b_1 b_2$ is a multiple of q. Since q is a prime number, either b_1 or

b_2 is a multiple of q. Without loss of generality, assume b_1 is a multiple of q, and we write $b_1 = cq$. Consider the probability $\frac{c}{q^{l_1-1}}$, which can be realized with C_1, using l_1 pswitches. Assume that the same probability can also be realized with another sp circuit C_3, using $l_1 - 1$ pswitches. By connecting C_3 and C_2 in series, we can realize $\frac{a}{q^n}$ with $l_1 - 1 + l_2 = l - 1$ pswitches, contradicting the assumption that $\frac{a}{q^n}$ cannot be realized with less than l pswitches. Therefore, we see that $\frac{c}{q^{l_1-1}}$ cannot be realized with $l_1 - 1$ pswitches, but it can be realized with l_1 pswitches. Since $l_1 < l$, this also contradicts our assumption that l is minimal.

If C_1 and C_2 are connected in parallel, we have

$$\frac{b_1}{q^{l_1}} + \frac{b_2}{q^{l_2}} - \frac{b_1}{q^{l_1}} \cdot \frac{b_2}{q^{l_2}} = \frac{a}{q^n}.$$

Therefore, $b_1 b_2 = b_1 q^{l_2} + b_2 q^{l_1} - a q^{l-n}$. Using a similar argument as above, we can conclude that either b_1 or b_2 is a multiple of q. Then either (1) $\frac{a}{q^l}$ can be realized with less than l pswitches or (2) l is not optimal, yielding a contradiction. This proves the lemma. □

Based on the lemma above, it is easy to get the following theorem.

Theorem 7.12 (When q is a prime number larger than 3)**.** *For a prime number $q > 3$, there exists an integer a, with $0 < a < q^n$, such that $\frac{a}{q^n}$ cannot be realized using an sp circuit whenever $n \geq 2$.*

Proof: The conclusion follows lemma 7.11 and the following result in [134]: For any $q > 3$, no pswitch set containing all $\frac{a}{q}$, with $0 < a < q$, can realize all $P_r(C) = \frac{b}{q^2}$, with $0 < b < q^2$, using at most 2 pswitches. □

7.5 Probability Approximation

In this section, we consider a general case where given an arbitrary pswitch set, we want to realize a desired probability. Clearly, not every desired probability p_d can be realized without any error using a finite number of pswitches for a fixed pswitch set S. So the question is whether we can construct a circuit with at most n pswitches such that it can approximate the desired probability very well.

Namely, the difference between the probability of the constructed circuit and the desired probability should be as small as possible.

7.5.1 Greedy Algorithm

Given an arbitrary pswitch set S with $|S| \geq 2$, it is not easy to find the optimal circuit (ssp circuit) with n pswitches which approximates the desired probability p_d. As we discussed in the last section, a backward algorithm provides $|S|$ choices for each successive insertion. To find the optimal circuit, we may have to search through $|S|^n$ different combinations. As $|S|$ or n increases, the number of combinations will increase dramatically. In order to reduce the search space, we propose a greedy algorithm: In each step, we insert m pswitches, which are the "best" locally. Normally, m is a very small constant. Since each step has complexity $|S|^m$, the total number of possible combinations is reduced to $|S|^m \frac{n}{m}$, which is much smaller than $|S|^n$ when $|S| \geq 2$ and n is large. Now, we describe this greedy algorithm briefly. The same notations $x_1, x_2, ...$ and $p_1, p_2, ...$ are used, as those described for the backward algorithms: x_k indicates the kth pswitch inserted and p_k indicates the desired probability of the subcircuit constructed by $x_k, x_{k+1}, ...$

Algorithm 7.13 (Greedy algorithm with step-length m)**.**

1. *Assume that the desired probability is p_1. Set $k = 1$ and start with an empty circuit.*

2. *Select the optimal $x^m = (x_1, x_2, \ldots, x_m) \in S^m$ to minimize $f(x^m, S, p_k)$, which will be specified later, and this x^m is denoted as $x^* = (x_1^*, x_2^*, \ldots, x_m^*)$.*

3. *Insert m pswitches $x_1^*, x_2^*, \ldots, x_m^*$ one by one into the circuit in backward direction. During this process, calculate $p_{k+1}, p_{k+2}, ..., p_{k+m}$ one by one and update k as $k + m$.*

4. *Repeat steps 2 and 3 for $\lfloor \frac{n}{m} \rfloor$ times.*

5. *Construct a new circuit with $n - \lfloor \frac{n}{m} \rfloor m$ pswitches such that its probability is closest to p_k, then replace p_k with this new circuit.*

So far, according to the backward algorithm described in section 7.4.1, we know how to finish step 3, including how to insert m pswitches one by one into a circuit in a backward direction, and how to update p_k. The only thing unclear in the procedure above is the expression of $f(x^m, S, p_k)$.

In order to get a good expression for $f(x^m, S, p_k)$, we study how errors propagate in a backward algorithm. Note that in a backward algorithm, we insert pswitches $x_1, x_2, ..., x_n$ one by one: if $x_k > p_k$, then x_k is inserted in series; if $x_k < p_k$, then x_k is inserted in parallel. Now, given a circuit C with size n constructed using a backward algorithm, we let $C^{(k)}$ denote the subcircuit constructed by $x_{k_1}, x_{k_1+1}, ..., x_n$ and call $|P(C^{(k)}) - p_k|$ as the approximation error of p_k, denoted by e_k. In the following theorem, we will show how e_{k_1} affects that of e_{k_2} for $k_2 < k_1$ after inserting pswitches $x_{k_2}, ..., x_{k_1-1}$.

Lemma 7.14. *In a backward algorithm, let p_k denote the desired probability of the subcircuit $C^{(k)}$ constructed by $x_k, x_{k+1}, ..., x_n$, and let e_k denote the approximation error of p_k. Then for any $k_2 < k_1 \leq n$, we have*

$$e_{k_2} = \left(\prod_{i=k_1}^{k_2-1} r(x_i) \right) e_{k_1},$$

where

$$r(x_i) = \begin{cases} x_i & \text{if } x_i \text{ is inserted in series,} \\ 1 - x_i & \text{if } x_i \text{ is inserted in parallel.} \end{cases}$$

Proof. We only need to prove that for any k less than the circuit size, the following result holds:

$$e_k = r(x_k) e_{k+1}.$$

When $x_k = p_k$, we have $e_k = e_{k+1} = 0$, so the result is trivial.

When $x_k > p_k$, then x_k is inserted in series. In this case, we have

$$p_{k+1} x_k = p_k,$$

and

$$P(C^{(k+1)})x_k = P(C^{(k)}).$$

As a result, the approximation error of p_k is

$$\begin{aligned} e_k &= |P(C^{(k)}) - p_k| \\ &= |P(C^{(k+1)})x_k - p_{k+1}x_k| \\ &= x_k e_{k+1}. \end{aligned}$$

When $x_k < p_k$, then x_k is inserted in parallel. In this case, we have

$$p_{k+1} + x_k - p_{k+1}x_k = p_k,$$

and

$$P(C^{(k+1)}) + x_k - P(C^{(k+1)})x_k = P(C^{(k)}).$$

As a result, the approximation error of p_k is

$$\begin{aligned} e_k &= |P(C^{(k)}) - p_k| \\ &= |P(C^{(k+1)}) + x_k - P(C^{(k+1)})x_k - (p_{k+1} + x_k - p_{k+1}x_k)| \\ &= (1 - x_k)e_{k+1}. \end{aligned}$$

This completes the proof. □

In each step of the greedy algorithm, our goal is to minimize e_k, the approximation error of p_k. According to the lemma above, we know that

$$e_k = \left(\prod_{i=k}^{k+m-1} r(x_i) \right) e_{k+m},$$

where the term e_{k+m} is unknown. But we can minimize $\prod_{i=k}^{k+m-1} r(x_i)$ such that e_k is as small as

possible.

Based on the above discussion, we express $f(x, S, p_k)$ as

$$f(x, S, p_k) = \prod_{i=1}^{m} r(x_i),$$

with

$$r(x_i) = \begin{cases} x_i & \text{if } x_i \text{ is inserted in series,} \\ 1 - x_i & \text{if } x_i \text{ is inserted in parallel.} \end{cases}$$

In the rest of this section, based on this expression for $f(x, S, p_k)$, we show that the greedy algorithm has good performance in reducing the approximation error of p_d.

7.5.2 Approximation Error when $|S| = 1$

When S has only one element, say $S = \{p\}$, the greedy algorithm above can become really simple. If $p_k > p_k$, then we insert one pswitch in parallel; otherwise, we insert it in series. Figure 7.11 demonstrates how to approximate $\frac{1}{2}$ using four pswitches with the same probability $\frac{1}{3}$. Initially, $p_1 = \frac{1}{2} > \frac{1}{3}$, so we insert $\frac{1}{3}$ in parallel. As a result, $p_2 = \frac{\frac{1}{2}-\frac{1}{3}}{1-\frac{1}{3}} = \frac{1}{4} < \frac{1}{3}$, so we insert the second pswitch in series. The final probability of the circuit in figure 7.11 is $\frac{37}{81}$, which is close to $\frac{1}{2}$.

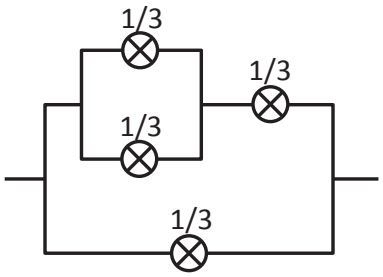


Figure 7.11. This circuit approximates $\frac{1}{2}$ using 4 pswitches of probability $\frac{1}{3}$.

Note that in the greedy algorithm, when p is close to $\frac{1}{2}$, the probability of the resulting circuit will quickly converge to the desired probability. But when p is close to 0 or 1, the convergence speed is slower. In the following theorem, we provide an upper bound for the approximation error of the desired probability when $|S| = 1$.

Theorem 7.15 (Approximation error when $|S| = 1$)**.** *Given n pswitches, each with probability p, and a desired probability p_d, the greedy algorithm (algorithm 7.13) with $m = 1$ generates an ssp circuit C with approximation error*

$$e = |p_d - P(C)| \leq \frac{(\max\{p, 1-p\})^n}{2},$$

where equality is achieved when

$$p_d = f_n(p) = \begin{cases} 1 - \frac{(\max\{p,1-p\})^n}{2} & \text{if } p < \frac{1}{2}, \\ \frac{(\max\{p,1-p\})^n}{2} & \text{if } p > \frac{1}{2}. \end{cases}$$

Proof. In the following proof, we only consider the case when $p < \frac{1}{2}$. From duality, the result will also hold for $p > \frac{1}{2}$.

We induct on the number of pswitches. For one pswitch, the result is trivial: the worst-case desired probability is $p + \frac{1-p}{2}$, with approximation error $\frac{1-p}{2}$. Now assume the result of the theorem holds for n pswitches, we want to prove that it also holds for $n + 1$ pswitches.

Let $p_1 = p_d$ be approximated with $n + 1$ pswitches using algorithm 7.13. At the beginning, one pswitch is inserted in series if $p_d < p$, or in parallel if $p_d > p$. According to lemma 7.14, we know that the approximation error of p_1 is

$$e_1 = r(p)e_2,$$

where $r(p) \leq \max\{p, 1-p\}$, and e_2 is the approximation error of p_2. According to our assumption, we know that

$$e_2 \leq \frac{(\max\{p, 1-p\})^n}{2}.$$

So we have

$$e_1 \leq \frac{(\max\{p, 1-p\})^{n+1}}{2}.$$

Note that equality is achieved if $r(p) = \max\{p, 1-p\}$ and $e_2 = \frac{(\max\{p,1-p\})^n}{2}$. In this case,

$p_2 = f_n(p) \geq \frac{1}{2} > p$ and the last pswitch is inserted in parallel. As a result, we have

$$f_{n+1}(p) = f_n(p) + p - f_n(p)p = 1 - \frac{(1-p)^{n+1}}{2}$$

as described in the theorem. This completes the proof. □

If we let $p = \frac{1}{2}$, the theorem shows that for any desired probability p_d and any integer n, we can find an ssp circuit with n pswitches to approximate p_d, such that the approximation error is at most $\frac{1}{2q^n}$. This agrees with the result in [134]: Given a pswitch set $S = \{\frac{1}{2}\}$, all rational $\frac{a}{2^n}$, with $0 < a < q^n$, can be realized using at most n pswitches.

7.5.3 Approximation Error when $|S| > 1$

In this subsection, we show that using the greedy algorithm (Algorithm 7.13) with small m, such as 1 or 2, we can construct a circuit to obtain a good approximation of any desired probability. Here, given a pswitch set $S = \{s_1, s_2, ..., s_{|S|}\}$, we define its maximal interval Δ as

$$\Delta = \max_{i=0}^{|S|} |s_{i+1} - s_i|,$$

where we let $s_0 = 0$ and $s_{|S|+1} = 1$. In the following theorems, we will see that the approximation error of the greedy algorithm depends on Δ, and can decrease rapidly as n increases.

Let us first consider the case $m = 1$:

Theorem 7.16 (Approximation error for $m = 1$)**.** *Assume we have the pswitch set $S = \{s_1, s_2, \ldots, s_{|S|}\}$ with maximal interval Δ. For any desired probability p_d and any integer n, algorithm 7.13 with $m = 1$ yields an ssp circuit with at most n pswitches, such that the approximation error e satisfies*

$$e \leq \frac{\Delta}{2}\left(\frac{(3+\Delta)\Delta}{2}\right)^{\lceil \frac{n}{2} \rceil - 1}.$$

Proof. In the following proof, we only consider the case that n is odd. If the result holds for odd n,

then the result will also hold for even n. In order to simplify the proof, we assume that $s_0 = 0$ and $s_{|S|+1} = 1$ also belong to S; i.e., there are pswitches with probability 0 or 1. This assumption will not affect our conclusion.

We write $n = 2k+1$ and induction on k. When $k = 0$, the result is trivial, since the approximation error e of one pswitch satisfies $e \leq \frac{\Delta}{2}$. Assume the result holds for $2k + 1$ pswitches. We want to show that the result also holds for $2(k + 1) + 1$ pswitches.

When $m = 1$ in the greedy algorithm, if we want to approximate $p_1 = p_d$ with $2(k + 1) + 1$ pswitches, we should insert a pswitch with probability $\arg\min_x f(x, S, p_1)$ in the first step.

Let $x_{\text{upper}} = \min\{x \in S | x > p_1\}$ and $x_{\text{lower}} = \max\{x \in S | x < p_1\}$. Since $0 \in S$ and $1 \in S$, we know that x_{upper} and x_{lower} exist.

(1) We first consider the case that $1 - x_{\text{lower}} \leq x_{\text{upper}}$. In this case, we insert x_{lower} in parallel as the first pswitch. Therefore, we can get

$$p_2 = \frac{p_1 - x_{\text{lower}}}{1 - x_{\text{lower}}}.$$

According to the definition of Δ, there exists a pswitch $x \in S$ such that $p_2 \leq x < p_2 + \Delta$. Assume in the algorithm, we insert pswitch x^* as the second one. Since x^* is locally optimal, we have

$$f(x^*, S, p_2) \leq f(x, S, p_2) < p_2 + \Delta.$$

Assume the approximation error of p_3 is e_3. According to lemma 7.14, we know that the approx-

imation error of $p_1 = p_d$ is

$$\begin{aligned} e_1 &\leq (p_2 + \Delta)(1 - x_{\text{lower}})e_3 \\ &= (\frac{p_1 - x_{\text{lower}}}{1 - x_{\text{lower}}} + \Delta)(1 - x_{\text{lower}})e_3 \\ &= ((p_1 - x_{\text{lower}}) + \Delta(1 - x_{\text{lower}}))e_3 \\ &\leq \Delta(2 - x_{\text{lower}})e_3 \\ &\leq \frac{\Delta(3 + x_{\text{upper}} - x_{\text{lower}})}{2}e_3 \\ &\leq \frac{\Delta(3 + \Delta)}{2}e_3. \end{aligned}$$

According to our assumption,

$$e_3 \leq \frac{\Delta}{2}(\frac{(3 + \Delta)\Delta}{2})^k.$$

So

$$e_1 \leq \frac{\Delta}{2}(\frac{(3 + \Delta)\Delta}{2})^{k+1}.$$

This completes the induction.

(2) When $1 - x_{\text{lower}} > x_{\text{upper}}$, we insert x_{upper} in series as the first pswitch. Using a similar argument as above, we can also prove that

$$e_1 \leq \frac{\Delta}{2}(\frac{(3 + \Delta)\Delta}{2})^{k+1}.$$

This completes the proof. □

In the next theorem, we show that if we increase m from 1 to 2, the upper bound of the approximation error can be reduced furthermore.

Theorem 7.17 (Approximation error for $m = 2$)**.** *Assume we have the pswitch set* $S = \{s_1, s_2, \ldots, s_{|S|}\}$ *with maximal interval* Δ*. For any desired probability* p_d *and any integer* n*, algorithm 7.13 with* $m = 2$

yields an ssp circuit with at most n pswitches, such that the approximation error e satisfies

$$e \leq \frac{\Delta}{2}\left(\frac{(2+\Delta)\Delta}{2}\right)^{\lceil \frac{n}{2} \rceil - 1}.$$

Proof. As in the proof for $m = 1$, we only consider the case when n is odd, so $n = 2k + 1$. In the proof, we use the same notations as those in the case of $m = 1$, and assume S includes 0 and 1.

Now we induct on k. When $k = 0$, the result of the theorem is trivial. Assume the result holds for $2k + 1$ pswitches; we want to prove that it also holds for $2(k + 1) + 1$ pswitches. Let $x_{\text{upper}} = \min\{x \in S | x > p_1\}$ and $x_{\text{lower}} = \max\{x \in S | x < p_1\}$, we will consider two different cases as follows.

(1) If $p_1 \leq \frac{x_{\text{upper}} + x_{\text{lower}} + \Delta(x_{\text{upper}} + x_{\text{lower}} - 1)}{2}$, we consider the following way to insert two pswitches: First insert $x_1 = x_{\text{lower}}$ in parallel, and we get

$$p_2 = \frac{p_1 - x_{\text{lower}}}{1 - x_{\text{lower}}}.$$

There exists a pswitch $x_2 \in S$ such that $p_2 \leq x_2 < p_2 + \Delta$. Then we insert x_2 in series as the second pswitch. In this case, letting $x = (x_1, x_2)$, we have

$$f(x, S, p_1) \leq (p_2 + \Delta)(1 - x_{\text{lower}}).$$

Let $x^* = (x_1^*, x_2^*)$ be the two pswitches inserted by the algorithm with $m = 2$, then the approximation error of $p_1 = p_d$ is

$$\begin{aligned} e_1 &= f(x^*, S, p_1)e_3 \leq f(x, S, p_1)e_3 \\ &\leq (p_2 + \Delta)(1 - x_{\text{lower}})e_3 \\ &= \left(\frac{p_1 - x_{\text{lower}}}{1 - x_{\text{lower}}} + \Delta\right)(1 - x_{\text{lower}})e_3. \end{aligned}$$

Since $p_1 \leq \frac{x_{\text{upper}}+x_{\text{lower}}+\Delta(x_{\text{upper}}+x_{\text{lower}}-1)}{2}$, we have

$$\begin{aligned} e_1 &\leq \frac{(x_{\text{upper}} - x_{\text{lower}})(1+\Delta)+\Delta}{2} e_3 \\ &\leq \frac{\Delta(2+\Delta)}{2} e_3. \end{aligned}$$

According to our assumption, we have $e_3 \leq \frac{\Delta}{2}\left(\frac{(2+\Delta)\Delta}{2}\right)^k$, so

$$e_1 \leq \frac{\Delta}{2}\left(\frac{(2+\Delta)\Delta}{2}\right)^{k+1}.$$

This completes the induction.

(2) If $p_1 > \frac{x_{\text{upper}}+x_{\text{lower}}+\Delta(x_{\text{upper}}+x_{\text{lower}}-1)}{2}$, we consider the following way to insert two pswitches: First insert $x_1 = x_{\text{upper}}$ in series, and we get

$$p_2 = \frac{p_1}{x_{\text{upper}}}.$$

There exists a pswitch $x_2 \in S$ such that $p_2 - \Delta \leq x_2 < p_2$. Then we insert x_2 in parallel as the second pswitch. In this case, letting $x = (x_1, x_2)$, we have

$$f(x, S, p_1) \leq (1 - (p_2 - \Delta))x_{\text{upper}}.$$

Let $x^* = (x_1^*, x_2^*)$ be the two pswitches inserted by the algorithm with $m = 2$, then the approximation error of $p_1 = p_d$ is

$$\begin{aligned} e_1 &= f(x_*, S, p_1)e_3 \\ &\leq f(x, S, p_1)e_3 \\ &\leq (1 - (p_2 - \Delta))x_{\text{upper}}e_3 \\ &= \left(\frac{x_{\text{upper}} - p_1}{x_{\text{upper}}} + \Delta\right) x_{\text{upper}}e_3. \end{aligned}$$

Since $p_1 > \frac{x_{\text{upper}}+x_{\text{lower}}+\Delta(x_{\text{upper}}+x_{\text{lower}}-1)}{2}$, we have

$$\begin{aligned} e_1 &\leq \frac{(x_{\text{upper}} - x_{\text{lower}})(1+\Delta)+\Delta}{2} e_3 \\ &\leq \frac{\Delta(2+\Delta)}{2} e_3. \end{aligned}$$

Then we have the same result as the first case. □

According to the two theorems above, when we let $\Delta \to 0$, the approximation error for $m = 1$ is upper bounded by $\frac{\Delta}{2}\left(\frac{3\Delta}{2}\right)^k$ where $k = \lceil \frac{n}{2} \rceil - 1$; and the approximation error for $m = 2$ is upper bounded by $\frac{\Delta}{2} \cdot \Delta^k$. It shows that the greedy algorithm has good performance in terms of approximation error, even when m is very small. Comparing with the case of $m = 1$, if we choose $m = 2$, the probability of the constructed circuit can converge to the desired probability faster as the circuit size n increases.

In the following theorem, we consider the special case $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$ for some integer q. In this case, we obtain a new upper bound for the approximation error when using the greedy algorithm with $m = 2$. This bound is slightly tighter than the one obtained in theorem 7.17.

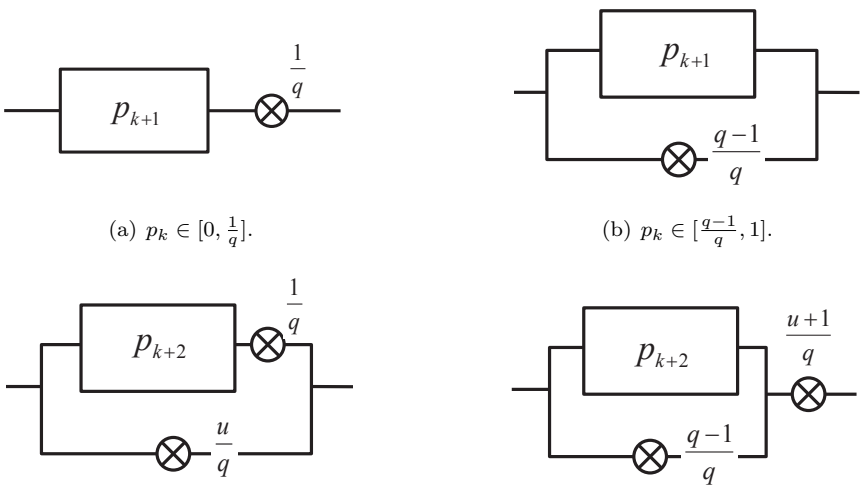
Theorem 7.18. *Suppose $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$ for some integer q, with $\Delta = \frac{1}{q}$. For any desired probability p_d and any integer n, algorithm 7.13 with $m = 2$ constructs an ssp circuit with at most n pswitches such that its approximation error*

$$e \leq \frac{\Delta}{2}\left(\Delta(1-\Delta)\right)^{\lceil \frac{n}{2} \rceil - 1}.$$

Proof. The proof is similar to the proof of theorem 7.17, so we simply provide a sketch. Assume that in each step, we insert two pswitches in the following way (see figure 7.12):

(1) If $p_k \in [0, \frac{1}{q}]$, we insert a pswitch $x_1 = \frac{1}{q}$ in series, and then insert a pswitch $x_2 = \frac{1}{q}$ in series or in parallel. In this case,

$$f\left(\left(\frac{1}{q}, \frac{1}{q}\right), S, p_k\right) \leq \frac{1}{q}\left(1 - \frac{1}{q}\right) = \Delta(1-\Delta).$$



(a) $p_k \in [0, \frac{1}{q}]$.

(b) $p_k \in [\frac{q-1}{q}, 1]$.

(c) $p_k \in [\frac{u}{q}, \frac{u}{q} + \frac{1}{q} - \frac{u^2}{q}]$ for some $u = \{1, \ldots, q-1\}$.

(d) $p_k \in [\frac{u}{q} + \frac{1}{q} - \frac{u^2}{q}, \frac{u}{q} + \frac{1}{q}]$ for some $u = \{0, 1, \ldots, q-2\}$.

Figure 7.12. Inserting pswitches for different values of p_k.

(2) If $p_k \in [\frac{q-1}{q}, 1]$, we insert a pswitch $x_1 = \frac{q-1}{q}$ in parallel, and then insert a pswitch $x_2 = \frac{q-1}{q}$ in series or in parallel. In this case,

$$f\left(\left(\frac{1}{q}, \frac{1}{q}\right), S, p_k\right) \leq \frac{1}{q}\left(1 - \frac{1}{q}\right) = \Delta(1-\Delta).$$

(3) If $p_k \in [\frac{u}{q}, \frac{u}{q} + \frac{1}{q} - \frac{u^2}{q}]$ for some $u = \{1, \ldots, q-1\}$, we insert a pswitch $x_1 = \frac{u}{q}$ in parallel, and then insert a pswitch $x_2 = \frac{1}{q}$ in series. In this case,

$$f\left(\left(\frac{u}{q}, \frac{1}{q}\right), S, p_k\right) \leq \left(1 - \frac{u}{q}\right)\frac{1}{q} \leq \Delta(1-\Delta).$$

(4) If $p_k \in [\frac{u}{q} + \frac{1}{q} - \frac{u^2}{q}, \frac{u}{q} + \frac{1}{q}]$ for some $u = \{0, 1, \ldots, q-2\}$, we insert a pswitch $x_1 = \frac{u+1}{q}$ in series, and then insert a pswitch $x_2 = \frac{q-1}{q}$ in parallel. In this case,

$$\begin{aligned} f\left(\left(\frac{u+1}{q}, \frac{q-1}{q}\right), S, p_k\right) &\leq \frac{u+1}{q}\left(1 - \frac{q-1}{q}\right) \\ &\leq \Delta(1-\Delta). \end{aligned}$$

Based on the above analysis, we know that for any $p_k \in (0,1)$, we can always find $x = (x_1, x_2)$

such that

$$f((x_1, x_2), S, p_k)) \leq \Delta(1 - \Delta).$$

Hence, the result of the theorem can be proved by induction. □

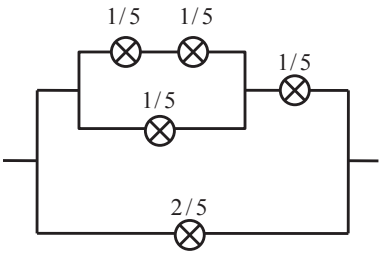


Figure 7.13. The circuit approximates $\frac{3}{7}$ with 5 pswitches from the pswitch set $S = \{\frac{1}{5}, \frac{2}{5}, \ldots, \frac{4}{5}\}$.

Figure 7.13 shows an example for demonstration. Assume $S = \{\frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}\}$, and suppose we want to realize $\frac{3}{7}$ using five pswitches. Using the greedy algorithm with $m = 2$, we can get the circuit in figure 7.13, whose probability is 0.4278, and approximation error is

$$e = \left|\frac{3}{7} - 0.4278\right| = 7.3 \times 10^{-4},$$

which is very small.

7.6 Conclusion

In this chapter, we have studied the robustness and synthesis of stochastic switching circuits. We have shown that ssp circuits are robust against small error perturbations, while general sp circuits are not. As a result, we focused on constructing ssp circuits to synthesize or approximate probabilities. We generalized the results in [134] and proved that when q is a multiple of 2 or 3, all rational fractions $\frac{a}{q^n}$ can be realized using ssp circuits when the pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, \ldots, \frac{q-1}{q}\}$. However, this property does not hold when q is a prime number greater than 3. For a more general case of an arbitrary pswitch set, we proposed a greedy algorithm to construct ssp circuits. This method can approximate any desired probability with low circuit complexity and small errors.

Many open problems remain concerning probability synthesis in stochastic switching circuits. For instance, if q is neither a prime number nor a multiple of 2 or 3, can we realize all rationals $\frac{a}{q^n}$ using ssp circuits with the pswitch set $S = \{\frac{1}{q}, \frac{2}{q}, ..., \frac{q-1}{q}\}$? Can we combine probability synthesis and probabilistic computing? Is it possible to design integrated systems with distributed and mixed storage and computing elements?

Chapter 8

Synthesis of Stochastic Flow Networks

This chapter designs optimal-sized stochastic flow networks for "synthesizing" target distributions. It shows that when each splitter (basic probabilistic element) has probability 1/2, an arbitrary rational probability $\frac{a}{b}$ with $a \leq b \leq 2^n$ can be realized by a stochastic flow network of size n, and its size is optimal.[1]

8.1 Introduction

There are a few works that considered the problem of probability transformation from a synthetic perspective, namely, designing a physical system for "synthesizing" target distributions, by connecting certain probabilistic elements. Such probabilistic elements can be electrical ones based on internal thermal noise or molecular ones based on inherent randomness in chemical reactions. In this scenario, the size of the construction becomes a central issue. Gill [44] [45] discussed the problem of generating rational probabilities using a sequential state machine. Sheng [107] considered applying threshold logic elements as a discrete probability transformer. Wilhelm and Bruck [134] proposed a procedure for synthesizing stochastic relay circuits to realize desired discrete probabilities. It was further discussed and analyzed in chapter 7 of this thesis. Qian et al. [92] studied combinational logic for transforming a set of given probabilities into target probabilities. Motivated by stochastic

[1] Some of the results presented in this chapter have been previously published in [144].

computation based on chemical reaction networks [108], in this chapter we study stochastic flow networks. A stochastic flow network is a directed graph with incoming edges (inputs) and outgoing edges (outputs); tokens enter through the input edges, travel stochastically in the network and can exit the network through the output edges. Each node in the network is a splitter, namely, a token can enter a node through an incoming edge and exit on one of the output edges according to a predefined probability distribution. We address the following synthesis question: Given a finite set of possible splitters and an arbitrary rational probability distribution, design an *optimal-sized* stochastic flow network, such that every token that enters the input edge will exit the outputs with the prescribed probability distribution.

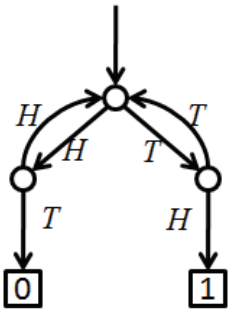


Figure 8.1. A stochastic flow network that consists of three p-splitters and generates probability $\frac{1}{2}$.

While stochastic flow networks can be easily implemented by chemical reaction networks, they demonstrate strong powers in expressing an arbitrary rational target distribution, stronger than any other synthetic stochastic systems described above. Figure 8.1 depicts von Neumann's algorithm in the language a stochastic flow network that consists of three p-splitters for any p and generates probability $\frac{1}{2}$. Here, a p-splitter indicates a splitter with two outgoing edges with probabilities p and $(1-p)$. In this construction, we have two outputs $\{\beta_1, \beta_2\} = \{0, 1\}$ (corresponding to the labels 0 and 1, respectively). For each incoming token, it has the same probability pq to reach either output 0 or output 1 directly, and it has probability $1 - 2pq$ to come back to the starting point. Eventually, the probability for the token to reach each of the outputs is $\frac{1}{2}$. In general, the outputs of a stochastic flow network have labels denoted by $\{\beta_1, \beta_2, ..., \beta_m\}$. A token will reach an output β_k $(1 \leq k \leq m)$ with probability q_k, and we call q_k the probability of β_k and call $\{q_1, q_2, ..., q_m\}$ the

output probability distribution of the network, where $\sum_{k=1}^{m} q_k = 1$.

In this chapter we assume, without loss of generality, that the probability of each splitter is $\frac{1}{2}$ ($\frac{1}{2}$-splitters can be implemented using three p-splitters for any p). Our goal is to realize the target probabilities or distributions by constructing a network of minimal size. In addition, we study the expected latency, namely the expected number of splitters a token need to pass before reaching the output (or we call it the expected operating time).

The main contributions of the chapter are

1. *General optimal construction:* For any desired rational probability, an *optimal-sized* construction of stochastic flow network is provided.

2. *The power of feedback:* We show that with feedback (loops), stochastic flow networks can generate much more probabilities than those without feedback.

3. *Constructions with well-bounded expected latency:* Two constructions with a few more splitters than the optimal-sized one are given, such that their expected latencies are well bounded by constants.

4. *Constructions for arbitrary rational distributions:* We generalize our constructions and results to arbitrary rational probability distributions $\{q_1, q_2, ..., q_m\}$.

The remainder of this chapter is organized as follows. In section 8.2 we introduce some preliminaries including Knuth and Yao's scheme and a few mathematical tools for calculating the distribution of a given stochastic flow network. Section 8.3 introduces an optimal-sized construction of stochastic flow networks for synthesizing an arbitrary rational probability and it demonstrates that feedback significantly enhances the expressibility of stochastic flow networks. Section 8.4 analyzes the expected latency of the optimal-sized construction. Section 8.5 gives two constructions with constant-bounded expected latencies, called size-relaxed construction and latency-oriented construction. Section 8.6 presents the generalizations of our results to arbitrary rational probability distributions. The concluding remarks and the comparison of different stochastic systems are given in section 8.7.

8.2 Preliminaries

In this section, we introduce some preliminaries, including Knuth and Yao's scheme for simulating an arbitrary distribution from a biased coin, and how using absorbing Markov chains or Mason' Rule to calculate the output distribution of a given stochastic flow network.

8.2.1 Knuth and Yao's Scheme

In 1976, Knuth and Yao proposed a simple procedure for simulating an arbitrary distribution from an unbiased coin (the probability of H and T is $\frac{1}{2}$) [71]. They introduced a concept called generating tree for representing the algorithm [27]. The leaves of the tree are marked by the output symbols, and the path from the root node to the leaves indicates the sequences of bits generated by the unbiased coin. Starting from the root node, the scheme selects edges to follow based on the coin tosses until it reaches one of the leaves. Then it outputs the symbol marked on that leaf.

In general, we assume that the target distribution is $\{p_1, p_2, ..., p_m\}$. Since all the leaves of the tree have probabilities of the form 2^{-k} (if the depth of the leaf is k), we split each probability p_i into atoms of this form. Specifically, let the binary expansion of the probability p_i be

$$p_i = \sum_{j \geq 1} p_i^{(j)},$$

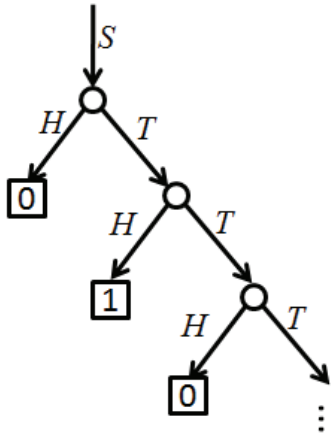


Figure 8.2. The generating tree to generate a $(\frac{2}{3}, \frac{1}{3})$ distribution.

where $p_i^{(j)} = 2^{-j}$ or 0. Then for each probability p_i, we get a group of atoms $\{p_i^{(j)} : j \geq 1\}$. For these atoms, we allot them to leaves with label β_i on the tree. Hence, the probability of generating β_i is p_i. We can see that the depths of all the atoms satisfy the Kraft inequality [27], i.e.

$$\sum_{i=1}^{m} \sum_{j \geq 1} p_i^{(j)} = 1.$$

So we can always construct such a tree with all the atoms allotted. Knuth and Yao showed that the expected number of fair bits required by the procedure (i.e., the expected depth of the tree) to generate a random variable X with distribution $\{p_1, p_2, ..., p_m\}$ lies between $H(X)$ and $H(X) + 2$ where $H(X)$ is the entropy of the target distribution.

Figure 8.2 depicts a generating tree that generates a distribution $\{\frac{2}{3}, \frac{1}{3}\}$, where the atoms for $\frac{2}{3}$ are $\{\frac{1}{2}, \frac{1}{8}, \frac{1}{32}, ...\}$, and the atoms for $\frac{1}{3}$ are $\{\frac{1}{4}, \frac{1}{16}, \frac{1}{64}, ...\}$. We see that the construction of generating trees is, in some sense, a special case of stochastic flow networks. If we consider each node in the generating tree as a splitter, then each token that enters the tree from the root node will reach the outputs with the target distribution. While Knuth and Yao's scheme aims to minimize the expected depth of the tree (or in our framework, we call it the expected latency of the network), our goal is to optimize the size of the construction, i.e., the number of nodes in the network.

8.2.2 Absorbing Markov Chain

Let us consider a stochastic flow network with n splitters and m outputs, in which each splitter is associated with a state number in $\{1, 2, ..., n\}$ and each output is associated with a state number in $\{n + 1, n + 2, ..., n + m\}$. When a token reaches splitter i with $1 \leq i \leq n$, we say that the current state of this network is i. When it reaches output k with $1 \leq k \leq m$, we say that the current state of this network is $n + k$. Note that the current state of the network only depends on the last state, and when the token reach one output it will stay there forever. So we can describe token flows in this network using an absorbing Markov chain. If the current state of the network is i, then the probability of reaching state j at the next instant of time is given by p_{ij}. Here, $p_{ij} = p_H$ ($p_{ij} = p_T$)

if and only if state i and state j is connected by an edge H (T).

Clearly, the network with n splitters and m outputs with different labels can be described by an absorbing Markov chain, where the first n states are transient states and the last m states are absorbing states. And we have

$$\begin{aligned} \textstyle\sum_{j=1}^{n+m} p_{ij} = 1 \qquad & i = 1, 2, ..., n+m, \\ p_{ij} = 0 \qquad & \forall i > n \text{ and } i \neq j, \\ p_{ii} = 1 \qquad & \forall i > n. \end{aligned}$$

The transition matrix of this Markov chain is given by

$$P = \begin{array}{c} \\ n \\ m \end{array} \begin{array}{c} \begin{array}{cc} n & m \end{array} \\ \begin{pmatrix} Q & R \\ 0 & I \end{pmatrix} \end{array},$$

where Q is an $n \times n$ matrix, R is an $n \times m$ matrix, 0 is an $m \times n$ zeros matrix and I is an $m \times m$ identity matrix.

Let B_{ij} be the probability for an absorbing Markov chain reaching the state $j + n$ if it starts in the transient state i. Then B is an $n \times m$ matrix, and

$$B = (I - Q)^{-1} R.$$

Assume this Markov chain starts from state 1 and let S_j be the probability for it reaching the absorbing state $j + n$. Then S is the distribution of the network

$$S = [1, 0, ..., 0]B = e_1(I - Q)^{-1} R.$$

Given a stochastic flow network, we can use the formula above to calculate its probability distribution. For example, the transition matrix of the network in figure 8.3 is

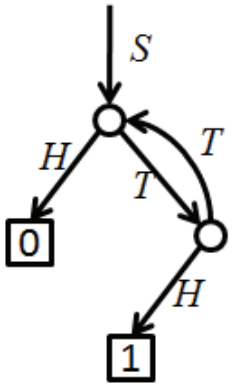


Figure 8.3. The stochastic flow network to generate a $(\frac{2}{3}, \frac{1}{3})$ distribution.

$$P = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

from which we can obtain the probability distribution

$$S = e_1(I - Q)^{-1}R = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \end{pmatrix}.$$

8.2.3 Mason's Rule

Mason's gain rule is a method used in control theory to find the transfer function of a given control system. It can be applied to any signal flow graph. Generally, we describe it as follows (see more details about Mason's rule in [120]):

Let $H(z)$ denote the transfer function of a signal flow graph. Define the following notations:

1. $\Delta(z)$ = determinant of the graph.

2. L = number of forward paths, with $P_k(z)$, $1 \leq k \leq L$ denoting the forward path gains.

3. $\Delta_k(z)$ = determinant of the graph that remains after deleting the kth forward path $P_k(z)$.

To calculate the determinant of a graph $\Delta(z)$, we list all the loops in the graph and their gains

denoted by L_i, all pairs of nontouching loops L_iL_j, all pairwise nontouching loops $L_iL_jL_k$, and so forth. Then

$$\Delta(z) = 1 - \sum_{i:\text{loops}} L_i + \sum_{(i,j):\text{nontouching}} L_iL_j - ...$$

The transfer function is

$$H(z) = \frac{\sum_{k=1}^{L} P_k(z)\Delta_k(z)}{\Delta(z)},$$

called Mason's rule.

Let us treat a stochastic flow network as a control system with input $U(z) = 1$. Applying Mason's rule to this system, we can get the probability that one token reaches output k with $1 \leq k \leq m$. Also having the network in figure 8.3 as an example: in this network, we want to calculate the probability for a token to reach output 1 (for short, we call it the probability of 1). Since there is only one loop with gain $= \frac{1}{4}$ and only one forward path with forward gain $\frac{1}{4}$, we can obtain that the probability of 1 is

$$P = \frac{\frac{1}{4}}{1 - \frac{1}{4}} = \frac{1}{3},$$

which accords with the result of absorbing Markov chains. In fact, it can be proved that the Mason's rule and the matrix form based on absorbing Markov chains are equivalent.

8.3 Optimal-Sized Construction and Feedback

In this section we present an optimal-sized construction of stochastic flow networks. It consists of splitters with probability 1/2 and computes an arbitrary rational probability. We demonstrate that feedback (loops) in stochastic flow networks significantly enhance their expressibility. To see that, let us first study stochastic flow networks without loops, and then those with loops.

8.3.1 Loop-Free Networks

Here, we want to study the expressive power of loop-free networks. We say that there are no loops in a network if no tokens can pass any position in the network more than once. For loop-free networks,

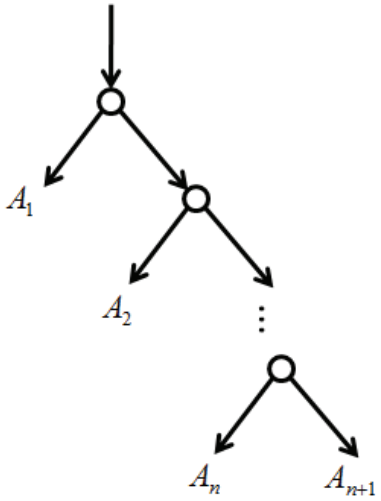


Figure 8.4. Tree structure used to realize probability $\frac{x}{2^n}$ for an integer $x(0 \leq x \leq 2^n)$.

we have the following theorem:

Theorem 8.1. *For a loop-free network with n $\frac{1}{2}$-splitters, any probability $\frac{x}{2^n}$ with integer $x(0 \leq x \leq 2^n)$ can be realized, and only probabilities $\frac{x}{2^n}$ with integer $x(0 \leq x \leq 2^n)$ can be realized.*

Proof. (a) In order to prove that all probability $\frac{x}{2^n}$ with integer $x(0 \leq x \leq 2^n)$ can be realized, we only need to provide the constructions of the networks.

1. Construct a tree, as shown in figure 8.4. In this tree structure, each token will reach $A_i(1 \leq i \leq n)$ with probability 2^{-i}, and reach A_{n+1} with probability 2^{-n}.

2. Let $\frac{x}{2^n} = \sum_{i=1}^{n} \gamma_i 2^{-i}$, where $\gamma_i = 0$ or 1. For each j with $1 \leq j \leq n$, $\gamma_j = 1$, we connect A_j to output 0; otherwise, we connect A_j to output 1. Then we connect A_{n+1} to output 1. Eventually, the probability for a token to reach output 0 is

$$P = \sum_{j=1}^{n} \frac{\gamma_{n-j}}{2^j} = \sum_{i=0}^{n-1} \frac{\gamma_i}{2^{n-i}} = \frac{x}{2^n}.$$

Using the procedure above, we can construct a network such that its probability is $\frac{x}{2^n}$. Actually, it is a special case of Knuth and Yao's construction [71].

(b) Now, we prove that only probability $\frac{x}{2^n}$ with integer $x(0 \leq x \leq 2^n)$ can be realized. If this

is true, then $\frac{x}{2^n}$ with odd x cannot be realized with less than n splitters. It means that in the construction above, the network size n is optimal.

According to Mason's rule, for a network without loops, the probability for a token reaching one output is

$$P = \sum_k P_k,$$

where P_k is the path gain of a forward path from the root to the output. Given n splitters, the length of each forward path should be at most n. Otherwise, there must be a loop along this forward path (have to pass the same splitter for at least two times). For each k, P_k can be written as $\frac{x_k}{2^n}$ for some x_k. As a result, we can get that P can be written as $\frac{x}{2^n}$ for some x. □

8.3.2 Networks with Loops

We showed that stochastic flow networks without loops can only realize binary probabilities. Here, we show that feedback (loops) plays an important rule in enhancing their expressibility. For example, with feedback, we can realize probability $\frac{2}{3}$ with only two splitters, as shown in figure 8.3. But without loops, it is impossible (or requires an infinite number of splitters) to realize $\frac{2}{3}$. To study the property of stochastic flow networks with loops, we first give the following lemma, whose proof will be given in next subsection.

Lemma 8.2. *Given Q an $n \times n$ matrix with each entry in $\{0, \frac{1}{2}, 1\}$, such that sum of each row is at most 1, then we have $0 \leq det(I - Q) \leq 1$, where I is an identity matrix and $det(\cdot)$ is the determinant of a matrix.*

For any desired rational probability $\frac{a}{b}$ with integers $0 \leq a \leq b \leq 2^n$, we have the following theorem:

Theorem 8.3. *For a network with n $\frac{1}{2}$-splitters, any rational probability $\frac{a}{b}$ with integers $0 \leq a \leq b \leq 2^n$ can be realized, and only rational probabilities $\frac{a}{b}$ with integers $0 \leq a \leq b \leq 2^n$ can be realized.*

Proof. (a) We prove that all rational probability $\frac{a}{b}$ with integers $0 \leq a \leq b \leq 2^n$ can be realized. When $b = 2^n$, the problem becomes trivial due to the result of theorem 8.1. In the following proof, without loss of generality (w.l.o.g), we only consider the case in which $2^{n-1} < b < 2^n$ for some n.

We first show that all probability distributions $\{\frac{x}{2^n}, \frac{y}{2^n}, \frac{z}{2^n}\}$ with integers x, y, z such that $(x + y + z = 2^n)$ can be realized with n splitters. Now let us construct the network iteratively.

When $n = 1$, by enumerating all the possible connections, we can verify that all the following probability distributions can be realized:

$$\{0, 0, 1\}, \{0, 1, 0\}, \{1, 0, 0\}, \{0, \frac{1}{2}, \frac{1}{2}\}, \{\frac{1}{2}, 0, \frac{1}{2}\}, \{\frac{1}{2}, \frac{1}{2}, 0\}.$$

So all the probability distributions $\{\frac{x}{2}, \frac{y}{2}, \frac{z}{2}\}$ with integers x, y, z such that $(x + y + z = 2)$ can be realized.

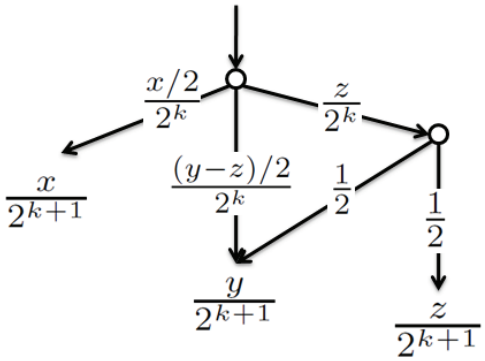
Assume that all the probability distribution $\{\frac{x}{2^k}, \frac{y}{2^k}, \frac{z}{2^k}\}$ with integers x, y, z s.t. $(x+y+z = 2^k)$ can be realized by a network with k splitters, then we show that any desired probability distribution $\{\frac{x}{2^{k+1}}, \frac{y}{2^{k+1}}, \frac{z}{2^{k+1}}\}$ s.t. $x+y+z = 2^{k+1}$ can be realized with one more splitter. Since $x+y+z = 2^{k+1}$, at least one of x, y, z is even. W.l.o.g, we let x be even. Then there are two cases to consider: either both y and z are even, or both y and z are odd.

When both y and z are even, the problem is trivial since the desired probability distribution can be written as $\{\frac{x/2}{2^k}, \frac{y/2}{2^k}, \frac{z/2}{2^k}\}$, which can be realized by a network with k splitters.

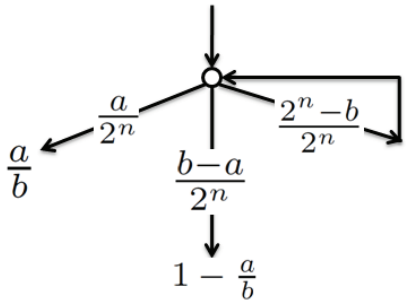
When both y and z are odd, w.l.o.g, we assume that $z \leq y$. In this case, we construct a network to realize probability distribution $\{\frac{x/2}{2^k}, \frac{(y-z)/2}{2^k}, \frac{z}{2^k}\}$ with k splitters. By connecting the last output with probability $\frac{z}{2^k}$ to an additional splitter, we can get a new distribution $\{\frac{x/2}{2^k}, \frac{(y-z)/2}{2^k}, \frac{z}{2^{k+1}}, \frac{z}{2^{k+1}}\}$. If we consider the second and the third output as a single output, then we can get a new network in figure 8.5(a), whose probability distribution is $\{\frac{x}{2^{k+1}}, \frac{y}{2^{k+1}}, \frac{z}{2^{k+1}}\}$.

Hence, for any probability distribution $\{\frac{x}{2^n}, \frac{y}{2^n}, \frac{z}{2^n}\}$ with $x+y+z = 2^n$, we can always construct a network with n splitters to realize it.

Now, in order to realize probability $\frac{a}{b}$ with $2^{n-1} < b < 2^n$ for some n, we can construct a network



(a) The network to realize $\{\frac{x}{2^{k+1}}, \frac{y}{2^{k+1}}, \frac{z}{2^{k+1}}\}$ iteratively.



(b) The network to realize $\{\frac{a}{b}, 1-\frac{a}{b}\}$.

Figure 8.5. The network to realize $\{\frac{a}{b}, 1-\frac{a}{b}\}$ with feedback.

with probability distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$ with n splitters and connect the last output (output 2) to the starting point of the network, as shown in figure 8.5(b). Using the method of absorbing Markov chains, we can obtain that the probability for a token to reach output 0 is $\frac{a}{b}$. A simple understanding for this result is that: (1) the ratio of the probabilities for a token to reach the first output and the second output is $\frac{a}{2^n} : \frac{b-a}{2^n}$ that equals $a : (b-a)$ (2) the sum of these two probabilities is 1, since the tokens will finally reach one of the two outputs.

(b) Now we prove that with n splitters, only rational probability $\frac{a}{b}$ with integers $0 \leq a \leq b \leq 2^n$ can be realized. For any flow network with n splitters, it can be described as an absorbing Markov

chain with n transient states and 2 absorbing states, whose transition matrix P can be written as

$$P = \begin{pmatrix} p_{11} & \cdots & p_{1n} & p_{1(n+1)} & p_{1(n+2)} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ p_{n1} & \cdots & p_{nn} & p_{n(n+1)} & p_{n(n+2)} \\ 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix},$$

where each row consists of two $\frac{1}{2}$ entries and n zeros.

Let

$$Q = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix}, R = \begin{pmatrix} p_{1(n+1)} & p_{1(n+2)} \\ \vdots & \vdots \\ p_{n(n+1)} & p_{n(n+2)} \end{pmatrix},$$

then the probability distribution of the network can be written as

$$e_1(I-Q)^{-1}R.$$

In order to prove the result in the theorem, we only need to prove that $(I-Q)^{-1}R$ can be written as $\frac{1}{b}A$ with $b \leq 2^n$, where A is an integer matrix (all the entries in A are integers).

Let $K = I - Q$, we know that K is invertible if and only $det(K) \neq 0$. In this case, we have

$$(K^{-1})_{ij} = \frac{K_{ji}}{det(K)},$$

where K_{ji} is defined as the determinant of the square matrix of order $(n-1)$ obtained from K by removing the ith row and the jth column multiplied by $(-1)^{i+j}$.

Since each entry of K is chosen from $\{0, \frac{1}{2}, 1\}$, K_{ji} can be written as $\frac{k_{ji}}{2^{n-1}}$ for some integer k_{ji} and $det(K)$ can be written as $\frac{b}{2^n}$ for some integer b. According to lemma 8.2, we have $0 \leq det(K) \leq 1$, which leads us to $0 < b \leq 2^n$ (note that $det(K) \neq 0$).

Then, we have that

$$\begin{aligned} K^{-1} &= \frac{1}{DEP(K)} \begin{pmatrix} K_{11} & K_{21} & \dots & K_{n1} \\ K_{12} & K_{22} & \dots & K_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ K_{1n} & K_{2n} & \dots & K_{nn} \end{pmatrix} \\ &= \frac{2}{b} \begin{pmatrix} k_{11} & k_{21} & \dots & k_{n1} \\ k_{12} & k_{22} & \dots & k_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{1n} & k_{2n} & \dots & k_{nn} \end{pmatrix}. \end{aligned}$$

Since each entry of R is also in $\{0, \frac{1}{2}, 1\}$, we know that

$$2R = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ r_{n1} & r_{n2} \end{pmatrix}$$

is an integer matrix.

As a result

$$K^{-1}R = \frac{2R}{b} \begin{pmatrix} k_{11} & k_{21} & \dots & k_{n1} \\ k_{12} & k_{22} & \dots & k_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{1n} & k_{2n} & \dots & k_{nn} \end{pmatrix}$$

$$= \frac{1}{b}\begin{pmatrix} k_{11} & k_{21} & \dots & k_{n1} \\ k_{12} & k_{22} & \dots & k_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{1n} & k_{2n} & \dots & k_{nn} \end{pmatrix}\begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ r_{n1} & r_{n2} \end{pmatrix}$$
$$= \frac{A}{b},$$

where each entry of A is an integer. So all the probabilities in the final distribution are of the form $\frac{a}{b}$.

This completes the proof. □

Based on the method in the theorem above, we can realize any arbitrary rational probability with an optimal-sized network. The construction has two steps:

1. Construct a network with output distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$ iteratively using at most n splitters.

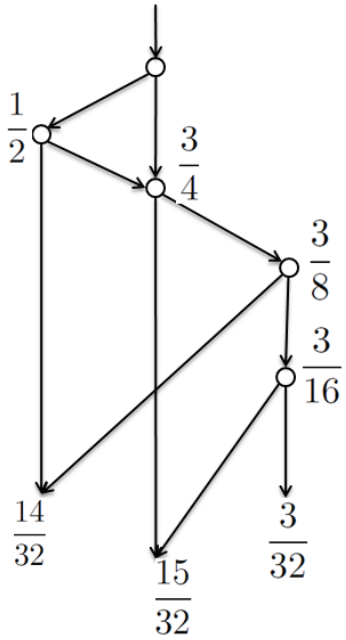
2. Connect the last output to the starting point, such that the distribution of the resulting network is $\{\frac{a}{b}, \frac{b-a}{b}\}$.

When $b = 2^n$ for some n, the construction above is exactly the generating tree construction in the Knuth and Yao's scheme as described in section 8.2. Now, assume we want to realize probability $\frac{14}{29}$. We can first generate a probability distribution $\{\frac{14}{32}, \frac{15}{32}, \frac{3}{32}\}$, which can be realized by adding one splitter to a network with probability distribution $\{\frac{7}{16}, \frac{6}{16}, \frac{3}{16}\}$... Recursively, we can have the following probability distributions:

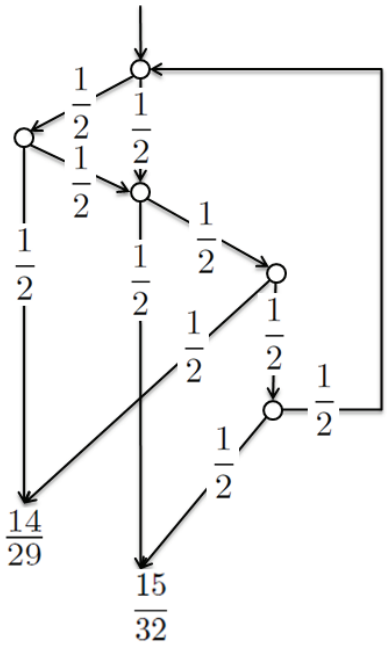
$$\{\frac{14}{32}, \frac{15}{32}, \frac{3}{32}\} \rightarrow \{\frac{7}{16}, \frac{6}{16}, \frac{3}{16}\} \rightarrow \{\frac{2}{8}, \frac{3}{8}, \frac{3}{8}\}$$

$$\rightarrow \{\frac{1}{4}, 0, \frac{3}{4}\} \rightarrow \{\frac{1}{2}, 0, \frac{1}{2}\}.$$

As a result, we get a network to generate probability distribution $\{\frac{14}{32}, \frac{15}{32}, \frac{3}{32}\}$, as shown in figure 8.6(a), where only 5 splitters are used. Connecting the last output to the starting point results in



(a) The network to realize probability distribution $\{\frac{14}{32}, \frac{15}{32}, \frac{3}{32}\}$.



(b) The network to realize probability $\frac{14}{29}$.

Figure 8.6. The network to realize probability $\frac{14}{29}$.

the network in figure 8.6(b) with probability $\frac{14}{29}$. Comparing the results in theorem 8.3 with those in theorem 8.1, we see that introducing loops into networks can strongly enhance their expressibility.

8.3.3 Proof of Lemma 8.2

Lemma 8.2. *Given Q an $n \times n$ matrix with each entry in $\{0, \frac{1}{2}, 1\}$, such that sum of each row is at most 1, then we have $0 \leq det(I-Q) \leq 1$, where I is an identity matrix and $det(\cdot)$ is the determinant of a matrix.*

Proof. Before proving this lemma, we can see that for any given matrix Q, it has the following properties: For any i, j such that $1 \leq i < j \leq n$, switching the ith row with the jth row then switching the ith column with the jth column, the determinant of $K = I - Q$ stays unchanged. And more, each entry of Q is still from $\{0, \frac{1}{2}, 1\}$ and sum of each row of Q is at most 1. Now, we call the transform above as equivalent transform of Q.

Let us prove this lemma by induction. When $n = 1$, we have that

$$Q = \begin{pmatrix} 0 \end{pmatrix} \text{ or } Q = \begin{pmatrix} \frac{1}{2} \end{pmatrix} \text{ or } Q = \begin{pmatrix} 1 \end{pmatrix}.$$

In all of the cases, we have $0 \leq det(I - Q) \leq 1$.

Assume the result of the lemma hold for $(n-1) \times (n-1)$ matrix, we want to prove that this result also holds for $n \times n$ matrix. Now, given a $n \times n$ matrix Q, according to the definition in the lemma, we know that the sum of all the entries in Q is at most n. As a result, there exists a column such that the sum of the entries in the column is at most 1. Using equivalent transform, we have that

- The sum of the entries in the 1st column of Q is at most 1.
- The sum of the entries in each row of Q is at most 1.

Now, for the 1st column of $I - Q$, let us continue using the equivalent transform to move all the nonzero entries to the beginning of this column. The possible nonzero entry set of the 1st column of $I - Q$ is

$$\phi, \{\frac{1}{2}\}, \{1\}, \{\frac{1}{2}, -\frac{1}{2}\}, \{1, -\frac{1}{2}\}, \{1, -1\}, \{1, -\frac{1}{2}, -\frac{1}{2}\}.$$

The first three cases, the result in the lemma can be easily proved. In the following proof, we only consider the other cases (let C_1 denote the nonzero entry set for the 1^{st} column of $I - Q$) :

(1) $C_1 = \{\frac{1}{2}, -\frac{1}{2}\}$.

In this case, we can write Q as

$$Q = \begin{pmatrix} \frac{1}{2} & A \\ \frac{1}{2} & B \\ O & C \end{pmatrix}$$

where A has at most one nonzero entry $-\frac{1}{2}$, the same as B.

Let

$$E_1 = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \end{pmatrix},$$

$$I_1 = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix},$$

then we have

$$\begin{aligned} & DEP(I-Q) \\ = \; & \frac{1}{2}\det\begin{pmatrix} -A \\ I_1 - C \end{pmatrix} + \frac{1}{2}\det\begin{pmatrix} E_1 - B \\ I_1 - C \end{pmatrix} \\ = \; & \frac{1}{2}\det\begin{pmatrix} E_1 - A - B \\ I_1 - C \end{pmatrix} \\ = \; & \frac{1}{2}\det(I - \begin{pmatrix} A+B \\ C \end{pmatrix}). \end{aligned}$$

Let $D = A + B$, since both A and B has at most one nonzero entry $\frac{1}{2}$, we know that each entry of D is from $\{0, \frac{1}{2}, 1\}$, and the sum of all the entries is at most one. According to our assumption,

we know that

$$0 \leq \det(I - \begin{pmatrix} D \\ C \end{pmatrix} \leq 1.$$

As a result, we have

$$0 \leq \det(I - Q) \leq \frac{1}{2}.$$

(2) $C_1 = \{1, -\frac{1}{2}\}$.

In this case, we can write Q as

$$Q = \begin{pmatrix} 0 & A \\ \frac{1}{2} & B \\ O & C \end{pmatrix}.$$

Then

$$\begin{aligned} & DEP(I-Q) \\ = & \frac{1}{2}\det\begin{pmatrix} -A \\ I_1 - C \end{pmatrix} + \det\begin{pmatrix} E_1 - B \\ I_1 - C \end{pmatrix} \\ = & \frac{1}{2}\det\begin{pmatrix} 2E_1 - A - 2B \\ I_1 - C \end{pmatrix} \\ = & \frac{1}{2}\det(I - \begin{pmatrix} A \\ C \end{pmatrix}) + \frac{1}{2}\det(I - \begin{pmatrix} 2B \\ C \end{pmatrix}). \end{aligned}$$

According to our assumption

$$0 \leq \det(I - \begin{pmatrix} A \\ C \end{pmatrix}) \leq 1,$$

$$0 \leq \det(I - \begin{pmatrix} 2B \\ C \end{pmatrix}) \leq 1,$$

so $det(I-Q)$ is also bounded by 0 and 1.

(3) $C_1 = \{1, -1\}$.

Using the same argument as case (1), we can get the result in the lemma.

(4) $C_1 = \{1, -\frac{1}{2}, -\frac{1}{2}\}$.

In this case, we can write Q as

$$Q = \begin{pmatrix} 0 & A \\ \frac{1}{2} & B \\ \frac{1}{2} & C \\ O & D \end{pmatrix}.$$

Let

$$E_2 = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \end{pmatrix},$$

$$I_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}.$$

Then

$$I - Q = \begin{pmatrix} 1 & -A \\ -\frac{1}{2} & E_1 - B \\ -\frac{1}{2} & E_2 - C \\ O & I_2 - D \end{pmatrix},$$

$$
\begin{aligned}
& DEP(I-Q) \\
= \quad & \det\begin{pmatrix} E_1 - B \\ E_2 - C \\ I_2 - D \end{pmatrix} + \frac{1}{2}\det\begin{pmatrix} -A \\ E_2 - C \\ I_2 - D \end{pmatrix} - \frac{1}{2}\det\begin{pmatrix} -A \\ E_1 - B \\ I_2 - D \end{pmatrix} \\
= \quad & \frac{1}{2}\det\begin{pmatrix} E_1 - B - A \\ E_2 - C \\ I_2 - D \end{pmatrix} + \frac{1}{2}\det\begin{pmatrix} E_1 - B \\ E_2 - C - A \\ I_2 - D \end{pmatrix}.
\end{aligned}
$$

Now, we can write $A = E + F$ such that both E and F has at most one nonzero entry, which is $\frac{1}{2}$. Therefore,

$$
\begin{aligned}
& DEP(I-Q) \\
= \quad & \frac{1}{2}\det\begin{pmatrix} E_1 - B - E - F \\ E_2 - C \\ I_2 - D \end{pmatrix} + \frac{1}{2}\det\begin{pmatrix} E_1 - B \\ E_2 - C - E - F \\ I_2 - D \end{pmatrix},
\end{aligned}
$$

where

$$
\begin{aligned}
& \det\begin{pmatrix} E_1 - B - E - F \\ E_2 - C \\ I_2 - D \end{pmatrix} \\
= \quad & \det\begin{pmatrix} E_1 - B - E \\ E_2 - C - F \\ I_2 - D \end{pmatrix} + \det\begin{pmatrix} -F \\ E_2 - C \\ I_2 - D \end{pmatrix} + \det\begin{pmatrix} E_1 - B - E \\ F \\ I_2 - D \end{pmatrix},
\end{aligned}
$$

and

$$\begin{aligned} & \det \begin{pmatrix} E_1 - B \\ E_2 - C - E - F \\ I_2 - D \end{pmatrix} \\ = \quad & \det \begin{pmatrix} E_1 - B - F \\ E_2 - C - E \\ I_2 - D \end{pmatrix} + \det \begin{pmatrix} E_1 - B \\ -F \\ I_2 - D \end{pmatrix} + \det \begin{pmatrix} F \\ E_2 - C - E \\ I_2 - D \end{pmatrix}. \end{aligned}$$

Finally, we can get that

$$\begin{aligned} & DEP(I - Q) \\ = \quad & \frac{1}{2} \det[I - \begin{pmatrix} B + E \\ C + F \\ D \end{pmatrix}] + \frac{1}{2} \det[I - \begin{pmatrix} B + F \\ C + E \\ D \end{pmatrix}]. \end{aligned}$$

According to our assumption, we have that

$$0 \leq \det[I - \begin{pmatrix} B + E \\ C + F \\ D \end{pmatrix}] \leq 1,$$

$$0 \leq \det[I - \begin{pmatrix} B + F \\ C + E \\ D \end{pmatrix}] \leq 1.$$

Therefore, the result of this lemma holds.

This completes the proof. □

8.4 Expected Latency of Optimal Construction

Besides of network size, anther important issue of a stochastic flow network is the expected operating time, or we call it expected latency, defined as the expected number of splitters a token need to pass before reaching one of the outputs. For the optimal-sized construction proposed in the above section, we have the following results about its expected latency.

Theorem 8.4. *Given a network with rational probability $\frac{a}{b}$ with $b \leq 2^n$ constructed using the optimal-sized construction, its expected latency ET is upper bounded by*[2]

$$ET \leq (\frac{3n}{4} + \frac{1}{4})\frac{2^n}{b} < \frac{3n}{2} + \frac{1}{2}.$$

Proof. For the optimal-sized construction, we first prove that the expected latency of the network with distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$ is bounded by $\frac{3n}{4} + \frac{1}{4}$.

Let us prove this by induction. When $n = 0$ or $n = 1$, it is easy to see that this conclusion is true. Assume when $n = k$, this conclusion is true, we want to show that the conclusion still holds for $n = k+2$. Note that in the optimal-sized construction, a network with size $k+2$ can be constructed by adding two more splitters to a network with size k. Let T_k denote the latency of the network with size k, then

$$E[T_{k+2}] = E[T_k] + p_1 + p_2,$$

where p_1 is the probability for a token to reach the first additional splitter and p_2 is the probability for a token to reach the second additional splitter. Assume the distribution of the network with size k is $\{q_1, q_2, q_3\}$, then

$$p_1 + p_2 \leq \max_{i \neq j}(q_i + (\frac{q_i}{2} + q_j)) \leq \frac{3}{2}.$$

So the conclusion is true for $n = k + 2$. By induction, we know that it holds for all $n \in \{0, 1, 2, ...\}$.

Secondly, we prove that if the expected latency of the network with distribution $\{q_1, q_2, q_3\}$ is

[2] By making the construction more sophisticated, we can reduce the upper bound to $(\frac{n}{2} + \frac{3}{4})\frac{2^n}{b}$.

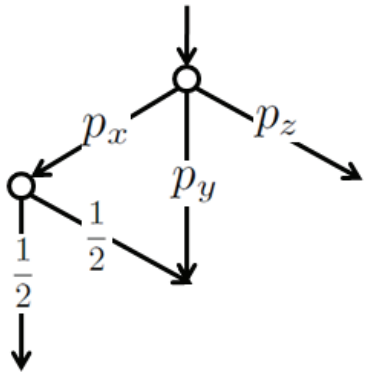


Figure 8.7. Illustration for the construction of a network with unbounded expected latency.

Table 8.1. The comparison of different construction, here $\frac{2^n}{b} < 2$

	Optimal-Sized Construction	Size-Relaxed Construction	Latency-Oriented Construction
Network size	$\leq n$	$\leq n+3$	$\leq 2(n-1)$
Expected latency	$\leq (\frac{3n}{4} + \frac{1}{4})\frac{2^n}{b}$	$\leq 6\frac{2^n}{b}$	$\leq 3.585\frac{2^n}{b}$

ET', then by connecting its last output to its starting point, we can get a network such that its expected latency is $ET = \frac{ET'}{q_1+q_2}$. This conclusion can be obtained immediately from

$$ET = ET' + q_3(ET).$$

This completes the proof. □

Theorem 8.5. *There exists a network of size n constructed using the optimal-sized construction such that its expected latency ET is lower bounded by*

$$ET \geq \frac{n}{3} + \frac{2}{3}.$$

Proof. We only need to construct a network with distribution $\{\frac{x}{2^n}, \frac{y}{2^n}, \frac{z}{2^n}\}$ for some integers x, y, z such that its expected latency is lower bounded by $\frac{n}{3} + \frac{2}{3}$.

Let us construct such a network in the following way: Starting from a network with single

splitter, and at each step adding one more splitter. Assume the current distribution is $\{p_x, p_y, p_z\}$ with $p_x \geq p_y \geq p_z$ (if this is not true, we can change the order of the outputs), then we can add an additional splitter to p_x as shown in figure 8.7. Iteratively, with n splitters, we can construct a network with distribution $\{\frac{x}{2^n}, \frac{y}{2^n}, \frac{z}{2^n}\}$ for some integers x, y, z and its expected latency is more than $\frac{n}{3} + \frac{2}{3}$.

By connecting one output with probability smaller than $\frac{1}{2}$ to the starting point, we can get such a network. □

The theorems above show that the upper bound of the expected latency of a stochastic flow network based on the optimal-sized construction is not well bounded. However, this upper bound only reflects the worst case. That does not mean that the optimal-sized construction always has a bad performance in expected latency when the network size is large. Let us consider the case that the target probability is $\frac{a}{b}$ with $b = 2^n$ for some n. In this case, the optimal-sized construction leads to a tree structure, whose expected latency can be written as

$$\begin{aligned} ET &= \sum_{i=1}^{n} \frac{i}{2^i} + \frac{n}{2^n} \\ &= [\sum_{i=1}^{n} x^{i+1}]' - \sum_{i=1}^{n-1} \frac{i}{2^i} \\ &= [\frac{x^2 - x^{n+2}}{1-x}]' - \frac{x - x^n}{1-x} \\ &= 2 - \frac{1}{2^{n-1}}, \end{aligned}$$

which is well bounded by 2.

8.5 Alternative Constructions

In the last section, we show that the expected latency of a stochastic flow network based on the optimal-sized construction is not always well bounded. In this section, we give two other constructions, called size-relaxed construction and latency-oriented construction. They take both the network size and the expected latency in consideration. Table 8.1 shows the summary of the results

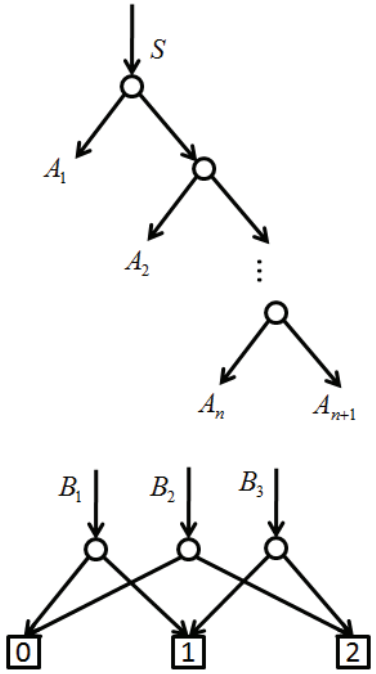


Figure 8.8. The framework to realize probability $\frac{a}{b}$.

in this section, from which we can see that there is a trade-off between the upper bound on the network size and the upper bound on the expected latency.

8.5.1 Size-Relaxed Construction

Assume that the desired probability is $\frac{a}{b}$ with $2^{n-1} < b \leq 2^n$ for some n. In this subsection, we give a construction, called size-relaxed construction for realizing $\frac{a}{b}$, with at most $n+3$ splitters and its expected latency is well bounded by a constant.

Assume a and b are relatively prime, and let $c = b - a$. Then $\frac{a}{2^n}$ and $\frac{c}{2^n}$ can be represented as binary expansions, namely

$$\frac{a}{2^n} = \sum_{i=1}^{n} a_i 2^{-i},$$

$$\frac{c}{2^n} = \frac{b-a}{2^n} = \sum_{i=1}^{n} c_i 2^{-i}.$$

Let us start from the structure in figure 8.8, where the probability of A_i with $1 \leq i \leq n$ is 2^{-i} and the probability of A_{n+1} is 2^{-n}. We connect A_i with $1 \leq i \leq n+1$ to one of $\{B_1, B_2, B_3$ and output $2\}$, such that the probability distribution of the outputs is $\{\frac{a}{2^{n+1}}, \frac{b-a}{2^{n+1}}, \frac{2^{n+1}-b}{2^{n+1}}\}$. Based on

the values of a_i, c_i with $1 \leq i \leq n$ (from binary expansions of $\frac{a}{2^n}$ and $\frac{c}{2^n}$), we have the following rules for these connections:

1. If $a_i = c_i = 1$, connect A_i with B_1.
2. If $a_i = 1, c_i = 0$,connect A_i with B_2.
3. If $a_i = 0, c_i = 1$, connect A_i with B_3.
4. If $a_i = c_i = 0$, connect A_i with output 2.
5. Connect A_{n+1} with output 2.

Assume that the probability for a token to reach B_j with $1 \leq j \leq 3$ is $P(B_j)$, then we have

$$P(B_1) = \sum_{i=1}^{n} I_{(a_i=c_i=1)} 2^{-i},$$

$$P(B_2) = \sum_{i=1}^{n} I_{(a_i=1,c_i=0)} 2^{-i},$$

$$P(B_3) = \sum_{i=1}^{n} I_{(a_i=0,c_i=1)} 2^{-i},$$

where $I_\phi = 1$ if and only if ϕ is true, otherwise $I_\phi = 0$.

As a result, the probability for a token to reach the first output is

$$P_1 = \frac{1}{2}(P(B_1) + P(B_2)) = \frac{1}{2} \sum_{i=1}^{n} I_{(a_i=1)} 2^{-i} = \frac{a}{2^{n+1}}.$$

Similarly, the probability for a token to reach the second output is

$$P_2 = \frac{b-a}{2^{n+1}}.$$

So far, we get that the distribution of the network is $\{\frac{a}{2^{n+1}}, \frac{b-a}{2^{n+1}}, \frac{2^{n+1}-b}{2^{n+1}}\}$. Similar to theorem 8.3, by connecting the output 2 to the starting point, we get a new network with probability $\frac{a}{b}$. Note that compared to the optimal-sized construction, 3 more splitters are used in the size-relaxed

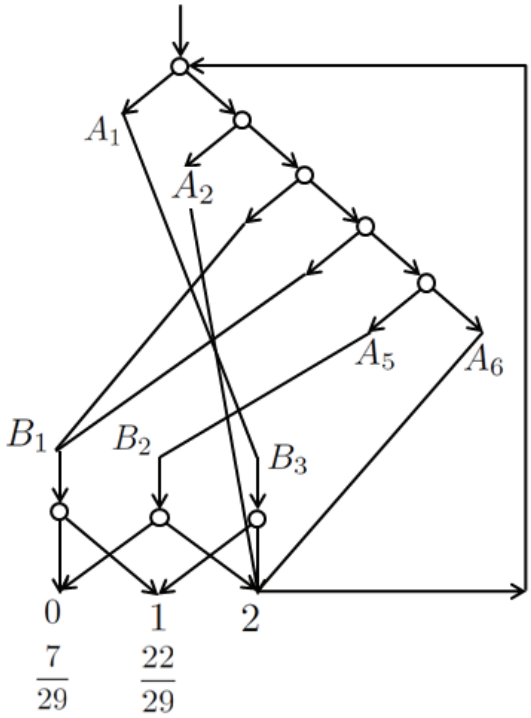


Figure 8.9. The network to realize probability $\frac{7}{29}$.

construction to realize the desired probability. But it has a much better upper bound on the expected latency as shown in the following theorem.

Theorem 8.6. *Given a network with probability* $\frac{a}{b}$ *(*$2^{n-1} < b < 2^n$*) constructed using the size-relaxed construction, its expected latency ET is bounded by*

$$ET \leq 6\frac{2^n}{b} < 12.$$

Proof. First, without the feedback, the expected latency for a token to reach B_1, B_2, B_3 or output 2 is less than 2. This can be obtained from the example in the last section. As a result, without the feedback, the expected latency for a token to reach one of the outputs is less than 3. Finally, we can get the theorem. □

Let us give an example of the size-relaxed construction. Assume the desired probability is $\frac{7}{29}$,

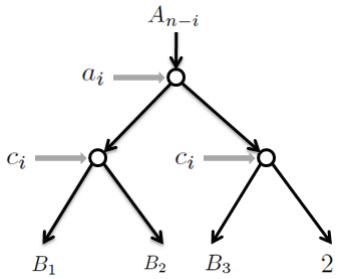


Figure 8.10. The deterministic device to control flow in UPI.

then we can write $\frac{a}{2^n}$ and $\frac{b-a}{2^n}$ into binary expansions:

$$\begin{aligned} \frac{a}{2^n} &= 0.00111, \\ \frac{b-a}{2^n} &= 0.10110. \end{aligned}$$

According to the rules above, we connect A_1 to B_3, A_2 to output 2, and so on. After connecting output 2 to the starting point, we can get a network with probability $\frac{7}{29}$, as shown in figure 8.9.

Another advantage of the size-relaxed construction is that from which we can build an Universal Probability Generator (UPI) efficiently with $a_i, c_i (1 \leq i \leq n)$ as inputs, such that its probability output is $\frac{a}{a+c} = \frac{a}{b}$. The definition and description of UPI can be found in [134]. Instead of connecting A_i with $1 \leq i \leq n$ to one of $\{B_1, B_2, B_3$ and output $2\}$ directly, we insert a deterministic device as shown in figure 8.10. At each node of this device, if its corresponding input is 1, all the incoming tokens will exit the left outgoing edge. If the input is 0, all the incoming tokens will exit the right outgoing edge. As a result, the connections between A_i and $\{B_1, B_2, B_3$, Output $2\}$ are automatically controlled by inputs a_i and c_i with $1 \leq i \leq n$. Finally, we can get an Universal Probability Generator (UPI), whose output probability is

$$\frac{\sum_{i=1}^{n} a_i 2^{-i}}{\sum_{i=1}^{n} (a_i + c_i) 2^{-i}} = \frac{a}{a+c} = \frac{a}{b}.$$

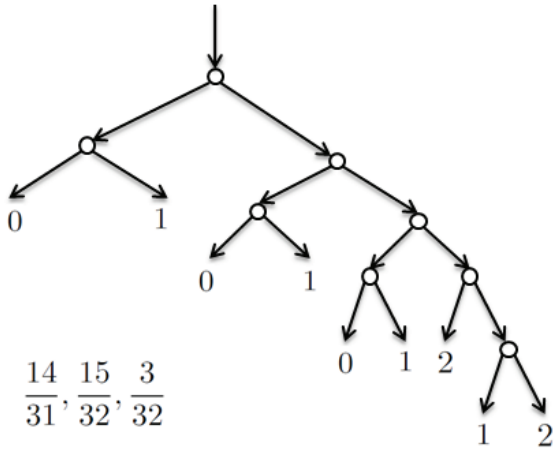


Figure 8.11. The network to realize $\{\frac{14}{32}, \frac{15}{32}, \frac{3}{32}\}$ using Knuth and Yao's scheme.

8.5.2 Latency-Oriented Construction

In this subsection, we propose another construction, called latency-orient construction. It uses more splitters than the size-relaxed construction, but achieves a better upper bound on the expected latency. Similar to the optimal-sized construction, this construction is first trying to realize the distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$, and then connecting the last output to the starting point. The difference is that in the latency-oriented construction, this distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$ is realized by applying Knuth and Yao's scheme [71] that was introduced in the section of preliminaries.

Let us go back to the example of realizing probability $\frac{14}{29}$. According to Knuth and Yao's scheme, we need first find the atoms for the binary expansions of $\frac{14}{32}, \frac{15}{32}, \frac{3}{32}$, i.e.,

$$\frac{14}{32} \rightarrow (\frac{1}{4}, \frac{1}{8}, \frac{1}{16}),$$

$$\frac{15}{32} \rightarrow (\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}),$$

$$\frac{3}{32} \rightarrow (\frac{1}{16}, \frac{1}{32}).$$

Then we allot these atoms to a binary tree, as shown in figure 8.11. In this tree, the probability for a token to reach outputs labeled 0 is $\frac{14}{32}$, the probability for a token to reach outputs labeled 1 is $\frac{15}{32}$, and the probability for a token to reach outputs labeled 2 is $\frac{3}{32}$. If we connect the outputs

labeled 2 to the starting point, the desired probability $\frac{14}{29}$ can be achieved.

Theorem 8.7. *Given a network with probability $\frac{a}{b}$ ($2^{n-1} < b < 2^n$) constructed the latency-oriented construction, its network size is bounded by $2(n-1)$ and its expected latency ET is bounded by*

$$ET \leq (log_2 3 + 2)\frac{2^n}{b} < 7.2.$$

Proof. Let us first consider the network with distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$, which is constructed using Knuth and Yao's scheme.

1) The network size is bounded by $2(n-1)$. To prove this, let us use k_j to denote the number of atoms with value 2^{-j}, and use a_j to denote the number of nodes with depth j in the tree. Then k_j and a_j have the following recursive relations,

$$a_n = k_n,$$

$$a_j = k_j + \frac{a_{j+1}}{2}, \quad \forall 1 \leq j \leq n-1.$$

As a result,

$$\sum_{j=1}^{n} a_j = \sum_{j=1}^{n} k_j + \sum_{j=1}^{n-1} \frac{a_{j+1}}{2}.$$

From which, we can get the total number of atoms in the tree is

$$N = \sum_{j=1}^{n} k_j = \sum_{j=1}^{n} \frac{a_j}{2} + \frac{a_1}{2}.$$

We know that k_j and a_j also satisfy the following constraints,

$$k_j \leq 3, \forall 1 \leq j \leq n,$$

$$a_j \bmod 2 = 0, \forall 1 \leq j \leq n.$$

From $j = n$ to $j = 1$, by induction, we can prove that

$$a_j \leq 4, \forall 1 \leq j \leq n.$$

That is because a_j is even, and if $a_{j+1} \leq 4$, then

$$\frac{a_j}{2} \leq \lfloor \frac{k_j + \frac{a_{j+1}}{2}}{2} \rfloor \leq 2.$$

Since $a_n, a_1 \leq 2$, we can get that

$$N \leq \frac{a_n}{2} + a_1 + \sum_{j=2}^{n-1} \frac{a_j}{2} \leq 2n - 1.$$

To create N atoms, we need $N - 1 = 2(n - 1)$ splitters.

2) The expected latency ET' of the network with distribution $\{\frac{a}{2^n}, \frac{b-a}{2^n}, \frac{2^n-b}{2^n}\}$ is bounded by $ET' \leq (log_2 3 + 2)$. That is because the expected latency ET' is equal to the expected number of fair bits required. According to the result of Knuth and Yao, it is not hard to get this conclusion.

Now we can get a new network by connecting the last output to the starting point. The size of the network is unchanged and the expected latency of the new network is $ET = ET' \frac{2^n}{b}$. So we can get the results in the theorem. □

8.6 Generating Rational Distributions

In this section, we want to generalize our results to generate an arbitrary rational probability distribution $\{q_1, q_2, ..., q_m\}$ with $m \geq 2$. Two different methods will be proposed and studied. The first method is based on Knuth and Yao's scheme and it is a direct generalization of the latency-oriented construction. The second method is based on a construction with a binary-tree structure. At each inner node of the binary tree, one probability is split into two probabilities. As a result, using a binary-tree structure, the probability one can be split into m probabilities (as a distribution) marked

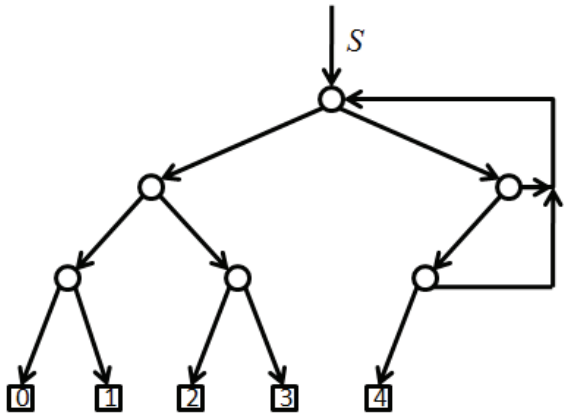


Figure 8.12. The network to realize probability distribution $\{\frac{1}{5}, \frac{1}{5}, ..., \frac{1}{5}\}$.

on all the m leaves. In the rest of this section, we will discuss and analyze these two methods. Since we consider rational probability distributions, we can write $\{q_1, q_2, ..., q_m\}$ as $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$ with integers $a_1, a_2, ..., b$ and b minimized.

8.6.1 Based on Knuth and Yao's Scheme

In order to generate distribution $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$ with $2^{n-1} < b \leq 2^n$ for some n, we can first construct a network with distribution $\{\frac{a_1}{2^n}, \frac{a_2}{2^n}, ..., \frac{a_m}{2^n}, \frac{2^n-b}{2^n}\}$ using Knuth and Yao's scheme. Then by connecting the last output to the starting point, we can obtain a network with distribution $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$. In order to study the properties of this method, we will analyze two extreme cases: (1) $m = b$ and (2) $m \ll b$.

When $m = b$, the target probability distribution can be written as $\{\frac{1}{b}, \frac{1}{b}, ..., \frac{1}{b}\}$. For this distribution, we have the following theorem about the network constructed using the method based on Knuth and Yao's scheme.

Theorem 8.8. *For a distribution $\{\frac{1}{b}, \frac{1}{b}, ..., \frac{1}{b}\}$, the method based on Knuth and Yao's scheme can construct a network with $b + h(b) - 1$ splitters. Here, we assume $b = 2^n - \sum_{i=0}^{n-1} \gamma_i 2^i$ and $h(b) = \sum_{i=0}^{n-1} \gamma_i$.*

Proof. See the network in figure 8.12 as an example of the construction.

First, let us consider a complete tree with depth n. The network size of such a tree (i.e., the

number of parent nodes) is $2^n - 1$, denoted by $N_{complete}$.

Let $N(b)$ be the network size of the construction above to realize distribution $\{\frac{1}{b}, \frac{1}{b}, ..., \frac{1}{b}\}$. Assume

$$2^n - b = 2^{a_1} + 2^{a_2} + ... + 2^{a_H},$$

with $n > a_1 > a_2 > ... > a_H$ is a binary expansion of $2^n - b$, then we can get the difference between the size of the construction and the size of the complete binary tree,

$$\Delta = N_{complete} - N(b) = \sum_{i=1}^{H}(2^{a_i} - 1) = 2^n - b - H.$$

So the network size of the construction $N(b)$ is

$$N(b) = 2^n - 1 - (2^n - b - H) = b + H - 1,$$

where $H = \sum_{i=0}^{n-1} \gamma_i = h(b)$. □

Let $N^*(b)$ be the optimal size of a network that realizes the distribution $\{\frac{1}{b}, \frac{1}{b}, ..., \frac{1}{b}\}$. It is easy to see that $N^*(b) \geq b - 1$. Note that $h(b)$ is at most the number of bits in the binary expansion of $2^n - b$ (which is smaller than b), so we can get the following inequality quickly

$$b - 1 \leq N^*(b) \leq N(b) \leq b - 1 + \log_2 b.$$

It shows that the construction based on Knuth and Yao's scheme is near-optimal when $m = b$. More generally, we believe that when m is large, this construction has a good performance in network size.

For a general m, we have the following results regarding to the network size and expected latency.

Theorem 8.9. *For a distribution* $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$ *with* $b \leq 2^n$*, the method based on Knuth and Yao's scheme can construct a network with at most* $m(n - \lfloor \log_2 m \rfloor + 1)$ *splitters, such that its expected latency ET is bounded by*

$$H(X')\frac{2^n}{b} \leq ET \leq [H(X') + 2]\frac{2^n}{b},$$

where $\frac{2^n}{b} < 2$. $H(X')$ *is the entropy of the distribution* $\{\frac{a_1}{2^n}, \frac{a_2}{2^n}, ..., \frac{a_m}{2^n}, \frac{2^n-b}{2^n}\}$.

Proof. We can use the same argument as that in theorem 8.7. The proof for the expected latency is straightforward. Here, we only briefly describe the proof for the network size.

In the network that realizes $\{\frac{a_1}{2^n}, \frac{a_2}{2^n}, ..., \frac{a_m}{2^n}, \frac{2^n-b}{2^n}\}$, let us use k_j to denote the number of atoms with value 2^{-j}, and use a_j to denote the number of nodes with depth j in the tree. It can be proved that the total number of atoms in the tree is

$$N = \sum_{j=1}^{n} k_j = \sum_{j=1}^{n} \frac{a_j}{2} + \frac{a_1}{2}.$$

Here, the constrains are

$$k_j \leq m+1, \forall 1 \leq j \leq n,$$

$$a_j \text{ is even}, \forall 1 \leq j \leq n.$$

Recursively, we can get that for all $1 \leq j \leq n-1$, $a_j \leq 2m$.

For the first $\lfloor \log_2 2m \rfloor$ levels, we have

$$\sum_{j=1}^{\lfloor \log_2 2m \rfloor} a_j \leq 4m.$$

Hence,

$$\begin{aligned} N &\leq \frac{\sum_{j=1}^{\lfloor \log_2 2m \rfloor} a_j}{2} + \frac{a_1}{2} + \frac{\sum_{j=\lfloor \log_2 2m \rfloor + 1}^{n} a_j}{2} \\ &\leq 2m + 1 + m(n - \lfloor \log_2 2m \rfloor) \\ &\leq m(n - \lfloor \log_2 m \rfloor + 1) + 1. \end{aligned}$$

So we can conclude that $m(n-\lfloor \log_2 m \rfloor+1)$ splitters are enough for realizing $\{\frac{a_1}{2^n}, \frac{a_2}{2^n}, ..., \frac{a_m}{2^n}, \frac{2^n-b}{2^n}\}$ as well as $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$. □

This theorem is a simple generalization of the results in theorem 8.7. Here, the upper bound for

Table 8.2. The comparison of different methods, here $\frac{2^n}{b} < 2$

	Based on Knuth and Yao's Scheme	Based on binary-tree structure
Network size	$\leq m(n - \lfloor \log_2 m \rfloor + 1)$	$\leq (m-1)n$
Expected latency	$\leq (\log_2(m+1) + 2)\frac{2^n}{b}$	$\leq (\log_2 m + 1)ET_{max}$

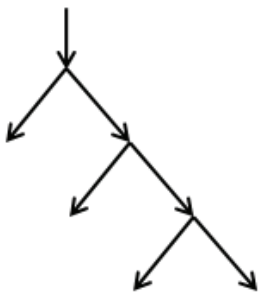
the network size is tight only for small m.

8.6.2 Based on Binary-Tree Structure

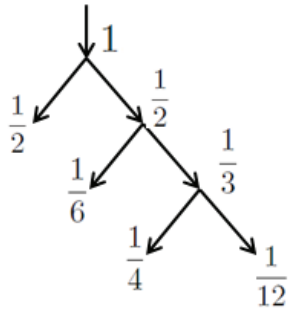
In this subsection, we propose another method to construct a stochastic flow network that generates an arbitrary rational distribution $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$. The idea of this method is based on binary-tree structure. We can describe the method in the following way: We construct a binary tree with m leaves, where the weight of the ith $(1 \leq i \leq m)$ leaf is $q_i = \frac{a_i}{b}$. For each parent (inner) node, its weight is sum of the weights of its two children. Recursively, we can get all the weights of the inner nodes in the tree and the weight of the root node is 1. For each parent node, assume the weights of its two children are w_1 and w_2, then we can replace this parent node by a subnetwork which implements a splitter with probability distribution $\{\frac{w_1}{w_1+w_2}, \frac{w_2}{w_1+w_2}\}$. For each leaf, we treat it as an output. In this new network, a token will reach the ith output with probability q_i.

For example, in order to realize the distribution $\{\frac{1}{2}, \frac{1}{6}, \frac{1}{4}, \frac{1}{12}\}$, we can first generate a binary tree with 4 leaves, as shown in figure 8.13(a). Then according to the method above, we can obtain the weight of each node in this binary tree, see figure 8.13(b). Based on these weights, we replace the three parent nodes with three subnetworks, whose probability distributions are $\{\frac{1}{2}, \frac{1}{2}\}, \{\frac{1}{3}, \frac{1}{3}\}, \{\frac{3}{4}, \frac{1}{4}\}$. Eventually, we construct a network with the desired distribution as shown in figure 8.13(c). It can be implemented with $1 + 2 + 2 = 5$ splitters.

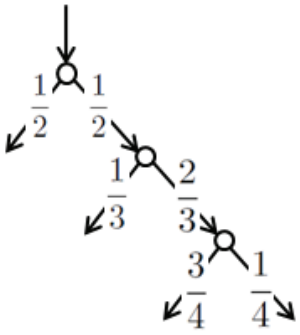
In the procedure above, any binary tree with m leaves works. Among all these binary trees, we need to find one such that the resulting network satisfies our requirements in network size and expected latency. For example, given the target distribution $\{\frac{1}{2}, \frac{1}{6}, \frac{1}{4}, \frac{1}{12}\}$, the binary tree depicted above does not result in an optimal-sized construction. When m is extremely small, such as $3, 4$, we can search all the binary trees with m leaves. However, when m is a little larger, such as 10,



(a) A binary tree with 4 leaves.



(b) Node weights in the binary tree.



(c) The network to realize probability distribution $\{\frac{1}{2}, \frac{1}{6}, \frac{1}{4}, \frac{1}{12}\}$, where $\{\frac{1}{3}, \frac{2}{3}\}, \{\frac{3}{4}, \frac{1}{4}\}$ can be realized using the methods in the sections above.

Figure 8.13. A demonstration of the method based on binary-tree structure.

the number of such binary trees grows exponentially. In this case, the method of brute-force search becomes impractical. In the rest of this section, we will show that Huffman procedure can create a binary tree with good performances in network size and expected latency for most of the cases.

Huffman procedure can be described as follows [27]:

1. Draw m nodes with weights $q_1, q_2, ..., q_m$.

2. Let S denote the set of nodes without parents. Assume node A and node B are the two nodes with the minimal weights in S, then we added a new node as the parent of A and B, with weight $w(A) + w(B)$, where $w(X)$ is the weight of node X.

3. Repeat 2) until the size of S is 1.

Figure 8.14 shows an example of a binary tree constructed by Huffman procedure, when the

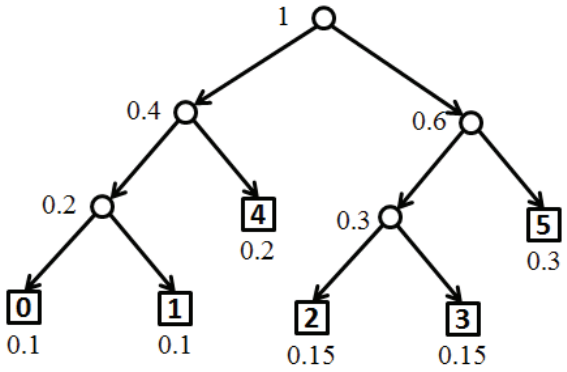


Figure 8.14. The tree constructed using Huffman procedure for $\{0.1, 0.1, 0.15, 0.15, 0.2, 0.3\}$.

desired distribution is $\{0.1, 0.1, 0.15, 0.15, 0.2, 0.3\}$. From [27], we know that when using Huffman procedure we can create a tree with minimal expected path length. Let EL^* denote this minimal expected path length, then its satisfies the following inequality,

$$H(X) \leq EL^* \leq H(X) + 1,$$

where $H(X)$ is the entropy of the desired probability distribution $\{q_1, q_2, ..., q_m\} = \{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$.

Let w_i denote the weight of the ith parent node in the binary tree. In order to simplify our analysis, we assume that this parent node can be replaced by a subnetwork with about $\log_2(bw_i)$ splitters. This simplification is reasonable from the statistical perspective and according to the results about our constructions for realizing rational probabilities in the sections above. Then the size of the resulting network is approximately $\sum_{i=1}^{m-1} \log_2(bw_i)$. According to lemma 8.10 as follows, when m is small, Huffman procedure can create a binary tree that minimizes $\sum_{i=1}^{m-1} \log_2 w_i$. As a result, among all the binary trees with m leaves, the one constructed based on Huffman procedure has an optimal network size – however, it is only true based on our assumption. For example, let us consider a desired distribution $\{q_1, q_2, ..., q_m\}$ with $\sum_{i \in S} q_i = \frac{1}{2}$ for some set S. In this case, the binary-tree structure based on Huffman procedure may not be the best one.

Lemma 8.10. *Given a desired probability distribution $\{q_1, q_2, ..., q_m\}$ and $m < 6$, Huffman procedure can construct a binary tree such that*

1. *It has m leaves with weight $q_1, q_2, ..., q_m$.*

2. *$L = \sum_{j=1}^{m-1} \log_2 w_j$ is minimized, where w_j is the weight of jth parent node in a binary tree with m leaves.*

Proof. It is easy to prove that the case for $m = 3$ or $m = 4$ is true. In the following proof, we only show the case for $m = 5$ briefly. W.l.o.g, we assume $q_1 \leq q_2 \leq ... \leq q_5$. Without considering the order of the leaves, we have only two binary-tree structures, as shown in figure 8.15.

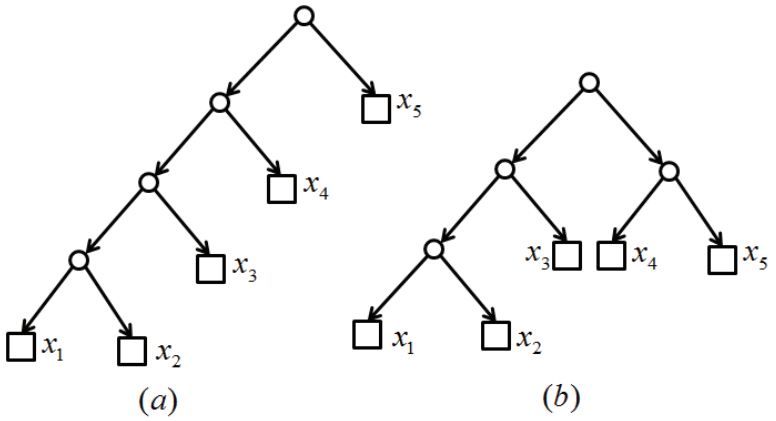


Figure 8.15. Two possible tree structures for $m = 5$.

In both of the structures, for any pair of leaves x_i and x_j, if x_i's sibling is x_j's ancestor then $x_i \geq x_j$. Otherwise, we can switch the position of x_i and x_j to reduce $\sum_{j=1}^{m-1} \log_2 w_j$. So if the tree structure (a) in figure 8.15 is the optimal one, we have $x_1 = q_1, x_2 = q_2$ or $x_1 = q_2, x_2 = q_1$. Now, we will show that if the tree structure (b) in figure 8.15 is the optimal one, we also have $x_1 = q_1, x_2 = q_2$ or $x_1 = q_2, x_2 = q_1$.

For the tree structure (b), we have the following relations:

$$x_3 \geq \max\{x_1, x_2\},$$

$$x_4 + x_5 \geq \max\{x_1 + x_2, x_3\}.$$

Then q_1 and q_2 is in $\{x_1, x_2, x_4, x_5\}$ and $x_1 + x_2 \leq \frac{1-x_3}{2}$.

Let $x = x_1 + x_2$, then L can be written as

$$\begin{aligned} L &= \min \log(x_1 + x_2) + \log(x_1 + x_2 + x_3) + \log(x_4 + x_5) \\ &= \min \log((x_1 + x_2)(x_1 + x_2 + x_3)(1 - x_1 - x_2 - x_3)) \\ &= \min \log x(1 - x_3 - x)(x + x_3). \end{aligned}$$

So we can minimize $x(1 - x_3 - x)(x + x_3)$ instead of minimizing L. Fixing x_3, we can see that $x(1-x_3-x)$ increases as x increases when $x \leq \frac{1-x_3}{2}$; $(x+x_3)$ also increases as x increases. So fixing x_3, $x(1-x_3-x)(x+x_3)$ is minimized if and only if x is minimized, which will cause $x_1 = q_1, x_2 = q_2$ or $x_1 = q_2, x_2 = q_1$.

Based on the discussion above, we know that in the optimal tree, q_1 and q_2 must be siblings. Let us replace q_1, q_2 and their parent node using a leaf with weight $q_1 + q_2$. Then we can get an optimal tree for distribution $\{q_1 + q_2, q_3, q_4, q_5\}$, whose L value is L_4^*. Assume the optimal L value for distribution $\{q_1, q_2, q_3, q_4, q_5\}$ is L_5^*, then

$$L_5^* = L_4^* + \log_2(q_1 + q_2).$$

Let us consider a tree constructed by Huffman procedure for $\{q_1, q_2, q_3, q_4, q_5\}$, whose L value is L_5. We want to show that this tree is optimal. According to the procedure, we know that q_1 and q_2 are also siblings. By combing q_1 and q_2 to a leaf with $q_1 + q_2$, we can get a new tree. This new tree can be constructed by applying Huffman procedure to distribution $\{q_1 + q_2, q_3, q_4, q_5\}$. Due to our assumption for $m = 4$, it is optimal, as a result the following result is true,

$$L_5 = L_4^* + \log_2(q_1 + q_2).$$

Finally, we can obtain $L_5 = L_5^*$, which shows that the L value of the tree constructed by Huffman procedure is minimized when $m = 5$.

This completes the proof. □

Now we can get the following conclusion about stochastic flow networks constructed using the method based on binary-tree structures.

Theorem 8.11. *For a distribution* $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$ *with* $b \leq 2^n$*, the method based on binary-tree structures constructs a network with at most* $(m-1)n$ *splitters. If the binary tree is constructed using Huffman procedure, then the expected latency of the resulting network, namely* ET*, is upper bounded by*

$$ET \leq (H(X)+1)ET_{\max},$$

where $H(X)$ *is the entropy of the target distribution and* $ET_{\max}$ *is the maximum expected latency of the inner nodes in the binary tree.*

Proof. 1) According to the optimal-sized construction, each inner node can be implemented using at most n splitters.

2) The upper bound on the expected latency is immediate following the result that the expected path length $EL^* \leq H(X)+1$. □

8.6.3 Comparison

Let us have a brief comparison between the method based on Knuth and Yao's scheme and the method based on binary-tree structure. Generally, when m is large, the method based Knuth and Yao's scheme may perform better. When m is small, the comparison between these two methods is given in table 8.2, where the desired distribution is $\{\frac{a_1}{b}, \frac{a_2}{b}, ..., \frac{a_m}{b}\}$ with $2^{n-1} < b \leq 2^n$. In this table, we assume that the binary tree (in the second method) is constructed using Huffman procedure. ET_{max} denotes the maximum expected latency of the parent nodes in a given binary tree. It is still hard to say that one of the two methods has an absolutely better performance than the other one. In fact, the performance of a construction is usually related to the number structure of the target distribution. In practice, we can compare both of the constructions based on real values and choose the better one.

Table 8.3. The comparison of different stochastic systems of size n

	Expressibility (probabilities)	Operating time
Sequential State Machine [44]	Converge to rational $\frac{a}{b}$ $(b \leq n)$	states traveled $O(n)$
Stochastic Switching Circuit [134]	Realize binary probability $\frac{a}{2^n}$	longest path $O(n)$
Combinational Logic [92]	Realize binary probability $\frac{a}{2^n}$	maximum depth $O(n)$
Stochastic Flow Network	Realize rational $\frac{a}{b}$ $(b \leq 2^n)$	expected latency $O(1)$

8.7 Concluding Remarks

Motivated by computing based on chemical reaction networks, we introduced the concept of stochastic flow networks and studied the synthesis of optimal-sized networks for realizing rational probabilities. We also studied the expected latency of stochastic flow networks, namely, the expected number of splitters a token need to pass before reaching the output. Two constructions with well-bounded expected latency are proposed. Finally, we generalize our constructions to realize arbitrary rational probability distributions. Beside of network size and expected latency, robustness is also an important issue in stochastic flow networks. Assume the probability error of each splitter is bounded by a constant ϵ, the robustness of a given network can be measured by the total probability error. It can be shown that most constructions in this chapter are robust against small errors in the splitters.

To end this chapter, we compare a few types of stochastic systems of the same size n in table 8.3. Here we assume that the basic probabilistic elements in these systems have probability $1/2$ and we want use them to synthesize the other probabilities. To unfairly compare different systems, we remove threshold logic circuits from the list, since their complexity is difficult to analyze. From this table, we see that stochastic flow networks have excellent performances in both expressibility and operating time. Future works include the synthesis of stochastic flow network to 'approximate' desired probabilities or distributions, and the study of the scenario that the probability of each splitter is not $\frac{1}{2}$.

Part IV

Coding for Data Storage

Chapter 9

Nonuniform Codes for Correcting Asymmetric Errors

This chapter introduces a new type of code called a nonuniform code, whose codewords can tolerate different numbers of asymmetric errors depending on their Hamming weights. The goal of nonuniform codes is to guarantee the reliability of every codeword while maximizing the code size for correcting asymmetric errors.[1]

9.1 Introduction

Asymmetric errors exist in many storage devices [21]. In optical disks, read only memories and quantum memories, the error probability from 1 to 0 is significantly higher than the error probability from 0 to 1, which is modeled by Z-channels where the transmitted sequences only suffer one type of errors, say $1 \to 0$. In some other devices, like flash memories and phase change memories, although the error probability from 0 to 1 is still smaller than that from 1 to 0, it is not ignorable. That means both types of errors, say $1 \to 0$ and $0 \to 1$ are possible, modeled by binary asymmetric channels. In contrast to symmetric errors, where the error probability of a codeword is context independent (since the error probability for 1s and 0s is identical), asymmetric errors are context dependent. For example, the all-one codeword is prone to have more errors than the all-zero codeword in both Z-channels and binary asymmetric channels.

[1] Some of the results presented in this chapter have been previously published in [146].

The construction of asymmetric error correcting codes is a topic that was studied extensively. In [67], Kløve summarized and presented several such codes. In addition, a large amount of efforts are contributed to the design of systematic codes [2, 16], constructing single or multiple error-correcting codes [8, 100, 110], increasing the lower bounds [35, 36, 39, 137] and applying LDPC codes in the context of asymmetric channels [129]. However, the existing approach for code construction is similar to the approach taken in the construction of symmetric error correcting codes, namely, it assumes that every codeword could sustain t asymmetric errors (or generally t_1 $1 \rightarrow 0$ errors and t_2 $0 \rightarrow 1$ errors). As a result, different codewords might have different reliability. To see this, let us consider errors to be i.i.d., where every bit that is a 1 can change to a 0 by an asymmetric error with crossover probability $p > 0$ and each bit that is a 0 keeps unchanged. For a codeword $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, let $w(\mathbf{x}) = |\{i \ : 1 \leq i \leq n, x_i = 1\}|$ denote the Hamming weight of $\mathbf{x}$. Then the probability for $\mathbf{x}$ to have at most t asymmetric errors is

$$P_t(\mathbf{x}) = \sum_{i=0}^{t} \binom{w(\mathbf{x})}{i} p^i (1-p)^{w(\mathbf{x})-i}.$$

Since $\mathbf{x}$ can correct t errors, $P_t(\mathbf{x})$ is the probability of correctly decoding $\mathbf{x}$ (assuming codewords with more than t errors are uncorrectable). It can be readily observed that the reliability of codewords decreases when their Hamming weights increase, for example, see figure 9.1.

While asymmetric errors are content dependent, in most applications of data storage the reliability of each codeword should be content independent. Namely, unaware of data importance, no matter what content is stored, it should be retrieved with very high probability. The reason is that once a block cannot be correctly decoded, the content of the block, which might be very important, will be lost forever. So we are interested in the worst-case performance rather than the average performance that is commonly considered in telecommunication, and we want to construct error-correcting codes that can guarantee the reliability of every codeword. In this case, it is not desired to let all the codewords tolerate the same number of asymmetric errors, since the codeword with the highest Hamming weight will become a 'bottleneck' and limit the code rate. We call the

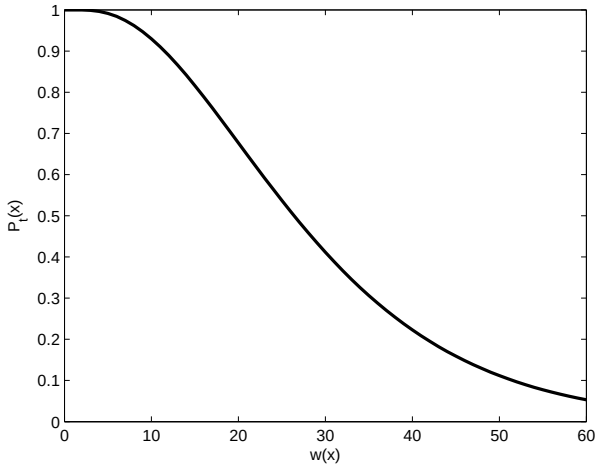


Figure 9.1. The relation between $P_t(\mathbf{x})$ and $w(\mathbf{x})$ when $p = 0.1$ and $t = 2$.

existing codes *uniform codes* while we focus on the notion of *nonuniform codes*, namely, codes whose codewords can tolerate different numbers of asymmetric errors depending on their Hamming weights. The goal of introducing nonuniform codes is to maximize the code size while guaranteeing the reliability of each codeword for combating asymmetric errors.

In a nonuniform code, given a codeword $\mathbf{x} \in \{0,1\}^n$ of weight w, we let $t_\downarrow(w)$ denote the number of $1 \to 0$ errors that it has to tolerate, and we let $t_\uparrow(w)$ denote the number of $0 \to 1$ errors that it has to tolerate. Both $t_\downarrow$ and $t_\uparrow$ are step functions on $\{0, 1, ..., n\}$ that can be predetermined by the channel, the types of errors and the required reliability. In this chapter, we consider $t_\downarrow$ a nondecreasing function and $t_\uparrow$ a nonincreasing function of codeword weight. As a result, we call such a code as a nonuniform code correcting $[t_\downarrow, t_\uparrow]$ errors. In particular, for Z-channels where $t_\uparrow(w) = 0$ for all $0 \le w \le n$, we call it a nonuniform code correcting $t_\downarrow$ asymmetric errors. Surprisingly, there is little in the literature that studies this type of codes although they are natural and much more efficient than traditional codes for correcting asymmetric errors in data storage applications.

Example 9.1. *In Z-channels, let p be the crossover probability of each bit from 1 to 0 and let $q_e < 1$ be maximum tolerated error probability for each codeword. If we consider the errors to be i.i.d., then we can get*

$$t_\downarrow(w) = \min\{s \in N | \sum_{i=0}^{s} \binom{w}{i} p^i (1-p)^{w-i} \ge 1 - q_e\} \tag{9.1}$$

for $0 \leq w \leq n$. *In this case, every erroneous codeword can be corrected with probability at least* $1 - q_e$.

The following notations will be used throughout of this chapter:

q_e	the maximal error probability for each codeword
$p, p_\downarrow$	the error probability of each bit from 1 to 0
$p_\uparrow$	the error probability of each bit from 0 to 1
$t_\downarrow$	a nondecreasing function that indicates the number of $1 \to 0$ errors to tolerate
$t_\uparrow$	a nonincreasing function that indicates the number of $0 \to 1$ errors to tolerate

In this chapter, we introduce the concept of nonuniform codes and study their basic properties, upper bounds on the rate, asymptotic performance, and code constructions. We first focus on Z-channels and study nonuniform codes correcting $t_\downarrow$ asymmetric errors. The chapter is organized as follows: In section 9.2, we provide some basic properties of nonuniform codes. In section 9.3, we give an almost explicit upper bound for the size of nonuniform codes. Section 9.4 studies and compares the asymptotic performances of nonuniform codes and uniform codes. Two general constructions, based on multiple layers or bit flips, are proposed in section 9.5 and section 9.6. Finally, we extend our discussions and results from Z-channels to general binary asymmetric channels in section 9.7, where we study nonuniform codes correcting $[t_\downarrow, t_\uparrow]$ errors, namely, $t_\downarrow$ $1 \to 0$ errors and $t_\uparrow$ $0 \to 1$ errors. Concluding remarks are presented in section 9.8.

9.2 Basic Properties of Nonuniform Codes for Z-Channels

Storage devices such as optical disks, read-only memories and quantum atomic memories can be modeled by Z-channels, in which the transmitted sequences only suffer one type of errors, namely $1 \to 0$. In this section, we study some properties of nonuniform codes for Z-channels, namely, codes

that only correct $t_\downarrow$ asymmetric errors. Typically, $t_\downarrow(w)$ is a nondecreasing function in w, the weight of the codeword. We prove it in the following lemma for the case of i.i.d. errors.

Lemma 9.1. *Assume the errors in a Z-channel are i.i.d., then given any $0 < p, q_e < 1$, the function $t_\downarrow$ defined in (9.1) satisfies $t_\downarrow(w+1) - t_\downarrow(w) \in \{0, 1\}$ for all $0 \le w \le n-1$.*

Proof. Let us define

$$P(k, w, p) = \sum_{i=0}^{k} \binom{w}{i} p^i (1-p)^{w-i}.$$

Then

$$P(k, w, p) = (w-k)\binom{w}{k} \int_0^{1-p} t^{w-k-1}(1-t)^k dt,$$

which leads us to

$$\begin{aligned} & P(k, w, p) - P(k, w+1, p) \\ = \quad & \frac{k+1}{w+1}[P(k+1, w+1, p) - P(k, w+1, p)]. \end{aligned} \tag{9.2}$$

First, let us prove that $t_\downarrow(w+1) \ge t_\downarrow(w)$. Since

$$P(k+1, w+1, p) - P(k, w+1, p) > 0,$$

we have $P(k, w, p) > P(k, w+1, p)$.

We know that $P(t_\downarrow(w+1), w+1, p) \ge 1 - q_e$, so

$$P(t_\downarrow(w+1), w, p) > 1 - q_e.$$

According to definition of $t_\downarrow(w)$, we can conclude that $t_\downarrow(w+1) \ge t_\downarrow(w)$.

Second, let us prove that $t_\downarrow(w+1) - t_\downarrow(w) \le 1$. Based on equation (9.2), we have

$$P(k, w, p) - P(k+1, w+1, p) = \frac{w-k}{w+1}[P(k, w+1, p) - P(k+1, w+1, p)].$$

So $P(k, w, p) < P(k+1, w+1, p)$.

We know that $P(t_{\downarrow}(w), w, p) \geq 1 - q_e$, therefore

$$P(t_{\downarrow}(w) + 1, w + 1, p) > 1 - q_e.$$

According to the definition of $t_{\downarrow}(w+1)$, we have $t_{\downarrow}(w+1) \leq t_{\downarrow}(w) + 1$.

This completes the proof. □

Given two binary vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, we say $\mathbf{x} \leq \mathbf{y}$ if and only if $x_i \leq y_i$ for all $1 \leq i \leq n$. Let $\mathcal{B}(\mathbf{x})$ be the (asymmetric) 'ball' centered at $\mathbf{x}$, namely, it consists of all the vectors obtained by changing at most $t_{\downarrow}(w(\mathbf{x}))$ 1s in $\mathbf{x}$ into 0s, i.e.,

$$\mathcal{B}(\mathbf{x}) = \{\mathbf{v} \in \{0,1\}^n | \mathbf{v} \leq \mathbf{x} \text{ and } N(\mathbf{x}, \mathbf{v}) \leq t_{\downarrow}(w(\mathbf{x}))\},$$

where $w(\mathbf{x})$ is the weight of $\mathbf{x}$ and

$$N(\mathbf{x}, \mathbf{y}) \triangleq |\{i : x_i = 1, y_i = 0\}|.$$

We have the following properties of nonuniform codes as the generalizations of those for uniform codes studied in [67].

Lemma 9.2. *Code C is a nonuniform code correcting $t_{\downarrow}$ asymmetric errors if and only if $\mathcal{B}(\mathbf{x}) \bigcap \mathcal{B}(\mathbf{y}) = \phi$ for all $\mathbf{x}, \mathbf{y} \in C$ with $\mathbf{x} \neq \mathbf{y}$.*

Proof. According to the definition of nonuniform codes, all the vectors in $\mathcal{B}(\mathbf{x})$ can be decoded as $\mathbf{x}$, and all the vectors in $\mathcal{B}(\mathbf{y})$ can be decoded as $\mathbf{y}$. Hence, $\mathcal{B}(\mathbf{x}) \bigcap \mathcal{B}(\mathbf{y}) = \phi$ for all $\mathbf{x}, \mathbf{y} \in C$. □

Lemma 9.3. *There always exists a nonuniform code of the maximum size that corrects $t_{\downarrow}$ asymmetric errors and contains the all-zero codeword.*

Proof. Let C be a nonuniform code correcting $t_{\downarrow}$ asymmetric errors, and assume that 00...00 $\notin C$.

If there exists a codeword $\mathbf{x} \in C$ such that $00...00 \in \mathcal{B}(\mathbf{x})$, then we can get a new nonuniform code C' of the same size by replacing $\mathbf{x}$ with 00...00 in C. If there does not exist a codeword $\mathbf{x} \in C$ such that $00...00 \in \mathcal{B}(\mathbf{x})$, then we can get a larger nonuniform code C' by adding 00...00 to C. □

Given a nonuniform code C, let A_r denote the number of codewords with Hamming weight r in C, i.e.,

$$A_r = |\{\mathbf{x} \in C | w(\mathbf{x}) = r\}|.$$

Given a nondecreasing function $t_\downarrow$, let R_r denote a set of weights that can reach weight r with at most $t_\downarrow$ asymmetric errors, namely,

$$R_r = \{0 \leq s \leq n | s - t_\downarrow(s) \leq r \leq s\}.$$

Lemma 9.4. *Let C be a nonuniform code correcting $t_\downarrow$ asymmetric errors. For $0 \leq r \leq n$, we have*

$$\sum_{j \in R_r} \binom{j}{r} A_j \leq \binom{n}{r}. \tag{9.3}$$

Proof. Let $V_r = \{\mathbf{x} \in \{0,1\}^n | w(\mathbf{x}) = r\}$ be the set consisting of all the vectors of length n and weight r. If $\mathbf{x} \in C$ with $w(\mathbf{x}) = j \in R_r$, according to the properties of $t_\downarrow$, $\mathcal{B}(\mathbf{x})$ contains $\binom{j}{r}$ vectors of weight r, namely

$$|V_r \bigcap \mathcal{B}(\mathbf{x})| = \binom{j}{r}.$$

According to lemma 9.2, we know that $\bigcup_{\mathbf{x} \in C}(V_r \bigcap \mathcal{B}(\mathbf{x}))$ is a disjoint union, in which the number of vectors is

$$\sum_{j \in R_r} \binom{j}{r} A_j.$$

Since $\bigcup_{\mathbf{x} \in C}(V_r \bigcap \mathcal{B}(\mathbf{x})) \subseteq V_r$ and there are at most $\binom{n}{r}$ vectors in V_r, the lemma follows. □

9.3 Upper Bounds

Let $B_\alpha(n,t)$ denote the maximal size of a uniform code correcting t asymmetric errors, and let $B_\beta(n,t_\downarrow)$ denote the maximal size of a nonuniform code correcting $t_\downarrow$ asymmetric errors, where t is a constant and $t_\downarrow$ is a nondecreasing function of codeword weight. In this section, we first present some existing results on the upper bounds of $B_\alpha(n,t)$ for uniform codes. Then we derive an almost explicit upper bound of $B_\beta(n,t_\downarrow)$ for nonuniform codes.

9.3.1 Upper Bounds for Uniform Codes

An explicit upper bound to $B_\alpha(n,t)$ was given by Varshamov [122]. In [67], Borden showed that $B_\alpha(n,t)$ is upper bounded by $\min\{A(n+t,2t+1),(t+1)A(n,2t+1)\}$, where $A(n,d)$ is the maximal number of vectors in $\{0,1\}^n$ with Hamming distance at least d. Goldbaum [46] pointed out that the upper bounds can be obtained using integer programming. By adding more constrains to the integer programming, the upper bounds were later improved by Delsarte and Piret [28] and Weber et al. [133] [132]. Kløve generalized the bounds of Delsarte and Piret, and gave an almost explicit upper bound which is very easy to compute by relaxing some of the constrains [68], in the following way.

Theorem 9.5. [68]] *For $n > 2t \geq 2$, let $y_0, y_1, ..., y_n$ be defined by*

1. $y_0 = 1$,
2. $y_r = 0, \quad \forall 1 \leq r \leq t$,
3. $y_{t+r} = \frac{1}{\binom{t+r}{t}}[\binom{n}{r} - \sum_{j=0}^{t-1} y_{r+j}\binom{r+j}{j}], \forall 1 \leq r \leq \frac{n}{2} - t$,
4. $y_{n-r} = y_r, \quad \forall 0 \leq r < \frac{n}{2}$.

Then $B_\alpha(n,t) \leq M_\alpha(n,t) \triangleq \sum_{r=0}^{n} y_r$.

This method obtains a good upper bound to $B_\alpha(n,t)$ (although it is not the best known one). Since it is easy to compute, when n and t are large, it is every useful for analyzing the sizes of uniform codes.

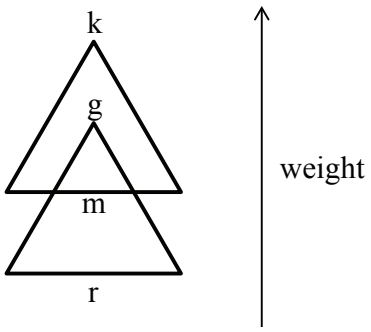


Figure 9.2. This diagram demonstrates the relative values of r, g, k, m.

9.3.2 Upper Bounds for Nonuniform Codes

We now derive an almost explicit upper bound for the size of nonuniform codes correcting $t_\downarrow$ asymmetric errors, followed the idea of Kløve [68] for uniform codes. According to the lemmas in the previous section, we can get an upper bound of $B_\beta(n, t_\downarrow)$, denoted by $M_\beta(n, t_\downarrow)$, such that

$$M_\beta(n, t_\downarrow) = \max \sum_{i=0}^{n} z_r,$$

where the maximum is taken over the following constraints:

1. z_r are nonnegative real numbers;

2. $z_0 = 1$;

3. $\sum_{j \in R_r} \binom{j}{r} z_j \leq \binom{n}{r}, \forall 0 \leq r \leq n.$

Here, condition 2) is given by lemma 9.3, and condition 3) is given by lemma 9.4. Our goal is to find an almost explicit way to calculate $M_\beta(n, t_\downarrow)$.

Lemma 9.6. *Assume $\sum_{r=0}^{n} z_r$ is maximized over $z_0, z_1, ..., z_n$ in the problem above. If $r = s - t_\downarrow(s)$ for some integer s with $0 \leq s, r \leq n$, then*

$$Z_r = \sum_{j \in R_r} \binom{j}{r} z_j = \binom{n}{r}.$$

Proof. Suppose that $Z_r < \binom{n}{r}$ for some r that satisfies the above condition. Let $g = \max R_r$ and $k = \min\{w | z_w > 0, w > g\}$, as indicated in figure 9.2, where a triangular denote the ball centered at the top vertex. Furthermore, we let $m = \max\{w | k - t_\downarrow(k) > w\}$. Note that in this case $r = g - t_\downarrow(g)$ and $m = k - t_\downarrow(k) - 1$.

We first prove that for all $r \leq w \leq m$, $Z_w < \binom{n}{w}$. In order to prove this, we let $s = w - r$, then we get

$$\begin{aligned} Z_w &= \sum_{j \in R_w} z_j \binom{j}{w} \\ &= \sum_{j=w}^{g} z_j \binom{j}{w} \\ &= \sum_{j=s}^{g-r} z_{r+j} \binom{r+j}{r+s}. \end{aligned}$$

It is easy to obtain that

$$\binom{r+j}{r+s} = \binom{r+j}{r} \frac{\binom{j}{s}}{\binom{r+s}{s}}.$$

So

$$\begin{aligned} Z_w &\leq \frac{\binom{g-r}{s}}{\binom{r+s}{s}} \sum_{j=s}^{g-r} z_{r+j} \binom{r+j}{r} \\ &< \frac{\binom{g-r}{s}}{\binom{r+s}{s}} \binom{n}{r} \\ &= \frac{(g-r)(g-r-1)...(g-r-s+1)}{(n-r)(n-r-1)...(n-r-s+1)} \binom{n}{r+s} \\ &\leq \binom{n}{w}. \end{aligned}$$

Now, we construct a new group of real numbers $z_0^*, z_1^*, ..., z_n^*$ such that

1. $z_g^* = z_g + \Delta$,

2. $z_k^* = z_k - \delta$,

3. $z_r^* = z_r$ for $r \neq h, r \neq k$,

with

$$\Delta = \min(\{\frac{\binom{n}{w} - Z_w}{\binom{g}{w}} | r \leq w \leq m\} \bigcup \{\frac{\binom{k}{w}}{\binom{g}{w}} z_k | m < w \leq g\}),$$

$$\delta = \frac{1}{\min\{\frac{\binom{k}{w}}{\binom{g}{w}} | m < w \leq g\}} \Delta.$$

For such Δ, δ, it is not hard to prove that $Z_r^* = \binom{n}{r}$ for $0 \leq r \leq n$. On the other hand,

$$\sum_{r=0}^{n} z_r^* = \sum_{r=0}^{n} z_r + \Delta - \delta > \sum_{r=0}^{n} z_r,$$

which contradicts our assumption that $\sum_{r=0}^{n} z_r$ is maximized over the constrains. So the lemma is true. □

Lemma 9.7. *Assume $\sum_{r=0}^{n} z_r$ is maximized over $z_0, z_1, ..., z_n$ in the problem above. If $r = s - t_{\downarrow}(s)$ for some integer s with $0 \leq s, r \leq n$, then*

$$Z_r = \sum_{j=r}^{h} \binom{j}{r} z_j = \binom{n}{r},$$

where

$$h = \min\{s \in N | s - t_{\downarrow}(s) = r\}.$$

Sketch of Proof: Let $g = \max\{s \in N | s - t_{\downarrow}(s) = r\}$. If $g = h$, then the lemma is true. So we only need to prove it for the case that $g > k$. Similar to lemma 9.6, we assume $Z_r < \binom{n}{r}$, to get the contradiction, we can construct a new group of real numbers $z_0^*, z_1^*, ..., z_n^*$ such that

1. $z_h^* = z_h + \Delta$,

2. $z_w^* = 0$ for $h < w \leq g$,

3. $z_w^* = z_w$ if $w \notin [h, g]$.

with

$$\Delta = \min\{\frac{\sum_{j=h+1}^{g} \binom{j}{w} z_j}{\binom{h}{w}} | r \leq w \leq h\}.$$

For this $z_0^*, z_1^*, ..., z_n^*$, they satisfy all the constrains and $Z_r^* = \sum_{j=r}^{h} \binom{j}{r} z_j^* = \binom{n}{r}$. At the same time, it can be proved that

$$\sum_{r=0}^{n} z_r^* > \sum_{r=0}^{n} z_r,$$

which contradicts with our assumption that $\sum_{r=0}^{n} z_r$ is maximized over the constrains. This completes the proof. □

Now let $y_0, y_1, ..., y_n$ be a group of optimal solutions to $z_0, z_1, ..., z_n$ that maximize $\sum_{r=0}^{n} z_r$. Then $y_0, y_1, ..., y_n$ satisfy the condition in Lemma 9.7. We see that $y_0 = 1$. Then based on lemma 9.7, we can get $y_1, ..., y_n$ uniquely by iteration. Hence, we have the following theorem for calculating the upper bound $M_\beta(n, t_\downarrow)$.

Theorem 9.8. *Let $y_0, y_1, ..., y_n$ be defined by*

1. $y_0 = 1$;
2. $y_r = \frac{1}{\binom{r}{t_\downarrow(r)}} [\binom{n}{r - t_\downarrow(r)} - \sum_{j=1}^{t_\downarrow(r)} y_{r-j} \binom{r-j}{t_\downarrow(r) - j}], \forall 1 \leq r \leq n.$

Then $B_\beta(n, t_\downarrow) \leq M_\beta(n, t_\downarrow) = \sum_{r=0}^{n} y_r$.

This theorem provides an almost explicit expression for the upper bound $M_\beta(n, t_\downarrow)$, which is much easier to calculate than the equivalent expression defined at the beginning of this subsection. Note that in the theorem, we do not have a constrain like the one (constraint 4) in theorem 9.5. It is because that the optimal nonuniform codes do not have symmetric weight distributions due to the fact that $t_\downarrow(w)$ monotonically increases with w.

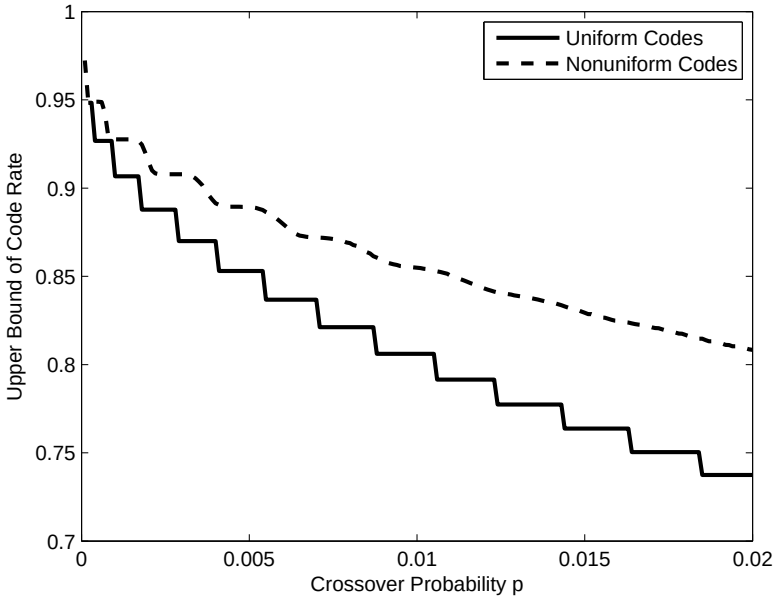


Figure 9.3. Upper bounds of the rates for uniform/nonuniform codes when $n = 255, q_e = 10^{-4}$.

9.3.3 Comparison of Upper Bounds

Here we focus on i.i.d. errors, i.e., given the crossover probability p from 1 to 0 and the maximal tolerated error probability q_e, the function $t_\downarrow$ is defined in equation (9.1). In this case, we can write the maximal size of a uniform code as $B_\alpha(n, t_\downarrow(n)) = B_\alpha(n, p, q_e)$, and write the maximal size of a nonuniform code as $B_\beta(n, t_\downarrow(n)) = B_\beta(n, p, q_e)$.

Now we let $\eta_\alpha(n, p, q_e)$ denote the maximal code rate defined by

$$\eta_\alpha(n, p, q_e) = \frac{\log B_\alpha(n, p, q_e)}{n}.$$

Similar, we let $\eta_\beta(n, p, q_e)$ denote the maximal code rate defined by

$$\eta_\beta(n, p, q_e) = \frac{\log B_\beta(n, p, q_e)}{n}.$$

By the definition of uniform and nonuniform codes, it is simple to see that $\eta_\beta(n, p, q_e) \geq \eta_\alpha(n, p, q_e)$.

Figure 9.3 depicts the upper bounds of $\eta_\alpha(n, p, q_e)$ and $\eta_\beta(n, p, q_e)$ for different values of p when

$n = 255$ and $q_e = 10^{-4}$. The upper bound of $\eta_\alpha(n, p, q_e)$ is obtained based on the almost explicit upper bound given by Kløve, and the upper bound of $\eta_\beta(n, p, q_e)$ is obtained based on the almost explicit method proposed in this section. It demonstrates that given the same parameters, the upper bound for nonuniform codes is substantially greater than that for uniform codes.

9.4 Asymptotic Performance

In this section, we study and compare the asymptotic rates of uniform codes and nonuniform codes. Note that the performance of nonuniform codes strongly depends on the selection of the function $t_\downarrow$. Here, we focus on i.i.d. errors, so given $0 < p, q_e < 1$, we study the asymptotic behavior of $\eta_\alpha(n, p, q_e)$ and $\eta_\beta(n, p, q_e)$ as $n \to \infty$. By the definition of nonuniform and uniform codes, the 'balls' containing up to $t_\downarrow(\mathbf{x})$ (or $t_\downarrow(n)$) errors that are centered at codewords $\mathbf{x}$ need to be disjoint.

Before giving the asymptotic rates, we first present the following known result: For any $\delta > 0$, when n is large enough, we have

$$2^{n(H(\frac{k}{n})-\delta)} \leq \binom{n}{k} \leq 2^{n(H(\frac{k}{n})+\delta)},$$

where $H(p)$ is the entropy function with

$$H(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p} \text{ for } 0 \leq p \leq 1,$$

and

$$H(p) = 0 \text{ for } p > 1 \text{ or } p < 0.$$

Lemma 9.9. *Let $A(n, d, w)$ be the maximum size of a constant-weight binary code of codeword length n, whose Hamming weight is w and minimum distance is d. Let $R(n, t, w)$ be the maximum size of a binary code with Hamming weight w and codeword length n where every codeword can correct t asymmetric errors. Then*

$$R(n, t, w) = A(n, 2(t+1), w).$$

Proof. Let C be a code of length n, constant weight w and size $R(n,t,w)$ that corrects t asymmetric errors. For all $\mathbf{x} \in \{0,1\}^n$, let's define $S_t(\mathbf{x})$ be the set consisting of all the vectors obtained by changing at most t 1s in $\mathbf{x}$ into 0s, i.e.,

$$S_t(\mathbf{x}) = \{\mathbf{v} \in \{0,1\}^n | \mathbf{v} < \mathbf{x} \text{ and } N(\mathbf{x},\mathbf{v}) \leq t\}.$$

Then $\forall \mathbf{x}, \mathbf{y} \in C$, we know that $S_t(\mathbf{x}) \bigcap S_t(\mathbf{y}) = \phi$.

Let $\mathbf{u} = (u_1, \ldots, u_n)$ be a vector such that $u_i = \min\{x_i, y_i\}$ for $1 \leq i \leq n$. Then $N(\mathbf{x},\mathbf{u}) = N(\mathbf{y},\mathbf{u})$ and $\mathbf{u} \notin S_t(\mathbf{x}) \bigcap S_t(\mathbf{y})$. W.l.o.g, suppose that $\mathbf{u} \notin S_t(\mathbf{x})$. Then $N(\mathbf{x},\mathbf{u}) > t$, and the Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ is

$$d(\mathbf{x},\mathbf{y}) = N(\mathbf{x},\mathbf{u}) + N(\mathbf{y},\mathbf{u}) \geq 2(t+1).$$

So the minimum distance of C is at least $2(t+1)$. As a result, $A(n, 2(t+1), w) \geq R(n,t,w)$.

On the other hand, if a constant-weight code has minimum distance at least $2(t+1)$, it can correct t asymmetric errors. As a result, $R(n,t,w) \geq A(n, 2(t+1), w)$. □

9.4.1 Bounds of $\lim_{n\to\infty} \eta_\alpha(n,p,q_e)$

Let us first give the lower bound of $\lim_{n\to\infty} \eta_\alpha(n,p,q_e)$ and then provide the upper bound.

Theorem 9.10 (Lower bound)**.** *Given $0 < q_e < 1$, if $0 < p \leq \frac{1}{4}$, we have*

$$\lim_{n\to\infty} \eta_\alpha(n,p,q_e) \geq 1 - H(2p).$$

Proof. We consider uniform codes that correct t asymmetric errors, where

$$t = \min\{s | \sum_{i=0}^{s} \binom{n}{i} p^i (1-p)^{n-i} \geq 1 - q_e\}.$$

According to Hoeffding's inequality, for any $\delta > 0$, as n becomes large enough, we have $(p-\delta)n \leq$

$t \le (p+\delta)n$. If we write $t = \gamma n$, then $p - \delta \le \gamma \le p + \delta$ for n large enough.

Since each codeword tolerates t asymmetric errors, we have

$$B_\alpha(n, p, q_e) = B_\alpha(n, t) \ge R(n, t, w) = A(n, 2(t+1), w),$$

for every w with $0 \le w \le n$. The Gilbert Bound gives that (see Graham and Sloane [49])

$$A(n, 2(t+1), w) \ge \frac{\binom{n}{w}}{\sum_{i=0}^{t} \binom{w}{i}\binom{n-w}{i}}.$$

Hence

$$\begin{aligned}
B_\alpha(n, p, q_e) &\ge \max_{w=0}^{n} \frac{\binom{n}{w}}{\sum_{i=0}^{t} \binom{w}{i}\binom{n-w}{i}} \\
&\ge \max_{w=0}^{n} \frac{\binom{n}{w}}{n \max_{i\in[0,t]} \binom{w}{i}\binom{n-w}{i}} \\
&\ge \max_{w:\, \frac{w(n-w)}{n} > t} \frac{\binom{n}{w}}{n \max_{i\in[0,t]} \binom{w}{i}\binom{n-w}{i}} \\
&\ge \max_{w:\, \frac{w(n-w)}{n} > t} \frac{\binom{n}{w}}{n\binom{w}{t}\binom{n-w}{t}}.
\end{aligned}$$

For a binomial term $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ and $\delta > 0$, when n is large enough,

$$2^{n(H(\frac{k}{n})-\delta)} \le \binom{n}{k} \le 2^{n(H(\frac{k}{n})+\delta)}.$$

Let $w = \theta n$ and $t = \gamma n$ with $0 \le \theta, \gamma \le 1$, as n becomes large enough, we have

$$\begin{aligned}
& \eta_\alpha(n, p, q_e) \\
= \; & \frac{1}{n} \log_2 B_\alpha(n, p, q_e) \\
\ge \; & \frac{1}{n} \log_2 \max_{w: \frac{w(n-w)}{n} > t} \frac{\binom{n}{w}}{n\binom{w}{t}\binom{n-w}{t}} \\
\ge \; & \frac{1}{n} \log_2 \max_{\theta: \theta(1-\theta) > \gamma} \frac{2^{(H(\theta)-\delta)n}}{n 2^{(H(\frac{\gamma}{\theta})+\delta)\theta n} 2^{(H(\frac{\gamma}{1-\theta})+\delta)(1-\theta)n}} \\
\ge \; & \max_{\theta: \theta(1-\theta) \ge \gamma} H(\theta) - \theta H(\frac{\gamma}{\theta}) - (1-\theta) H(\frac{\gamma}{1-\theta}) - 2\delta + \frac{1}{n} \log \frac{1}{n}.
\end{aligned}$$

From $\theta(1-\theta) \ge \gamma$, we get $\theta > \gamma > 0$; then $H(\frac{\gamma}{\theta})$ is a continuous function of γ. As n becomes large, we have $p - \delta \le \gamma \le p + \delta$, so we can approximate $H(\frac{\gamma}{\theta})$ with $H(\frac{p}{\theta})$. Similarly, we can approximate $H(\frac{\gamma}{1-\theta})$ with $H(\frac{p}{1-\theta})$. Then we can get as $n \to \infty$,

$$\eta_\alpha(n, p, q_e) \ge \max_{\theta: \theta(1-\theta) > p} H(\theta) - \theta H(\frac{p}{\theta}) - (1-\theta) H(\frac{p}{1-\theta}).$$

If $0 \le p \le \frac{1}{4}$, the maximum value can be achieve at $\theta^* = \frac{1}{2}$. Hence we have

$$\lim_{n \to \infty} \eta_\alpha(n, p, q_e) \ge 1 - H(2p).$$

This completes the proof. □

Theorem 9.11 (Upper bound)**.** *Given $0 < p, q_e < 1$, we have*

$$\lim_{n \to \infty} \eta_\alpha(n, p, q_e) \le (1+p)[1 - H(\frac{p}{1+p})].$$

Proof. For a uniform code correcting t asymmetric errors, we have the following observations:

1. There is at most one codeword with Hamming weight at most t;

2. For $t+1 \leq w \leq n$, the number of codewords with Hamming weight w is at most $\dfrac{\binom{n}{w-t}}{\binom{w}{t}}$.

Consequently, the total number of codewords is

$$\begin{aligned} B_\alpha(n,p,q_e) &\leq 1+\sum_{w=t+1}^{n}\frac{\binom{n}{w-t}}{\binom{w}{t}} \\ &= 1+\sum_{w=t+1}^{n}\frac{\binom{n+t}{w}}{\binom{n+t}{t}} \\ &\leq \frac{2^{n+t}}{\binom{n+t}{t}}. \end{aligned}$$

So as $n \to \infty$, we have

$$\begin{aligned} \eta_\alpha(n,p,q_e) &\leq \frac{1}{n}\log[\frac{2^{n+t}}{\binom{n+t}{t}}] \\ &\leq \frac{1}{n}\log\frac{2^{(1+\gamma)n}}{2^{H(\frac{\gamma}{1+\gamma})(1+\gamma)n}} \\ &= (1+\gamma)-H(\frac{\gamma}{1+\gamma})(1+\gamma) \\ &= (1+p)[1-H(\frac{p}{1+p})], \end{aligned}$$

where the last step is due to the continuousness of $(1+\gamma)-H(\frac{\gamma}{1+\gamma})(1+\gamma)$ over γ. □

We see that when $n \to \infty$, $\eta_\alpha(n,p,q_e)$ does not depends on q_e as long as $0 < q_e < 1$. It is because that when $n \to \infty$, we have $t \to pn$, which does not depend on q_e. This property is also hold by $\eta_\beta(n,p,q_e)$ when $n \to \infty$.

9.4.2 Bounds of $\lim_{n\to\infty}\eta_\beta(n,p,q_e)$

In this subsection, we study the bounds of the asymptotic rates of nonuniform codes. Here, we use the same idea as that for uniform codes, besides that we need also prove that the 'edge effect' can be ignored, i.e., the number of codewords with Hamming weight $w \ll n$ does not dominate the final

result.

Theorem 9.12 (Lower bound)**.** *Given* $0 < p, q_e < 1$*, we have*

$$\lim_{n\to\infty} \eta_\beta(n, p, q_e) \geq \max_{0\leq\theta\leq 1-p} H(\theta) - \theta H(p) - (1-\theta)H(\frac{p\theta}{1-\theta}).$$

Proof. We consider nonuniform codes that corrects $t_\downarrow$ asymmetric errors, where

$$t_\downarrow(w) = \min\{s| \sum_{i=0}^{s} \binom{w}{i} p^i (1-p)^{w-i} \geq 1 - q_e\},$$

for all $0 \leq w \leq n$.

Based on Hoeffding's inequality, for any $\delta > 0$, as w becomes large enough, we have $(p-\delta)w \leq t_\downarrow(w) \leq (p+\delta)w$. In another word, for any $\epsilon, \delta > 0$, when n is large enough and $w \geq \epsilon n$, we have $(p-\delta)w \leq t_\downarrow(w) \leq (p+\delta)w$.

Let $w = \theta n$ and $t_\downarrow(w) = \gamma w$, then when n is large enough, if $\theta > \epsilon$, we have

$$(p-\delta) \leq \gamma \leq (p+\delta).$$

If $\theta < \epsilon$, we call it the 'edge' effect. In this case $0 \leq \gamma \leq 1$.

Since each codeword with Hamming weight w can tolerate $t_\downarrow(w)$ errors,

$$B_\beta(n, p, q_e) \geq R(n, t_\downarrow(w), w) \geq A(n, 2(t_\downarrow(w)+1), w),$$

for every w with $0 \leq w \leq n$.

Applying the Gilbert Bound, we have

$$\begin{aligned} B_\beta(n,p,q_e) &\geq \max_w \frac{\binom{n}{w}}{\sum_{i=0}^{t_\downarrow(w)} \binom{w}{i}\binom{n-w}{i}} \\ &\geq \max_w \frac{\binom{n}{w}}{\max_{i\in[0,t_\downarrow(w)]} n\binom{w}{i}\binom{n-w}{i}} \\ &\geq \max_{w:\frac{w(n-w)}{n}\geq t_\downarrow(w)} \frac{\binom{n}{w}}{n\binom{w}{t_\downarrow(w)}\binom{n-w}{t_\downarrow(w)}}. \end{aligned}$$

When $n \to \infty$, we have

$$\begin{aligned} & \eta_\beta(n,p,q_e) \\ = \;& \frac{1}{n}\log_2 B_\beta(n,p,q_e) \\ \geq \;& \frac{1}{n}\log_2 \max_{\theta:(1-\theta)\geq\gamma} \frac{2^{(H(\theta)-\delta)n}}{n2^{(H(\gamma)+\delta)\theta n}2^{(H(\frac{\gamma\theta}{1-\theta})+\delta)(1-\theta)n}} \\ \geq \;& \max_{\theta:(1-\theta)\geq\gamma} H(\theta) - \theta H(\gamma) - (1-\theta)H(\frac{\gamma\theta}{1-\theta}) - 2\delta + \frac{1}{n}\log\frac{1}{n} \\ = \;& \max_{\theta:(1-\theta)\geq\gamma} H(\theta) - \theta H(\gamma) - (1-\theta)H(\frac{\gamma\theta}{1-\theta}). \end{aligned}$$

Note that when $\theta < \epsilon$ for small ϵ, we have

$$H(\theta) - \theta H(\gamma) - (1-\theta)H(\frac{\gamma\theta}{1-\theta}) \sim 0.$$

So we can ignore this edge effect. That implies that we can write

$$p - \delta \leq \gamma \leq p + \delta,$$

for any θ with $0 \leq \theta \leq 1$.

Table 9.1. Upper bounds and lower bounds for the maximum rates of uniform codes and nonuniform codes

	Lower Bound	Upper Bound
$\lim_{n\to\infty} \eta_\alpha(n,p,q_e)$	$[1-H(2p)]I_{0\le p\le \frac{1}{4}}$	$(1+p)[1-H(\frac{p}{1+p})]$
$\lim_{n\to\infty} \eta_\beta(n,p,q_e)$	$\max_{0\le\theta\le 1-p} H(\theta)-\theta H(p)$ $-(1-\theta)H(\frac{p\theta}{1-\theta})$	$\max_{0\le\theta\le 1} H((1-p)\theta)-\theta H(p)$

Since $1-\theta \ge \gamma > 0$, for any fixed θ,

$$H(\theta)-\theta H(\gamma)-(1-\theta)H(\frac{\gamma\theta}{1-\theta})$$

is a continuous function of γ. As $n\to\infty$, we have

$$\eta_\beta(n,p,q_e) \ge \max_{\theta:(1-\theta)\ge p} H(\theta)-\theta H(p)-(1-\theta)H(\frac{p\theta}{1-\theta}).$$

This completes the proof. □

Theorem 9.13 (Upper bound)**.** *Given $0<p,q_e<1$, we have*

$$\begin{aligned}\lim_{n\to\infty}\eta_\beta(n,p,q_e) &\le \max_{0\le\theta\le 1} H((1-p)\theta)-\theta H(p)\\ &= H(\frac{1}{2^{s(p)}+1})+\frac{s(p)}{2^{s(p)}+1},\end{aligned}$$

with $s(p)=H(p)/(1-p)$.

Proof. Here we use the same notations as above. Similar as the proof in theorem 9.11, given (n,p,q_e),

the maximal number of codewords is

$$\begin{aligned} B_\beta(n,p,q_e) &\leq 1+\sum_{w=\overline{h}(0)+1}^{n} \frac{\binom{n}{w-t_\downarrow(w)}}{\binom{w}{t_\downarrow(w)}} \\ &= \sum_{w=\overline{h}(0)}^{n} \frac{\binom{n}{w-t_\downarrow(w)}}{\binom{w}{t_\downarrow(w)}} \\ &\leq \max_{w=0}^{n} n\frac{\binom{n}{w-t_\downarrow(w)}}{\binom{w}{t_\downarrow(w)}}. \end{aligned}$$

As $n \to \infty$, we have

$$\begin{aligned} & \eta_\beta(n,p,q_e) \\ = & \frac{1}{n}\log_2 B_\beta(n,p,q_e) \\ \leq & \frac{1}{n}\log_2 \max_{0\leq\theta\leq 1} n\frac{2^{H((1-\gamma)\theta+\delta)n}}{2^{(H(\gamma)\theta-\delta)n}} \\ = & \max_{0\leq\theta\leq 1} H((1-\gamma)\theta)-\theta H(\gamma)+2\delta+\frac{1}{n}\log n \\ = & \max_{0\leq\theta\leq 1} H((1-\gamma)\theta)-\theta H(\gamma). \end{aligned}$$

Note that when $\theta < \epsilon$ for small ϵ, we have

$$H((1-\gamma)\theta)-\theta H(\gamma) \sim 0.$$

So we can ignore the edge effect. That implies that we can write

$$p-\delta \leq \gamma \leq p+\delta,$$

for any θ with $0 \leq \theta \leq 1$.

Since for any fixed θ with $0 \leq \theta \leq 1$, $H((1-\gamma)\theta)-\theta H(\gamma)$ is a continuous function of γ. When

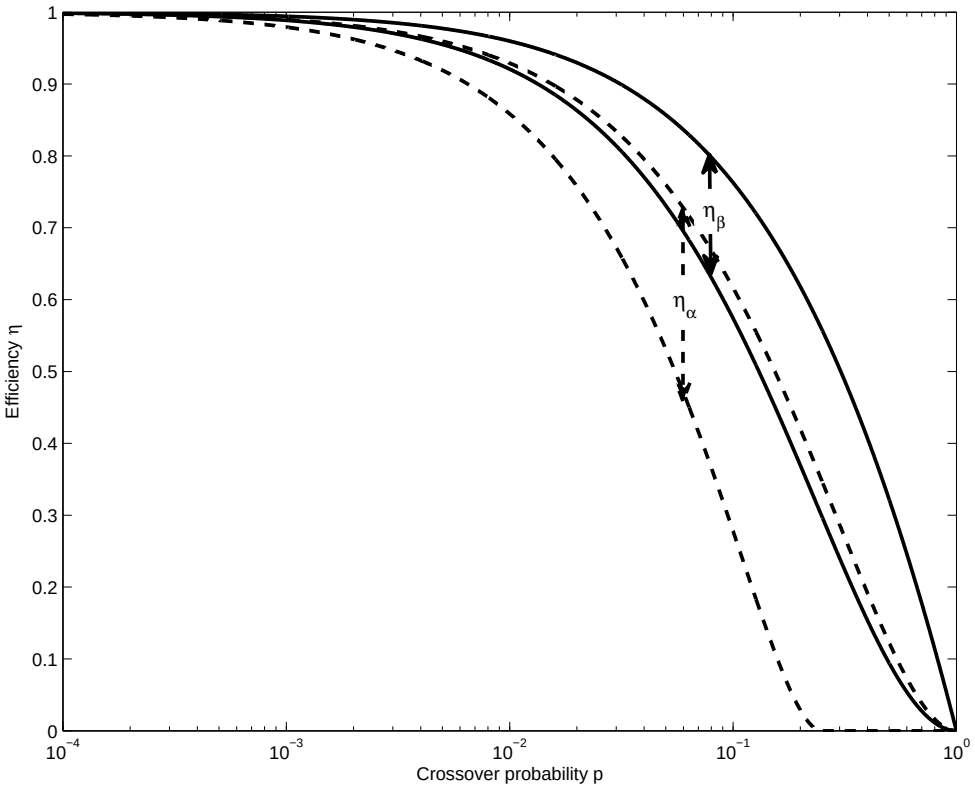


Figure 9.4. Bounds of $\lim_{n\to\infty} \eta_\alpha(n,p,q_e)$ and $\lim_{n\to\infty} \eta_\beta(n,p,q_e)$.

$n \to \infty$, we have

$$\eta_\beta(n,p,q_e) \lesssim \max_{0\le\theta\le 1} H((1-p)\theta) - \theta H(p),$$

which equals to

$$H(\frac{1}{2^{s(p)}+1}) + \frac{s(p)}{2^{s(p)}+1},$$

with $s(p) = H(p)/(1-p)$.

This completes the proof. □

9.4.3 Comparison of Asymptotic Performances

Table 9.1 summarizes the analytic upper bounds and lower bounds of $\lim_{n\to\infty} \eta_\alpha(n,p,q_e)$ and $\lim_{n\to\infty} \eta_\beta(n,p,q_e)$ obtained in this section. For the convenience of comparison, we plot them

in figure 9.4. The dashed curves represent the lower and upper bounds to $\lim_{n\to\infty} \eta_\alpha(n, p, q_e)$, and the solid curves represent the lower and upper bounds to $\lim_{n\to\infty} \eta_\beta(n, p, q_e)$. The gap between the bounds for the two codes indicate the potential improvement in efficiency (code rate) by using the nonuniform codes (compared to using uniform codes) when the codeword length is large. We see that the upper bound in Theorem 9.13 is also the capacity of the Z-channel, derived in [125]. It means that nonuniform codes may be able to achieve the Z-channel capacity as n becomes large, while uniform codes cannot (here we assume that they have codewords of high weights and worst-case performance is considered, so the constructions of uniform codes cannot achieve the capacity of Z-channel).

9.5 Layered Codes Construction

In [67], Kløve summarized some constructions of uniform codes for correcting asymmetric errors. The code of Kim and Freiman was the first one constructed for correcting multiple asymmetric errors. Varshamov [121] and Constrain and Rao [24] presented some constructions based group theory. Later, Delsarte and Piret [28] proposed a construction based on 'expurgating/puncturing' with some improvements given by Weber et al. [132]. It is natural for us to ask whether it is possible to construct nonuniform codes based on existing constructions of uniform codes. In this section, we propose a general construction of nonuniform codes based on multiple layers. It shows that the sizes of the codes can be significantly increased by equalizing the reliability of all the codewords.

9.5.1 Layered Codes

Let us start from a simple example: Assume we want to construct a nonuniform code with codeword length $n = 10$ and

$$t_\downarrow(w) = \begin{cases} 0 & \text{for } w = 0, \\ 1 & \text{for } 1 \le w \le 5, \\ 2 & \text{for } 6 \le w \le 10. \end{cases}$$

In this case, how can we construct a nonuniform code efficiently? Intuitively, we can divide all the codewords into two layers such that each layer corresponds to a uniform code, namely, we get a nonuniform code

$$C = \{\mathbf{x} \in \{0,1\}^n | w(\mathbf{x}) \leq 5, \mathbf{x} \in C_1\} \bigcup \{\mathbf{x} \in \{0,1\}^n | w(\mathbf{x}) \geq 6, \mathbf{x} \in C_2\},$$

where C_1 is a uniform code correcting 1 asymmetric error and C_2 is a uniform code correcting 2 asymmetric errors. So we can obtain a nonuniform code by combining multiple uniform codes, each of which corrects a number of asymmetric errors. We call nonuniform codes constructed in this way as *layered codes*. However, the simple construction above has a problem – due to the interference of neighbor layers, the codewords at the bottom of the higher layer may violate our requirement of reliability, namely, they cannot correct sufficient asymmetric errors. To solve this problem, we can construct a layered code in the following way: Let us first construct a uniform code correcting 2 asymmetric errors. Then we add more codewords into the code such that

1. The weights of these additional codewords are less than $4 = 6 - t_{\downarrow}(6)$. This condition can guarantee that in the resulting nonuniform code all the codewords with weights at least 6 can tolerate 2 errors.

2. These additional codewords are selected such that the codewords with weights at most 5 can tolerate 1 error.

9.5.2 Construction

Generally, given a nondecreasing function $t_{\downarrow}$, we can get a nonuniform code with $t_{\downarrow}(n)$ layers by iterating the process above. Based on this idea, given $n, t_{\downarrow}$, we construct layered codes as follows.

Let $k = t_{\downarrow}(n)$ and let $C_1, ..., C_k$ be k binary codes of codeword length n, where

$$C_1 \supset ... \supset C_k,$$

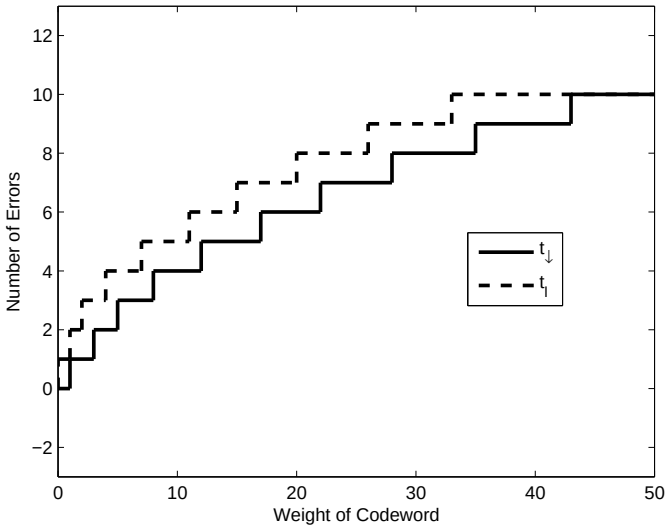


Figure 9.5. A demonstration of function $t_{\downarrow}$ and t_l.

and for $1 \leq t \leq k$, the code C_t can correct t asymmetric errors. Given $t_{\downarrow}$, we can construct a layered code C such that

$$C = \{\mathbf{x} \in \{0,1\}^n | \mathbf{x} \in C_{t_l(w(\mathbf{x}))}\},$$

where

$$\begin{aligned} t_l(w(\mathbf{x})) &= t_{\downarrow}(\max R_{w(\mathbf{x})}) \\ &= t_{\downarrow}(\max\{s | s - t_{\downarrow}(s) \leq w(\mathbf{x})\}). \end{aligned}$$

We see that there is a shift of the layers (corresponding to the function t_l and the function $t_{\downarrow}$), see figure 9.5 as a demonstration. The following theorem shows that the construction above satisfies our requirements of nonuniform codes, i.e., it corrects $t_{\downarrow}$ asymmetric errors.

Theorem 9.14. *Let C be a layered code based on the above construction, then for all $\mathbf{x} \in C$, $\mathbf{x}$ can tolerate $t_{\downarrow}(w(\mathbf{x}))$ asymmetric errors.*

Proof. We prove that for all $\mathbf{x}, \mathbf{y} \in C$ with $\mathbf{x} \neq \mathbf{y}$, $\mathcal{B}(\mathbf{x}) \bigcap \mathcal{B}(\mathbf{y}) = \phi$. W.l.o.g., we assume $w(\mathbf{x}) \geq w(\mathbf{y})$.

If $w(\mathbf{x}) - t_{\downarrow}(w(\mathbf{x})) > w(\mathbf{y})$, the conclusion is true.

If $w(\mathbf{x}) - t_{\downarrow}(w(\mathbf{x})) \leq w(\mathbf{y})$ and $w(\mathbf{x}) \geq w(\mathbf{y})$, then $\mathbf{x}, \mathbf{y} \in C_{t_l(w(\mathbf{y}))}$. That means there does not exist a word $\mathbf{z} \in \{0,1\}^n$ such that $\mathbf{x}, \mathbf{y} \geq \mathbf{z}$ and $N(\mathbf{x}, \mathbf{z}) \leq t_l(w(\mathbf{y}))$ and $N(\mathbf{y}, \mathbf{z}) \leq t_l(w(\mathbf{y}))$. Since $w(\mathbf{x}) - t_{\downarrow}(w(\mathbf{x})) \leq w(\mathbf{y})$, according to the definition of t_l, it is easy to get $t_l(w(\mathbf{y})) \geq t_{\downarrow}(w(\mathbf{x})) \geq t_{\downarrow}(w(\mathbf{y}))$. So there does not exist a word $\mathbf{z} \in \{0,1\}^n$ such that $\mathbf{x}, \mathbf{y} \geq \mathbf{z}$ and $N(\mathbf{x}, \mathbf{z}) \leq t_{\downarrow}(w(\mathbf{x}))$ and $N(\mathbf{y}, \mathbf{z}) \leq t_{\downarrow}(w(\mathbf{y}))$, namely, $\mathcal{B}(\mathbf{x}) \bigcap \mathcal{B}(\mathbf{y}) = \phi$.

This completes the proof. □

We see that the constructions of layered codes are based on the provided group of codes $C_1, ..., C_k$ such that $C_1 \supset C_2 \supset ... \supset C_k$ and for $1 \leq t \leq k$, and the code C_t corrects t asymmetric errors. Examples of such codes include Varshamov codes [121], BCH codes, etc.

The construction of Varshamov codes can be described as follows: Let $\alpha_1, \alpha_2, ..., \alpha_n$ be distinct nonzero elements of F_q, and let $\alpha := (\alpha_1, \alpha_2, ..., \alpha_n)$. For $\mathbf{x} = (x_1, x_2, ..., x_n) \in \{0,1\}^n$, let $\mathbf{x}\alpha = (x_1\alpha_1, x_2\alpha_2, ..., x_n\alpha_n)$. For $g_1, g_2, ..., g_t \in F_q$ and $1 \leq t \leq k$, let

$$C_t := \{\mathbf{x} \in \{0,1\}^n | \sigma_l(\mathbf{x}\alpha) = g_l \text{ for } 1 \leq l \leq t\},$$

where the elementary symmetric function $\sigma_l(\mathbf{u})$ for $l \geq 0$ are defined by

$$\prod_{i=1}^{r}(z + u_i) = \sum_{l=0}^{\infty} \sigma_l(\mathbf{u}) z^{r-l}.$$

Then C_t can correct t asymmetric errors (for $1 \leq t \leq k$), and $C_1 \supset C_2 \supset ... \supset C_k$.

Such a group of codes can also be constructed by BCH codes: Let $(\alpha_0, \alpha_1, ..., \alpha_{n-1})$ be n distinct nonzero elements of G_{2^m} with $n = 2^m - 1$. For $1 \leq t \leq k$, let

$$C_t := \{\mathbf{x} \in \{0,1\}^n | \sum_{i=1}^{n} x_i \alpha_i^{(2l-1)} = 0 \text{ for } 1 \leq l \leq t\}.$$

9.5.3 Decoding Algorithm

Assume $\mathbf{x}$ is a codeword in C_t and $\mathbf{y} = \mathbf{x} + \mathbf{e}$ is a received erroneous word with error vector e, then there is an efficient algorithm to decode $\mathbf{y}$ into a codeword, which is denoted by $D_t(\mathbf{y})$. If $\mathbf{y}$ has at most t asymmetric errors, then $D_t(\mathbf{y}) = \mathbf{x}$. We show that the layered codes proposed above also have an efficient decoding algorithm if $D_t(\cdot)$ (for $1 \leq t \leq k$) are provided and efficient.

Theorem 9.15. *Let C be a layered code based on the above construction, and let $\mathbf{y} = \mathbf{x} + \mathbf{e}$ be a received word such that $\mathbf{x} \in C$ and $|e| \leq t_{\downarrow}(w(\mathbf{x}))$. To recover $\mathbf{x}$ from $\mathbf{y}$, we enumerate the integers in $[t_l(w(\mathbf{y})), t_l(w(\mathbf{y}) + t_l(w(\mathbf{y})))]$. If we can find an integer t such that $D_t(\mathbf{y}) \in C$ and $N(D_t(\mathbf{y}), \mathbf{y}) \leq t_{\downarrow}(w(D_t(\mathbf{y})))$, then $D_t(\mathbf{y}) = \mathbf{x}$.*

Proof. If we let $t = t_{\downarrow}(w(\mathbf{x}))$, then we can get that t satisfies the conditions and $D_t(\mathbf{y}) = \mathbf{x}$. So such t exists.

Now we only need to prove that once there exists t satisfying the conditions in the theorem, we have $D_t(\mathbf{y}) = \mathbf{x}$. We prove this by contradiction. Assume there exists t satisfying the conditions but $\mathbf{z} = D_t(\mathbf{y}) \neq \mathbf{x}$. Then $N(\mathbf{z}, \mathbf{y}) \leq t_{\downarrow}(w(\mathbf{z}))$. Since we also have $N(\mathbf{x}, \mathbf{y}) \leq t(w(\mathbf{x}))$, $\mathcal{B}(\mathbf{x}) \bigcap \mathcal{B}(\mathbf{z}) \neq \phi$, which contradicts the property of the layered codes.

This completes the proof. □

In the above method, to decode an erroneous word $\mathbf{y}$, we can check all the integers between $t_l(w(\mathbf{y}))$ and $t_l(w(\mathbf{y}) + t_l(w(\mathbf{y})))$ to find the value of t. Once we find the integer t satisfying the conditions in the theorem, we can decode $\mathbf{y}$ into $D_t(\mathbf{y})$ directly. (Note that the length of the interval for t, namely $t_l(w(\mathbf{y}) + t_l(w(\mathbf{y}))) - t_l(w(\mathbf{y}))$, is normally much smaller than $w(\mathbf{y})$. It is approximately $\frac{p^2}{(1-p)^2} w(\mathbf{y})$ for i.i.d. errors when $w(\mathbf{y})$ is large.) We see that this decoding process is efficient if $D_t(.)$ is efficient for $1 \leq t \leq k$.

9.5.4 Layered vs.Uniform

Typically, nonlinear codes, like Varshamov codes are superior to BCH codes. But it is still not well-known how to estimate the sizes of Varshamov codes and their weight distributions. To compare

Table 9.2. BCH codes with codeword length 255

n	k	t	n	k	t
255	247	1	255	115	21
255	239	2	255	107	22
255	231	3	255	99	23
255	223	4	255	91	25
255	215	5	255	87	26
255	207	6	255	79	27
255	199	7	255	71	29
255	191	8	255	63	30
255	187	9	255	55	31
255	179	10	255	47	42
255	171	11	255	45	43
255	163	12	255	37	45
255	155	13	255	29	47
255	147	14	255	21	55
255	139	15	255	13	59
255	131	18	255	9	63
255	123	19			

[40]

uniform constructions and nonuniform constructions for correcting asymmetric errors, we focus on BCH codes, namely, we compare normal BCH codes with layered BCH codes. Here, we consider i.i.d. errors, and we assume that the codeword length is $n = 255$, the crossover probability is p and the maximal tolerated error probability is q_e.

Table 9.2 shows the relations between the dimension k and the number of errors t that can be corrected in BCH codes when $n = 255$. According to [78], many BCH codes have approximated binomial weight distribution. So given an $(255, k, t)$ BCH code, the number of codewords of weight i is approximately

$$b_i \sim 2^k \frac{\binom{n}{i}}{2^n}.$$

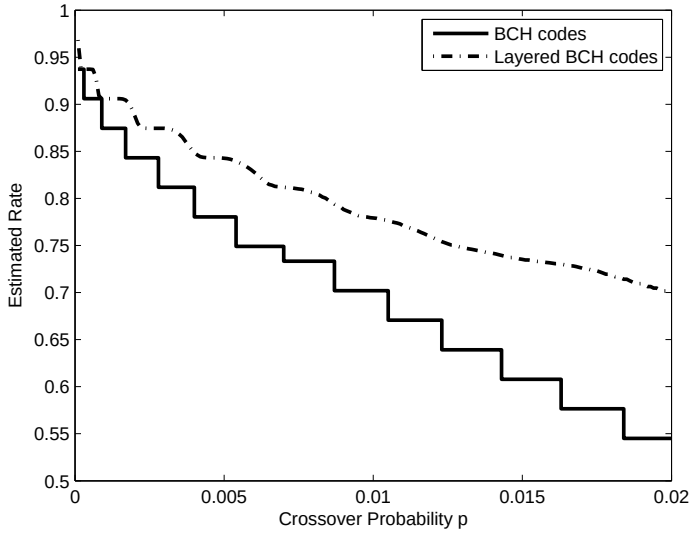


Figure 9.6. The estimated rates of BCH codes and layered BCH codes when $n = 255, q_e = 10^{-4}$.

For a normal BCH code, it has to correct t errors with

$$t = \min\{s \in N | \sum_{i=0}^{s} \binom{n}{i} p^i (1-p)^{n-i} \geq 1 - q_e\},$$

then it has 2^k codewords where k can be obtained from table 9.2 based on the value of t.

For a layered BCH code, the codewords with Hamming weight w have to correct $t_{\downarrow}(w)$ asymmetric errors such that

$$t_{\downarrow}(w) = \min\{s \in N | \sum_{i=0}^{s} \binom{w}{i} p^i (1-p)^{w-i} \geq 1 - q_e\},$$

for all $0 \leq w \leq n$. Based on the approximated weight distribution of BCH codes, the number of codewords in a layered BCH codes can be estimated by summing up the numbers of codewords with different weights.

Figure 9.6 plots the estimated rates of BCH codes and layered BCH codes for different p when $n = 255$ and $q_e = 10^{-4}$. Here, for a code C, let $\#C$ be the number of codewords, then the rate of C is defined as $\frac{\log_2(\#C)}{n}$. From this figure, we see that under the same parameters (n, p, q_e), the rates of layered BCH codes are much higher than those of BCH codes. By constructing nonuniform codes instead of uniform codes, the code rate can be significantly increased. Comparing figure 9.6 with

figure 9.3, it can be seen that the rates of layered BCH codes are very close to the upper bounds of uniform codes. It implies that we can gain more by considering nonuniform codes rather than nonlinear uniform codes.

9.6 Flipping Codes Construction

Many nonlinear codes designed to correct asymmetric errors like Varshamov codes are superior to linear codes. However, they do not yet have efficient encoding algorithms, namely, it is not easy to find an efficient encoding function $f : \{0,1\}^k \to C$ with $k \backsimeq \lfloor \log |C| \rfloor$. In this section, we focus on the approach of designing nonuniform codes for asymmetric errors with efficient encoding schemes, by utilizing the well-studied linear codes.

A simple method is that we can use a linear code to correct $t_{\downarrow}(n)$ asymmetric errors directly, but this method is inefficient not only because the decoding sphere for symmetric errors is greater than the sphere for asymmetric errors (and therefore an overkill), but also because for low-weight codewords, the number of asymmetric errors they need to correct can be much smaller than $t_{\downarrow}(n)$.

Our idea is to build a *flipping code* that uses only low-weight codewords (specifically, codewords of Hamming weight no more than $\sim \frac{n}{2}$), because they need to correct fewer asymmetric errors and therefore can increase the code's rate. In the rest of this section, we present two different constructions.

9.6.1 First Construction

First, we construct a linear code C (like BCH codes) of length n with generator matrix G that corrects $t_{\downarrow}(\lfloor \frac{n}{2} \rfloor)$ symmetric errors. Assume the dimension of the code is k. For any binary message $\mathbf{u} \in \{0,1\}^k$, we can map it to a codeword $\mathbf{x}$ in C such that $\mathbf{x} = \mathbf{u}G$. Next, let $\overline{\mathbf{x}}$ denote a word obtained by flipping all the bits in $\mathbf{x}$ such that if $x_i = 0$ then $\overline{x}_i = 1$ and if $x_i = 1$ then $\overline{x}_i = 0$; and let $\mathbf{y}$ denote the final codeword corresponding to $\mathbf{u}$. We check whether $w(\mathbf{x}) < \lfloor \frac{n}{2} \rfloor$ and construct $\mathbf{y}$

in the following way:

$$\mathbf{y} = \begin{cases} \mathbf{x}00...0 & \text{if } w(\mathbf{x}) < \lfloor \frac{n}{2} \rfloor, \\ \overline{\mathbf{x}}11...1 & \text{otherwise.} \end{cases}$$

Here, the auxiliary bits (0s or 1s) are added to distinguish that whether $\mathbf{x}$ has been flipped or not, and they form a repetition code to tolerate errors.

The corresponding decoding process is straightforward: Assume we received a word $\mathbf{y}'$. If there is at least one 1 in the auxiliary bits, then we "flip" the word by changing all 0s to 1s and all 1s to 0s; otherwise, we keep the word unchanged. Then we apply the decoding scheme of the code C to the first n bits of the word. Finally, the message $\mathbf{u}$ can be successfully decoded if $\mathbf{y}'$ has at most $t_{\downarrow}(\lfloor \frac{n}{2} \rfloor)$ errors in the first n bits.

9.6.2 Second Construction

In the previous construction, several auxiliary bits are needed to protect one bit of information, which is not very efficient. Here we try to move this bit into the message part of the codewords in C. This motivates us to give the following construction.

Let C be a systematic linear code with length n that corrects t' symmetric errors (we will specify t' later). Assume the dimension of the code is k. Now, for any binary message $\mathbf{u} \in \{0,1\}^{k-1}$ of length $k-1$, we get $\mathbf{u}' = 0\mathbf{u}$ by adding one bit 0 in front of $\mathbf{u}$. Then we can map $\mathbf{u}'$ to a codeword $\mathbf{x}$ in C such that

$$\mathbf{x} = (0\mathbf{u})G = 0\mathbf{u}\mathbf{v},$$

where G is the generator matrix of C in systematic form and the length of $\mathbf{v}$ is $n-k$. Let α be a codeword in C such that the first bit $\alpha_1 = 1$ and its weight is the maximal one among all the codeword in C, i.e.,

$$\alpha = \arg \max_{\mathbf{x} \in C, x_1 = 1} w(\mathbf{x}).$$

Generally, $w(\alpha)$ is very close to n. For example, in any primite BCH code of length 255, α is the all-one vector. In order to reduce the weights of the codewords, we use the following operations:

Calculate the relative weight

$$w(\mathbf{x}|\alpha) = |\{1 \leq i \leq n | x_i = 1, \alpha_i = 1\}|.$$

Then we get the final codeword

$$\mathbf{y} = \begin{cases} \mathbf{x} + \alpha & \text{if } w(\mathbf{x}|\alpha) > \frac{w(\alpha)}{2}, \\ \mathbf{x} & \text{otherwise,} \end{cases}$$

where $+$ is the binary sum, so $\mathbf{x} + \alpha$ is to flip the bits in $\mathbf{x}$ corresponding the ones in α. So far, we see that the maximal weight for $\mathbf{y}$ is $\lfloor n - \frac{w(\alpha)}{2} \rfloor$. That means we need to select t' such that

$$t' = t_{\downarrow}(\lfloor n - \frac{w(\alpha)}{2} \rfloor).$$

For many linear codes, α is the all-one vector, so $t' = t_{\downarrow}(\lfloor \frac{n}{2} \rfloor)$.

In the above encoding process, for different binary messages, they have different codewords. And for any codeword $\mathbf{y}$, we have $\mathbf{y} \in C$. That is because either $\mathbf{y} = \mathbf{x}$ or $\mathbf{y} = \mathbf{x} + \alpha$, where both $\mathbf{x}$ and α are codewords in C and C is a linear code. So the resulting flipping code is a subset of code C.

The decoding process is very simple: Given the received word $\mathbf{y}' = \mathbf{y} + \mathbf{e}$, we can always get $\mathbf{y}$ by applying the decoding scheme of the linear code if $|\mathbf{e}| \leq t'$. If $y_1 = 1$, that means $\mathbf{x}$ has been flipped based on α, so we have $\mathbf{x} = \mathbf{y} + \alpha$; otherwise, $\mathbf{x} = \mathbf{y}$. Then the initial message $\mathbf{u} = x_2 x_3 ... x_k$.

We see that the second construction is a little more efficient than the first one, by moving the 'flipping' bit from the outside of a codeword (of an error-correcting code) to the inside. Here is an example of the second construction: Let C be the $(7, 4)$ Hamming code, which is able to correct

single-bit errors. The generating matrix of the $(7, 4)$ Hamming code is

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Here we have $t' = 1$ and $k = 4$. Assume the binary message is $\mathbf{u} = 011$, then we have $\mathbf{x} = (0\mathbf{u})G = 0011100$. It is easy to see that α is the all-one codeword, i.e., $\alpha = 1111111$. In this case, $w(\mathbf{x}|\alpha) <= \frac{w(\alpha)}{2}$, so the final codeword $\mathbf{y} = 0011100$. Assume the binary message is $\mathbf{u} = 110$, then we have $\mathbf{x} = (0\mathbf{u})G = 0110110$. In this case, $w(\mathbf{x}|\alpha) > \frac{w(\alpha)}{2}$, so the final codeword $\mathbf{y} = \mathbf{x}+\alpha = 1001001$.

Assume the received word is $\mathbf{y}' = 0001001$. By applying the decoding algorithm of Hamming codes, we get $\mathbf{y} = 1001001$. Since $y_1 = 1$, we have $\mathbf{x} = \mathbf{y} + \alpha$, and as a result, $\mathbf{u} = 110$.

9.6.3 Flipping vs.Layered

When n is sufficiently large, the flipping codes above become nearly as efficient (in terms of code rate) as a linear codes correcting $t_{\downarrow}(\lfloor \frac{n}{2} \rfloor)$ symmetric errors. It is much more efficient than designing a linear code correcting $t_{\downarrow}(n)$ symmetric errors. Note that when n is large and p is small, these codes can have very good performance on code rate. That is because when n is sufficiently large, the rate of an optimal nonuniform code is dominated by the codewords with the same Hamming weight w_d ($\leq \frac{n}{2}$), and w_d approaches $\frac{n}{2}$ as p gets close to 0. We can intuitively understand it based on two facts when n is sufficiently large: (1) There are at most $n2^{n(H(\frac{w_d}{n})+\delta)}$ codewords in this optimal nonuniform code. (2) When p becomes small, we can get a nonuniform code with at least $2^{n(1-\delta)}$ codewords. So when n is sufficiently large and p is small, we have $w_d \to \frac{n}{2}$. Hence, an optimal nonuniform code has almost the same asymptotic performance with an optimal weight-bounded code (Hamming weight is at most n/2) that corrects $t_{\downarrow}(n/2)$ asymmetric errors.

Let us consider a flipping BCH code based on the second construction. Similar as the previous section, we assume that the codeword length is $n = 255$ and the number of codewords with weight

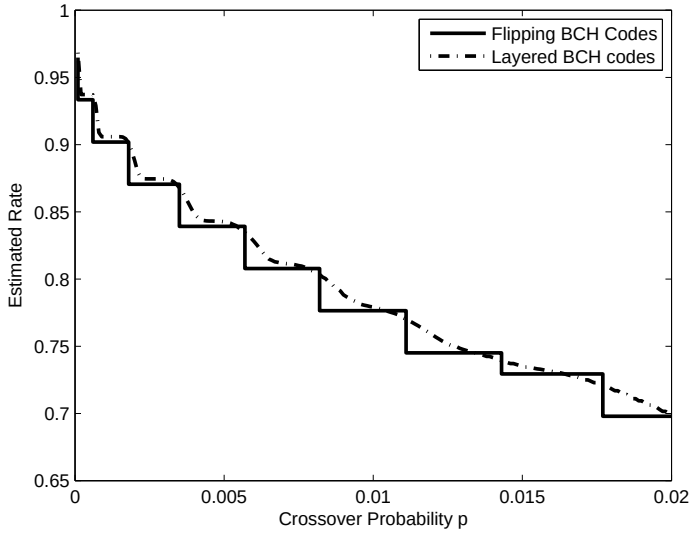


Figure 9.7. The estimated rates of flipping/layered BCH codes when $n = 255, q_e = 10^{-4}$.

i can be approximated by

$$2^k \frac{\binom{n}{i}}{2^n},$$

where k is the dimension of the code. Figure 9.7 compares the estimated rates of flipping BCH codes and those of layered BCH codes when $n = 255$ and $q_e = 10^{-4}$. Surprisingly, the flipping BCH codes achieves almost the same rates as layered BCH codes. Note that, for the layered codes, we are able to further improve the efficiency (rates) by replacing BCH codes with Varshamov codes.

9.7 Extension to Binary Asymmetric Channels

In the previous sections, we have introduced and studied nonuniform codes for Z-channels. The concept of nonuniform codes can be extended from Z-channels to general binary asymmetric channels, where the error probability from 0 to 1 is smaller than the error probability from 1 to 0 but it may not be ignorable. In this case, we are able to construct nonuniform codes correcting a big number of $1 \to 0$ errors and a small number of $0 \to 1$ errors. Such codes can be used in flash memories or phase change memories, where the change in data has an asymmetric property. For example, the stored data in flash memories is represented by the voltage levels of transistors, which drift in one direction

because of charge leakage. In phase change memories, another class of nonvolatile memories, the stored data is determined by the electrical resistance of the cells, which also drifts due to thermally activated crystallization of the amorphous material. This asymmetric property will introduce more $1 \to 0$ errors after a long duration.

In this section, we first investigate binary asymmetric channels where the probability from 0 to 1 is much smaller than that from 1 to 0, namely, $p_\uparrow \ll p_\downarrow$, but $p_\uparrow$ is not ignorable. In this case, we can let $t_\uparrow$ be a constant function. Later, we consider general binary asymmetric channels, where $t_\uparrow$ can be an arbitrary nonincreasing step function.

9.7.1 $t_\uparrow$ Is a Constant Function

We show that if $t_\uparrow$ is a constant function, then correcting $[t_\downarrow, t_\uparrow]$ errors is equivalent to correcting $t_\downarrow + t_\uparrow$ asymmetric errors, where $t_\downarrow$ can be an arbitrary step functions on $\{0, 1, ..., n\}$.

Theorem 9.16. *Let $t_\uparrow$ be a constant function, a code C is a nonuniform code correcting $[t_\downarrow, t_\uparrow]$ errors if and only if it is a nonuniform code correcting $t_\downarrow + t_\uparrow$ asymmetric errors.*

Proof. 1) We first show that if C is a nonuniform code correcting $[t_\downarrow, t_\uparrow]$ errors where $t_\uparrow$ is a constant function, then it can correct $t_\downarrow + t_\uparrow$ asymmetric errors. We need to prove that there does not exists a pair of codewords $\mathbf{x}, \mathbf{y} \in C$ such that

$$N(\mathbf{x}, \mathbf{y}) \leq t_\downarrow(w(\mathbf{x})) + t_\uparrow,$$

$$N(\mathbf{y}, \mathbf{x}) \leq t_\downarrow(w(\mathbf{y})) + t_\uparrow,$$

where

$$N(\mathbf{x}, \mathbf{y}) \triangleq |\{i : x_i = 1, y_i = 0\}|.$$

Let us prove it by contradiction. Assume that their exists a pair of codewords $\mathbf{x}, \mathbf{y}$ that satisfy the inequalities above. By adding at most $t_\uparrow$ $0 \to 1$ errors, we get a vector $\mathbf{x}'$ from $\mathbf{x}$ such that the Hamming distance between $\mathbf{x}'$ and $\mathbf{y}$ is minimized; also we get a vector $\mathbf{y}'$ from $\mathbf{y}$ such that the

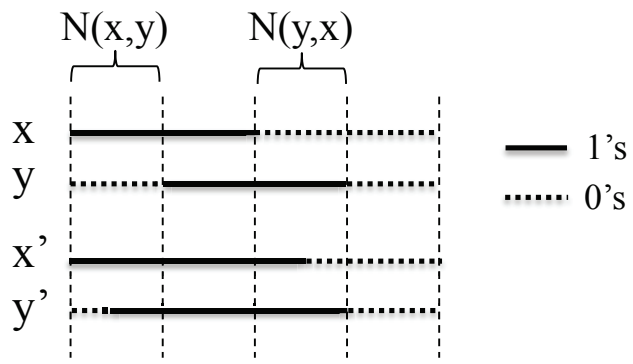


Figure 9.8. A demonstration of $\mathbf{x}, \mathbf{y}, \mathbf{x}', \mathbf{y}'$.

Hamming distance between $\mathbf{y}'$ and $\mathbf{x}$ is minimized. In this case, we only need to show that

$$N(\mathbf{x}', \mathbf{y}') \leq t_{\downarrow}(w(\mathbf{x})), N(\mathbf{y}', \mathbf{x}') \leq t_{\downarrow}(w(\mathbf{y})),$$

which contradicts with our assumption that C can correct $[t_{\downarrow}, t_{\uparrow}]$ errors. The intuitive way of understanding $\mathbf{x}', \mathbf{y}'$ is shown in figure 9.8. In the figure, we present each vector as a line, in which the solid part is for 1s and the dashed part is for 0s.

If $N(\mathbf{x}', \mathbf{x}) < t_{\uparrow}$ and $N(\mathbf{y}', \mathbf{y}) < t_{\uparrow}$, then

$$x'_i = \max(x_i, y_i) = y'_i,$$

so $\mathbf{x}' = \mathbf{y}'$. The statement is true.

If $N(\mathbf{x}', \mathbf{x}) < t_{\uparrow}$ and $N(\mathbf{y}', \mathbf{y}) = t_{\uparrow}$, then $\mathbf{y}' \leq \mathbf{x}'$. In this case,

$$N(\mathbf{x}', \mathbf{y}') \leq N(\mathbf{x}, \mathbf{y}) - t_{\uparrow} \leq t_{\downarrow}(w(\mathbf{x})).$$

We get the statement.

Similarly, if $N(\mathbf{y}', \mathbf{y}) < t_{\uparrow}$ and $N(\mathbf{x}', \mathbf{x}) = t_{\uparrow}$, we have $\mathbf{x}' \leq \mathbf{y}'$ and

$$N(\mathbf{y}', \mathbf{x}') \leq N(\mathbf{y}, \mathbf{x}) - t_{\uparrow} \leq t_{\downarrow}(w(\mathbf{y})).$$

If $N(\mathbf{x}', \mathbf{x}) = t_\uparrow$ and $N(\mathbf{y}', \mathbf{y}) = t_\uparrow$, we can get

$$N(\mathbf{x}', \mathbf{y}') \leq N(\mathbf{x}, \mathbf{y}) - t_\uparrow \leq t_\downarrow(w(\mathbf{x})),$$

$$N(\mathbf{y}', \mathbf{x}') \leq N(\mathbf{y}, \mathbf{x}) - t_\uparrow \leq t_\downarrow(w(\mathbf{y})).$$

Based on the discussions above, we can conclude that if C is a nonuniform code correcting $[t_\downarrow, t_\uparrow]$ errors where $t_\uparrow$ is a constant function, then it is also a nonuniform code correcting $t_\downarrow + t_\uparrow$ asymmetric errors.

2) We show that if C is a nonuniform codes correcting $t_\downarrow + t_\uparrow$ asymmetric errors where $t_\uparrow$ is a constant function, then it is also a nonuniform code correcting $[t_\downarrow, t_\uparrow]$ errors. That means for any $\mathbf{x}, \mathbf{y} \in C$, there does not exist a vector $\mathbf{v}$ such that

$$N(\mathbf{v}, \mathbf{x}) \leq t_\uparrow, \quad N(\mathbf{x}, \mathbf{v}) \leq t_\downarrow(w(\mathbf{x})),$$

$$N(\mathbf{v}, \mathbf{y}) \leq t_\uparrow, \quad N(\mathbf{y}, \mathbf{v}) \leq t_\downarrow(w(\mathbf{y})).$$

Let us prove this by contradiction. We assume there exists a vector $\mathbf{v}$ satisfies the above conditions. Now, we define a few vectors $\mathbf{x}', \mathbf{y}', \mathbf{u}$ such that

$$x'_i = \min(x_i, v_i) \quad \forall 1 \leq i \leq n,$$

$$y'_i = \min(y_i, v_i) \quad \forall 1 \leq i \leq n,$$

$$u_i = \min(x_i, y_i, v_i) \quad \forall 1 \leq i \leq n.$$

The intuitive way of understanding these vectors is shown in figure 9.9. In the figure, we present each vector as a line, in which the solid part is for 1s and the dashed part is for 0s.

Then

$$\mathbf{x}' \leq \mathbf{x}, \mathbf{x}' \leq \mathbf{v}, N(\mathbf{x}, \mathbf{x}') \leq t_\downarrow(w(\mathbf{x})), N(\mathbf{v}, \mathbf{x}') \leq t_\uparrow,$$

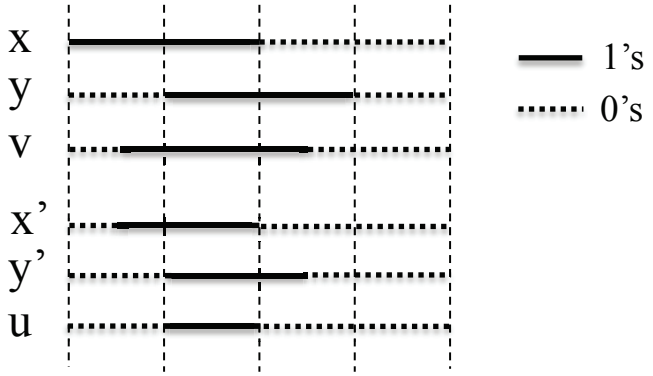


Figure 9.9. A demonstration of $\mathbf{x}, \mathbf{y}, \mathbf{x}', \mathbf{y}', \mathbf{v}, \mathbf{u}$.

$$\mathbf{y}' \leq \mathbf{y}, \mathbf{y}' \leq \mathbf{v}, N(\mathbf{y}, \mathbf{y}') \leq t_{\downarrow}(w(\mathbf{y})), N(\mathbf{v}, \mathbf{y}') \leq t_{\uparrow}.$$

Now we want to show that

$$N(\mathbf{x}, \mathbf{u}) \leq t_{\downarrow}(w(\mathbf{x})) + t_{\uparrow}.$$

Since

$$N(\mathbf{x}, \mathbf{u}) \leq N(\mathbf{x}, \mathbf{x}') + N(\mathbf{x}', \mathbf{u}),$$

we only to show that

$$N(\mathbf{x}', \mathbf{u}) \leq t_{\uparrow}.$$

According to the definition of $\mathbf{u}$, it is easy to get that

$$\begin{aligned} N(\mathbf{v}, \mathbf{x}') + N(\mathbf{x}', \mathbf{u}) &= N(\mathbf{v}, \mathbf{y}') + N(\mathbf{y}', \mathbf{u}) \\ &\leq N(\mathbf{v}, \mathbf{x}') + N(\mathbf{v}, \mathbf{y}') \end{aligned}$$

So $N(\mathbf{x}', \mathbf{u}) \leq t_{\uparrow}$, which leads us to

$$N(\mathbf{x}, \mathbf{u}) \leq t_{\downarrow}(w(\mathbf{x})) + t_{\uparrow}.$$

Similarly, we can also get

$$N(\mathbf{y}, \mathbf{u}) \leq t_{\downarrow}(w(\mathbf{y})) + t_{\uparrow}.$$

In this case, C is not a nonuniform codes correcting $t_{\downarrow} + t_{\uparrow}$ asymmetric errors, which contradicts with our assumption.

Based on the discussions above, we can get the conclusion in the theorem. □

According to the above theorem, all our results for Z-channels, like upper bounds and constructions of nonuniform codes, can apply to nonuniform codes correcting $[t_{\downarrow}, t_{\uparrow}]$ errors if $t_{\uparrow}$ is a constant function.

9.7.2 $t_{\uparrow}$ Is a Nonincreasing Function

Another case of binary asymmetric channel is that $p_{\uparrow} < p_{\downarrow}$ but $p_{\uparrow}$ is not much smaller than $p_{\downarrow}$. In this case, it is not efficient to write $t_{\uparrow}$ as a constant function. Instead, we consider it as a nonincreasing step function.

Theorem 9.17. *Let $t_{\downarrow}$ be a nondecreasing function and $t_{\uparrow}$ be a nonincreasing function. A code C is a nonuniform code correcting $[t_{\downarrow}, t_{\uparrow}]$ errors if it is a nonuniform code correcting $t_{\downarrow} + \overline{t_{\uparrow}}$ asymmetric errors. Here, for all $0 \leq w \leq n$,*

$$\overline{t_{\uparrow}}(w) = t_{\uparrow}(\max\{s | t_{\uparrow}(s) + s \leq w - t_{\downarrow}(w)\}).$$

Proof. Let C be a nonuniform code correcting $t_{\downarrow} + \overline{t_{\uparrow}}$ errors. For any $\mathbf{x}, \mathbf{y} \in C$, w.l.o.g, we assume $w(\mathbf{x}) \leq w(\mathbf{y})$. If $w(\mathbf{x}) + t_{\uparrow}(w(\mathbf{x})) < w(\mathbf{y}) - t_{\downarrow}(w(\mathbf{y}))$, then there does not exist a vector $\mathbf{v}$ such that

$$N(\mathbf{v}, \mathbf{x}) \leq t_{\uparrow}, \quad N(\mathbf{x}, \mathbf{v}) \leq t_{\downarrow}(w(\mathbf{x})),$$

$$N(\mathbf{v}, \mathbf{y}) \leq t_{\uparrow}, \quad N(\mathbf{y}, \mathbf{v}) \leq t_{\downarrow}(w(\mathbf{y})).$$

If $w(\mathbf{x}) + t_{\uparrow}(w(\mathbf{x})) \geq w(\mathbf{y}) - t_{\downarrow}(w(\mathbf{y}))$, according to the proof in theorem 9.16, we can get that

there does not exist a vector $\mathbf{v}$ such that

$$N(\mathbf{v},\mathbf{x}) \le t_{\uparrow}(w(\mathbf{x})),$$

$$N(\mathbf{x},\mathbf{v}) \le t_{\downarrow}(w(\mathbf{x})) + \overline{t_{\uparrow}}(w(\mathbf{x})) - t_{\uparrow}(w(\mathbf{x}));$$

$$N(\mathbf{v},\mathbf{y}) \le t_{\uparrow}(w(\mathbf{x})),$$

$$N(\mathbf{y},\mathbf{v}) \le t_{\downarrow}(w(\mathbf{y})) + \overline{t_{\uparrow}}(w(\mathbf{y})) - t_{\uparrow}(w(\mathbf{x})).$$

Since

$$\overline{t_{\uparrow}}(w(\mathbf{x})) - t_{\uparrow}(w(\mathbf{x})) \ge 0,$$

$$t_{\uparrow}(w(\mathbf{x})) \ge t_{\uparrow}(w(\mathbf{y})),$$

$$\overline{t_{\uparrow}}(w(\mathbf{y})) \ge t_{\uparrow}(w(\mathbf{x})),$$

we can get that there does not exist a vector $\mathbf{v}$ such that

$$N(\mathbf{v},\mathbf{x}) \le t_{\uparrow}, \quad N(\mathbf{x},\mathbf{v}) \le t_{\downarrow}(w(\mathbf{x})),$$

$$N(\mathbf{v},\mathbf{y}) \le t_{\uparrow}, \quad N(\mathbf{y},\mathbf{v}) \le t_{\downarrow}(w(\mathbf{y})).$$

Finally, we conclude that C is a nonuniform code correcting $[t_{\downarrow}, t_{\uparrow}]$ errors. □

According to the above theorem, we can convert the problem of constructing a nonuniform codes for an arbitrary binary asymmetric channel to the problem of constructing a nonuniform correcting only $1 \to 0$ errors. Note that this conversion results in a little loss of code efficiency, but typically it is very small. Both layered codes and flipping codes can be applied for correcting errors in binary asymmetric channels. A little point to notice is that $t_{\downarrow} + \overline{t_{\uparrow}}$ might not be a strict nondecreasing function of codeword weight. In this case, we can find a nondecreasing function t_h which is slightly larger than $t_{\downarrow} + \overline{t_{\uparrow}}$, and construct a nonuniform code correcting t_h asymmetric errors.

When we apply flipping codes for correcting errors in binary asymmetric channels, we do not have to specify $t_\downarrow$ and $t_\uparrow$ separately. For example, assume that i.i.d. errors are considered. If the maximal tolerated error probability is q_e, then given a codeword of weight w, it has to tolerate total $t_f(w)$ errors. For $0 \leq w \leq n$, $t_f(w)$ can be obtained by calculating the minimal integer t such that

$$\sum_{i=0}^{t}\sum_{j=0}^{t-i}\binom{w}{i}\binom{n-w}{j}p_\downarrow^i(1-p_\downarrow)^{w-i}p_\uparrow^j(1-p_\uparrow)^{(n-w-j)} \geq 1-q_e.$$

To construct a flipping code, we only need to find a linear code such that it corrects $t_f(\lfloor n-\frac{\alpha}{2}\rfloor)$ symmetric errors, where α is the codeword with the maximum weight in the linear code.

Theorem 9.18. *Let $t_\downarrow$ be a nondecreasing function and $t_\uparrow$ be a nonincreasing function. If a code C is a nonuniform code correcting $[t_\downarrow, t_\uparrow]$ errors, then it corrects $t_\downarrow + \underline{t_\uparrow}$ asymmetric errors. Here,*

$$\underline{t_\uparrow}(w) = t_\uparrow(\min\{s|s - t_\uparrow(s) - t_\downarrow(s) \leq w\}).$$

Proof. The proof of this theorem is very similar as that for the previous theorem. It follows the conclusion in theorem 9.16. □

According to the theorem above, to calculate the upper bound of nonuniform codes correcting $[t_\downarrow, t_\uparrow]$ errors, we can first calculate the upper bound of nonuniform codes correcting $t_\downarrow + \underline{t_\uparrow}$ asymmetric errors. Generally speaking, nonuniform codes correcting $[t_\downarrow, t_\uparrow]$ errors (considering the optimal case) are more efficient than nonuniform codes correcting $t_\downarrow + \overline{t_\uparrow}$ asymmetric errors, but less efficient than those correcting $t_\downarrow + \underline{t_\uparrow}$ asymmetric errors. According to the definitions of $\underline{t_\uparrow}$ and $\overline{t_\uparrow}(w)$, it is easy to get that

$$\underline{t_\uparrow}(w) \leq t_\uparrow(w) \leq \overline{t_\uparrow}(w),$$

for $0 \leq w \leq n$. Typically, if $p_\downarrow, p_\uparrow \ll 1$, then $\overline{t_\uparrow}(w) - \underline{t_\uparrow}(w) \ll t_\uparrow(w)$. It implies that nonuniform codes correcting $[t_\downarrow, t_\uparrow]$ errors are roughly as efficient as those correcting $t_\downarrow + t_\uparrow$ asymmetric errors. If we consider i.i.d. errors and long codewords, it is equally difficult to correct errors introduced by a binary asymmetric channel with crossover probabilities $p_\downarrow$ and $p_\uparrow$ or a Z-channel with a crossover

probability $p_{\downarrow} + p_{\uparrow}$.

9.8 Conclusion

In storage systems with asymmetric errors, it is very desirable to design a code such that the reliability of each codeword is guaranteed and the size of the code is maximized. This motivates us to propose the concept of nonuniform codes, whose codewords can tolerate different numbers of asymmetric errors depending on their Hamming weights. In this chapter, we gave an almost explicit upper bound for the sizes of nonuniform codes and studied the asymptotic performances of nonuniform codes and uniform codes, which shows the potential performance gain by nonuniform codes. We also presented two general constructions of nonuniform codes, including layered codes and flipping codes. Finally, we showed that nonuniform codes for Z-channels and those for binary asymmetric channels can convert to each other. Since more needs to be known on the efficient mapping between information bits and codewords for layered codes, and the efficiency of flipping codes still needs improvement when p is not small, how to design simple and efficient nonuniform codes is still an open problem.

Chapter 10

Balanced Modulation for Nonvolatile Memories

This chapter presents a practical writing/reading scheme in nonvolatile memories, called balanced modulation, for minimizing the asymmetric component of errors. The main idea is to encode data using a balanced error-correcting code. When reading information from a block, it adjusts the reading threshold such that the resulting word is also balanced or approximately balanced. Balanced modulation has suboptimal performance for any cell-level distribution and it can be easily implemented in the current systems of nonvolatile memories.[1]

10.1 Introduction

Nonvolatile memories, like EPROM, EEPROM, Flash memory or Phase-change memory (PCM), are memories that can keep the data content even without power supply. This property enables them to be used in a wide range of applications, including cellphones, consumers, automotive and computers. Many research studies have been carried out on nonvolatile memories because of their unique features, attractive applications and huge marketing demands.

An important challenge for most nonvolatile memories is data reliability. The stored data can be lost due to many mechanisms, including cell heterogeneity, programming noise, write disturbance, read disturbance, etc. [12, 89]. From a long-term view, the change in data has an asymmetric

[1] Some of the results presented in this chapter have been previously published in [145].

property. For example, the stored data in flash memories is represented by the voltage levels of transistors, which drift in one direction because of charge leakage. In PCM, another class of nonvolatile memories, the stored data is determined by the electrical resistance of the cells, which drifts due to thermally activated crystallization of the amorphous material [135]. All these mechanisms make the errors in nonvolatile memories be heterogeneous, asymmetric, time dependent and unpredictable. These properties bring substantial difficulties to researchers attempting to develop simple and efficient error-correcting schemes.

To date, existing coding schemes for nonvolatile memories commonly use fixed thresholds to read data. For instance, in flash memories, a threshold voltage level $\mathbf{v}$ is predetermined; when reading data from a cell, it gets '1' if the voltage level is higher than $\mathbf{v}$, and otherwise it gets '0'. To increase data reliability, error-correcting codes such as Hamming code, BCH code, Reed-Solomon code and LDPC code are applied in nonvolatile memories to combat errors. Because of the asymmetric feature of nonvolatile memories, a fixed threshold usually introduces too many asymmetric errors after a long duration [85], namely, the number of $1 \to 0$ errors is usually much larger than the number of $0 \to 1$ errors. To overcome the limitations of fixed thresholds in reading data in nonvolatile memories, dynamic thresholds are introduced in this chapter. To better understand this, we use flash memories for illustration, see figure 10.1. The top figure is for newly written data, and the bottom figure is for old data that has been stored for a long time T. In the figures, assume the left curve indicates the voltage distribution for bit '0' (a bit '0' is written during programming) and the right curve indicates the voltage distribution for bit '1'. At time 0 (the moment after programming), it is best to set the threshold voltage as $\mathbf{v} = v_1$, for separating bit '1' and '0'. But after a period of time, the voltage distribution will change. In this case, v_1 is no longer the best choice, since it will introduce too many $1 \to 0$ errors. Instead, we can set the threshold voltage as $\mathbf{v} = v_2$ (see the second plot in the figure), to minimize the error probability. This also applies to other nonvolatile memories, such as PCMs.

Although best dynamic reading thresholds lead to much less errors than fixed ones, certain difficulties exist in determining their values at a time t. One reason is that the accurate level

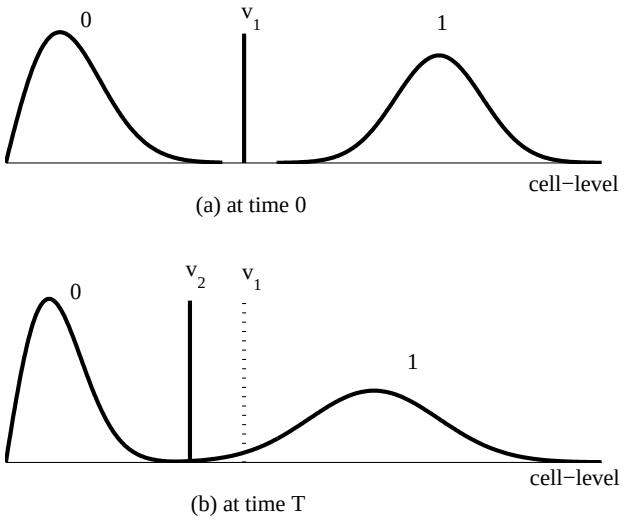


Figure 10.1. An illustration of the voltage distributions for bit "1" and bit "0" in flash memories.

distributions for bit '1' and '0' at any the current time are hard to obtain due to the lack of time records, the heterogeneity of blocks, and the unpredictability of exceptions. Another possible method is to classify all the cell levels into two groups based on unsupervised clustering and then map them into '1's and '0's. But when the border between bit '1's and '0's becomes fuzzy, mistakes of clustering may cause significant number of reading errors. In view of these considerations, in this chapter, we introduce a simple and practical writing/reading scheme in nonvolatile memories, called *balanced modulation*, which is based on the construction of balanced codes (or balanced error-correcting codes) and it aims to minimize the asymmetric component of errors in the current block.

Balanced codes, whose codewords have an equal number of 1s and 0s, have been studied in several literatures. Knuth, in 1986, proposed a simple method of constructing balanced codes [69]. In his method, given an information word of k-bits (k is even), the encoder inverts the first i bits such that the modified word has an equal number of 1s and 0s. Knuth showed that such an integer i always exists, and it is represented by a balanced word of length p. Then a codeword consists of an p-bit prefix word and an k-bit modified information word. For decoding, the decoder can easily retrieve the value of i and then get the original information word by inverting the first i

bits of the k-bit information word again. Knuth's method was later improved or modified by many researchers [6, 55, 111, 130]. Based on balanced codes, we have a scheme of balanced modulation. It encodes the stored data as balanced codewords; when reading data from a block, it adjusts the reading threshold dynamically such that the resulting word to read is also balanced (namely, the number of 1s is equal to the number of 0s) or approximately balanced. Here, we call this dynamic reading threshold as a *balancing threshold*.

There are several benefits of applying balanced modulation in nonvolatile memories. First, it increases the *safety gap* of 1s and 0s. With a fixed threshold, the *safety gap* is determined by the minimum difference between cell levels and the threshold. With balanced modulation, the safety gap is the minimum difference between cell levels for 1 and those for 0. Since the cell level for an individual cell has a random distribution due to the cell-programming noise [17, 76], the actual value of the charge level varies from one write to another. In this case, balanced modulation is more robust than the commonly used fixed-threshold approach in combating programming noise. Second, as we discussed, balanced modulation can is a very simple solution that minimizes the influence of cell-level drift. It was shown in [19] that cell-level drift in flash memories introduces the most dominating errors. Third, balanced modulation can efficiently reduce errors introduced by some other mechanisms, such as the change of external temperatures and the current leakage of other reading lines, which result in the shift of cell levels in a same direction. Generally, balanced modulation is a simple approach that minimizes the influence of noise asymmetries, and it can be easily implemented on current memory devices without hardware changes. The balanced condition on codewords enables us to select a much better threshold dynamically than the commonly used fixed threshold when reading data from a block.

The main contributions of the chapter are

1. We study balanced modulation as a simple, practical and efficient approach to minimize asymmetric component of errors in nonvolatile memories.

2. A new construction of balanced error-correcting codes, called balanced LDPC code, is introduced and analyzed, which has a higher rate than prior constructions.

3. We investigate partial-balanced modulation, for its simplicity of constructing error-correcting codes, and then we extend our discussions from binary cells to multi-level cells.

10.2 Scope of This Chapter

10.2.1 Performance and Implementation

In the first part of this chapter, including section 10.3, section 10.4 and section 10.5, we focus on the introduction and performance of balanced modulation. In particular, we demonstrate that balanced modulation introduces much less errors than the traditional approach based on fixed thresholds. For any cell-level distributions, the balancing threshold used in balanced modulation is suboptimal among all the possible reading thresholds, in the term of total number of errors. It enables balanced modulation to be adaptive to a variety of channels characters, hence, it makes balanced modulation applicable for most types of nonvolatile memories. Beyond storage systems, balanced modulation can also be used in optimal communication, where the strength of received signals shifts due to many factors like the transmitting distance, temperature, etc.

A practical and very attractive aspect of balanced modulation is that it can be easily implemented in the current systems of nonvolatile memories. The only change is that, instead of using a fixed threshold in reading a binary vector, it allows this threshold to be adaptive. Fortunately, this operation can be implemented physically, making the process of data reading reasonably fast. In this case, the reading process is based on hard decision.

If we care less about reading speed, we can have soft-decision decoding, namely, reading data without using a threshold. We demonstrate that the prior knowledge that the stored codeword is balanced is very useful. It helps us to better estimate the current cell-level distributions, hence, resulting in a better performance in bit error rate.

10.2.2 Balanced LDPC Code

Balanced modulation can efficiently reduce bit error rate when reading data from a block. A further question is how to construct balanced codes that are capable of correcting errors. We call such codes *balanced error-correcting codes*. Knuth's method cannot correct errors. In [119], van Tilborg and Blaum presented a family of balanced binary error-correcting codes. The idea is to consider balanced blocks as symbols over an alphabet and to construct error-correcting codes over that alphabet by concatenating n blocks of length $2l$ each. Due to the constraint in the code construction, this method achieves only moderate rates. Error-correcting balanced codes with higher rates were presented by Al-Bassam and Bose in [6], however, their construction considers only the case that the number of errors is at most 4. In [82], Mazumdar, Roth, and Vontobel studied linear balancing sets, namely, balancing sets that are linear subspaces $\mathbb{F}^n$, which are applied in obtaining coding schemes that combine balancing and error correction. Recently, Weber, Immink and Ferreira extent Knuth's method to let it equipped with error-correcting capabilities [131]. Their idea is to assign different error protection levels to the prefix and modified information word in Knuth's construction. So their construction is a concatenation of two error-correct codes with different error correcting capabilities. In section 10.6, we introduce a new construction of balanced error-correcting codes, which is based on LDPC code, so called balanced LDPC code. Such a construction has a simple encoding algorithm and its decoding complexity based on message-passing algorithm is asymptotically equal to the decoding complexity of the original (unbalanced) LDPC code. We demonstrate that balanced LDPC code has error-correcting capability very close to the original (unbalanced) LDPC code.

10.2.3 Partial-Balanced Modulation and Its Extension

Our observation is that the task of constructing efficient balanced error-correcting codes with simple encoding and decoding algorithms is not simple, but it is much easier to construct error-correcting codes that are partially balanced, namely, only a certain segment (or subsequence) of each codeword is balanced. Motivated by this observation, we propose a variant of balanced modulation, called partial-balanced modulation. When reading from a block, it adjusts the reading threshold such that

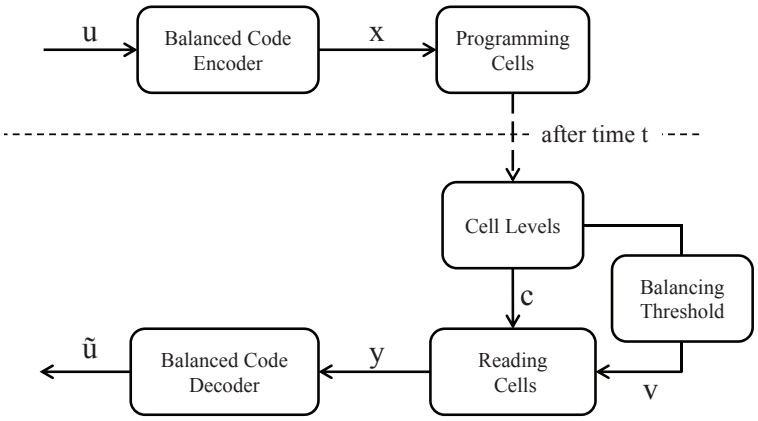


Figure 10.2. The diagram of balanced modulation.

the segment of the resulting word is balanced. Partial-balanced modulation has a performance very close to that of balanced modulation, and it has much simpler constructions of error-correcting codes than balanced modulation. Another question that we address in the third part is how to extend the scheme of balanced modulation or partial-balanced modulation to be used in nonvolatile memories with multi-level cells. Details will be provided in section 10.7 and section 10.8.

10.3 Balanced Modulation

For convenience, we consider different types of nonvolatile memories in the same framework where data is represented by cell levels, such as voltages in flash memories and resistance in phase-change memories. The scheme of balanced modulation is sketched in figure 10.2. It can be divided into two steps: programming step and reading step.

(1) In the programming step, we encode data based a balanced (error-correcting) code. Let k denote the dimension of the code and n denote the number of cells in a block, then given a message $\mathbf{u} \in \{0,1\}^n$, it is mapped to a balanced codeword $\mathbf{x} \in \{0,1\}^n$ such that $|\mathbf{x}| = \frac{n}{2}$ where $|\mathbf{x}|$ is the Hamming weight of $\mathbf{x}$.

(2) In the reading step, we let $\mathbf{c} = c_1c_2...c_n \in \mathcal{R}^n$ be the current levels of the n cells to read. A balancing threshold $\mathbf{v}$ is determined based on $\mathbf{c}$ such that the resulting word, denoted by $\mathbf{y} = y_1y_2...y_n$, is also balanced, namely, $|\mathbf{y}| = \frac{n}{2}$. For each $i \in \{1,2,...,n\}$, $y_i = 1$ if and only if $c_i \geq \mathbf{v}$,

otherwise $y_i = 0$. By applying the decoder of the balanced (error-correcting) code, we get a binary output $\tilde{\mathbf{u}}$, which is the message that we read from the block.

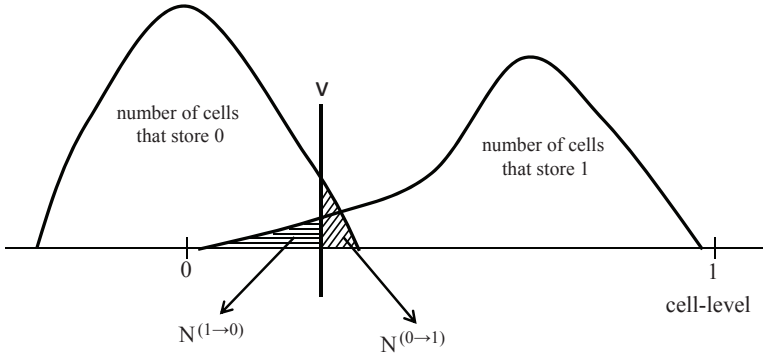


Figure 10.3. Cell-level distributions for 1 and 0, and the reading threshold.

Let us intuitively understanding the function of balanced modulation based on the demonstration of figure 10.3, which depicts the cell-level distributions for those cells that store 0 or 1. Given a reading threshold $\mathbf{v}$, we use $N^{(1\to 0)}$ denote the number of $1 \to 0$ errors and use $N^{(0\to 1)}$ denote the number of $0 \to 1$ errors, as the tails marked in the figure. Then

$$N^{(1\to 0)} = |\{i : x_i = 1, y_i = 0\}|,$$

$$N^{(0\to 1)} = |\{i : x_i = 0, y_i = 1\}|.$$

We are ready to see

$$|\mathbf{y}| = |\mathbf{x}| - N^{(1\to 0)} + N^{(0\to 1)},$$

where $|\mathbf{x}|$ is the Hamming weight of $\mathbf{x}$.

According to the definition, a balancing threshold is the one that makes $\mathbf{y}$ being balanced, hence,

$$N^{(1\to 0)}(\mathbf{v}) = N^{(0\to 1)}(\mathbf{v}),$$

i.e., a balancing threshold results in the same number of $1 \to 0$ errors and $0 \to 1$ errors.

We define $N_e(\mathbf{v})$ as the total number of errors based on a reading threshold $\mathbf{v}$, then

$$N_e(\mathbf{v}) = N^{(1\to 0)}(\mathbf{v}) + N^{(0\to 1)}(\mathbf{v}).$$

If the cell-level distributions for those cells that store 1 and those cells that store 0 are known, then the balancing threshold may not be the best reading threshold that we can have, i.e., $N_e(\mathbf{v})$ may not be minimized based on the balancing threshold. Let v_b denote the balancing threshold, as a comparison, we can have an optimal threshold v_o, which is defined by

$$v_o = \arg\min_{\mathbf{v}} N_e(\mathbf{v}).$$

Unfortunately, it is almost impossible for us to know the cell-level distributions for those cells that store 1 and those cells that store 0 without knowing the original word $\mathbf{x}$. From this sense, the optimal threshold v_o is imaginary. Although we are not able to determine v_o, the following result shows that the balancing threshold v_b has performance comparable to that of v_o. Even in the worst case, the number of errors introduced based on v_b is at most two times that introduced by v_o, implying the suboptimality of the balancing threshold v_b.

Theorem 10.1. *Given any balanced codeword $\mathbf{x} \in \{0,1\}^n$ and cell-level vector $\mathbf{c} \in \mathcal{R}^n$, we have*

$$N_e(v_b) \leq 2N_e(v_o).$$

Proof. Given the balancing threshold v_b, the number of $0 \to 1$ errors equals the number of $1 \to 0$ errors, hence, the total number of errors is

$$N_e(v_b) = 2N^{(1\to 0)}(v_b) = 2N^{(0\to 1)}(v_b).$$

If $v_o \geq v_b$, the number of $1 \to 0$ errors $N^{(1\to 0)}(v_o) \geq N^{(1\to 0)}(v_b)$. Therefore,

$$N_e(v_b) \leq 2N^{(1\to 0)}(v_o) \leq 2N_e(v_o).$$

Similarly, if $v_o < v_b$, by considering only $0 \to 1$ errors, we get the same conclusion. □

Now we compare the balancing threshold v_b with a fixed threshold, denoted by v_f. As shown in figure 10.3, if we set the reading threshold as fixed $v_f = \frac{1}{2}$, then it will introduce much more errors then the balancing threshold. Given a fixed threshold v_f, after a long duration, we can characterize the storage channel as a binary asymmetric channel, as shown in figure 10.4(a), where $p_1 > p_2$. Balanced modulation is actually a process of modifying the channel to make it being symmetric. As a result, balanced modulation results in a binary symmetric channel with crossover probability p such that $p_2 < p < p_1$. When $p_2 \ll p_1$, it has $p - p_2 \ll p_1 - p$. In this case, the bit error rate is reduced from $\frac{p_1+p_2}{2}$ to p, where $p \ll \frac{p_1+p_2}{2}$.

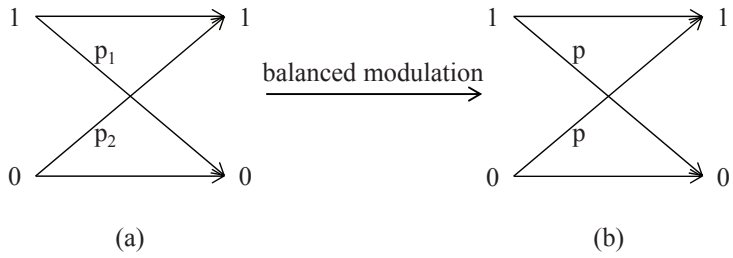


Figure 10.4. Balanced modulation to turn a binary asymmetric channel with crossover probabilities $p_1 > p_2$ into a binary symmetric channel with $p_2 < p < p_1$.

10.4 Bit-Error-Rate Analysis

To better understand different types of reading thresholds as well as their performances, we study them from the expectation (statistical) perspective. Assume that we write n bits (including k ones) into a block at time 0, let $g_t(v)$ denote the probability density function (p.d.f.) of the cell level at time t that stores a bit 0, and let $h_t(v)$ denote the p.d.f. of the cell level at time t that stores 1.

Then at time t, the bit error rate of the block based on a reading threshold $\mathbf{v}$ is given by

$$p_e(\mathbf{v}) = \frac{1}{2}\int_{\mathbf{v}}^{\infty} g_t(u)du + \frac{1}{2}\int_{-\infty}^{\mathbf{v}} h_t(v)dv.$$

According to our definition, a balancing threshold v_b is chosen such that $N^{(1\to 0)}(v_b) = N^{(0\to\infty)}(v_b)$, i.e., the number of $1 \to 0$ errors is equal to the number of $0 \to 1$ errors. As the block length n becomes sufficiently large, we can approximate $N^{(1\to 0)}(v_b)$ as $\frac{n}{2}\int_{-\infty}^{\mathbf{v}} h_t(v)dv$ and approximate $N^{(0\to\infty)}(v_b)$ as $\frac{n}{2}\int_{\mathbf{v}}^{\infty} g_t(u)du$. So when n is large, we have approximately

$$\int_{v_b}^{\infty} g_t(u)du = \int_{-\infty}^{v_b} h_t(v)dv.$$

Differently, an optimal reading threshold v_o is the one that minimizes the total number of errors. When n is large, we have approximately

$$v_o = \arg\min_{\mathbf{v}} p_e(\mathbf{v}).$$

When $g_t(v)$ and $h_t(v)$ are continuous functions, the solutions of v_o are

$$v_o = \pm\infty \text{ or } g_t(v_o) = h_t(v_o).$$

That means v_o is one of the intersections of $g_t(v)$ and $h_t(v)$ or one of the infinity points.

Generally, $g_t(v)$ and $h_t(v)$ are various for different nonvolatile memories and different blocks, and they have different dynamics over time. It is not easy to find a perfect model to characterize $g_t(v)$ and $h_t(v)$, but there are two trends about them in timescale. The change of a cell level can be treated as a superposition of these two trends. First, due to cell-level drift, the difference between the means of $g_t(v)$ and $h_t(v)$ becomes smaller. Second, due to the existence of different types of noise and disturbance, their variances increases over time. To study the performance of balanced modulation, we consider both of the effects separately in some simple scenarios.

Example 10.1. *Let $g_t(v) = \mathcal{N}(0, \sigma)$ and $h_t(v) = \mathcal{N}(1 - t, \sigma)$, as illustrated in figure 10.5. We assume that the fixed threshold is $v_f = \frac{1}{2}$, which satisfies $g_0(v_f) = h_0(v_f)$.*

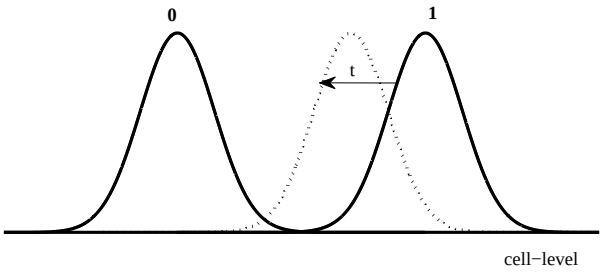


Figure 10.5. An illustration of the first model with $g_t(v) = \mathcal{N}(0, \sigma)$ and $h_t(v) = \mathcal{N}(1 - t, \sigma)$.

In the above example, the cell-level distribution corresponding to bit '1' drifts but its variance does not change. We have

$$v_b = v_o = \frac{1-t}{2}, \quad v_f = \frac{1}{2}.$$

At time t, the bit error rate based on a reading threshold $\mathbf{v}$ is

$$p_e(\mathbf{v}) = \frac{1}{2}\Phi(-\frac{\mathbf{v}}{\sigma}) + \frac{1}{2}\Phi(-\frac{1-t-\mathbf{v}}{\sigma}),$$

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

For different selections of reading thresholds, $p_e(\mathbf{v})$ is plotted in figure 10.6. It shows that the balancing threshold and the optimal threshold have the same performance, which is much better than the performance of a fixed threshold. When cell levels drift, balanced modulation can significantly reduce the bit error rate of a block.

Example 10.2. *Let $g_t(v) = \mathcal{N}(0, \sigma)$ and $h_t(v) = \mathcal{N}(1, \sigma + t)$, as illustrated in figure 10.7. We assume that the fixed threshold is $v_f = \frac{1}{2}$, which satisfies $g_0(v_f) = h_0(v_f)$.*

In this example, the variance of the cell-level distribution corresponding to bit '1' increases as

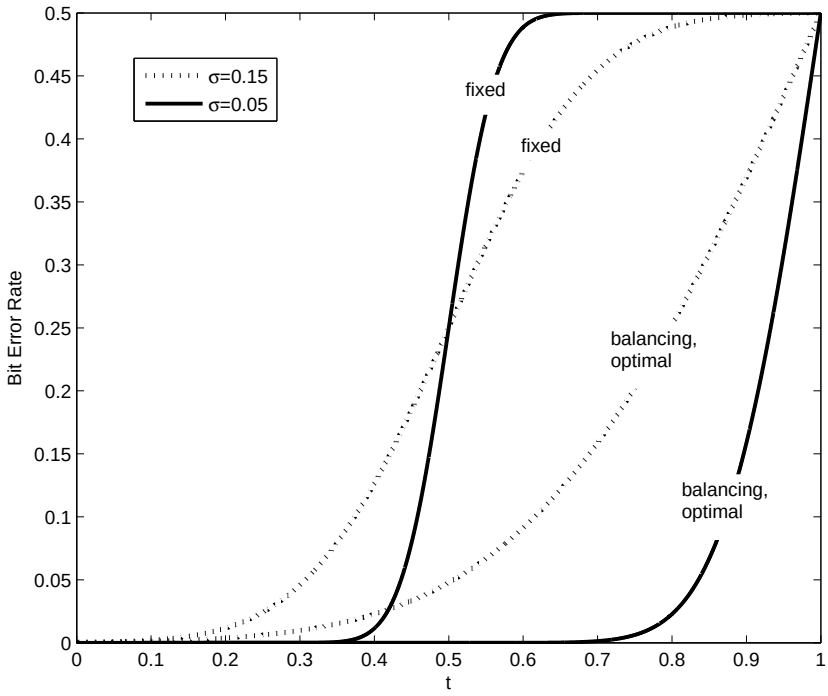


Figure 10.6. Bit error rates as functions of time t, under the first model with $g_t(v) = \mathcal{N}(0, \sigma)$ and $h_t(v) = \mathcal{N}(1 - t, \sigma)$.

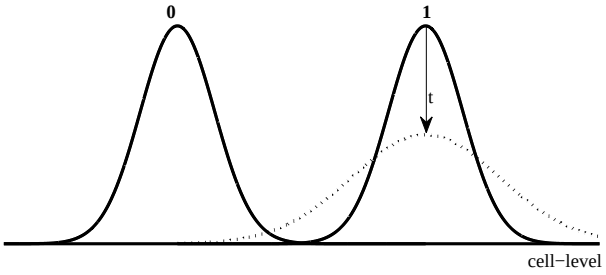


Figure 10.7. An illustration of the second model with $g_t(v) = \mathcal{N}(0, \sigma)$ and $h_t(v) = \mathcal{N}(1, \sigma + t)$.

the time t increases. We have

$$e^{-\frac{v_o{}^2}{2\sigma^2}} = \frac{\sigma}{\sigma + t} e^{-\frac{(1-v_o)^2}{2(\sigma+t)^2}}, \quad v_b = \frac{1}{2 + t/\sigma}, \quad v_f = \frac{1}{2}.$$

At time t, the bit error rate based on a threshold $\mathbf{v}$ is

$$p_e(\mathbf{v}) = \frac{1}{2}\Phi(-\frac{\mathbf{v}}{\sigma}) + \frac{1}{2}\Phi(-\frac{1-\mathbf{v}}{\sigma+t}),$$

which is plotted in figure 10.8 for different thresholds. It shows that balancing thresholds introduce much less errors than fixed thresholds when bit '1' and '0' have different reliability (reflected by their variances), although they introduce slightly more errors than optimal thresholds.

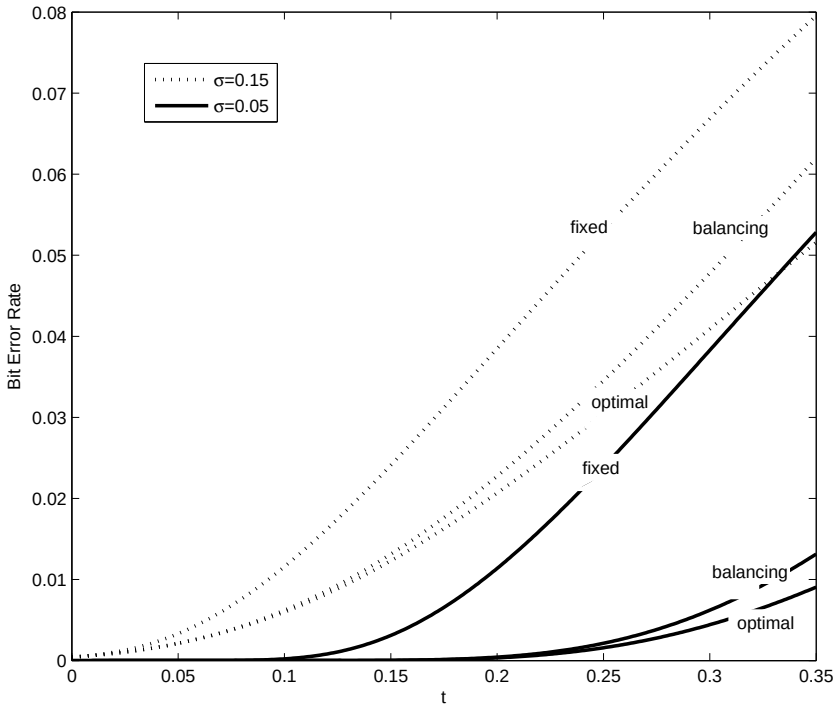


Figure 10.8. Bit error rates as functions of time t, under the second model with $g_t(v) = \mathcal{N}(0, \sigma)$ and $h_t(v) = \mathcal{N}(1, \sigma + t)$.

In practice, the cell-level distributions at a time t are much more complex than the simple Gaussian distributions, and the errors introduced are due to many complex mechanisms. However, the above analysis based two simple models are still useful, because they reflect the trends of the cell level changes, which is helpful for analyzing the time-dependent errors in nonvolatile memories.

10.5 Implementation

Balanced modulation can be easily implemented on the current architecture of nonvolatile memories. The process described in the previous sections can be treated as a hard decision approach, where a reading threshold is selected to separate all the cell levels as zeros and ones. In this section, we discuss a few methods of determining balancing thresholds quickly, as well as their implementations in nonvolatile memories. Furthermore, we discuss soft decision implementation of balanced modulation, namely, we do not read data based on a reading threshold, and the decoder can get access into all the cell levels (cell-level vector $\mathbf{c}$) directly. In this case, we want to know how the prior information that the stored codeword is balanced can help us to increase the success rate of decoding.

10.5.1 Balancing Threshold for Hard Decision

Given a block of n cells, assume their current levels are $\mathbf{c} = c_1 c_2 ... c_n$. Our problem is to determine a threshold v_b such that there are $\frac{n}{2}$ cells or approximately $\frac{n}{2}$ cells will be read as ones. A trivial method is to sort all the n cell levels in the decreasing order such that $c_{i_1} \geq c_{i_2} \geq ... \geq c_{i_n}$. Then $v_b = \frac{c_{i_k} + c_{i_{k+1}}}{2}$ is our desired balancing threshold. The disadvantage of this method is that it needs $O(n \log n)$ computational time, which may slow down the reading speed when n is large. To reduce the reading time, we hope that the balancing threshold can be controlled by hardware.

Half-interval search is a simple approach of determining the balancing threshold. Assume it is known that v_b is $\in [l_1, l_2]$ with $l_1 < l_2$. First, we set the reading threshold as $\frac{l_1+l_2}{2}$, based on which a simple circuit can quickly detect the number of ones in the resulting word, denoted by k. If $k < \frac{n}{2}$, we reset the interval $[l_1, l_2]$ as $[l_1, \frac{l_1+l_2}{2}]$. If $k > \frac{n}{2}$, we reset the interval $[l_1, l_2]$ as $[\frac{l_1+l_2}{2}, l_2]$. Then we repeat this procedure until we get a reading threshold such that $k = \frac{n}{2}$ or $l_2 - l_1 \leq \epsilon$ for a reading precision ϵ.

10.5.2 Relaxed Balancing Threshold

Half-interval search is an iterative approach of determining the balancing threshold such that the resulting word is well balanced. To further reduce the reading time, we can relax the constraint on

the weight of the resulting word, namely, we can let the number of ones in the resulting word be approximately $\frac{n}{2}$, instead of accurately $\frac{n}{2}$.

For instance, we can simply set the balancing threshold as

$$v_b = \frac{\sum_{i=1}^{n} c_i}{n} = \text{mean}(\mathbf{c}).$$

Obviously, such v_b reflects the cell-level drift and it can be easily implemented by a simple circuit.

More precisely, we can treat $\text{mean}(\mathbf{c})$ as the first-order approximation, in this way, we write v_b as

$$v_b = \text{mean}(\mathbf{c}) + a(\frac{1}{2} - \text{mean}(\mathbf{c}))^2,$$

where a is a constant depending on the noise model of memory devices.

10.5.3 Prior Probability for Soft Decision

Reading data based on hard decision is preferred in nonvolatile memories, regarding to its advantages in reading speed and computational complexity compared to soft decision decoding. However, in some occasions, soft decision decoding is still useful for increasing the decoding success rate. We demonstrate that the prior knowledge that the stored codewords are balanced can help us to better estimate the cell-level probability distributions for 0 or 1. Hence, it leads to a better soft decoding performance.

We assume that given a stored bit, either 0 or 1, its cell level is Gaussian distributed. (We may also use some other distribution models according to the physical properties of memory devices, and our goal is to have a better estimation of model parameters). Specifically, we assume that the cell-level probability distribution for 0 is $\mathcal{N}(u_0, \sigma_0)$ and the cell-level probability distribution for 1 is $\mathcal{N}(u_1, \sigma_1)$. Since the codewords are balanced, the probability for a cell being 0 or 1 is equal. So we can describe cell levels by a Gaussian Mixture Model. Our goal is to find the maximum likelihood

$u_0, \sigma_0, u_1, \gamma_1$ based on the cell-level vector $\mathbf{c}$, namely, the parameters that maximize

$$P(\mathbf{c}|u_0, \sigma_0, u_1, \sigma_1).$$

Expectation-Maximization (EM) algorithm is an iterative method that can easily find the maximum likelihood $u_0, \sigma_0, u_1, \gamma_1$. The EM iteration alternates between performing an expectation (E) step and a maximization (M) step. Let $\mathbf{x} = x_1 x_2 ... x_n$ be the codeword stored in the current block, and let $\lambda_t = [u_0(t), \sigma_0(t), u_1(t), \gamma_1(t)]$ be the estimation of the parameters in the tth iteration. In the E-step, it computes the probability for each cell being 0 or 1 based on the current estimation of the parameters, namely, for all $i \in \{1, 2, ..., n\}$, it computes

$$P(x_i = k|c_i, \lambda_t) = \frac{\frac{1}{\sigma_k(t)} e^{-\frac{(c_i - u_k(t))^2}{2\sigma_k(t)^2}}}{\sum_{k=0}^{1} \frac{1}{\sigma_k(t)} e^{-\frac{(c_i - u_k(t))^2}{2\sigma_k(t)^2}}}.$$

In the M-step, it computes parameters maximizing the likelihood with given the probabilities obtained in the E-step. Specifically, for $k \in \{0, 1\}$,

$$u_k(t+1) = \frac{\sum_{i=1}^{n} P(x_i = k|c_i, \lambda_t) c_i}{\sum_{i=1}^{n} P(x_i = k|c_i, \lambda_t)},$$

$$\sigma_k(t+1)^2 = \frac{\sum_{i=1}^{n} P(x_i = k|c_i, \lambda_t)(c_i - u_k(t+1))^2}{\sum_{i=1}^{n} P(x_i = k|c_i, \lambda_t)}.$$

These estimations of parameters are then used to determine the distribution of x_i in the next E-step.

Assume $u_0, \sigma_0, u_1, \sigma_1$ are the maximum-likelihood parameters, based on which we can calculate the log-likelihood for each variable x_i, that is

$$\lambda_i = \frac{\log f(c_i|x_i = 0)}{\log f(c_i|x_i = 1)} = \frac{\log \frac{1}{\sigma_0} - \frac{(c_i - u_0)^2}{2\sigma_0^2}}{\log \frac{1}{\sigma_1} - \frac{(c_i - u_1)^2}{2\sigma_1^2}},$$

where f is the probability density function. Based on the log-likelihood of each variable x_i, some soft decoding algorithms can be applied to read data, including message-passing algorithms [83],

linear programming [37], etc. It will be further discussed in the next section for decoding balanced LDPC code.

10.6 Balanced LDPC Code

Balanced modulation can significantly reduce the bit error rate of a block in nonvolatile memories, but error correction is still necessary. So we study the construction of balanced error-correcting codes. In the programming step, we encode the information based on a balanced error-correcting code and write it into a block. In the reading step, the reading threshold is adjusted such that it yields a balanced word, but probably erroneous. Then we pass this word to the decoder to further retrieve the original information.

10.6.1 Construction

In this section, we introduce a simple construction of balanced error-correcting codes, which is based on LDPC codes, called *balanced LDPC code*. LDPC codes, first introduced by Gallager [42] in 1962 and rediscovered in 1990s, achieve near Shannon-bound performances and allow reasonable decoding complexities. Our construction of balanced LDPC code is obtained by inverting the first i bits of each codeword in a LDPC code such that the codeword is balanced, where i is different for different codewords. It is based on Knuth's observation [69], that is, given an arbitrary binary word of length k with k even, one can always find an integer i with $0 \leq i < k$ such that by inverting the first i bits the word becomes balanced. Different from the current construction in [131], where i is stored and protected by a lower-rate balanced error-correcting codes (the misdecoding of i may lead to catastrophic error propagation in the information word), we do not store i in our construction. The main idea is that certain redundancy exists in the codewords of LDPC codes that enables us to locate i or at last find a small set that includes i with a very high probability, even some errors exist in the codewords. It is wasteful to store the value of i with a lower-rate balanced error-correcting code. As a result, our construction is more efficient than the recent construction proposed in [131].

Let $\mathbf{u}$ be the message to encode and its length is k, according to the description above, the

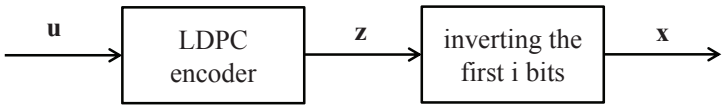


Figure 10.9. Encoding of balanced LDPC codes.

encoding procedure consists of two steps, as shown in figure 10.9:

1. Apply an (n, k) LDPC code $\mathcal{L}$ to encode the message $\mathbf{u}$ into a codeword of length n, denoted by $\mathbf{z} = G\mathbf{u}$, where G is the generator matrix of $\mathcal{L}$.

2. Find the minimal integer i in $\{0, 1, ..., n-1\}$ such that inverting the first i bits of $\mathbf{z}$ results in a balanced word

$$\mathbf{x} = \mathbf{z} + \mathbf{1}^i\mathbf{0}^{n-i},$$

where $\mathbf{1}^i\mathbf{0}^{n-i}$ denotes a run of i bits $\mathbf{1}$ and $n-i$ bits $\mathbf{0}$. Then we denote $\mathbf{x}$ as $\phi(\mathbf{z})$. This word $\mathbf{x}$ is a codeword of the resulting balanced LDPC code, denoted by $\mathcal{C}$.

We see that a balanced LDPC code is constructed by simply balancing the codewords of a LDPC code, which is called the original LDPC code. Based on the procedure above we can encode any message $\mathbf{u}$ of length k into a balanced codeword $\mathbf{x}$ of length n. The encoding procedure is very simple, but how to decode a received word? Now, we focus on the decoding of this balanced LDPC code. Let $\mathbf{y}$ be an erroneous word received by the decoder, then the output of the maximum likelihood decoder is

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathcal{C}} D(\mathbf{y}, \mathbf{x}),$$

where $D(\mathbf{y}, \mathbf{x})$ is the distance between $\mathbf{y}$ and $\mathbf{x}$ depending on the channel, for instance, Hamming distance for binary symmetric channels.

The balanced code $\mathcal{C}$ is not a linear code, so the constraint $\mathbf{x} \in \mathcal{C}$ is not easy to deal with. A simpler way is to think about the codeword $\mathbf{z} \in \mathcal{L}$ that corresponds to $\mathbf{x}$. By inverting the first j

bits of $\mathbf{y}$ with $0 \leq j < n$, we can get a set of words $S_{\mathbf{y}}$ of size n, namely,

$$S_{\mathbf{y}} = \{\mathbf{y^{(0)}}, \mathbf{y^{(1)}}, ..., \mathbf{y^{(n-1)}}\},$$

in which

$$\mathbf{y^{(j)}} = \mathbf{y} + \mathbf{1}^j\mathbf{0}^{n-j},$$

for all $j \in \{0, 1, 2, ..., n\}$. Then there exists an $i \in \{0, 1, 2, ..., n-1\}$ such that

$$\mathbf{y^{(i)}} - \mathbf{z} = \mathbf{y} - \mathbf{x}.$$

The output of the maximum likelihood decoder is

$$(\hat{\mathbf{z}}, \hat{\mathbf{i}}) = \arg \min_{\mathbf{z'} \in \mathcal{L}, i' \in \{0,1,2...,n\}} D(\mathbf{y^{(i')}}, \mathbf{z'}),$$

subject to i' is the minimum integer that makes $\mathbf{z'} + \mathbf{1}^{i'}\mathbf{0}^{n-i'}$ being balanced.

If we ignore the constraint that i has to be the minimum integer, then the output of the decoder is the codeword in $\mathcal{L}$ that has the minimum distance to $S_{\mathbf{y}}$. Figure 10.10 provides a simple demonstration, where the solid circles are for the codewords of the LPDC code $\mathcal{L}$, the triangles are for the words in $S_{\mathbf{y}}$ that are connected by lines. Our goal is to find the solid circle that is the closest one to the set of triangles. It is different from traditional decoding of linear codes whose goal is to find the closest codeword to a single point.

10.6.2 An Extreme Case

LDPC codes achieve near Shannon bound performances. A natural question is whether balanced LDPC codes hold this property. Certain difficulties exist in proving it by following the method in [43] (section 2 and section 3), since balanced LDPC codes are not linear codes and the distance distributions of balanced LDPC codes are not easy to characterize. Fortunately, this statement looks correct because if the first i bits of a codeword have been inverted (we assume that the interger i

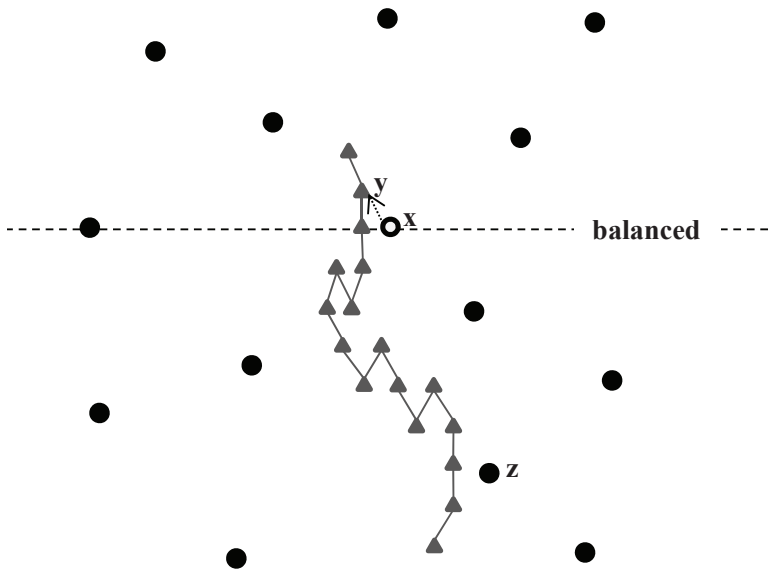


Figure 10.10. Demonstration for the decoding of balanced LDPC codes.

is unknown), then the codeword can be recovered with only little cost, i.e., a very small number of additional redundant bits.

Let us consider the ensemble of an (n, a, b) parity-check matrix given by Gallager [43], which has a ones in each column, b ones in each row, and zeros elsewhere. According to this construction, the matrix is divided into a submatrices, each containing a single 1 in each column. All the submatrices are random column permutations of a matrix that has a single one in each column and b ones in each row. As a result, we have (n, a, b) LDPC codes.

Theorem 10.2. *Given a codeword* $\mathbf{z}$ *of an* (n, a, b) *LDPC code, we get*

$$\mathbf{x} = \mathbf{z} + \mathbf{1}^i\mathbf{0}^{n-i}$$

by inverting the first i *bits of* $\mathbf{z}$ *with* $0 \leq i < n$. *Let* $P_e(\mathbf{x})$ *be the error probability that* $\mathbf{z}$ *cannot be correctly recovered from* $\mathbf{x}$ *if* i *is unknown. As* $n \to \infty$,

$$P_e(\mathbf{x}) \to 0,$$

for any integers a and b.

Proof. Let H be the parity-check matrix of the LDPC code, and let

$$\mathbf{y^{(j)}} = \mathbf{x} + \mathbf{1}^j\mathbf{0}^{n-j},$$

for all $j \in \{0, 1, 2, ..., n-1\}$.

We can recover $\mathbf{z}$ from $\mathbf{x}$ if and only if

$$H\mathbf{y^{(j)}} \neq 0,$$

for all $j \neq i$ and $0 \leq j \leq n-1$.

Hence,

$$P_e(\mathbf{x}) = P(\exists j \neq i, s.t., H\mathbf{y^{(j)}} = 0)$$

$$\leq \sum_{j \neq i} P(H\mathbf{y^{(j)}} = 0).$$

Let us first consider the case of $j > i$. We have $H\mathbf{y^{(j)}} = 0$ if and only if

$$H(\mathbf{y^{(j)}} + \mathbf{z}) = 0,$$

where

$$\mathbf{y^{(j)}} + \mathbf{z} = \mathbf{0}^i\mathbf{1}^{j-i}\mathbf{0}^{n-j}.$$

So $H\mathbf{y^{(j)}} = 0$ is equivalent to

$$H(\mathbf{0}^i\mathbf{1}^{j-i}\mathbf{0}^{n-j}) = 0.$$

As we described, H is constructed by a submatrices, namely, we can write H as

$$H = \begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_a \end{pmatrix}.$$

Let H_s be one of the a submatrices of H, then H contains a single one in each columns and b ones in each row. And it satisfies

$$H_s(\mathbf{0}^i\mathbf{1}^{j-i}\mathbf{0}^{n-j}) = 0,$$

i.e., in each row of H_s, there are even number of ones from the $i+1$th column to the jth column.

According to the construction of (n, a, b) LDPC codes,

$$P(H_s(\mathbf{0}^i\mathbf{1}^{j-i}\mathbf{0}^{n-j}) = 0) = P(H_s(\mathbf{1}^{j-i}\mathbf{0}^{n-j+i}) = 0).$$

So we can use $P(n, j-i)$ to denote $P(H_s(\mathbf{0}^i\mathbf{1}^{j-i}\mathbf{0}^{n-j}) = 0)$.

First, we consider the case that b is even. In this case,

$$P(n, j-i) = P(n, n-j+i).$$

Hence, without loss of generality, we can assume that $j - i = d \leq \frac{n}{2}$.

It is easy to see that $P(n, j-i) > 0$ only if d is even. Assume that the one in the first column of H_s is in the tth row, and let u be the number of ones in the tth row from the first $j-i$ columns. Then we can get

$$P(n, d) = \sum_{u=2,4,\ldots} \binom{b}{u-1} (\frac{d-1}{n-1})^{u-1} (\frac{n-d}{n-1})^{b-u} P(n-b, d-u),$$

where $P(n, d) = 1$ if $n = d$ or $d = 0$.

If $d < \log n$, then $P(n,d) = O(\frac{\log n}{n})$.

If $\log n \leq d \leq \frac{n}{2}$, then

$$\sum_{u=2,4,\ldots} \binom{b}{u-1} (\frac{d-1}{n-1})^{u-1} (\frac{n-d}{n-1})^{b-u} \leq \frac{b-1}{b}.$$

Iteratively, we can prove that

$$P(n,d) = O((\frac{b-1}{b})^{\frac{\log n}{2b}}).$$

Similar as above, when $j < i$, we can get

$$P(H\mathbf{y}^{(\mathbf{j})} = 0) \leq P(n, i-j).$$

Finally, we have

$$P_e(\mathbf{x}) \leq \sum_{s=1}^{n-1-i} P(n,s) + \sum_{s=1}^{i} P(n,s) = O(\frac{\log n}{n}).$$

So if b is even, as $n \to \infty$, $P_e(\mathbf{x}) \to 0$.

If b is odd, in each row, there exists at least one 1 in the last $n-j+i$ elements. As a result, $n-j+i \geq \frac{n}{b}$. Using a same idea as above, we can also prove that as $n \to \infty$, $P_e(\mathbf{x}) \to 0$.

So the statement in the theorem is true for any rate $R = \frac{b-a}{b} < 1$. This completes the proof. □

The above theorem considers an extreme case that if the codeword of a balanced LDPC code does not have errors, then we can recover the original message with little cost of redundancy. It implies that balanced LDPC codes may achieve almost the same rates as the original unbalanced LDPC codes. In the following subsections, we discuss some decoding techniques for binary erasure channels and binary symmetric channels. Simulation results on these channels support the above statement.

10.6.3 Decoding for Erasure Channels

In this subsection, we consider binary erasure channels (BEC), where a bit (0 or 1) is either successfully received or it is deleted, denoted by "?". Let $\mathbf{y} \in \{0, 1, ?\}^n$ be a word received by a decoder after transmitting a codeword $\mathbf{x} \in \mathcal{C}$ over a BEC. Then the key of decoding $\mathbf{y}$ is to determine the value of the integer i such that $\mathbf{x}$ can be obtained by inverting the first i bits of a codeword in $\mathcal{L}$.

A simple idea is to search all the possible values of i, i.e., we decode all the possible words $\mathbf{y^{(0)}}, \mathbf{y^{(1)}}, ..., \mathbf{y^{(n-1)}}$ separately and select the best resulting codeword that satisfies all the constraints as the final output. This idea is straightforward, but the computational complexity of the decoding increases by a factor of n, which is not acceptable for most practical applications.

Our observation is that we might be able to determine the value of i or at least find a feasible set that includes i, based on the unerased bits in $\mathbf{y}$. For example, given $\mathbf{x} \in \mathcal{L}$, assume that one parity-check constraint is

$$x_{i_1} + x_{i_2} + ... + x_{i_4} = 0.$$

If all $y_{i_1}, y_{i_2}, ..., y_{i_4}$ are observed (not erased), then we can have the following statement about i:

(1) If $y_{i_1} + y_{i_2} + ... + y_{i_4} = 0$, then

$$i \in [0, i_1) \bigcup [i_2, i_3) \bigcup [i_4, n].$$

(2) If $y_{i_1} + y_{i_2} + ... + y_{i_4} = 1$, then

$$i \in [i_1, i_2) \bigcup [i_3, i_4).$$

By combining this observation with the message-passing algorithm, we get a decoding algorithm for balanced LDPC codes under BEC. Similar as the original LDPC code, we present a balanced LDPC code as a sparse bipartite graph with n variable nodes and r check nodes, as shown in figure 10.11. Additionally, we add an inversion node for representing the value or the feasible set of i. Let us describe a modified message-passing algorithm on this graph. In each round of the algorithm,

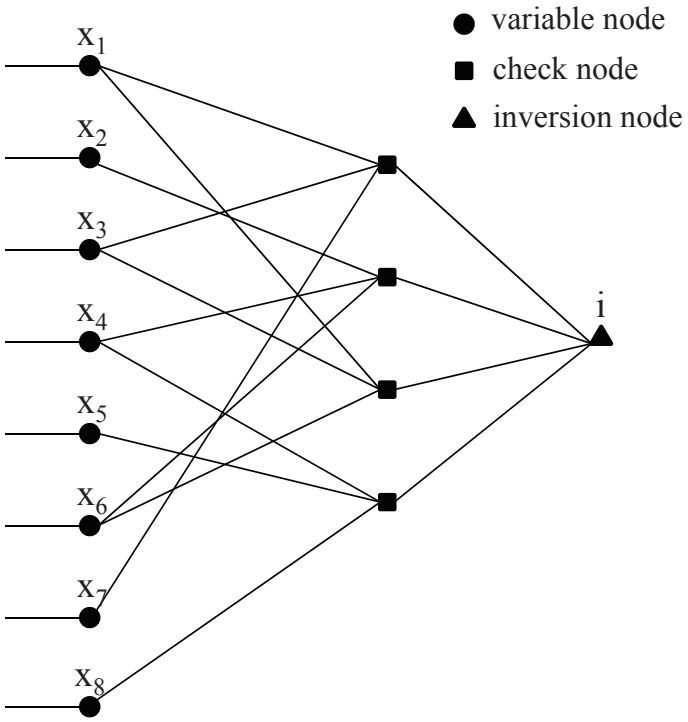


Figure 10.11. Graph for balanced LDPC codes.

messages are passed from variable nodes and inversion nodes to check nodes, and then from check nodes back to variable nodes and inversion nodes.

We use $\mathcal{I}$ denote the feasible set consisting of all possible values for the integer i, called inversion set. At the first round, we initialize the jth variable node $y_j \in \{0, 1, ?\}$ and initialize the inversion set as $\mathcal{I} = [0, n]$. Then we pass message and update the graph iteratively. In each round, we do the following operations.

(1) For each variable node **v**, if its value x_v is in $\{0, 1\}$, it sends x_v to all its check neighbors. If $x_v =?$ and any incoming message u is 0 or 1, it updates x_v as u and sends u to all its check neighbors. If $x_v =?$ and all the incoming messages are ?, it sends ? to all its check neighbors.

(2) For each check node **c**, assume the messages from its variable neighbors are $x_{i_1}, x_{i_2}, ..., x_{i_b}$, where $i_1, i_2, ..., i_b$ are the indices of these variable nodes s.t. $i_1 < i_2 < ... < i_b$. Then we define

$$S_c^0 = [0, i_1) \bigcup [i_2, i_3) \bigcup ...,$$

$$S_c^1 = [i_1, i_2) \bigcup [i_3, i_4) \bigcup ....$$

If all the incoming messages are in $\{0, 1\}$, then we update $\mathcal{I}$ in the following way: If $x_{i_1} + x_{i_2} + ... + x_{i_b} = 0$, we update $\mathcal{I}$ as $\mathcal{I} \bigcap S_c^0$; otherwise, we update $\mathcal{I}$ as $\mathcal{I} \bigcap S_c^1$. In this case, this check node $\mathbf{c}$ is no longer useful, so we can remove this check node from the graph.

(3) For each check node $\mathbf{c}$, if there are exactly one incoming message from its variable neighbor which is $x_j =?$ and all other incoming messages are in $\{0, 1\}$, we check whether $\mathcal{I} \subseteq S_c^0$ or $\mathcal{I} \subseteq S_c^1$. If $\mathcal{I} \subseteq S_c^0$, then the check node sends the XOR of the other incoming messages except ? to x_j. If $\mathcal{I} \subseteq S_c^1$, then the check node sends the XOR of the other incoming messages except ? plus one to x_j. In this case, the check node $\mathbf{c}$ is also no longer useful, so we can remove this check node from the graph.

The procedure above continues until all erasures are filled in, or no erasures are filled in the current iteration. Different from the message-passing decoding algorithm for LDPC codes, where in each iteration both variable nodes and check nodes are processed only once, here, we process variable nodes once but check nodes twice in each iteration. If all erasures are filled in, $\mathbf{x}$ is the binary vector labeled on the variable nodes. In this case, if $|\mathcal{I}| = 1$, then i is the only element in $\mathcal{I}$, and we can get $\mathbf{z} \in \mathcal{L}$ by calculating

$$\mathbf{z} = \mathbf{x} + \mathbf{1}^i\mathbf{0}^{n-i}.$$

If there are still some unknown erasures, we enumerate all the possible values in $\mathcal{I}$ for the integer i. Usually, $|\mathcal{I}|$ is small. For a specific i, it leads to a feasible solution $\mathbf{z}$ if

(1) Given $\mathcal{I} = \{i\}$, with the message-passing procedure above, all the erasures can be filled in.

(2) $\mathbf{x}$ is balanced, namely, the numbers of ones and zeros are equal for the variable nodes.

(3) Let $\mathbf{z} = \mathbf{x} + \mathbf{1}^i\mathbf{0}^{n-i}$. Then i is the minimal integer in $\{0, 1, 2, ..., n\}$ subject to $\mathbf{z} + \mathbf{1}^i\mathbf{0}^{n-i}$ is balanced.

We say that a word $\mathbf{y}$ with erasures is uniquely decodable if and only if there exists $i \in \mathcal{I}$ that leads to a feasible solution, and for all such integers i they result in the unique solution $\mathbf{z} \in \mathcal{L}$. The following simple example is provided for the purpose of demonstrating the decoding process.

Example 10.3. *Based on figure 10.11, we have a codeword* $\mathbf{x} = 01111000$*, which is transmitted over an erasure channel. We assume that the received word is* $\mathbf{y} = 011110??$*.*

In the first round of the decoding, we have

$$\mathbf{x^{(1)}} = 011110??, \mathcal{I} = [0, 8].$$

Considering the $2nd$ *check node, we can update* $\mathcal{I}$ *as*

$$\mathcal{I} = \{0, 1, 4, 5\}.$$

Considering the $3nd$ *check node, we can continue updating* $\mathcal{I}$ *as*

$$\mathcal{I} = \mathcal{I} \bigcap \{1, 2, 6, 7, 8\} = \{1\}.$$

Based on (3), we can fill $0, 0$ *for the* $7th$ *and* $8th$ *variable nodes. Finally, we get* $\mathbf{z} = 11111000$ *and* $i = 1$*.*

Regarding to the decoding algorithm described above, there are two important issues that need to consider, including the decoding complexity of the algorithm and its performance. First, the decoding complexity of the algorithm strongly depends on the size of $\mathcal{I}$ when it finishes iterations. Figure 10.12 simulates the average size of the inversion set $\mathcal{I}$ for decoding three balanced LDPC codes. It shows that when the crossover probability is lower than a threshold, the size of $\mathcal{I}$ is smaller than a constant with a very high probability. In this case, the decoding complexity of the balanced LDPC code is very close to the decoding complexity of the original unbalanced LDPC code.

Another issue is about the performance of the decoding algorithm for balanced LDPC codes. In particular, we want to figure out the cost of additional redundancy in correcting the inversion of the first i bits when i is unknown. In figure 10.13, it presents the word error rate of balanced LDPC codes and the corresponding original unbalanced LDPC codes for different block lengths. It is interesting to see that as the block length increases, the balanced LDPC codes and the original unbalanced

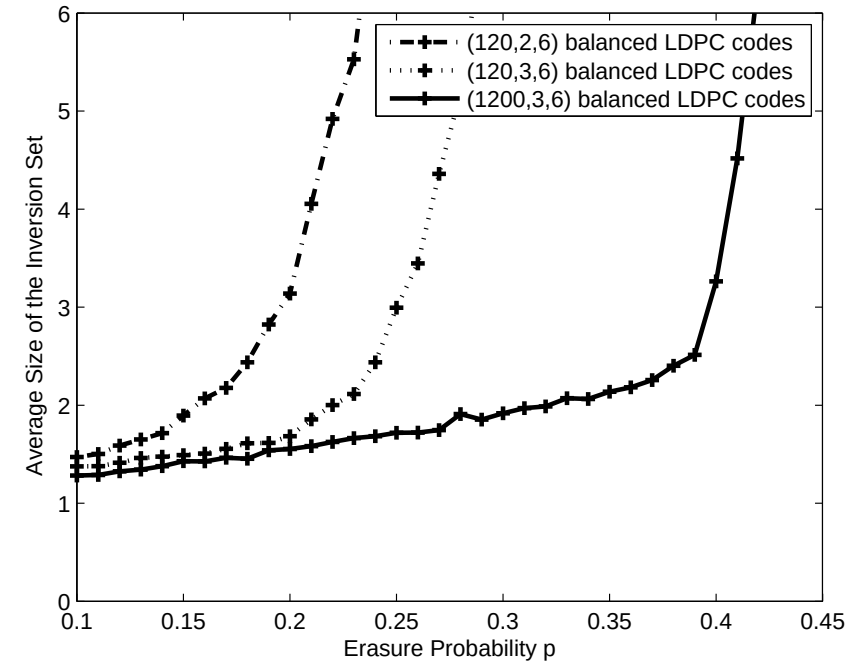


Figure 10.12. The average size of the inversion set $\mathcal{I}$ after iterations in the message-passing algorithm for decoding balanced LDPC codes.

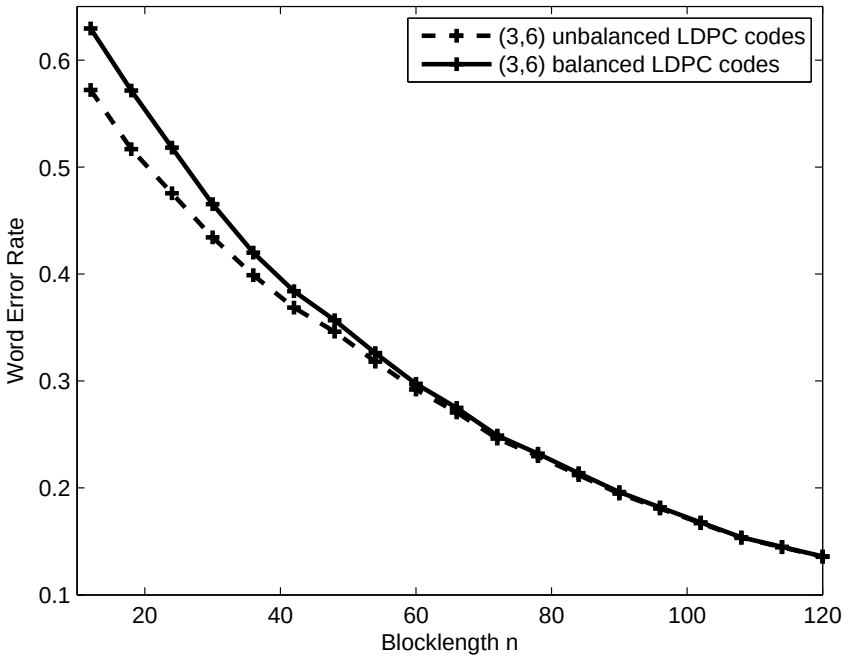


Figure 10.13. Word error rate of balanced LDPC codes and unbalanced LDPC codes when the erasure probability $p = 0.35$.

LDPC codes have almost the same performance, that is, the cost of correcting the inversion of the first i bits is ignorable.

10.6.4 Decoding for Symmetric Channels

In this subsection, we study and analyze the decoding of balanced LDPC codes for symmetric channels, including binary symmetric channels (BSC) and AWGN (Additive White Gaussian Noise) channels. Different from binary erasure channels (BEC), here we are not able to determine a small set that definitely includes the integer i. Instead, we want to figure out the most possible values for i. Before presenting our decoding algorithm, we first introduce belief propagation algorithm for decoding LDPC codes.

Belief propagation [83], where messages are passed iteratively across a factor graph, has been widely studied and recommended for the decoding of LDPC codes. In each iteration, each variable node passes messages (probabilities) to all the adjacent check nodes and then each check node passes messages (beliefs) to all the adjacent variable nodes. Specifically, let $\mathbf{m}_{vc}^{(\ell)}$ be the message passed from a variable node $\mathbf{v}$ to a check node $\mathbf{c}$ at the ℓth round of the algorithm, and let $\mathbf{m}_{cv}^{(\ell)}$ be the message from a check node $\mathbf{c}$ to a variable node $\mathbf{v}$. At the first round, $\mathbf{m}_{vc}^{(0)}$ is the log-likelihood of the node $\mathbf{v}$ conditioned on its observed value, i.e., $\log \frac{P(y|x=0)}{P(y|x=1)}$ for variable x and its observation y. This value is denoted by $\mathbf{m}_v$. Then the iterative update procedures can be described by the following equations

$$\mathbf{m}_{vc}^{(\ell)} = \begin{cases} \mathbf{m}_v & \ell = 0, \\ \mathbf{m}_v + \sum_{c' \in N(v)/c} \mathbf{m}_{c'v}^{(\ell-1)} & \ell \geq 1, \end{cases}$$

$$\mathbf{m}_{cv}^{(\ell)} = 2 \tanh^{-1}\Big(\prod_{v' \in N(c)/v} \tanh\big(\frac{\mathbf{m}_{v'c}^{(\ell)}}{2}\big)\Big),$$

where $N(v)$ is the set of check nodes that connect to variable node $\mathbf{v}$ and $N(c)$ is the set of variable nodes that connect to check node $\mathbf{c}$. In practice, the belief-propagation algorithm stops after a certain number of iterations or until the passed likelihoods are close to certainty. Typically, for a BSC with crossover probability p, the log-likelihood $\mathbf{m}_v$ for each variable node $\mathbf{v}$ is a constant

depending on p. Let x be the variable on $\mathbf{v}$ and let y be its observation, then

$$\mathbf{m}_v = \begin{cases} \log \frac{1-p}{p} & \text{if } y = 0, \\ -\log \frac{1-p}{p} & \text{if } y = 1. \end{cases}$$

Let us consider the decoding of balanced LDPC codes. Assume $\mathbf{x} \in \mathcal{C}$ is a codeword of a balanced LDPC code, obtained by inverting the first i bits of a codeword $\mathbf{z}$ in a LDPC code $\mathcal{L}$. The erroneous word received by the decoder is $\mathbf{y} \in \mathcal{Y}^n$ for an alphabet $\mathcal{Y}$. For example, $\mathcal{Y} = \{0, 1\}$ for BSC channels, and $\mathcal{Y} = \mathbb{R}$ for AWGN channels. Here, we consider a symmetric channel, i.e., a channel for which there exists a permutation π of the output alphabet $\mathcal{Y}$ such that (1) $\pi^{-1} = \pi$, and (2) $P(y|1) = P(\pi(y)|0)$ for all $y \in \mathcal{Y}$, where $P(y|x)$ is the probability of observing y when the input bit is x.

The biggest challenge of decoding a received word $\mathbf{y} \in \mathcal{Y}^n$ is lacking of the location information about where the inversion happens, i.e., the integer i. We let

$$\mathbf{y^{(i)}} = \pi(y_1)\pi(y_2)...\pi(y_i)y_{i+1}...y_n,$$

for all $i \in \{0, 1, 2, ..., n-1\}$. A simple idea is to search all the possibilities for the integer i from 0 to $n-1$, i.e, decoding all the words

$$\mathbf{y^{(0)}}, \mathbf{y^{(1)}}, ..., \mathbf{y^{(n-1)}}$$

separately. Assume their decoding outputs based on belief propagation are

$$\hat{\mathbf{z}}^{\mathbf{(0)}}, \hat{\mathbf{z}}^{\mathbf{(1)}}, ...\hat{\mathbf{z}}^{\mathbf{(n)}},$$

then the final output of the decoder is $\hat{\mathbf{z}} = \hat{\mathbf{z}}^{\mathbf{(j)}}$ such that $P(\mathbf{y^{(j)}}|\hat{\mathbf{z}}^{\mathbf{(j)}})$ is maximized. The drawback of this method is its high computational complexity, which is about n times the complexity of decoding the original unbalanced LDPC code. To reduce computational complexity, we want to estimate the

value of i in a simpler and faster way, even sacrificing a little bit of performance on bit error rate.

The idea is that when we are using belief propagation to decode a group of words $\mathbf{y^{(0)}}, \mathbf{y^{(1)}}, ..., \mathbf{y^{(n-1)}}$, some information can be used to roughly compare their goodness, namely, their distances to the nearest codewords. To find such information, given each word $\mathbf{y^{(i)}}$ (here, we denote it as $\mathbf{y}$ for simplicity), we run belief propagation for ℓ rounds (iterations), where ℓ is very small, e.g., $\ell = 2$. There are several ways of estimating the goodness of $\mathbf{y}$, and we introduce one of them as follows.

Given a word $\mathbf{y}$, we define

$$\lambda(\mathbf{y}, \ell) = \sum_{c \in C} \prod_{v \in N(c)} \tanh(\mathbf{m}_{vc}^{(\ell)}/2),$$

where C is the set of all the variable nodes, $N(c)$ is the set of neighbors of a check node $\mathbf{c}$, and $\mathbf{m}_{vc}^{(\ell)}$ is the message passed from a variable node $\mathbf{v}$ to a check node $\mathbf{c}$ at the ℓth round of the belief-propagation algorithm. Roughly, $\lambda(\mathbf{y}, \ell)$ is a measurement of the number of correct parity checks for the current assignment in belief propagation (after $\ell - 1$ iterations). For instance,

$$\lambda(\mathbf{y}, \ell = 1) = \alpha(r - 2|H\mathbf{y}|),$$

for a binary symmetric channel. In this expression, α is a constant, $r = n - k$ is the number of redundancies, and $|H\mathbf{y}|$ is the number of ones in $H\mathbf{y}$, i.e., the number of unsatisfied parity checks.

Generally, the bigger $\lambda(\mathbf{y^{(j)}}, \ell)$ is, the more likely $j = i$ is. So we can get the most likely i by calculating

$$\hat{i} = \arg\max_{j=0}^{n-1} \lambda(\mathbf{y^{(j)}}, \ell).$$

Then we decode $\mathbf{y^{(\hat{i})}}$ as the final output. However, the procedure requires to calculate $\lambda(\mathbf{y^{(j)}}, \ell)$ with $0 \leq j \leq n - 1$. The following theorem shows that the task of computing all $\lambda(\mathbf{y^{(j)}}, \ell)$ with $0 \leq j \leq n - 1$ can be finished in linear time if ℓ is a small constant.

Theorem 10.3. *The task of computing all $\lambda(\mathbf{y^{(j)}}, \ell)$ with $0 \leq j \leq n - 1$ can be finished in linear time if ℓ is a small constant.*

Proof. First, we calculate $\lambda(\mathbf{y^{(0)}}, \ell)$. Based on the belief-propagation algorithm described above, it can be finished in $O(n)$ time. In this step, we save all the messages including $\mathbf{m}_v$, $\mathbf{m}_{cv}^{(l)}$, $\mathbf{m}_{vc}^{(l)}$ for all $c \in C, v \in V$ and $1 \leq l \leq \ell$.

When we calculate $\lambda(\mathbf{y^{(1)}}, \ell)$, the only change on the inputs is $\mathbf{m}_{v_1}$, where v_1 is the first variable node (the sign of $\mathbf{m}_{v_1}$ is flipped). As a result, we do not have to calculate all $\mathbf{m}_v$, $\mathbf{m}_{cv}^{(l)}$, $\mathbf{m}_{vc}^{(l)}$ for all $c \in C, v \in V$ and $1 \leq l \leq \ell$. Instead, we only need to update those messages that are related with $\mathbf{m}_{v_1}$. It needs to be noted that the number of messages related to $\mathbf{m}_{v_1}$ has an exponential dependence on ℓ, so the value of ℓ should be small. In this case, based on the calculation of $\lambda(\mathbf{y}^{(0)}, \ell)$, $\lambda(\mathbf{y^{(1)}}, \ell)$ can be calculated in a constant time. Similarly, each of $\lambda(\mathbf{y^{(j)}}, \ell)$ with $2 \leq j \leq n-1$ can be obtained iteratively in a constant time.

Based on the process above, we can compute all $\lambda(\mathbf{y^{(j)}}, \ell)$ with $0 \leq j \leq n-1$ in $O(n)$ time. □

To increase the success rate of decoding, we can also create a set of most likely values for i, denoted by $\mathcal{I}_c$. $\mathcal{I}_c$ consists of at most c local maximums with the highest values of $\lambda(\mathbf{y^{(i)}}, \ell)$. Here, we say that $j \in \{0, 1, 2, 3, ..., n-1\}$ is a local maximum if and only if

$$\lambda(\mathbf{y^{(j)}}, \ell) > \lambda(\mathbf{y^{(j-1)}}, \ell), \lambda(\mathbf{y^{(j)}}, \ell) \geq \lambda(\mathbf{y^{(j+1)}}, \ell).$$

Note that $\mathcal{I}_1 = \{\hat{i}\}$, where $\hat{i}$ is the global maximum as defined above. If $c > 1$, for all $j \in \mathcal{I}_c$, we decode $\mathbf{y^{(j)}}$ separately and choose the output with the maximum likelihood as the final output of the decoder. It is easy to see that the the above modified belief-propagation algorithm for balanced LDPC codes has asymptotically the same decoding complexity as the belief-propagation algorithm for LDPC codes, that is, $O(n \log n)$.

In figure 10.14, it shows the performance of the above algorithm for decoding balanced LDPC codes under BSC and the performance of belief propagation algorithm for the original LDPC codes. From which, we see that when $\ell = 2$ and $c = 4$, the performance gap between balanced $(280, 4, 7)$ LDPC code and unbalanced $(280, 4, 7)$ LDPC code is very small. This comparison implies that the cost of correcting the inversion of the first i bits (when i is unknown) is small for LDPC codes.

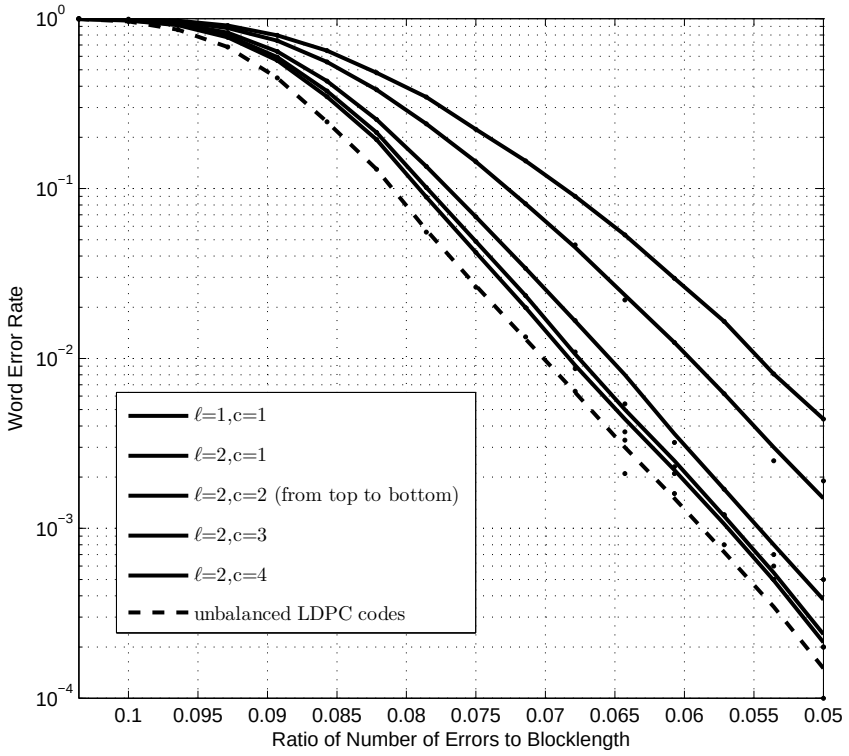


Figure 10.14. World error rate of $(280, 4, 7)$ LDPC codes with maximal 50 iterations.

Let us go back the scheme of balanced modulation. The following examples give the log-likelihood of each variable node when the reading process is based on hard decision and soft decision, respectively. Based on them, we can apply the modified propagation algorithm in balanced modulation.

Example 10.4. *If the reading process is based on hard decision, then it results in a binary symmetric channel with crossover probability p. In this case, let y be the observation on a variable node $\mathbf{v}$, the log-likelihood for $\mathbf{v}$ is*

$$\mathbf{m}_v = \begin{cases} \log \frac{1-p}{p} & \text{if } y = 0, \\ -\log \frac{1-p}{p} & \text{if } y = 1. \end{cases}$$

Example 10.5. *If the reading process is based on soft decision, then we can approximate cell-level distributions by Gaussian distributions, which are characterized by 4 parameters $u_0, \sigma_0, u_1, \sigma_1$. These parameters can be obtained based on the cell-level vector $\mathbf{y} = \mathbf{c}$, following the steps in subsection 10.5.3. In this case, if the input of the decoder is $\mathbf{y}$, then the log-likelihood of the ith variable node*

$\mathbf{v}$ *is*

$$\mathbf{m}_v = \lambda_i = \frac{\log \frac{1}{\sigma_0} - \frac{(c_i - u_0)^2}{2\sigma_0^2}}{\log \frac{1}{\sigma_1} - \frac{(c_i - u_1)^2}{2\sigma_1^2}}$$

where c_i *is the current level of the* i*th cell. If the input of the decoder is* $\mathbf{y^{(i)}}$ *(we don't have to care about its exact value), then the log-likelihood of the* i*th variable node* $\mathbf{v}$ *is*

$$\mathbf{m}_v = \begin{cases} \lambda_i & \text{if } i > j, \\ -\lambda_i & \text{if } i \leq j, \end{cases},$$

for all $0 \leq i < n$.

10.7 Partial-Balanced Modulation

Constructing balanced error-correcting codes is more difficult than constructing normal error-correcting codes. A question is: is it possible to design some schemes that achieve similar performances with balanced modulation and have simple error-correcting code constructions? With this motivation, we propose a variant of balanced modulation, called partial-balanced modulation. The main idea is to construct an error-correcting code whose codewords are partially balanced, namely, only a certain segment of each codeword is balanced. When reading information from a block, we adjust the reading threshold to make this segment of the resulting word being balanced or being approximately balanced.

One way of constructing partial-balanced error-correcting codes is shown in figure 10.15. Given an information vector $\mathbf{u}$ of k bits (k is even), according to Knuth's observation [69], there exists an integer i with $0 \leq i < k$ such that inverting the first i bits of $\mathbf{u}$ results in a balanced word $\widetilde{\mathbf{u}}$. Since our goal is to construct a codeword that is partially balanced, it is not necessary to present i in a balanced form. Now, we use $\mathbf{i}$ denote the binary representation of length $\lceil \log_2 k \rceil$ for i. To further correct potential errors, we consider $[\widetilde{\mathbf{u}}, \mathbf{i}]$ as the information part and add extra parity-check bits by applying a systematic error-correcting code, like BCH code, Reed-Solomon code, etc. As a result, we obtain a codeword $\mathbf{x} = [\widetilde{\mathbf{u}}, \mathbf{i}, \mathbf{r}]$ where $\mathbf{r}$ is the redundancy part. In this codeword, $\widetilde{\mathbf{u}}$ is balanced,

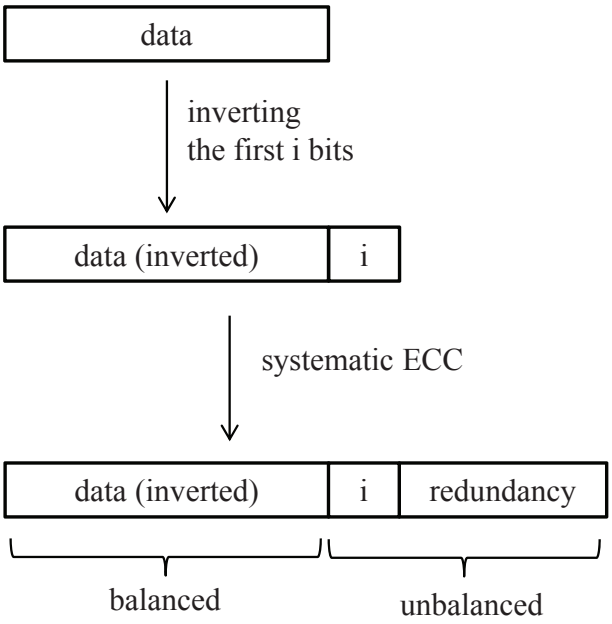


Figure 10.15. Partial balanced code.

$[\mathbf{i}, \mathbf{r}]$ is not balanced.

Note that in most data-storage applications, the bit error rate of a block is usually very small. The application of modulation schemes can further reduce the bit error rate. Hence, the number of errors in real applications is usually much smaller than the block length. In this case, the total length of $[\mathbf{i}, \mathbf{r}]$ is smaller or much smaller than the code dimension k. As the block length n becomes large, like one thousand, the reading threshold determined by partial-balanced modulation is almost the same as the one determined by balanced modulation. One assumption that we made is that all the cells in the same block have similar noise properties. To make this assumption being sound, we can reorder the bits in $\mathbf{x} = [\widetilde{\mathbf{u}}, \mathbf{i}, \mathbf{r}]$ such that the k cells of storing $\widetilde{\mathbf{u}}$ is (approximately) randomly distributed among all the n cells. Compared to balanced modulation, partial-balanced modulation can achieve almost the same performance, and its code construction is much easier (the constraints on the codewords are relaxed). In the following two examples, it compares the partial-balanced modulation scheme with the traditional one based on a fixed threshold.

Example 10.6. *Let us consider a nonvolatile memory with block length* $n = 255$. *To guarantee the data reliability, each block has to correct* 18 *errors if the reading process is based on a fixed reading*

threshold. Assume $(255, 131)$ *primitive BCH code is applied for correcting errors, then the data rate (defined by the ratio between the number of available information bits and the block length) is*

$$\frac{131}{255} = 0.5137.$$

Example 10.7. *For the block discussed in the previous example, we assume that it only needs to correct* 8 *errors based on partial-balanced modulation. In this case, we can apply* $(255, 191)$ *primitive BCH code for correcting errors, and the data rate is*

$$\frac{191 - 8}{255} = 0.7176,$$

which is much higher than the one obtained in the previous example.

The reading/decoding process of partial-balanced modulation is straightforward. First, the reading threshold v_b is adjusted such that among the cells corresponding to $\mathbf{u}$ there are $k/2$ cells or approximately $k/2$ cells with higher levels than v_b. Based on this reading threshold v_b, the whole block is read as a binary word $\mathbf{y}$, which can be further decoded as $[\widetilde{\mathbf{u}}, \mathbf{i}]$ if the total number of errors is well bounded. Then we obtain the original message $\mathbf{u}$ by inverting the first i bits of $\widetilde{\mathbf{u}}$.

10.8 Balanced Codes for Multi-Level Cells

In order to maximize the storage capacity of nonvolatile memories, multi-level cells (MLCs) are used, where a cell of q discrete levels can store $\log_2 q$ bits [17]. Flash memories with 4 and 8 levels have been used in products, and MLCs with 16 levels have been demonstrated in prototypes. For PCMs, cells with 4 or more levels have been in development.

The idea of balanced modulation and partial-balanced modulation can be extended to multi-level cells. For instance, if each cell has 4 levels, we can construct a balanced code in which each codeword has the same number of 0s, 1s, 2s, and 3s. When reading data from the block, we adjust three reading thresholds such that the resulting word also has the same number of 0s, 1s, 2s, and

3s. The key question is how to construct balanced codes or partial-balanced codes for an alphabet size $q > 2$.

10.8.1 Construction based on Rank

A simple approach of constructing balanced codes for a nonbinary case is to consider the message as the rank of its codeword among all its permutations, based on the lexicography order. If the message is $\mathbf{u} \in \{0,1\}^k$, then the codeword length n is the minimum integer such that $n = qm$ and $\binom{qm}{m \quad m \quad \ldots \quad m} > 2^k$. The following examples are provided for demonstrating the encoding and decoding processes.

Example 10.8. *Assume the message is* $\mathbf{u} = 1010010010$ *of length* 10 *and* $q = 3$. *Since* $\binom{9}{3 \quad 3 \quad 3} > 2^{10}$, *we can convert* $\mathbf{u}$ *to a balanced word* $\mathbf{x}$ *of length* 9 *and alphabet size* $q = 3$. *Let* S *denote the set that consists of all the balanced words of length* 9 *and alphabet size* $q = 3$. *To map* $\mathbf{u}$ *into a word in* S, *we write* $\mathbf{u}$ *into the decimal form* $r = 658$ *and let* r *be the rank of* $\mathbf{x}$ *in* S *based on the lexicographical order.*

Let us consider the first symbol of $\mathbf{x}$. *In* S, *there are totally* $\binom{8}{2 \quad 3 \quad 3} = 560$ *sequences starting with* 0, *or* 1, *or* 2. *Since* $560 \leq r < 560 + 560$, *the first symbol in* $\mathbf{x}$ *would be* 1, *then we update* r *as* $r - 560 = 98$, *which is the rank of* $\mathbf{x}$ *among all the sequences starting with* 1.

Let us consider the second symbol of $\mathbf{x}$. *There are totally* $\binom{8}{2 \quad 2 \quad 3}$ *sequences starting with* 10, *and it is larger than* r, *so the second symbol of* $\mathbf{x}$ *is* 0.

Repeating this process, we can convert $\mathbf{u}$ *into a balanced word* $\mathbf{x} = 101202102$.

Example 10.9. *We use the same notations as the above example. Given* $\mathbf{x} = 101202102$, *it is easy to calculate its rank in* S *based on the lexicographical order (via enumerative source coding [26]). It*

is

$$\begin{aligned} r &= \binom{8}{2\ 3\ 3} + \binom{6}{1\ 2\ 3} + \binom{5}{1\ 1\ 3} + \binom{5}{2\ 0\ 3} \\ &\quad + \binom{3}{0\ 1\ 2} + \binom{3}{1\ 0\ 2} + \binom{2}{0\ 1\ 1} \\ &= 656, \end{aligned}$$

where $\binom{8}{2\ 3\ 3}$ *is the number of* $\mathbf{x}$*'s permutations starting with* 0, $\binom{6}{1\ 2\ 3}$ *is the number of* $\mathbf{x}'$ *permutations starting with* 100, ...

Then from r*, we can get its binary representation* $\mathbf{u} = 1010010010$*. In [99], Ryabko and Matchikina showed that if the length of* $\mathbf{x}$ *is* n*, then we can get the message* $\mathbf{u}$ *in* $O(n \log^3 n \log\log n)$ *time.*

The above approach is simple and information efficient, but the encoding is not computationally fast.

10.8.2 Generalizing Knuth's Construction

An alternative approach is to generalize Knuth's idea to the nonbinary case due to its operational simplicity. Generally, assume that we are provided a word $\mathbf{u} \in G_q^k$ with $G_q = \{0, 1, 2, ..., q-1\}$ and $k = qm$, our goal is to generalize Knuth's idea to make $\mathbf{u}$ being balanced.

Let us consider a simple case, $q = 4$. Given a word $\mathbf{u} \in G_4^k$, we let n_i with $0 \leq i \leq 3$ denote the number of is in $\mathbf{u}$. To balance all the cell levels, we first balance the total number of 0s and 1s, such that $n_0 + n_1 = 2m$. It also results in $n_2 + n_3 = 2m$. To do this, we can treat 0 and 1 as an identical state and treat 2 and 3 as another identical state. Based on Knuth's idea, there always exists an integer i such that by operating on the first i symbols ($0 \to 2$, $1 \to 3$, $2 \to 0$, $3 \to 1$) it yields $n_0 + n_1 = 2m$. We then consider the subsequence consisting of 0s and 1s, whose length is $2m$. By applying Knuth's idea, we can make this subsequence being balanced. Similarly, we can also balance the subsequence consisting of 2s and 3s. Consequently, we convert any word in G_4^k into a balanced word. In order to decode this word, three additional integers of length at most $\lceil \log k \rceil$ need to be stored, indicating the locations of having operations. The following example is constructed for the

purpose of demonstrating this procedure.

Example 10.10. *Assume* $\mathbf{u} = 0110230210110003$*, we convert it into a balanced word with the following steps:*

(1) By operating the first 4 *symbols in* $\mathbf{u}$*, it yields* 2332230210110003*, where* $n_0 + n_1 = 8$*.*

(2) Considering the subsequence of 0*s and* 1*s, i.e., the underlined part in* $2332230\underline{2}\underline{10110003}$*. By operating the first bit of this subsequence* $(0 \to 1, 1 \to 0)$*, it yields* $2332231\underline{2}\underline{10110003}$*, where* $n_0 = n_1 = 4$*.*

(3) Considering the subsequence of 0*s and* 1*s, i.e., the underlined part in* $\underline{2332231}\underline{2}10110003$*. By operating the first* 0 *bit of this subsequence* $(2 \to 3, 3 \to 2)$*, it yields* $\underline{2332231}\underline{2}10110003$*, which is balanced.*

To recover 0110230210110003 *from* 2332231210110003 *(the inverse process), we need to record the three integers* $[4, 1, 0]$ *whose binary lengths are* $[\log_2 16, \log_2 8, \log_2 8]$*.*

It can be observed that the procedure above can be easily generalized for any $q = 2^a$ with $a \geq 2$. If $m = 2^b$ with $b \geq a$, then the number of bits to store the integers (locations) is

$$\sum_{j=0}^{\log_2 q - 1} 2^j \log_2 \frac{qm}{2^j} = (q-1)ab - q(a-2) - 2.$$

For instance, if $q = 2^3 = 8$ and $m = 2^7 = 128$, then $k = 1024$ and it requires 137 bits to represent the locations. These bits can be stored in 46 cells without balancing.

In fact, the above idea can be generalized for an arbitrary $q > 2$. For instance, when $q = 3$, given an binary word $\mathbf{u} \in G_3^{3m}$, there exists an integer i such that $\mathbf{u} + \mathbf{1}^i\mathbf{0}^{3m-i}$ has exactly m 0s or m 1s. Without loss of generality, we assume that it has exactly m 0s, then we can further balance the subsequence consisting of 1s and 2s. Finally, we can get a balanced word with alphabet size 3. More generally, we have the following result.

Theorem 10.4. *Given an alphabet size* $q = \alpha\beta$ *with two integers* α *and* β*, we divide all the levels into* β *groups, denoted by* $\{0, \beta, 2\beta, ...\}$*,* $\{1, \beta + 1, 2\beta + 1, ...\}$*, ...,* $\{\beta - 1, 2\beta - 1, 3\beta - 1, ...\}$*. Given*

any word $\mathbf{u} \in G_q^{qm}$*, there exists an integer* i *such that* $\mathbf{u} + \mathbf{1}^i\mathbf{0}^{qm-i}$ *has exactly* αm *symbols in one of the first* $\beta - 1$ *groups.*

Proof. Let us denote all the groups as $S_0, S_1, ..., S_{\beta-1}$. Given a sequence $\mathbf{u}$, we use n_j denote the number of symbols in $\mathbf{u}$ that belong to S_j. Furthermore, we let n'_j denote the number of symbols in $\mathbf{u} + \mathbf{1}^{qm}$ that belong to S_j. It is easy to see that $n'_{j+1} = n_j$ for all $j \in \{0, 1, ..., \beta - 1\}$, where $(\beta - 1) + 1 = 0$. We prove that that there exists $j \in \{0, 1, ..., \beta - 2\}$ such that $n_j \geq \alpha m \geq n'_j$ or $n_j \leq \alpha m \leq n'_j$ by contradiction. Assume this statement is not true, then either $\min(n_j, n'_j) > \alpha m$ or $\max(n_j, n'_j) < \alpha m$ for all $j \in \{0, 1, ..., \beta - 2\}$. So if $n_1 > \alpha m$, we can get $n_j > \alpha m$ for all $j \in \{0, 1, ..., \beta - 1\}$ iteratively. Similarly, if $n_1 < \alpha m$, we can get $n_j < \alpha m$ for all $j \in \{0, 1, ..., \beta - 1\}$ iteratively. Both cases contradict with the fact that $\sum_{j=0}^{\beta} n_j = \alpha m \beta = qm$.

Note that the number of symbols in $\mathbf{u} + \mathbf{1}^i\mathbf{0}^{qm-i}$ that belong to S_j changes by at most 1 if we increase i by one. So if there exists $j \in \{0, 1, ..., \beta - 2\}$ such that $n_j \geq \alpha m \geq n'_j$ or $n_j \leq \alpha m \leq n'_j$, there always exists an integer i such that $\mathbf{u} + \mathbf{1}^i\mathbf{0}^{qm-i}$ has exactly αm symbols in S_j.

This completes the proof. □

Based on the above result, given any q, we can always split all the levels into two groups and make them being balanced (the number of symbols belonging to a group is proportional to the number of levels in that group). Then we can balance the levels in each group. Iteratively, all the levels will be balanced. In order to recover the original message, it requires roughly

$$(q-1)\log_2 q \log_2 m$$

bits for storing additional information when m is large. If we store this additional information as a prefix using a shorter balanced code, then we get a generalized construction of Knuth's code. If we follow the steps in section 10.7 by further adding parity-check bits, then we get a partial-balanced code with error-correcting capability, based on which we can implement partial-balanced modulation for multiple-level cells.

Now, if we have a code that uses 'full' sets of balanced codewords, then the redundancy is

$$\log_2 q^{qm} - \log_2 \binom{qm}{m, m, ..., m} \simeq \frac{q - \log_2 q}{2} \log_2 m$$

bits. So given an alphabet size q, the redundancy of the above method is about $\frac{2(q-1)\log_2 q}{q-\log_2 q}$ times as high as that of codes that uses 'full' sets of balanced codewords. For $q = 2, 3, 4, 5, ..., 10$, we list these factors as follows:

$$2.0000, 4.4803, 6.0000, 6.9361, 7.5694,$$

$$8.0351, 8.4000, 8.6995, 8.9539.$$

It shows that as q increases, the above method becomes less information efficient. How to construct balanced codes for a nonbinary alphabet in a simple, efficient and computationally fast way is still an open question. It is even more difficult to construct balanced error-correcting codes for nonbinary alphabets.

10.9 Conclusion

In this chapter, we introduced balanced modulation for reading/writing in nonvolatile memories. Based on the construction of balanced codes or balanced error-correcting codes, balanced modulation can minimize the effect of asymmetric noise, especially those introduced by cell-level drifts. Hence, it can significantly reduce the bit error rate in nonvolatile memories. Compared to the other schemes, balanced modulation is easy to be implemented in the current memory systems and it does not require any assumptions about the cell-level distributions, which makes it very practical. Furthermore, we studied the construction of balanced error-correcting codes, in particular, balanced LDPC codes. It has very efficient encoding and decoding algorithms, and it is more efficient than prior construction of balanced error-correcting codes.

Chapter 11

Systematic Error-Correcting Codes for Rank Modulation

This chapter explores systematic error-correcting codes for rank modulation while considering the Kendall τ-distance. It presents $(k+2;k)$ systematic codes for correcting a single error, and proves that systematic codes for rank modulation can achieve the same capacity as general error-correcting codes.[1]

11.1 Introduction

The rank modulation scheme has been proposed recently for efficiently and robustly writing and storing data in nonvolatile memories (NVMs) [58, 60]. Its applications include flash memories [20], which are currently the most widely used family of NVMs, and several emerging NVM technologies, such as phase-change memories [18]. The rank modulation scheme uses the relative order of cell levels to represent data, where a cell level denotes a floating-gate cell's threshold voltage for flash memories and denotes a cell's electrical resistance for resistive memories (such as phase-change memories). Consider n memory cells, where for $i = 1, 2, ..., n$, let $c_i \in \mathbb{R}$ denote the level of the ith cell. It is assumed that no two cells have the same level, which is easy to realize in practice. Let $\mathcal{S}_n$ denote the set of all $n!$ permutations of $\{1, 2, ..., n\}$. The n cell levels induce a permutation $[x_1, x_2, ..., x_n] \in \mathcal{S}_n$, where $c_{x_1} > c_{x_2} > ... > c_{x_n}$. The rank modulation scheme uses such permutations to represent

[1]Some of the results presented in this chapter have been previously published in [147].

data. It enables memory cells to be programmed efficiently and robustly from lower levels to higher levels, without the risk of overprogramming. It also makes it easier to adjust cell levels when noise appears without erasing/resetting cells, and makes the stored data be more robust to asymmetric errors that change cell levels in the same direction [58, 60].

Error-correcting codes for rank modulation are very important for data reliability [20, 59]. Errors are caused by noise in cell levels, and the smallest error that can happen is for two adjacent cell levels to switch their order in the permutation, which is called an *adjacent transposition* [29]. An adjacent transposition changes a permutation $[x_1, x_2, ..., x_n] \in \mathcal{S}_n$ to $[x_1, ..., x_{i-1}, x_{i+1}, x_i, x_{i+2}, ..., x_n]$ for some $i \in \{1, 2, ..., n-1\}$. In this chapter, as in [10, 59, 60], we measure the distance between two permutations $\mathbf{x} = [x_1, x_2, ..., x_n] \in \mathcal{S}_n$ and $\mathbf{y} = [y_1, y_2, ..., y_n] \in \mathcal{S}_n$ by the minimum number of adjacent transpositions needed to change $\mathbf{x}$ into $\mathbf{y}$ (and vice versa), and denote it by $d_\tau(\mathbf{x}, \mathbf{y})$. This distance metric is called the Kendall's τ-distance [29]. For example, if $\mathbf{x} = [2, 1, 3, 4]$ and $\mathbf{y} = [3, 1, 4, 2]$, then $d_\tau(\mathbf{x}, \mathbf{y}) = 4$, because to change the permutation from $\mathbf{x}$ to $\mathbf{y}$ (or vice versa), we need at least 4 adjacent transpositions: $[2, 1, 3, 4] \rightarrow [1, 2, 3, 4] \rightarrow [1, 3, 2, 4] \rightarrow [1, 3, 4, 2] \rightarrow [3, 1, 4, 2]$. Based on this distance metric, an error-correcting code that can correct t errors is a subset of $\mathcal{S}_n$ whose minimum distance is at least $2t + 1$.

There have been some results on error-correcting codes for rank modulation equipped with the Kendall's τ-distance. In [59], a one-error-correcting code is constructed based on metric embedding, whose size is provably within half of the optimal size. In [10], the capacity of rank modulation codes is derived for the full range of minimum distance between codewords, and the existence of codes whose sizes are within a constant factor of the sphere-packing bound for any fixed number of errors is shown. Some explicit constructions of error-correcting codes have been proposed and analyzed in [80] and [81]. There has also been some work on error-correcting codes for rank modulation equipped with the L_∞ distance [104, 112]. The distance metric is more appropriate for cells where the noise in cell levels has limited magnitudes.

In this chapter, we study *systematic* error-correcting codes for rank modulation as a new approach for code design. Let k and n be two integers such that $2 \leq k < n$. In an (n, k) systematic code,

we use the permutation induced by the levels of n cells to store data. The first k cells are called *information cells*, whose induced permutation has a one-to-one mapping to information bits. The last $n-k$ cells are called *redundant cells*, which are used to add redundancy to the codewords. Compared to the existing constructions of error-correcting codes for rank modulation, systematic codes have the benefit that they support efficient data retrieval, because when there is no error (or when error correction is not considered), data can be retrieved by only reading the information cells. And since every permutation induced by the information cells represents a unique value of the data, the permutations can be mapped to data (and vice versa) very efficiently via enumerative source coding (e.g., by ordering permutations alphabetically and map them to data) [26, 79]. In addition, the encoding algorithm of the error-correcting code can potentially be made very efficient by defining the positions of the redundant cells in the permutation as a function of the corresponding positions of the information cells.

We study the design of systematic codes, and analyze their performance. We present a family of $(k+2, k)$ systematic codes for correcting one error, where either k or $k+1$ is a prime number. We show that they have optimal rates among systematic codes, unless *perfect* systematic one-error-correcting codes, which meet the sphere-packing bound, exist. We also study the design of systematic codes that correct multiple errors, and prove that for any $2 \leq k < n$, there exists a systematic code of minimum distance $n-k$. Furthermore, we prove that for rank modulation, systematic codes have the same capacity as general error-correcting codes. This result establishes that asymptotically, systematic codes are as strong in their error correction capability as general codes.

The rest of the chapter is organized as follows. In section 11.2, we define some terms and show properties of systematic codes. In section 11.3, we study systematic codes that correct one error. In section 11.4, we study codes that correct multiple errors. In section 11.5, we present the capacity of systematic codes, which matches the capacity of general codes. In section 11.7, we present the concluding remarks.

11.2 Terms and Properties

In this section, we define some terms for systematic codes, and show its basic properties. Let $C \subseteq \mathcal{S}_n$ denote a general (n, k) systematic error-correcting code for rank modulation. Given a codeword $\mathbf{x} = [x_1, x_2, ..., x_n] \in C$, we call the permutation induced by the first k cells (i.e., the information cells) $\mathbf{a} = [a_1, a_2, ..., a_k] \in \mathcal{S}_k$ the *information sector* of the codeword $\mathbf{x}$. More specifically, if $c_1, c_2, ..., c_n$ are the n cells' levels that induce the permutation $[x_1, x_2, ..., x_n] \in C$, then we have $c_{a_1} > c_{a_2} > ... > c_{a_k}$. Clearly, the information sector $[a_1, a_2, ..., a_k]$ is a subsequence of its codeword $[x_1, x_2, ..., x_n]$; namely, $[a_1, a_2, ..., a_k] = [x_{i_1}, x_{i_2}, ..., x_{i_k}]$ for some $1 \leq i_1 < i_2 < ... < i_k \leq n$.

Example 11.1. *Let $k = 4$ and $n = 6$. Let $c_1 = 1.0$, $c_2 = 2.1$, $c_3 = 0.8$, $c_4 = 0.2$, $c_5 = 1.5$, $c_6 = 0.6$. Then the permutation induced by the $n = 6$ cells is $[2, 5, 1, 3, 6, 4]$. The permutation induced by the $k = 4$ information cells is $[2, 1, 3, 4]$. We can see that $[2, 1, 3, 4]$ is a subsequence of $[2, 5, 1, 3, 6, 4]$.* □

Given a permutation $\mathbf{x} = [x_1, x_2, ..., x_n] \in \mathcal{S}_n$, we can see it as constructed by sequentially inserting $1, 2, ..., n$ into an initially empty permutation. Hence, we define the *insertion vector* of $\mathbf{x}$ as the positions of inserting $1, 2, ..., n$. Specifically, for $1 \leq i \leq n$, let $g_i(\mathbf{x})$ denote the position of the insertion of the integer i. That is, if $p \in \{1, 2, ..., n\}$ denotes the integer such that $x_p = i$, then

$$g_i(\mathbf{x}) = |\{j | 1 \leq j < p, x_j < i\}|.$$

Then we have the insertion vector

$$\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), g_2(\mathbf{x}), ..., g_n(\mathbf{x})] \in \mathbb{Z}_1 \times \mathbb{Z}_2 \times ... \times \mathbb{Z}_n,$$

where $\mathbb{Z}_i = \{0, 1, 2, ..., i-1\}$. Note that given $\mathbf{g}(\mathbf{x})$, we can reconstruct $\mathbf{x}$ uniquely. It has been shown that for any $\mathbf{x}, \mathbf{y} \in \mathcal{S}_n$ [10],

$$d_\tau(\mathbf{x}, \mathbf{y}) \geq \sum_{i=1}^{n} |g_i(\mathbf{x}) - g_i(\mathbf{y})|.$$

For an (n, k) systematic code, it is required that for every permutation $\mathbf{a} = [a_1, a_2, ..., a_k] \in \mathcal{S}_k$, there is exactly one codeword with $\mathbf{a}$ as its information sector, which we will denote by $\mathbf{x_a}$. The code has $k!$ codewords, and we define its *rate* as $\frac{\ln k!}{\ln n!}$. Given an information sector $\mathbf{a} \in \mathcal{S}_k$, we can get the insertion vector of its codeword $\mathbf{x_a}$, namely,

$$\begin{aligned} \mathbf{g}(\mathbf{x_a}) &= [g_1(\mathbf{x_a}), g_2(\mathbf{x_a}), ..., g_n(\mathbf{x_a})] \\ &= [g_1(\mathbf{a}), ..., g_k(\mathbf{a}), g_{k+1}(\mathbf{x_a}), ...g_n(\mathbf{x_a})]. \end{aligned}$$

It means that $\mathbf{x_a}$ can be constructed from $\mathbf{a}$ in the following way: First, we insert $k+1$ (namely, the $(k+1)$th cell) into the permutation $[a_1, a_2, ..., a_k]$ at the position $g_{k+1}(\mathbf{x_a}) \in \mathbb{Z}_{k+1}$; next, we insert the integer $k+2$ (namely, the $(k+2)$th cell) at the position $g_{k+2}(\mathbf{x_a}) \in \mathbb{Z}_{k+2}$; and so on. (The last integer to insert is n.) To design good systematic codes, given the information permutation $\mathbf{a}$, we need to find $[g_{k+1}(\mathbf{x_a}), g_{k+2}(\mathbf{x_a}), ..., g_n(\mathbf{x_a})]$ appropriately to maximize the code's minimum distance.

Example 11.2. *Let* $k = 4$ *and* $n = 6$. *If* $\mathbf{a} = [1, 3, 2, 4]$, $g_5(\mathbf{x_a}) = 3$ *and* $g_6(\mathbf{x_a}) = 0$, *then* $\mathbf{x_a} = [6, 1, 3, 2, 5, 4]$. □

The following theorem shows how the insertion of redundant cells into the information sector affects the Kendall's τ-distance between codewords.

Theorem 11.1. *Given two permutations* $\mathbf{a}, \mathbf{b} \in \mathcal{S}_k$, *the Kendall's* τ*-distance between* $\mathbf{x_a}$ *and* $\mathbf{x_b}$ *satisfies the inequality*

$$d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq d_\tau(\mathbf{a}, \mathbf{b}) + \sum_{i=k+1}^{n} |g_i(\mathbf{x_a}) - g_i(\mathbf{x_b})|.$$

Proof. The proof is by induction. As the base case, the inequality is clearly satisfied if $n = k$. Now consider the inductive step. Suppose that the inequality holds for any integer n with $n < k + r$. (Here r is a nonnegative integer.) We need to show that it also holds for $n = k + r$.

Consider a sequence of $d_\tau(\mathbf{x_a}, \mathbf{x_b})$ adjacent transpositions that changes the permutation $\mathbf{x_a} \in \mathcal{S}_n$

into the permutation $\mathbf{x_a} \in \mathcal{S}_n$. Among them, assume that α adjacent transpositions involve the integer n, and β adjacent transpositions do not involve n. (Clearly, $d_\tau(\mathbf{x_a}, \mathbf{x_b}) = \alpha + \beta$.) Since the integer n needs to be moved from position $g_n(\mathbf{x_a})$ to position $g_n(\mathbf{x_b})$, we get $\alpha \geq |g_n(\mathbf{x_a}) - g_n(\mathbf{x_b})|$. Note that those adjacent transpositions that involve n do not change the relative order of the integers $\{1, 2, ..., n-1\}$ in the permutation. So to transform the integers $\{1, 2, ..., n-1\}$ from their relative order in permutation $\mathbf{x_a}$ to their relative order in permutation $\mathbf{x_b}$, by the induction assumption, we get

$$\beta \geq d_\tau(\mathbf{a}, \mathbf{b}) + \sum_{i=k+1}^{n-1} |g_i(\mathbf{x_a}) - g_i(\mathbf{x_b})| .$$

That leads to the conclusion. □

Example 11.3. *Let $n = 3$ and $k = 2$. If $\mathbf{a} = [1, 2]$, $\mathbf{b} = [2, 1]$, $g_3(\mathbf{x_a}) = 1$ and $g_3(\mathbf{x_b}) = 2$, then $\mathbf{x_a} = [1, 3, 2]$ and $\mathbf{x_b} = [2, 1, 3]$. In this case, the inequality in theorem 11.1 becomes equality:*

$$d_\tau(\mathbf{x_a}, \mathbf{x_b}) = d_\tau(\mathbf{a}, \mathbf{b}) + |g_3(\mathbf{x_a}) - g_3(\mathbf{x_b})| = 2.$$

The equality, however, does not always hold. For instance, if $\mathbf{a} = [1, 2]$, $\mathbf{b} = [2, 1]$ and $g_3(\mathbf{a}) = g_3(\mathbf{b}) = 1$, then $\mathbf{x_a} = [1, 3, 2]$ and $\mathbf{x_b} = [2, 3, 1]$. We have

$$d_\tau(\mathbf{x_a}, \mathbf{x_b}) = 3 > d_\tau(\mathbf{a}, \mathbf{b}) + |g_3(\mathbf{x_a}) - g_3(\mathbf{x_b})| = 1.$$

□

We now present an inequality for ball sizes in $\mathcal{S}_n$, which will be useful for the analysis of systematic codes. Given a permutation $\mathbf{x} \in \mathcal{S}_n$, the ball of radius r centered at $\mathbf{x}$, denoted by $\mathfrak{B}_r(\mathbf{x})$, is the set of permutations in $\mathcal{S}_n$ that are within distance r from $\mathbf{x}$. Namely, $\mathfrak{B}_r(\mathbf{x}) = \{\mathbf{y} \in \mathcal{S}_n | d_\tau(\mathbf{x}, \mathbf{y}) \leq r\}$, for $0 \leq r \leq \frac{n(n-1)}{2}$. (The maximum Kendall's τ-distance for any two permutations in $\mathcal{S}_n$ is $\frac{n(n-1)}{2}$. [60]) A simple relabeling argument suffices to show that the size of a ball does not depend on the choice of its center. So we use $|\mathfrak{B}_r(n)|$ to denote $|\mathfrak{B}_r(\mathbf{x})|$ for any $\mathbf{x} \in S_n$.

The value of $|\mathfrak{B}_r(n)|$ is provided in [60]. It is shown that $|\mathfrak{B}_r(n)| = \sum_{i=0}^{r} e_i$, where e_i is the

coefficient of x^i in the polynomial $\prod_{j=1}^{n-1} \frac{x^{j+1}-1}{x-1}$. When $1 \leq r \leq n$, e_r can be obtained explicitly [10]. In this chapter, we will use the following inequality for ball sizes in the analysis of systematic codes.

Lemma 11.2. *For any* $0 \leq r \leq \frac{n(n-1)}{2}$,

$$|\mathfrak{B}_r(n)| \leq \binom{n+r-1}{n-1}.$$

Proof. Given a permutation $\mathbf{x} = [x_1, x_2, ..., x_n] \in \mathcal{S}_n$, we have $(g_1(\mathbf{x}), g_2(\mathbf{x}), ..., g_n(\mathbf{x})) \in \mathbb{Z}_1 \times \mathbb{Z}_2 \times ... \times \mathbb{Z}_n$.

For any two permutations $\mathbf{x}, \mathbf{y} \in \mathcal{S}_n$, we have

$$d_\tau(\mathbf{x}, \mathbf{y}) \geq \sum_{i=1}^{n} |g_i(\mathbf{x}) - g_i(\mathbf{y})|.$$

Let us consider a ball $\mathfrak{B}_r(\mathbf{x})$ with the center $\mathbf{x} = [n, n-1, ..., 1]$. Since $g_1(\mathbf{x}) = g_2(\mathbf{x}) = ... = g_n(\mathbf{x}) = 0$, for any permutation $\mathbf{y} \in \mathcal{S}_n$, we have

$$d_\tau(\mathbf{x}, \mathbf{y}) \geq \sum_{i=1}^{n} |g_i(\mathbf{y}) - g_i(\mathbf{x})| = \sum_{i=1}^{n} g_i(\mathbf{y}) = \sum_{i=2}^{n} g_i(\mathbf{y}),$$

with $g_i(\mathbf{y}) \in \mathbb{Z}_i$. (Note that $g_1(\mathbf{y}) = 0$.)

To compute $|\mathfrak{B}_r(\mathbf{x})|$, we let $d_\tau(\mathbf{x}, \mathbf{y}) \leq r$. It yields the relaxed condition

$$\sum_{i=2}^{n} g_i(\mathbf{y}) \leq r.$$

If we further relax the constraint that $g_i(\mathbf{y}) \leq i-1$ and only consider the constraint that $g_i(\mathbf{y}) \geq 0$, then there are $\binom{n+r-1}{n-1}$ different solutions to $(g_2(\mathbf{y}), g_3(\mathbf{y}), ..., g_n(\mathbf{y}))$ for the inequality $\sum_{i=2}^{n} g_i(\mathbf{y}) \leq r$. (It is equivalent to the problem of placing r balls in n boxes.) Since every permutation $\mathbf{y} \in \mathcal{S}_n$ can be distinctly determined by its corresponding vector $(g_2(\mathbf{y}), g_3(\mathbf{y}), ..., g_n(\mathbf{y}))$, there are at most $\binom{n+r-1}{n-1}$ permutations in $\mathcal{S}_n$ whose distance to $\mathbf{x}$ is at most r. □

11.3 One-Error-Correcting Codes

In this section, we analyze and design systematic codes for correcting one error. Such codes have minimum distance 3. In particular, we present a family of $(k+2, k)$ systematic codes, where either k or $k+1$ is a prime number. It will be shown that the codes have optimal rates among systematic codes, unless perfect systematic one-error-correcting codes, which meet the sphere-packing bound, exist.

11.3.1 Properties of One-Error-Correcting Codes

A r-error-correcting code $C \subseteq \mathcal{S}_n$ for rank modulation needs to satisfy the sphere-packing bound: $|C| \leq \frac{n!}{|\mathfrak{B}_r(n)|}$. If the inequality in the above bound becomes equality, we call the code *perfect*. For one-error-correcting codes, since $|\mathfrak{B}_1(n)| = n$, the following result holds.

Theorem 11.3. *A systematic (n, k) one-error-correcting code for rank modulation is perfect if and only if $n = k + 1$. More generally, a perfect one-error-correcting code (systematic or not) of length n has $(n-1)!$ codewords.*

It is known that perfect codes are often rare. Well-known examples include binary codes, where the only perfects codes are Hamming codes and Golay codes, and Lee metric codes in three-dimensional and higher-dimensional spaces [48]. For rank modulation, there is a simple $(3, 2)$ one-error-correcting code that is perfect: $\{[1, 2, 3], [3, 2, 1]\}$. However, beside this trivial code, no other perfect code has been found yet. If we add the requirement that the code needs to be systematic, it will be even harder for such codes to exist. For instance, it can be proved that there does not exist any perfect systematic one-error-correcting code when $k = 3$.

Theorem 11.4. *There does not exist any $(4, 3)$ systematic one-error-correcting code for rank modulation.*

Proof. The proof is by contradiction. Suppose that there exists a perfect $(4, 3)$ systematic one-error-correcting code, which we denote by C. As before, for any permutation $\mathbf{a} \in S_3$, we let $\mathbf{x_a} \in \mathcal{S}_4$ denote the unique codeword in C with $\mathbf{a}$ as its information sector, and we write $g_4(\mathbf{x_a})$ as $h(\mathbf{a})$. And

for convenience of expression in the following analysis, given any two information sectors $\mathbf{a}, \mathbf{b} \in \mathcal{S}_3$, we denote the distance between their corresponding codewords by $d_\tau^{(f)}(\mathbf{a}, \mathbf{b})$.

We first prove that at least one of the codewords in C does not start or end with 4; namely, there exists a permutation $\mathbf{a} \in S_3$ such that $h(\mathbf{a}) \notin \{0, 3\}$. This statement can be proved by contradiction. Assume that every codeword in C either starts with 4 or ends with 4. Without loss of generality, we can let $h([1, 2, 3]) = 3$. Then the only possible choice for $h([2, 1, 3])$ and $h([1, 3, 2])$ is 0 because otherwise, $d_\tau^{(f)}([1, 2, 3], [2, 1, 3])$ and $d_\tau^{(f)}([1, 2, 3], [1, 3, 2])$ would equal 1, which would contradict the requirement C has minimum distance at least 3. Hence we get two codewords $[4, 2, 1, 3]$ and $[4, 1, 3, 2]$. However, in this case, their distance equals 2, which contradicts our assumption.

So there exists at least one permutation $\mathbf{a} \in \mathcal{S}_3$ such that $h(\mathbf{a}) \in \{1, 2\}$. Without loss of generality (by symmetry), we can let $\mathbf{a} = [1, 2, 3]$ and let $h(\mathbf{a}) = 2$. Its corresponding codeword is $[1, 2, 4, 3]$.

We now consider the codewords whose information sectors are $[2, 1, 3]$, $[1, 3, 2]$, $[3, 1, 2]$, $[3, 2, 1]$, $[2, 3, 1]$, respectively.

1. $[2, 1, 3]$ is at distance one from $[1, 2, 3]$. Hence the only possible codeword with $[2, 1, 3]$ as its information sector is $[4, 2, 1, 3]$ because otherwise, we would have $d_\tau^{(f)}([2, 1, 3]), [1, 2, 3]) < 3$.

2. $[1, 3, 2]$ is also at distance one from $[1, 2, 3]$. To make $d_\tau^{(f)}([1, 3, 2], [1, 2, 3]) \geq 3$, we have $h([1, 3, 2]) \in \{0, 2\}$. Since it is required that $d_\tau^{(f)}([1, 3, 2], [2, 1, 3]) \geq 3$, the only possible value for $h([1, 3, 2])$ is 2. Therefore, the codeword with $[1, 3, 2]$ as its information sector is $[1, 3, 4, 2]$.

3. With a similar analysis, we get $h([3, 1, 2]) = 0$. Its corresponding codeword is $[4, 3, 1, 2]$.

4. Since it is required that $d_\tau^{(f)}([3, 2, 1], [3, 1, 2]) \geq 3$, we need $h([3, 2, 1]) \in \{2, 3\}$. Since it is required that $d_\tau^{(f)}([2, 3, 1], [2, 1, 3]) \geq 3$, we need $h([2, 3, 1]) \in \{2, 3\}$. However, in this case, by enumerating all the possible values for $h[3, 2, 1]$ and $h([2, 3, 1])$, we can see that

$$d_\tau^{(f)}([3, 2, 1], [2, 3, 1]) < 3,$$

which is a contradiction.

Based on the above analysis, it can be concluded that there does not exists any $(4, 3)$ systematic code correcting one error for rank modulation. □

For any given $k \geq 3$, if the perfect $(k+1, k)$ code does not exist, then the $(k+2, k)$ code becomes the optimal code. We show such an $(6, 4)$ systematic code in the appendix. In the following subsection, we present a family of $(k+2, k)$ systematic codes, where either k or $k+1$ is a prime number.

11.3.2 Construction of $(k+2, k)$ One-Error-Correcting Codes

We now present the construction that builds a family of $(k+2, k)$ systematic one-error-correcting codes.

Construction 11.1. *Let $k \geq 3$ be an integer such that either k or $k+1$ is a prime number. Given any information sector $\mathbf{a} = [a_1, a_2, ..., a_k] \in \mathcal{S}_k$, let $g_{k+1}(\mathbf{x_a}) \in \mathbb{Z}_{k+1}, g_{k+2}(\mathbf{x_a}) \in \mathbb{Z}_{k+2}$ be the positions of inserting $k+1$ and $k+2$. We set*

$$\begin{aligned} g_{k+1}(\mathbf{x_a}) &= \textstyle\sum_{i=1}^{k}(2i-1)a_i \ mod\ m, \\ g_{k+2}(\mathbf{x_a}) &= \textstyle\sum_{i=1}^{k}(2i-1)^2 a_i \ mod\ m, \end{aligned} \tag{11.1}$$

where $m = k$ if k is a prime number and $m = k+1$ if $k+1$ is a prime number. □

The following theorem shows that the above code can correct one error.

Theorem 11.5. *The $(k+2, k)$ systematic code in construction 11.1 has minimum distance at least 3. Hence it is a one-error-correcting code.*

Proof. In the $(k+2, k)$ code of construction 11.1, either k or $k+1$ is a prime number. Let us first consider the case that k is a prime number. Assume that $\mathbf{a} = [a_1, a_2, ..., a_k] \in \mathcal{S}_k$ and $\mathbf{b} = [b_1, b_2, ..., b_k] \in \mathcal{S}_k$ are two distinct information sectors, whose corresponding codewords are $\mathbf{x_a}, \mathbf{x_b} \in \mathcal{S}_n$, respectively. Our goal is to prove that $d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq 3$. We consider three cases:

1. Case 1: $d_\tau(\mathbf{a}, \mathbf{b}) \geq 3$. In this case, we have $d_\tau(\mathbf{x}, \mathbf{y}) \geq d_\tau(\mathbf{a}, \mathbf{b}) \geq 3$.

2. Case 2: $d_\tau(\mathbf{a}, \mathbf{b}) = 1$. In this case, we can write $\mathbf{b}$ as $\mathbf{b} = [b_1, b_2, ..., b_k] = [a_1, a_2, ..., a_{i+1}, a_i, ..., a_k]$ for some $i \in \{1, 2, ..., k-1\}$. If we define $\Delta = a_{i+1} - a_i$, then we get

$$g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b}) = 2\Delta \pmod{\text{k}}.$$

Since $1 \leq |\Delta| \leq k-1$ and $k \geq 3$ is a prime number, we know that 2Δ is not a multiple of k. As a result, we get

$$|g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b})| \geq 1.$$

Similarly, we have

$$\begin{aligned} & g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b}) \\ = \; & (2i-1)^2 a_i + (2i+1)^2 (a_i + \Delta) \\ & -(2i-1)^2 (a_i + \Delta) - (2i+1)^2 a_i \\ = \; & 8i\Delta \pmod{\text{k}}, \end{aligned}$$

where $8i\Delta$ is not a multiple of k, either, because $1 \leq i, |\Delta| \leq k-1$ and $k \geq 3$ is a prime number. This implies that $|g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b})| \geq 1$.

So by theorem 11.1, we get $d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq d_\tau(\mathbf{a}, \mathbf{b}) + |g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b})| + |g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b})| \geq 1 + 1 + 1 = 3$.

3. Case 3: $d_\tau(\mathbf{a}, \mathbf{b}) = 2$. In this case, it takes at least two adjacent transpositions to change the permutation $\mathbf{a}$ into $\mathbf{b}$. These two transpositions can be either separated (which means that the two pairs of integers involved in the two transposition do not share any common integer) or adjacent to each other (which means that the two pairs of integers involved in the two transpositions share one common integer). We consider the two cases.

In the first case that the two adjacent transpositions are separated, we can write $\mathbf{b}$ as

$$\mathbf{b} = [a_1, ..., a_{i+1}, a_i, ..., a_{j+1}, a_j, ..., a_k]$$

for some $1 < i + 1 < j < k$. Let us define $\Delta_1 = a_{i+1} - a_i$ and $\Delta_2 = a_{j+1} - a_j$. Then we get

$$g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b}) = 2(\Delta_1 + \Delta_2) \pmod{\text{k}}.$$

If $\Delta_1 + \Delta_2$ is not a multiple of k, then $|g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b})| \geq 1$. This leads to $d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq d_\tau(\mathbf{a}, \mathbf{b}) + |g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b})| \geq 2 + 1 = 3$. If $\Delta_1 + \Delta_2$ is a multiple of k, we can write Δ_2 as $\Delta_2 = tk - \Delta_1$ for some integer $t \in \{-1, 0, 1\}$. Hence

$$\begin{aligned}
& g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b}) \\
= \; & (2i-1)^2 a_i + (2i+1)^2 (a_i + \Delta_1) \\
& + (2j-1)^2 a_j + (2j+1)^2 (a_j + tk - \Delta_1) \\
& - (2i-1)^2 (a_i + \Delta_1) - (2i+1)^2 a_i \\
& - (2j-1)^2 (a_j + tk - \Delta_1) - (2j+1)^2 a_j \\
= \; & 8(j-i)\Delta_1 \pmod{\text{k}},
\end{aligned}$$

where $8(j - i)\Delta_1$ is not a multiple of k. So $|g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b})| \geq 1$, which leads to $d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq d_\tau(\mathbf{a}, \mathbf{b}) + |g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b})| \geq 2 + 1 = 3$.

In the second case that the two transpositions are adjacent to each other, we have either

$$\mathbf{b} = [a_1, ..., a_{i+2}, a_i, a_{i+1}, ..., a_k],$$

or

$$\mathbf{b} = [a_1, ..., a_{i+1}, a_{i+2}, a_i, ..., a_k],$$

for some $1 \leq i \leq k-2$.

By defining $\Delta_1 = a_{i+2} - a_{i+1}$ and $\Delta_2 = a_{i+2} - a_i$ (or $\Delta_1 = a_{i+1} - a_i$ and $\Delta_2 = a_{i+2} - a_i$), with the same argument as above, it can be proved that either $|g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b})| \geq 1$ or $|g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b})| \geq 1$. Therefore we again have $d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq d_\tau(\mathbf{a}, \mathbf{b}) + |g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b})| + |g_{k+2}(\mathbf{x_a}) - g_{k+2}(\mathbf{x_b})| \geq 2 + 1 = 3$.

Therefore, we can conclude that when k is a prime number, for any two distinct codewords $\mathbf{x_a}, \mathbf{x_b}$, their distance is at least 3. When $k+1$ is a prime number, we can apply the same procedure for the proof, with only replacing "mod k" by "mod $k+1$". And we get the result that $d_\tau(\mathbf{x_a}, \mathbf{x_b}) \geq 3$.

This completes the proof. □

We now present the encoding and decoding algorithms of the $(k+2, k)$ systematic code. Let $L = \{0, 1, ..., k! - 1\}$ denote the set of information symbols to encode. (If the input is information bits, they can be easily mapped to the information symbols in L.) For encoding, given an information symbol $\ell \in L$, it can be mapped to its corresponding permutation (i.e., information sector) $\mathbf{a} \in \mathcal{S}_k$ in time linear in k [79]. Based on construction 11.1, the insertion vector $(g_{k+1}(\mathbf{x_a}), g_{k+2}(\mathbf{x_a}))$ can be directly computed, which gives us the codeword $\mathbf{x_a}$. That completes the encoding algorithm.

We now describe the decoding algorithm. Let $\mathbf{x_a} \in \mathcal{S}_{k+2}$ denote the correct codeword, and let $\mathbf{a} = [a_1, a_2, ..., a_k] \in \mathcal{S}_k$ be its information sector. Let $\mathbf{y} \in \mathcal{S}_{k+2}$ denote the received (possibly noisy) codeword, and let $\mathbf{b} = [b_1, b_2, ..., b_k] \in \mathcal{S}_k$ be its information sector. Suppose that there is at most one error in $\mathbf{y}$. A straightforward decoding algorithm is to check all the $k+2$ permutations within distance one from $\mathbf{y}$ (including $\mathbf{y}$ itself), and verify which one of them is the correct codeword. There is, however, a more efficient decoding algorithm that avoids checking the $k+2$ candidate permutations, which we describe below.

Given the received codeword $\mathbf{y}$, let $g_1 \in \mathbb{Z}_{k+1}$ and $g_2 \in \mathbb{Z}_{k+2}$ denote the positions of the insertion of the integers $k+1$ and $k+2$, respectively. Let $\mathbf{x_b}$ be the codeword corresponding to the information sector $\mathbf{b}$, which can be computed based on construction 11.1. If $d_\tau(\mathbf{x_b}, \mathbf{y}) \leq 1$, then $\mathbf{x_b} = \mathbf{x}$ is the correct codeword and $\mathbf{b} = \mathbf{a}$ is the correct information sector; otherwise, there is an error in $\mathbf{b}$, which

we will find as follows. We can write $\mathbf{a}$ as $\mathbf{a} = [b_1, ..., b_{i+1}, b_i, ..., b_k]$ for some i with $1 \leq i \leq k-1$. In this case, we have $g_{k+1}(\mathbf{x_a}) = g_1$ and $g_{k+2}(\mathbf{x_a}) = g_2$ because

$$d_\tau(\mathbf{a}, \mathbf{b}) + |g_{k+1}(\mathbf{x_a}) - g_1| + |g_{k+2}(\mathbf{x_a}) - g_2| \leq d_\tau(\mathbf{x_a}, \mathbf{y}) \leq 1,$$

which implies $|g_{k+1}(\mathbf{x_a}) - g_1| = 0$ and $|g_{k+2}(\mathbf{x_a}) - g_2| = 0$.

According to the proof of theorem 11.5, we know that

$$g_1 - g_{k+1}(\mathbf{x_b}) = g_{k+1}(\mathbf{x_a}) - g_{k+1}(\mathbf{x_b}) = 2(b_i - b_{i+1}) \pmod{\text{m}},$$

$$g_2 - g_{k+2}(\mathbf{x_b}) = 8i(b_i - b_{i+1}) \pmod{\text{m}},$$

where m is the prime number in $\{k, k+1\}$. Based on these two equations, we get

$$g_2 - g_{k+2}(\mathbf{x_b}) = 4i(g_1 - g_{k+1}(\mathbf{x_b})) \pmod{\text{m}}. \tag{11.2}$$

By solving this equation, we can obtain the value for $i \in \{1, 2, ..., k-1\}$ that gives us the correct information sector $\mathbf{a}$ and its codeword $\mathbf{x_a}$.

We illustrate the decoding algorithm with the following example.

Example 11.4. *Let $k = 4$ and the correct information sector be $\mathbf{a} = [4, 1, 3, 2]$. Based on equation (11.1) in construction 11.1, we get its codeword $\mathbf{x_a} = [4, 5, 6, 1, 3, 2]$. Assume that one error happened and we receive the noisy word $\mathbf{y} = [4, 5, 6, 3, 1, 2]$, which we decode in the following way. First, from $\mathbf{y}$, we get $\mathbf{b} = [4, 3, 1, 2]$ and $g_1 = 1, g_2 = 1$. And we have $g_{k+1}(\mathbf{x_b}) = 2$, $g_{k+2}(\mathbf{x_b}) = 4$. Since here*

$$d_\tau(\mathbf{x_b}, \mathbf{y}) \geq |g_1 - g_{k+1}(\mathbf{x_b})| + |g_2 - g_{k+2}(\mathbf{x_b})| > 1,$$

there is one error in $\mathbf{b}$. From equation (11.2), we get $2 = -4i \mod 5$, which gives us $i = 2 \in \{1, 2, 3\}$. So it is determined that the correct information sector is $[4, 1, 3, 2]$. □

Given k, the $(k+2, k)$ code uses the minimum amount of redundancy among systematic codes,

unless there exists a perfect and systematic $(k+1, k)$ one-error-correcting code. And compared to the one-error-correcting code presented in [60], the $(k+2, k)$ codes presented here have more efficient encoding and decoding algorithms.

11.4 Multi-Error-Correcting Codes

In this section, we study the design of systematic codes that correct multiple errors, and prove that for any $2 \leq k < n$, there exists an (n, k) systematic code of minimum distance $n - k$.

The one-error-correcting code in construction 11.1 can be generalized for correcting multiple errors in the following way. Given any information sector $\mathbf{a} = [a_1, a_2, ..., a_k] \in \mathcal{S}_k$, we set its insertion vector $(g_{k+1}(\mathbf{x_a}), g_{k+2}(\mathbf{x_a}), ..., g_n(\mathbf{x_a}))$ as follows: For $j = 1, 2, ..., n-k$,

$$g_{k+j}(\mathbf{x_a}) = \sum_{i=1}^{k} (2i-1)^j a_i \bmod m,$$

where $m = k$ if k is a prime number and $m = k+1$ if $k+1$ is a prime number. This gives us a sequence of codes, including a $(10, 4)$ code of minimum distance 5, a $(14, 4)$ code of minimum distance 7, etc. In this section, we explore the existence of more efficient systematic codes.

We present a generic scheme for constructing an (n, k) systematic code of minimum distance d. The scheme is based on greedy searching. Although it is beyond the scope of this chapter to obtain efficient encoding and decoding algorithms for it, the analysis of this scheme is very useful for proving the existence of codes with certain parameters, and for deriving the capacity of systematic codes.

Construction 11.2. *Let $2 \leq k < n$ and $d \geq 1$. In this scheme, we construct an (n, k) systematic code of minimum distance d. It uses a greedy approach for choosing codewords as follows. Let $\mathbf{s_1}, \mathbf{s_2}, ..., \mathbf{s_{k!}}$ denote the $k!$ permutations in $\mathcal{S}_k$, respectively. For $i = 1, 2, ..., k!$, we choose the codeword $\mathbf{x_{s_i}}$ whose information sector is $\mathbf{s_i}$ as follows: Among all the permutations in $\mathcal{S}_n$ that contain*

$\mathbf{s_i}$ *as their information sector, choose a permutation* $\mathbf{x_{s_i}}$ *such that*

$$\forall j \in \{1, 2, ..., i-1\}, d_\tau(\mathbf{x_{s_i}}, \mathbf{x_{s_j}}) \geq d. \tag{11.3}$$

If all the $k!$ *codewords* $\mathbf{x_{s_1}}, \mathbf{x_{s_2}}, ..., \mathbf{x_{s_{k!}}}$ *can be generated successfully this way, we obtain an* (n, k) *systematic code of minimum distance* d. □

Note that given any $\mathbf{a} \in \mathcal{S}_k$, there are $(k+1) \times (k+2) \times ... \times n = \frac{n!}{k!}$ permutations in $\mathcal{S}_n$ that have $\mathbf{a}$ as their information sector. For the above code construction to succeed, $n-k$ needs to be sufficiently large. In the following theorem, we derive a bound for the parameters.

Theorem 11.6. *Construction 11.2 can successfully build an* (n, k) *systematic code of minimum distance* d *if*

$$\sum_{i=1}^{d-1} \binom{k+i-2}{i} 2^{\min(d-i-1, n-k)} \binom{d-i-1+n-k}{n-k} < \frac{n!}{k!}. \tag{11.4}$$

Proof. In construction 11.2, for any information sector $\mathbf{s_i} \in \mathcal{S}_k$ (where $1 \leq i \leq k!$), there are $\frac{n!}{k!}$ possible choices for the vector $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$. Our goal is to make sure that at least one of them, which will become the corresponding codeword $\mathbf{x_{s_i}}$, can guarantee to satisfy the requirement in (11.3).

Let us consider the maximum number of choices for the vector $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$ whose corresponding permutations in $\mathcal{S}_n$ are at distance less than d from at least one permutation in $\{\mathbf{x_{s_1}}, \mathbf{x_{s_2}}, ..., \mathbf{x_{s_{i-1}}}\}$. Such insertion vectors cannot be chosen for the codeword $\mathbf{x_{s_i}}$. For any word $\mathbf{b} = \mathbf{s_u}$ with $u < i$, if $d_\tau(\mathbf{s_i}, \mathbf{b}) = j \leq d-1$, to make $d_\tau(\mathbf{x_{s_i}}, \mathbf{x_b}) \geq d$, it is enough to let

$$\sum_{t=1}^{n-k} |g_{k+t}(\mathbf{x_{s_i}}) - g_{k+t}(\mathbf{x_b})| \geq d - j.$$

Now we are interested in the number of solutions to $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$ that satisfy the inequality

$$\sum_{t=1}^{n-k} |g_{k+t}(\mathbf{x_{s_i}}) - g_{k+t}(\mathbf{x_b})| \leq d - j - 1.$$

We call such solutions *unavailable combinations* for $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$. Note that there are at most $\binom{d-j-1+n-k}{n-k}$ possible choices for

$$[|g_{k+1}(\mathbf{x_{s_i}}) - g_{k+1}(\mathbf{x_b})|, |g_{k+2}(\mathbf{x_{s_i}}) - g_{k+2}(\mathbf{x_b})|,$$

$$..., |g_n(\mathbf{x_{s_i}}) - g_n(\mathbf{x_b})|].$$

Among them, at most $\min(d-j-1, n-k)$ elements are not zero. Hence the number of unavailable combinations for $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$ (due to the constraint imposed by $\mathbf{y}$) is at most

$$2^{\min(d-j-1,n-k)}\binom{d-j-1+n-k}{n-k}.$$

Let N_j be the number of permutations in $\mathcal{S}_k$ whose distance to $\mathbf{s_i}$ is j. Based on the union bound, the total number of unavailable combinations for $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$ is at most

$$N = \sum_{j=1}^{d-1} N_j 2^{\min(d-j-1,n-k)}\binom{d-j-1+n-k}{n-k}.$$

According to lemma 11.2, there are at most $\binom{k+j-1}{k-1}$ permutations in $\mathcal{S}_k$ for which the distance between their information sectors and $\mathbf{s_i}$ is at most j, namely,

$$1 + \sum_{t=1}^{j} N_t \leq \binom{k+j-1}{k-1},$$

for $1 \leq j \leq d-1$.

In this case, it is not hard to prove that N is maximized when

$$N_j = \binom{k+j-1}{k-1} - \binom{k+j-2}{k-1} = \binom{k+j-2}{k},$$

for $k \geq 2$ and $1 \leq j \leq d-1$ because $2^{\min(d-j-1,n-k)}\binom{d-j-1+n-k}{n-k}$ is a deceasing function of j.

As a result, we get

$$N \leq \sum_{j=1}^{d-1} \binom{k+j-2}{j} 2^{\min(d-j-1,n-k)} \binom{d-j-1+n-k}{n-k}.$$

Since the total number of possible combinations for $[g_{k+1}(\mathbf{x_{s_i}}), g_{k+2}(\mathbf{x_{s_i}}), ..., g_n(\mathbf{x_{s_i}})]$ is $\frac{n!}{k!}$, if $N < \frac{n!}{k!}$, we can always find an available combination such that equation (11.3) is satisfied. And this is true for all information sectors. So the conclusion holds. □

Given k and d, we can calculate the minimum value of n that satisfies the inequality in theorem 11.6.

Example 11.5. *When $d = 3$ and $n = k + 2$, the inequality in theorem 11.6 can be simplified as*

$$6\binom{k-1}{1} + \binom{k}{2} < (k+1)(k+2),$$

which holds for any $k \geq 2$. Therefore, there exists a $(k+2, k)$ systematic code that corrects one error for any $k \geq 2$. (Note that this result is consistent with the $(k+2, k)$ systematic one-error-correcting code built in construction 11.1.) □

Example 11.6. *When $d = 4$ and $n = k + 3$, the inequality in theorem 11.6 can be simplified as*

$$40\binom{k-1}{1} + 8\binom{k}{2} + \binom{k+1}{3} < (k+1)(k+2)(k+3),$$

which holds for all $k \geq 2$. Therefore, there exists a $(k+3, k)$ systematic code of minimum distance 4 for any $k \geq 2$. □

We now prove that for any $2 \leq k < n$, there exists an (n, k) systematic code of minimum distance $n - k$.

Theorem 11.7. *For any $k \geq 2$ and $d \geq 1$, there exists a $(k + d, k)$ systematic code of minimum distance d.*

Proof. Based on theorem 11.6, to show that there exists a $(k+d,k)$ systematic code of minimum distance d, we only need to prove

$$\sum_{i=1}^{d-1}\binom{k+i-2}{i}2^{d-i-1}\binom{2(d-1)-i}{d-1}<\frac{(k+d)!}{k!},$$

for $k\geq 2, d\geq 2$. (The case of $d=1$ is trivial.)

Here, we consider a stronger condition,

$$\sum_{i=1}^{d-1}\binom{k+i}{i}2^{d-i-1}\binom{2(d-1)-i}{d-1}<\frac{(k+d)!}{k!}. \tag{11.5}$$

We define

$$\psi_d(k)=\frac{\sum_{i=1}^{d-1}\binom{k+i}{i}2^{d-i-1}\binom{2(d-1)-i}{d-1}}{\frac{(k+d)!}{k!}}.$$

Then we would like to show that the ratio between $\psi(k+1)$ and $\psi(k)$ is at most 1. That is true because

$$\begin{aligned}\frac{\psi_d(k+1)}{\psi_d(k)} &= \frac{\sum_{i=1}^{d-1}\binom{k+1+i}{i}2^{d-i-1}\binom{2(d-1)-i}{d-1}}{\sum_{i=1}^{d-1}\binom{k+i}{i}2^{d-i-1}\binom{2(d-1)-i}{d-1}}\\ &\quad\times\frac{\frac{(k+d)!}{k!}}{\frac{(k+1+d)!}{(k+1)!}}\\ &\leq \max_{i=1}^{d-1}\frac{\binom{k+1+i}{i}}{\binom{k+i}{i}}\frac{\frac{(k+d)!}{k!}}{\frac{(k+1+d)!}{(k+1)!}}\\ &\leq \max_{i=1}^{d-1}\frac{k+1+i}{1+k}\frac{1+k}{1+k+d}\leq 1.\end{aligned}$$

This implies that given any $d\geq 2$, $\psi_d(k)$ is a nonincreasing function of k. If $\psi_d(2)<1$ for all $d\geq 2$, then for any $k,d\geq 2$, we have $\psi_d(k)<1$, which proves the condition in equation (11.5). So our task is to prove $\psi_d(2)<1$, namely,

$$\sum_{i=1}^{d-1}\binom{2+i}{i}2^{d-i-1}\binom{2(d-1)-i}{d-1}<\frac{(2+d)!}{2!},$$

for $d \geq 2$.

The left side of the inequality is

$$\begin{aligned}
& \sum_{i=1}^{d-1} \binom{2+i}{i} 2^{d-i-1} \binom{2(d-1)-i}{d-1} \\
\leq & \sum_{i=1}^{d-1} 3 \times 2^{d-2} \binom{2d-3}{d-1} \\
& \times (\frac{(i+2)(i+1)}{6} 2^{1-i} \prod_{j=2}^{i} \frac{d-j}{2d-1-j}) \\
\leq & \sum_{i=1}^{d-1} 3 \times 2^{d-2} \binom{2d-3}{d-1} (\frac{1}{2})^{i-1} \\
\leq & 6 \times 2^{d-2} \binom{2d-3}{d-1}.
\end{aligned}$$

Now, we need to show that

$$6 \times 2^{d-2} \binom{2d-3}{d-1} < \frac{(2+d)!}{2!},$$

for any $d \geq 2$. When $2 \leq d \leq 8$, we can show that the inequality holds by computing the exact values. When $d \geq 8$, we define

$$\phi(d) = \frac{6 \times 2^{d-2} \binom{2d-3}{d-1}}{\frac{(2+d)!}{2!}}.$$

Then

$$\frac{\phi(d+1)}{\phi(d)} = \frac{2(2d-1)(2d-2)}{d(d+1)(d+3)} \leq \frac{8}{d} \leq 1.$$

Since $\phi(8) < 1$, we get $\phi(d) < 1$ when $d \geq 8$.

Based on the above analysis, we see that the condition in equation (11.5) always holds when $d, k \geq 2$. That leads to the conclusion. □

Now, we present an explicit construction of systematic multi-error-correcting codes, by slightly modifying the multi-error-correcting codes derived in [80]. The idea is that given any two integers

$g_i(\mathbf{x_a}), g_i(\mathbf{x_b}) < 2^m$, there exists a function $\phi_m : \mathbb{Z}_{2^m} \to \{0,1\}^m$ (called Gray map) such that

$$|g_i(\mathbf{x_a}) - g_i(\mathbf{x_b})| \geq d_H(\phi_m(g_i(\mathbf{x_a})), \phi_m(g_i(\mathbf{x_b}))),$$

where d_H indicates the Hamming distance between two binary vectors. As a result, we can convert the problem of constructing rank modulation codes to the problem of constructing binary error-correcting codes in Hamming space. To make the code being systematic, we use $\phi_{\lceil \log_2 i \rceil}$ with $1 \leq i \leq k$ for the mapping of information part, instead of using $\phi_{\lfloor \log_2 i \rfloor}$ in the original construction.

Construction 11.3. *Let $2 \leq k < n$, we construct an (n,k) systematic rank modulation code, denoted by $C_\tau \subset \mathcal{S}_n$. Given any information sector $\mathbf{a} \in \mathcal{S}_k$, to construct its codeword $\mathbf{x_a} \in C_\tau$, we first construct $\mathbf{x_a}$'s image in a binary systematic code C_H, that is*

$$\begin{aligned} f(\mathbf{x_a}) =& [\phi_{\lceil \log_2 1 \rceil}(g_1(\mathbf{a})), ..., \phi_{\lceil \log_2 k \rceil}(g_k(\mathbf{a})), \\ & \phi_{\lfloor \log_2 (k+1) \rfloor}(g_{k+1}(\mathbf{x_a})), ..., \phi_{\lfloor \log_2 n \rfloor}(g_n(\mathbf{x_a}))]. \end{aligned}$$

In $f(\mathbf{x_a})$, the first $k' = \sum_{i=1}^{k} \lceil \log_2 i \rceil$ bits are the information bits and they can be obtained from the information sector $\mathbf{a}$ directly. The rest $r' = \sum_{i=k+1}^{n} \lfloor \log_2 i \rfloor$ bits are the parity-check bits based on the encoding of C_H. Then we can get $\mathbf{x_a} \in \mathcal{S}_n$ from $f(\mathbf{x_a})$ uniquely. If C_H is an $(k'+r', k')$ binary systematic code correcting t errors, then C_τ is an (n,k) systematic rank modulation code correcting t errors.

11.5 Capacity of Systematic Codes

In this section, we prove that for rank modulation, systematic error-correcting codes achieve the same capacity as general error-correcting codes. So they have the same asymptotic performance in terms of the error correction capability.

In [10], Barg and Mazumdar have derived the capacity of general error-correcting codes for rank modulation. Let $A(n,d)$ denote the maximum size of a code of length n and minimum distance d.

(So the code is a subset of $\mathcal{S}_n$.) Define the capacity of error-correcting codes of minimum distance d as

$$C(d) = \lim_{n\to\infty} \frac{\ln A(n,d)}{\ln n!}.$$

It is shown in [10] that

$$C(d) = \begin{cases} 1, & \text{if } d = O(n), \\ 1-\epsilon, & \text{if } d = \Theta(n^{1+\epsilon}) \text{ with } 0 < \epsilon < 1, \\ 0, & \text{if } d = \Theta(n^2). \end{cases} \tag{11.6}$$

For systematic codes, let $k(n,d)$ denote the maximum number of information cells that can exist in systematic codes of length n and minimum distance d. (Such codes are $(n, k(n,d))$ systematic codes, and have $k(n,d)!$ codewords.) The *capacity* of systematic codes of minimum distance d is

$$C_{sys}(d) = \lim_{n\to\infty} \frac{\ln k(n,d)!}{\ln n!}.$$

The following theorem shows that systematic codes have the same capacity as general codes.

Theorem 11.8. *The capacity of systematic codes of minimum distance d is*

$$C_{sys}(d) = \begin{cases} 1, & \textit{if } d = O(n), \\ 1-\epsilon, & \textit{if } d = \Theta(n^{1+\epsilon}) \textit{ with } 0 < \epsilon < 1, \\ 0, & \textit{if } d = \Theta(n^2). \end{cases}$$

Proof. Since systematic codes are a special case of general error-correcting codes, by equation (11.6), it is sufficient to prove

$$C_{sys}(d) \geq \begin{cases} 1, & \text{if } d = O(n), \\ 1-\epsilon, & \text{if } d = \Theta(n^{1+\epsilon}) \text{ with } 0 < \epsilon < 1, \\ 0, & \text{if } d = \Theta(n^2). \end{cases}$$

According to theorem 11.6, there exists an (n, k) systematic code of minimum distance d if k is the maximum integer that satisfies

$$\binom{k+d}{d} 2^n \binom{d+n-k}{n-k} < \frac{n!}{k!}.$$

That is because

$$\binom{k+d}{d} 2^n \binom{d+n-k}{n-k}$$

$$\geq \sum_{i=1}^{d-1} \binom{k+i-2}{i} 2^{\min(d-i-1, n-k)} \binom{d-i-1+n-k}{n-k},$$

for all $n > k \geq 2$ and $d \geq 2$.

For such k, we have $k(n, d) \geq k$. For convenience, let $\alpha = \lim_{n\to\infty} \frac{k}{n}$ be a constant. In this case, if $\alpha > 0$,

$$\begin{aligned} C_{sys}(d) &= \lim_{n\to\infty} \frac{\ln k(n,d)!}{\ln n!} \geq \lim_{n\to\infty} \frac{\ln k!}{\ln n!} \\ &= \lim_{n\to\infty} \frac{\alpha n \log(\alpha n)}{n \log n} = \alpha. \end{aligned}$$

To prove the final conclusion, we will show that if $d = O(n)$, then $\alpha = 1$; if $d = \Theta(n^{1+\epsilon})$, then $\alpha \geq 1 - \epsilon$. (If $d = \Theta(n^2)$, the result $\alpha \geq 0$ is trivial).

Based on the definition of k, we can get

$$\lim_{n\to\infty} \frac{\ln \binom{k+d}{d} 2^n \binom{d+n-k}{n-k}}{\ln \frac{n!}{k!}} = 1. \tag{11.7}$$

We consider two cases:

1) If $d = O(n)$, we have $d \leq \beta n$ for some $\beta > 0$. By Stirling's approximation, the formula above yields

$$\lim_{n\to\infty} \frac{(\alpha+\beta) n \ln \frac{\alpha+\beta}{\alpha\beta} + n \ln 2 + (\beta + 1 - \alpha) n \ln \frac{\beta+1-\alpha}{(1-\alpha)\beta}}{n \ln n - \alpha n \ln(\alpha n)} \geq 1,$$

which shows that $n \ln n - \alpha n \ln(\alpha n) = O(n)$. Hence α approaches 1 as $n \to \infty$.

2) If $d = \Theta(n^{1+\epsilon})$ for $0 < \epsilon < 1$, by applying Stirling's approximation to equation (11.7), we get

$$\lim_{n\to\infty} \frac{n\ln d - k\ln k - (n-k)\ln(n-k) + O(n)}{n\ln n - k\ln k + O(n)} = 1.$$

Since $k = \alpha n$ and $d = \Theta(n^{1+\epsilon})$, we get

$$\lim_{n\to\infty} \frac{(1+\epsilon)n\ln n - \alpha n\ln n - (1-\alpha)n\ln}{(1-\alpha)n\ln n} = 1.$$

That leads to $\alpha \geq 1 - \epsilon$.

Based on the above analysis and the fact that $S_{sys}(d) \geq \alpha$, we get the final conclusion. □

11.6 Appendix

In this appendix, we present an alternative $(6, 4)$ systematic code, and prove that it can correct one error.

The code is constructed as follows. Let us first show the adjacency graph for the permutations of $\mathcal{S}_k = \mathcal{S}_4$ in figure 11.1 (a), where two permutations are connected by an edge if and only if their Kendall's τ-distance is 1. The permutations in $\mathcal{S}_4$ are the permutations induced by the $k = 4$ information cells. And for any two permutations $\mathbf{a}, \mathbf{b} \in \mathcal{S}_4$, their Kendall's τ-distance $d_\tau(\mathbf{x_a}, \mathbf{x_b})$ equals the shortest-path distance in the adjacency graph in figure 11.1 (a).

Next, we insert a redundant cell (the 5th cell) into the permutations. For every permutation, we place the 5th cell right in the middle. As a result, we get the permutations in figure 11.1 (b). For any two permutations in figure 11.1 (b), they are connected by an edge if and only if their Kendall's τ-distance is 1. (An interesting thing to notice is that here every node has degree 2 and is in a cycle of length 4.)

In the final step, we insert another redundant cell (the 6th cell) into the permutations. As a result, we get the code in figure 11.1 (c), where the integer beside every codeword is the position of the 6th cell in that codeword (which equals $g_6(\mathbf{a}) + 1$ with $\mathbf{a}$ being the information sector). The

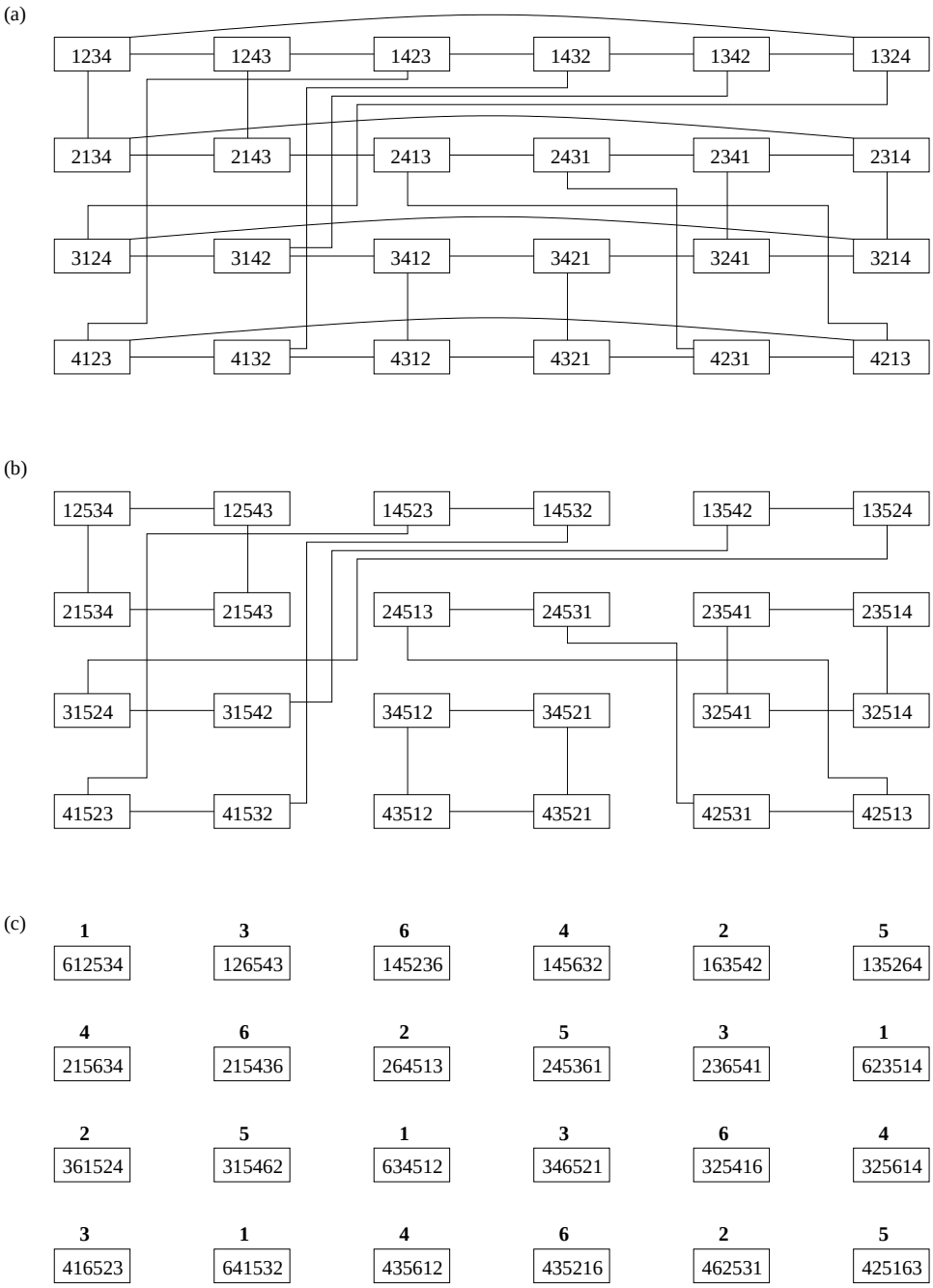


Figure 11.1. The construction of an (n, k) systematic one-error-correcting code for $n = 6$ and $k = 4$.

code is a $(6, 4)$ systematic code. The following theorem shows that it has minimum distance 3, and therefore is a one-error-correcting code.

Theorem 11.9. *The* $(6, 4)$ *systematic code in figure 11.1 (c) has minimum distance 3. So it is a one-error-correcting code.*

Proof. Since inserting redundant cells into permutations will only increase the distance between permutations, we just need to focus on the permutation pairs in figure 11.1 (a) that are at distance at most 2 from each other, and show that after adding the $n - k = 2$ redundant cells, their distance is at least 3.

First, consider the permutation pairs at distance one (i.e., adjacent permutations) in figure 11.1 (a). Every permutation $\mathbf{a} \in \mathcal{S}_4$ in figure 11.1 (a) has three neighbors, and they are contained in two cycles: a cycle of length 6 and a cycle of length 4. (For example, the permutation $[1, 2, 3, 4]$ has three neighbors: $[1, 2, 4, 3], [1, 3, 2, 4]$ and $[2, 1, 3, 4]$. The permutations $[1, 2, 3, 4], [1, 2, 4, 3], [1, 3, 2, 4]$ are in a cycle of length 6: $[1, 2, 3, 4] - [1, 2, 4, 3] - [1, 4, 2, 3] - [1, 4, 3, 2] - [1, 3, 4, 2] - [1, 3, 2, 4]$. The permutations $[1, 2, 3, 4], [1, 2, 4, 3], [2, 1, 3, 4]$ are in a cycle of length 4: $[1, 2, 3, 4] - [1, 2, 4, 3] - [2, 1, 4, 3] - [2, 1, 3, 4]$.) We consider the two cases:

- Consider a cycle of length 6. Let $S = (s_1, s_2, s_3, s_4, s_5, s_6)$ denote the positions of the number "6" in the final permutations in figure 11.1 (c). (Those positions are the numbers beside the permutations in figure 11.1 (c).) We can see that either $S = (1, 3, 6, 4, 2, 5)$ or $S = (6, 4, 1, 3, 5, 2)$ (or its cyclic shifts or inversions).

  For example, consider the cycle $[3, 4, 1, 2]-[3, 4, 2, 1]-[3, 2, 4, 1]-[3, 2, 1, 4]-[3, 1, 2, 4]-[3, 1, 4, 2]$ in figure 11.1 (a). The corresponding set of permutations in figure 11.1 (c) is $[6, 3, 4, 5, 1, 2] - [3, 4, 6, 5, 2, 1] - [3, 2, 5, 4, 1, 6] - [3, 2, 5, 6, 1, 4] - [3, 6, 1, 5, 2, 4] - [3, 1, 5, 4, 6, 2]$. For this cycle, we have $S = (1, 3, 6, 4, 2, 5)$.

  As another example, consider the cycle $[2, 1, 4, 3]-[2, 1, 3, 4]-[2, 3, 1, 4]-[2, 3, 4, 1]-[2, 4, 3, 1]-[2, 4, 1, 3]$ in figure 11.1 (a). The corresponding set of permutations in figure 11.1 (c) is $[2, 1, 5, 4, 3, 6] - [2, 1, 5, 6, 3, 4] - [6, 2, 3, 5, 1, 4] - [2, 3, 6, 5, 4, 1] - [2, 4, 5, 3, 6, 1] - [2, 6, 4, 5, 1, 3]$. For this cycle, we have $S = (6, 4, 1, 3, 5, 2)$.

  We see that any two adjacent numbers in the cycle S differ by at least 2. The two corresponding permutations in figure 11.1 (a) have distance 1. (Also note that the adjacency graph has no cycle of length less than 4.) So after inserting the redundant cells, their distance is at least

$2 + 1 = 3$.

- Similarly, consider a cycle of length 4. Let $S = (s_1, s_2, s_3, s_4)$ denote the positions of the number "6" in the final permutations in figure 11.1 (c). (Those positions are the numbers beside the permutations in figure 11.1 (c).) We can see that either $S = (1, 3, 6, 4)$ or $S = (2, 5, 2, 5)$ (or its cyclic shifts or inversions).

  For example, consider the cycle $[1, 2, 3, 4] - [1, 2, 4, 3] - [2, 1, 4, 3] - [2, 1, 3, 4]$ in figure 11.1 (a). The corresponding set of permutations in figure 11.1 (c) is $[6, 1, 2, 5, 3, 4] - [1, 2, 6, 5, 4, 3] - [2, 1, 5, 4, 3, 6] - [2, 1, 5, 6, 3, 4]$. For this cycle, we have $S = (1, 3, 6, 4)$.

  As another example, consider the cycle $[2, 4, 1, 3] - [2, 4, 3, 1] - [4, 2, 3, 1] - [4, 2, 1, 3]$ in figure 11.1 (a). The corresponding set of permutations in figure 11.1 (c) is $[2, 6, 4, 5, 1, 3] - [2, 4, 5, 3, 6, 1] - [4, 6, 2, 5, 3, 1] - [4, 2, 5, 1, 6, 3]$. For this cycle, we have $S = (2, 5, 2, 5)$.

  We see that any two adjacent numbers in the cycle S differ by at least 2. The two corresponding permutations in figure 11.1 (a) have distance 1. So after inserting the redundant cells, their distance is at least $2 + 1 = 3$.

So for any two adjacent permutations in figure 11.1 (a), after inserting the redundant cells, their distance is at least 3.

Next, consider the permutation pairs at distance two in figure 11.1 (a). Let $\mathbf{a} = [a_1, a_2, a_3, a_4] \in \mathcal{S}_4$ and $\mathbf{b} = [b_1, b_2, b_3, b_4] \in \mathcal{S}_4$ be two permutations at distance two in figure 11.1 (a). After inserting the 5th cell into them, they become $\mathbf{a}' = [a_1, a_2, 5, a_3, a_4] \in \mathcal{S}_5$ and $\mathbf{b}' = [b_1, b_2, 5, b_3, b_4] \in \mathcal{S}_5$. (See figure 11.1 (b).) After inserting the 6th cell into them, they become $\mathbf{a}'' \in \mathcal{S}_6$ and $\mathbf{b}'' \in \mathcal{S}_6$. Let $s_{\mathbf{a}}, s_{\mathbf{b}} \in \{1, 2, 3, 4, 5, 6\}$ denote the positions of the number "6" in $\mathbf{a}''$ and $\mathbf{b}''$, respectively. If $s_{\mathbf{a}} \neq s_{\mathbf{b}}$, then clearly $d_\tau(\mathbf{a}'', \mathbf{b}'') \geq 2 + 1 = 3$. So we only need to consider the case $s_{\mathbf{a}} = s_{\mathbf{b}}$. From figure 11.1, we can see it happens only in a cycle of length 4. For example, consider the cycle $[2, 4, 1, 3] - [2, 4, 3, 1] - [4, 2, 3, 1] - [4, 2, 1, 3]$ in figure 11.1 (a). If $\mathbf{a} = [2, 4, 1, 3]$ and $\mathbf{b} = [4, 2, 3, 1]$, then we have $d_\tau(\mathbf{a}, \mathbf{b}) = 2$, $\mathbf{a}' = [2, 4, 5, 1, 3]$, $\mathbf{b}' = [4, 2, 5, 3, 1]$, $\mathbf{a}'' = [2, 6, 4, 5, 1, 3]$, $\mathbf{b}'' = [4, 6, 2, 5, 3, 1]$, $s_{\mathbf{a}} = 2$, $s_{\mathbf{b}} = 2$. It is easy to see that $d_\tau(\mathbf{a}'', \mathbf{b}'') = d_\tau([2, 6, 4, 5, 1, 3], [4, 6, 2, 5, 3, 1]) > d_\tau([2, 6, 4], [4, 6, 2]) =$

3. Similarly, if $\mathbf{a} = [2, 4, 3, 1]$ and $\mathbf{b} = [4, 2, 1, 3]$, then we have $d_\tau(\mathbf{a}, \mathbf{b}) = 2$, $\mathbf{a}' = [2, 4, 5, 3, 1]$, $\mathbf{b}' = [4, 2, 5, 1, 3]$, $\mathbf{a}'' = [2, 4, 5, 3, 6, 1]$, $\mathbf{b}'' = [4, 2, 5, 1, 6, 3]$, $s_{\mathbf{a}} = 5$, $s_{\mathbf{b}} = 5$. It is easy to see that $d_\tau(\mathbf{a}'', \mathbf{b}'') = d_\tau([2, 4, 5, 3, 6, 1], [4, 2, 5, 1, 6, 3]) > d_\tau([3, 6, 1], [1, 6, 3]) = 3$. All the other permutation pairs are in similar cases. (Note that either $s_{\mathbf{a}} = s_{\mathbf{b}} = 2$, or $s_{\mathbf{a}} = s_{\mathbf{b}} = 5$.) So for any two permutations at distance two in figure 11.1 (a), after inserting the redundant cells, their distance is at least 3.

Since $[6, 1, 2, 5, 3, 4]$ and $[1, 2, 6, 5, 4, 3]$ are two codewords, and their distance is 3, the minimum distance of the code is exactly 3. It is a one-error-correcting code. □

11.7 Conclusion

In this chapter, we study systematic error-correcting codes for rank modulation. We present $(k+2, k)$ systematic codes for correcting one error, and analyze systematic codes that correct multiple errors. We prove that systematic codes have the same capacity as general codes. There are still many open problems for systematic codes for rank modulation. It is important to design multi-error-correcting codes of high rates with efficient encoding and decoding algorithms. It is also important to study codes equipped with distance metrics other than the Kendall's τ-distance, based on the different types of noise that are common in nonvolatile memories.

Bibliography

[1] E. Abbe,"Polarization and randomness extraction," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 184–188, 2011.

[2] K. A. S. Abdel-Ghaffar and H. C. Ferreira, "Systematic encoding of the Varshamov-Tenengol'ts codes and the Constantin-Rao codes," *IEEE Trans. Inform. Theory*, vol. 44, pp. 340–345, 1998.

[3] J. Abrahams, "Generation of discrete distributions from biased coins," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1541–1546, 1996.

[4] A. V. Aho, L. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley, 1976.

[5] Y. Akizawa, T. Yamazaki, A. Uchida, T. Harayama, S. Sunada, K. Arai, K. Yoshimura, and P. Davis, "Fast random number generation with bandwidth-enhanced chaotic semiconductor lasers at 8×50 Gb/s," *IEEE Photonics Technology Letters*, vol. 24, no. 12, pp. 1042–1044, 2012.

[6] S. Al-Bassam and B. Bose, "On balanced codes," *IEEE Trans. Inform. Theory*, vol. 36, pp. 406–408, Mar. 1990.

[7] S. Al-Bassam, B. Bose, "Design of efficient error-correcting balanced codes," *IEEE Trans. Comput.*, vol. 42, pp. 1261–1266, 1993.

[8] S. Al-Bassam, R. Venkatesan, and S. Al-Muhammadi, "New single asymmetric error-correcting codes," *IEEE Trans. Inform. Theory*, vol. 43, pp. 1619–1623, 1997.

[9] B. Barak, R. Impagliazzo, and A. Wigderson, "Extracting randomness using few independent sources," *SIAM J. Comput.*, 36:1095–1118, 2006.

[10] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Trans. Inform. Theory*, vol. 56, no. 7, pp. 3158–3165, 2010.

[11] M. Ben-Or and N. Linial, "Collective coin flipping," *Randomness and Computation*, New York, Academic Press, pp. 91–115, 1990.

[12] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, pp. 489–502, 2003.

[13] L. Blum, M. Blum, and M. Shub, "A simple, unpredictable pseudorandom generator," *SIAM J. Comput.*, vol. 15, no. 2, pp. 364–383, 1986.

[14] M. Blum, "Independent unbiased coin flips from a correlated biased source: A finite state Markov chain," *Combinatorica*, vol. 6, pp. 97–108, 1986.

[15] J. Bourgain, "More on the sum-product phenomenon in prime fields and its applications," *International Journal of Number Theory*, 1:1–32, 2005.

[16] B. Bose and S. Al-Bassam, "On systematic single asymmetric errorcorrecting codes," *IEEE Trans. Inform. Theory*, vol. 46, pp. 669–672, 2000.

[17] J. E. Brewer and M. Gill, *Nonvolatile Memory Technologies with Emphasis on Flash*, John Wiley & Sons, Hoboken, New Jersey, 2008.

[18] G. W. Burr et al., "Phase change memory technology," *Journal of Vacuum Science and Technology*, vol. 28, no. 2, pp. 223–262, March 2010.

[19] Y. Cai, E. F. Haratsch, O. Mutlu, K. Mai, "Error patterns in MLC NAND Flash memory: Measurement, characterization, and analysis," in *Proc. Design, Automation, and Test in Europe (DATE)*, 2012.

[20] P. Cappelletti, C. Golla, P. Olivo and E. Zanoni (Ed.), *Flash Memories*, Kluwer Academic Publishers, 1st Edition, 1999.

[21] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for asymmetric limited-magnitude errors with application to multilevel flash memories," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1582–1595, 2010.

[22] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem, "Probabilistic system-on-a-chip architecture," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–28, 2007.

[23] A. Cohen and A. Wigderson, "Dispersers, deterministic amplification, and weak random sources," in *Proc. Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.

[24] S. D. Constantin and T. R. N. Rao, "On the theory of binary asymmetric error-correcting codes," *Inform. Contr.*, vol. 40, pp. 20–36, 1979.

[25] M. Cook, D. Soloveichik, E. Winfree, J. Bruck, "Programmability of chemical reaction networks," *Algorithmic Bioprocesses, Natural Computing Series*, pp. 543–584, 2009.

[26] T. M. Cover, "Enumerative source coding," *IEEE Trans. Inform. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[27] T. M. Cover, J. A. Thomas, *Elements of Information Theory*, Second Edition, Wiley, July 2006.

[28] P. Delsarte and P. Piret, "Bounds and constructions for binary asymmetric error-correcting codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 125–128, 1981.

[29] M. Deza and H. Huang, "Metrics on permutations, a survey," *J. Comb. Inf. Sys. Sci.*, vol. 23, pp. 173–185, 1998.

[30] E. Dijkstra, "Making a fair roulette from a possibly biased coin," *Inform. Processing Lett.*, vol. 36, no. 4, pp. 193, 1990.

[31] Y. Dodis, "Impossibility of black-box reduction from non-adaptively to adaptively secure coin-flipping," *Technical Report 039*, Electronic Colloquium on Computational Complexity, 2000.

[32] Z. Dvir and A. Wigderson, "Kakeya sets, new mergers and older extractors," in *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.

[33] P. Elias, "The efficient construction of an unbiased random sequence," *Ann. Math. Statist.*, vol. 43, pp. 865–870, 1972.

[34] Y. El-Kurdi, D. Fernández, E. Souleimanov, D. Giannacopoulos, W. J. Gross, "FPGA architecture and implementation of sparse matrix-vector multiplication for the finite element method," *Computer Physics Communications 178(8)*: 558–570, 2008.

[35] T. Etzion, "Lower bounds for asymmetric and unidirectional codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1696–1704, 1991.

[36] G. Fang and H. C. A. van Tilborg, "Bounds and constructions of asymmetric or unidirectional error-correcting codes," *Appl. Algebra Engrg. Comm. Comput.*, vol. 3, no. 4, pp. 269–300, 1992.

[37] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes", *IEEE Trans. Inform. Theory*, vol. 51, pp. 954–972, Mar. 2005.

[38] B. Fett, J. Bruck, and M. D. Riedel, "Synthesizing stochasticity in biochemical systems," in *Proc. Annual Conference on Design Automation (DAC)*, pp. 640–645, 2007.

[39] F. Fu, S. Ling, and C. Xing, "New lower bounds and constructions for binary codes correcting asymmetric errors," *IEEE Trans. Inform. Theory*, vol. 49, pp. 3294–3299, 2003.

[40] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications*, John Wiley & Sons, 2006.

[41] A. Gabizon, R. Raz, and R. Shaltiel, "Deterministic extractors for bit-fixing sources by obtaining an independent seed," *SIAM J. Comput.*, 36:1072–1094, 2006.

[42] R. Gallager, "Low density parity check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[43] R. Gallager, *Low Density Parity Check Codes*, no. 21 in Research Monograph Series. Cambridge, MA: MIT Press, 1963.

[44] A. Gill, "Synthesis of probability transformers," *Journal of the Franklin Institute*, vol. 274, no. 1, pp. 1–19, 1962.

[45] A. Gill, "On a weight distribution problem, with application to the design of stochastic generators," *Journal of the ACM*, vol. 10, no. 1, pp. 110–121, 1963.

[46] I. Ya. Goldbaum, "Bounds on the number of signals in codes with asymmetrical error correction," *Automat. Tekmekh.*, vol. 32, pp. 94–97, 1971.

[47] O. Goldreich, H. Krawczyk, and M. Luby, "On the existence of pseudorandom generators," *SIAM J. Comput.*, vol. 22, no. 6, pp. 1163–1175, Dec. 1993.

[48] S. W. Golomb and L. R. Welch, "Perfect codes in the Lee metric and the packing of polyominoes," *SIAM J. Appl. Math.*, vol. 18, no. 2, pp. 302–317, 1970.

[49] R. L. Graham, and N. J. A. Sloane, "Lower bounds for constant weight codes," *IEEE Trans. Inform. Theory*, vol. 26, no. 1, pp. 37–43, 1980.

[50] V. Guruswami, C. Umans, and S. Vadhan, "Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes," in *Proc. IEEE Conference on Computational Complexity (CCC)*, pp. 96–108, 2007.

[51] T. S. Han, "Folklore in source coding: Information-spectrum approach," *IEEE Trans. Inform. Theory*, vol. 51, no. 2, pp. 747–753, 2005.

[52] T. S. Han and M. Hoshi, "Interval algorithm for random number generation," *IEEE Trans. Inform. Theory*, vol. 43, No. 2, pp. 599–611, 1997.

[53] N. Hessler, A. Shirke, and R. Malinow, "The probability of transmitter release at a mammalian central synapse," *Nature*, 366, pp. 569–572, 1993.

[54] W. Hoeffding and G. Simon, "Unbiased coin tossing with a biased coin," *Ann. Math. Statist.*, vol. 41, pp. 341–352, 1970.

[55] K. S. Immink and J. Weber, "Very efficient balanced codes," *IEEE Journal on Selected Areas in Communications*, vol. 28, pp. 188–192, 2010.

[56] R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudo-random generation from one-way functions," in *Proc. Annu. ACM Symp. Theory of Computing (STOC)*, Seattle, WA, May 15-17, pp. 12–24, 1989.

[57] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Trans. Inform. Theory*, vol. 40, no. 3, pp. 716–720, 1994.

[58] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 1731–1735, July 2008.

[59] A. Jiang, M. Schwartz and J. Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Trans. Inform. Theory*, vol. 56, no. 5, pp. 2112–2120, 2010.

[60] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 1736–1740, July 2008.

[61] A. Juels, M. Jakobsson, E. Shriver, B. K. Hillyer, "How to turn loaded dice into fair coins," *IEEE Trans. Inform. Theory*, vol. 46, pp. 911–921, 2000.

[62] B. Jun and P. Kocher, "The Intel random number generator," *http://www.cryptography.com/resources/whitepapers/IntelRNG.eps*, 1999.

[63] J. Kamp, A. Rao, S. Vadhan and D. Zuckerman, "Deterministic extractors for small-space sources," *Journal of Computer and System Sciences*, vol. 77, pp. 191–220, 2011.

[64] J. Kamp and D. Zuckerman, "Deterministic extractors for bit-fixing sources and exposure-resilient cryptography," *SIAM J. Comput.*, 36:1231–1247, 2006.

[65] I. Kanter, Y. Aviad, I. Reidler, E. Cohen, and M. Rosenbluh, "An optical ultrafast random bit generator," *Nat. Photon.*, vol. 4, no. 1, pp. 58–61, 2010.

[66] P. M. Kareiva and N. Shigesada, "Analyzing insect movement as a correlated random walk," *Oecologia* vol. 56, pp. 234–238, 1983.

[67] T. Kløve, "Error correcting codes for the asymmetric channel," Technical Report, Dept. of Informatics, University of Bergen, 1981. (Updated in 1995.)

[68] T. Kløve, "Upper bounds on codes correcting asymmetric errors," *IEEE Trans. Inform. Theory*, vol. 27, no. 1, pp. 128–131, 1981.

[69] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 32, no. 1, pp. 51–53, 1986.

[70] D. E. Knuth, *Things a Computer Scientist Rarely Talks About*, CSLI Publications, Stanford, 2001.

[71] D. E. Knuth and A. Yao, "The complexity of nonuniform random number generation," *Algorithms and Complexity: New Directions and Recent Results*, pp. 357–428, 1976.

[72] P. Lacharme, "Analysis and construction of correctors," *IEEE Trans. Inform. Theory*, vol. 55, pp. 4742–4748, 2009.

[73] P. Lacharme, "Post-processing functions for a biased physical random number generator," in *Proc. FSE*, vol. 5086, pp. 334–342, 2008.

[74] J. C. Lawrence, "A new universal coding scheme for the binary memoryless source," *IEEE Trans. Inform. Theory*, vol. 23, pp. 466–472, 1977.

[75] P. Loh, H. Zhou, and J. Bruck, "The robustness of stochastic switching networks," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2009.

[76] H. T. Lue et al., "Study of incremental step pulse programming (ISPP) and STI edge effect of BE-SONOS NAND flash," in *Proc. IEEE Int. Symp. on Reliability Physics*, pp. 693–694, May 2008.

[77] P. A. MacMahon, "The combinations of resistances," *The Electrician*, 28:601–602, 1892. (Reprinted in: Discr. Appl. Math., 54:225–228, 1994.).

[78] F. J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland: New York, NY, 1977.

[79] M. Mares and M. Straka, "Linear-time ranking of permutations," *Algorithms-ESA*, pp. 187–193, 2007.

[80] A. Mazumdar, A. Barg and G. Zémor, "Construction of rank modulation codes," in *Proc. IEEE Internatinal Sympos. Inform. Theory*, pp. 834–838, 2011.

[81] A. Mazumdar, A. Barg and G. Zémor, "Parameters of rank modulation codes: Examples," in *Proc. Annual Allerton conference on communication, control and computing (Allerton)*, pp. 13–17, 2011.

[82] A. Mazumdar, R. M. Roth, and P. O. Vontobel, "On linear balancing sets," in *Proc. IEEE Int. Symp. Information Theory*, pp. 2699–2703, 2009.

[83] R. McEliece, D. MacKay, and J. Cheng, "Turbo decoding as an instance of Pearl's belief propagation algorithm," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 140–152, Feb. 1998.

[84] N. Merhav and D. L. Neuhoff, "Variable-to-fixed length codes provide better large deviations performance than fixed-to-variable length codes," *IEEE Trans. Inform. Theory*, vol. 38, pp. 135–140, Jan. 1992.

[85] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit error rate in NAND Flash memories," in *IEEE International Reliability Physics Symposium*, pp. 9–19, 2008.

[86] R. Motwani and P. Pgahavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[87] N. Nisan, "Extracting randomness: How and why. A survey," in *Proc. IEEE conference on Computational Complexity*, pp. 44–58, 1996.

[88] S. Pae and M. C. Loui, "Optimal random number generation from a biased coin," in *Proc. ACM-SIAM Symp. Discrete Algorithms*, pp. 1079–1088, 2005.

[89] A. Pirovano, A. Redaelli, et al., "Reliability study of phase-change nonvolatile memories," *IEEE Transactions on Device and Materials Reliability*, vol. 4, pp. 422–427, 2004.

[90] Y. Peres, "Iterating von Neumann's procedure for extracting random bits," *Ann. Statist.*, vol. 20, pp. 590–597, 1992.

[91] K. C. Pohlmann, *Principles of Digital Audio*, McGraw-Hill/TAB Electronics, 6 edition, 2010.

[92] W. Qian, M. D. Riedel, H. Zhou, and J. Bruck, "Transforming probabilities with combinational logic," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, pp. 1279–1292, 2011.

[93] J. Radhakrishnan and A. Ta-Shma, "Bounds for dispersers, extractors, and depth-two supper-concentrators," *SIAM Journal on Discrete Mathmatics*, 13(1): 2–24, 2000.

[94] A. Rao, "Extractors for a constant number of polynomially small min-entropy independent sources," in *Proc. ACM Symposium on Theory of Computing (STOC)*, 2006.

[95] A. Rao, "Randomness extractors for independent sources and applications," Ph.D. thesis, Department of Computer Science, University of Texas at Austin, 2007.

[96] A. Rao, and D. Zuckerman, "Extractors for three uneven-length sources," in *Proc. Workshop on Randomization and Computation (RANDOM)*, pp. 557–570, 2008.

[97] R. Raz, "Extractors with weak random seeds," in *Proc. Annual ACM Symposium on Theory of Computing (STOC)*, pp. 11–20, 2005.

[98] I. Reidler, Y. Aviad, M. Rosenbluh, and I. Kanter, "Ultrahigh speed random number generation based on a chaotic semiconductor laser," *Phys. Rev. Lett.*, vol. 103, no. 2, pp. 024102-1C024102-4, 2009.

[99] B. Y. Ryabko and E. Matchikina, "Fast and efficient construction of an unbiased random sequence," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1090–1093, 2000.

[100] Y. Saitoh, K. Yamaguchi, and H. Imai, "Some new binary codes correcting asymmetric/unidirectional errors," *IEEE Trans. Inform. Theory*, vol. 36, pp. 645–647, 1990.

[101] P. A. Samuelson, "Constructing an unbiased random sequence," *J. Amer. Statist. Assoc*, pp. 1526–1527, 1968.

[102] M. Santha and U. V. Vazirani, "Generating quasi-random sequences from semi-random sources," *Journal of Computer and Symstem Science*, 33:75–87, 1986.

[103] S. A. Savari and R. G. Gallager, "Generalized Tunstall codes for sources with memory," *IEEE Trans. Inform. Theory*, vol. 43, pp. 658–668, Mar. 1997.

[104] M. Schwartz and I. Tamo, "Optimal permutation anticodes with the infinity norm via permanents of $(0,1)$-matrices," *Journal of Combinatorial Theory*, Series A, vol. 118, pp. 1761–1774, 2011.

[105] R. Shaltiel, "Recent developments in explicit constructions of extractors," in *Current trends in theoretical computer science. The Challenge of the New Century*, vol 1: Algorithms and Complexity, 2004.

[106] C. E. Shannon. "A symbolic analysis of relay and switching circuits," *Trans. AIEE*, 57:713–723, 1938.

[107] C. L. Sheng, "Threshold logic elements used as a probability transformer," *Journal of the ACM*, vol. 12, no. 2, pp. 262–276, April, 1965.

[108] D. Soloveichik, G. Seelig, and E. Winfree, "DNA as a universal substrate for chemical kinetics," *PNAS 107*, pp. 5393–5398, 2010.

[109] Q. Stout and B. Warren, "Tree algorithms for unbiased coin tosssing with a biased coin," *Ann. Probab.*, vol. 12, pp. 212–222, 1984.

[110] L. G. Tallini, B. Bose, "On a new class of error control codes and symmetric functions," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 980–984, 2008.

[111] L. G. Tallini, R. M. Capocelli, and B. Bose, "Design of some new balanced codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 790–802, May 1996.

[112] I. Tamo and M. Schwartz, "Correcting limited-magnitude errors in the rank-modulation scheme," *IEEE Trans. Inform. Theory*, vol. 56, no. 6, pp. 2551–2560, June 2010.

[113] G. Taylor and G. Cox, "Behind Intel's new random-number generator", *IEEE Spectrum*, Sep. 2011.

[114] T. J. Tjalkens and F. M. J. Willems, "A universal variable-to-fixed-length source codes based on Lawrence's algorithm," *IEEE Trans. Inform. Theory*, vol. 38, pp. 247–253, Mar. 1992.

[115] T. J. Tjalkens and F. M. J. Willems, "Variable to fixed-length codes for Markov sources," *IEEE Trans. Inform. Theory*, vol. 33, pp. 246–257, Mar. 1987.

[116] L. Trevisan and S. P. Vadhan, "Extracting randomness from samplable distributions," in *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 32–42, 2000.

[117] B. P. Tunstall, *Synthesis of noiseless compression codes*, Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, Sept. 1967.

[118] A. Uchida, et al., "Fast physical random bit generation with chaotic semiconductor lasers," *Nat. Photon.*, vol. 2, no. 12, pp. 728–732, 2008.

[119] H. van Tilborg and M. Blaum, "On error-correcting balanced codes," *IEEE Trans. Inf. Theory*, vol. 35, no. 5, pp. 1091–1095, Sep. 1989.

[120] M. E. Van Valkenburg, *Network Analysis*, 3rd Edition, Prentice-Hall, Englewood Cliffs, NJ, USA, 1974.

[121] R. R. Varshamov, "A class of codes for asymmetric channels and a problem from the additive theory of numbers," *IEEE Trans. Inform. Theory*, vol. 19, no. 1, pp. 92–95, 1973.

[122] R. R. Varshamov, "Some features of linear codes that correct asymmetric errors" (in Russian), *Doklady Akad. Nauk. SSSR*, vol. 157, no. 3, pp. 546–548, 1964. (Trans: *Soviet Physics-Doklady 9*, pp. 538–540, 1965.)

[123] U. V. Vazirani, "Efficiency consideration in using semi-random sources," in *Proc. ACM Symposium on the Theory of Computing (STOC)*, pp. 160–168, 1987.

[124] U. V. Vazirani and V. V. Vazirani, "Efficient and secure pseudo-random number generation," in *Proc. IEEE Symp. Foundations of Computer Science (FOCS)*, Singer Island, FL, Oct. 24–26, 1984, pp. 458–463.

[125] S. Verdú, "Channel capacity," Ch. 73.5 in the *Electrical Enginnering Handbook*, IEEE and CRC Press, pp. 1671–1678, 1997.

[126] K. Visweswariah, S. R. Kulkarni and S. Verdú, "Source codes as random number generators," *IEEE Trans. Inform. Theory*, vol. 44, no. 2, pp. 462–471, 1998.

[127] K. Visweswariah, S. R. Kulkarni and S. Verdú, "Univerasal variable-to-fixed length source codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 4, pp. 1461–1472, 2001.

[128] J. von Neumann, "Various techniques used in connection with random digits," *Appl. Math. Ser.*, Notes by G.E. Forstyle, Nat. Bur. Stand., vol. 12, pp. 36–38, 1951.

[129] C. Wang, S. R. Kulkarni, and H. V. Poor, "Density evolution for asymmetric memoryless channels," *IEEE Trans. Inform. Theory*, vol. 51, pp. 4216–4236, 2005.

[130] J. H. Weber and K. A. S. Immink, "Knuth's balanced code revisited," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1673–1679, Apr. 2010.

[131] J. Weber, K. S. Immink and H. Ferreira, "Error-correcting balanced Knuth codes," *IEEE Trans. Inform. Theory*, vol. 58, no. 1, pp. 82–89, 2012.

[132] J. H. Weber, C. de Vroedt, and D. E. Boekee, "Bounds and constructions for binary codes of length less than 24 and asymmetric distance less than 6," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1321–1331, 1988.

[133] J. H. Weber, C. de Vroedt, and D. E. Beokee, "New upper bounds on the size of codes correcting asymmetric errors," *IEEE Trans. Inform. Theory*, vol. 33, no. 3, pp. 434–437, 1987.

[134] D. Wilhelm and J. Bruck, "Stochastic switching circuit synthesis," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 1388–1392, 2008.

[135] H. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[136] J. Ziv and A. Lempel, "Compression of individual squences by varaible rate coding," *IEEE Trans. Inform. Theory*, vol. 24, pp. 530–536, 1978.

[137] Z. Zhang and X. Xia, "New lower bounds for binary codes of asymmetric distance two," *IEEE Trans. Inform. Theory*, vol. 38, pp. 1592–1597, 1992.

[138] H. Zhou and J. Bruck, "Efficiently generating random bits from finite state markov chains," *IEEE Trans. Inform. Theory*, vol. 58, pp. 2490–2506, 2012.

[139] H. Zhou and J. Bruck, "Generalizing the Blum-Elias method for generating ran- dom bits from Markov chains," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2010.

[140] H. Zhou and J. Bruck, "Linear extractors for extracting randomness from noisy sources," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011.

[141] H. Zhou and J. Bruck, "On the expressibility of stochastic switching circuits," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2009.

[142] H. Zhou and J. Bruck, "Streaming Algorithms for Optimal Generation of Random Bits," Technical Report, Electrical Engineering, California Institute of Technology, 2012.

[143] H. Zhou and J. Bruck, "Variable-length extractors," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2012.

[144] H. Zhou, H. Chen and J. Bruck, "On the synthesis of stochastic flow networks," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2010.

[145] H. Zhou, A. Jiang and J. Bruck, "Error-correcting schemes with dynamic thresholds in non-volatile memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011.

[146] H. Zhou, A. Jiang and J. Bruck, "Nonuniform codes for correcting asymmetric errors," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011.

[147] H. Zhou, A. Jiang and J. Bruck, "Systematic error-correcting codes for rank modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2012.

[148] L. Zhuo, V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," In *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 63–74, 2005.

[149] D. Zuckerman, "General weak random sources," in *Proc. IEEE Symposium on Foundations of Computer Science*, pp. 534–543, 1990.