UNIVERSITY OF CALIFORNIA SAN DIEGO

Topics in Polar Coding: Structural Properties, Construction, and Decoding

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

in

Electrical Engineering
(Communication Theory and Systems)

by

Hanwen Yao

Committee in charge:

Professor Paul Siegel, Chair
Professor Shachar Lovett
Professor Arya Mazumdar
Professor Alon Orlitsky

2022

The Dissertation of Hanwen Yao is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

Dedicated to the memory of Prof. Alexander Vardy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There are many people who have contributed to my work during my study in UCSD in the past six years. This dissertation would not be possible without them.

First and foremost, I would like to express my deepest gratitude to my two advisors, Prof. Paul Siegel and Prof. Alexander Vardy. I feel very fortunate to have the opportunity to interact with two of the best researchers and educators during my Ph.D. journey. Words are not enough to express my thanks to their guidance, patience and encouragement along the way. I learned from their knowledge, experience and research taste, to which I'll always be indebted. They led me into the door of the world of coding theory in this early stage of my career, and helped me become the researcher I am today.

My sincere gratitude extends to my committee members Prof. Shachar Lovett, Prof. Arya Mazumdar, and Prof. Alon Orlitsky. I'm thankful to their support, and their invaluable recommendations on my research. I would also like to express my special thanks to one of my favorite teachers, Prof. Orlitsky. He accepted me as his teaching assistant in my first year in UCSD. In that period, I was deeply impressed and influenced by his critical thinking, attention to detail, and his humor. I'm also grateful to Prof. Lovett for his enlightening classes on computational complexity and expander graphs, where I enjoyed learning all the fascinating theories.

During my Ph.D. study, I was fortunate to participate in a lot of collaborative projects and work with many amazing co-authors. I feel lucky to have Arman Fazeli as one of my senior colleagues in my early years. I'm extremely grateful to his guidance and tutoring that helped me grow as a researcher. It was a pleasure collaborating with him on all the fruitful projects along this journey. I would like to thank Prof. Han Mao Kiah for accepting me to his lab in NTU in my third summer. His supervision helped me expand my research into new and fascinating directions. I would also like to express my gratitude to Prof. Hessam Mahdavifar. Besides an enjoyable collaboration, his advises and support helped me in an immeasureable way. I am grateful to Dr. Jinfeng Du for his mentoring during my summer internship in Nokia Bell Lab.

His guidance on my research work and many other aspects of my life were extremely valuable. I also want to thank my other collaborators: Johan Chrisnata, Utkarsh Gupta, Bhaskar Gupta, Hengjie Yang, Ethan Liang, Jacob King, Prof. Richard Wesel, and many others. Having the opportunities to collaborate with them was a great honor for me.

I have taken plenty of classes along this journey in UCSD. I would also like to express my sincere gratitude for the teachers I have met. Thank you Prof. Laurence Milstein, Prof. Tara Javidi, Prof. Piya Pal, Prof. Ramamohan Paturi, Prof. Jacques Verstraete, Prof. Sanjoy Dasgupta, Prof. Alireza Salehi Golsefidy, Prof. Young-Han Kim, Prof. Jason Schweinsberg, and Prof. Lutz Warnke. Their classes deepened my knowledge, encouraged me to stay curious, and sharpened my way of thinking towards a clear scientific perspective.

I also wish to thank my loving and caring family: my parents Hong Yao and Hongwen Wen, my grandparents Minfan Yao and Junfang Shang, my aunt Wei Yao and my cousin Weijie Lin. I’m immensely grateful for their unconditional love and endless support in every aspect of my life, which made this trip so much easier.

My thanks also go to all my friends in San Diego: Chen Chen, Junjie Zhu, Junxiong Mao, Enhao Cui, Meng Wei, Manzhi Li, Gege Shang, Yuchen Fang, Haolan Liu, Meihan Li, Yizhou Shan, Zesen Zhang, among many others. It is impossible to list out all the names, but I’m tremendously grateful to their company and support, which makes my life during this journey so much more colorful and memorable. I wish them nothing but the best. My special thanks go to Huili Chen. I’m deeply indebted to her continuously and patiently support for me in almost every stage of the work on this dissertation. She has been my inspiration and motivation along this way, and will be for the rest of my life.

Lastly, I would like to cherish the memory of Alexander Vardy and dedicate him this work. I feel fortunate to have the opportunity to work under his supervision. His door was always open for inspiring discussions and advices. His guidance will be remembered with me forever.

This dissertation was supported in part by the United States National Science Foundation (NSF) under Grants CCF-1405119 and Grant CCF-1719139.

Chapter 2, in part, has been published at 2021 IEEE International Symposium on Information Theory (ISIT) and appeared as: Hanwen Yao, Arman Fazeli, and Alexander Vardy "A Deterministic Algorithm for Computing the Weight Distribution of Polar Codes" [YFV21a]. The dissertation author was the primary author of this conference paper.

Chapter 3, in part, has been published at 2019 IEEE International Symposium on Information Theory (ISIT) and appeared as: Hanwen Yao, Arman Fazeli, and Alexander Vardy "Explicit Polar Codes with Small Scaling Exponent" [YFV19]. The dissertation author was the primary author of this conference paper.

Chapter 4, in part, has been published at 2021 IEEE Globecom Workshops (GC Wkshps) and appeared as: Bhaskar Gupta, Hanwen Yao, Arman Fazeli,and Alexander Vardy "Polar List Decoding for Large Polarization Kernels" [GYFV21]. The dissertation author was the primary author of this conference paper. Bhaskar Gupta contributed to the algorithm design of this work.

Chapter 5, in part, has been published at Entropy 2021 and appeared as: Hanwen Yao, Arman Fazeli, and Alexander Vardy "List Decoding of Arıkan's PAC Codes" [YFV21b]. The dissertation author was the primary author of this journal paper.

Chapter 6, in part, has been published at 2022 IEEE International Symposium on Information Theory (ISIT) and appeared as: Hanwen Yao, Jinfeng Du, and Alexander Vardy "Polar Coded Modulation via Hybrid Bit Labeling" [YDV22]. The dissertation author was the primary author of this conference paper.

VITA

2016	Bachelor of Science in Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
2018	Master of Science in Electrical and Computer Engineering (Communication Theory and Systems), University of California San Diego
2022	Doctor of Philosophy in Electrical and Computer Engineering (Communication Theory and Systems), University of California San Diego

PUBLICATIONS

H. Yao, A. Fazeli and A. Vardy, “List decoding of Arıkan’s PAC codes,” Entropy, 2021.

J. Chrisnata, H.M. Kiah, S. Rao, A. Vardy, E. Yaakobi and **H. Yao**, “On the Number of Distinct k-Decks: Enumeration and Bounds”, Advances in Mathematics of Communications (AMC), 2021.

H. Yao, A. Fazeli and A. Vardy, “A Deterministic Algorithm for Computing the Weight Distribution of Polar Codes,” submitted to IEEE Transaction on Information Theory.

H. Yao, J. Du and A. Vardy, “Polar Coded Modulation via Hybrid Bit Labeling,” 2022 IEEE International Symposium on Information Theory (ISIT).

B. Gupta, **H. Yao**, A. Fazeli and A. Vardy, “Polar List Decoding for Large Polarization Kernels,” 2021 IEEE Global Communications Conference (GLOBECOM).

H. Yao, A. Fazeli and A. Vardy, “A Deterministic Algorithm for Computing the Weight Distribution of Polar Codes,” 2021 IEEE International Symposium on Information Theory (ISIT).

H. Yao, H. Mahdavifar, A. Fazeli and A. Vardy, “Channel Combining for Nonstationary Polarization on Erasure Channels,” 2021 IEEE International Symposium on Information Theory (ISIT).

A. Fazeli, A. Vardy and **H. Yao**, “List Decoding of Polar Codes: How Large Should the List Be to Achieve ML Decoding?,” 2021 IEEE International Symposium on Information Theory (ISIT).

H.M. Kiah, A. Vardy and **H. Yao**, “Efficient Bee Identification,” 2021 IEEE International Symposium on Information Theory (ISIT).

H. Yao, A. Fazeli and A. Vardy, “List decoding of Arıkan’s PAC codes,” 2020 IEEE International Symposium on Information Theory (ISIT).

A. Fazeli, A. Vardy and **H. Yao**, “Hardness of Successive-Cancellation Decoding of Linear Codes,” 2020 IEEE International Symposium on Information Theory (ISIT).

U. Gupta, H.M. Kiah, A. Vardy and **H. Yao**, “Polar Codes with Balanced Codewords,” 2020 IEEE International Symposium on Information Theory (ISIT).

H. Yao, A. Fazeli and A. Vardy, “Explicit Polar Codes with Small Scaling Exponent,” 2019 IEEE International Symposium on Information Theory (ISIT).

A. Fazeli, A. Vardy and **H. Yao**, “Convolutional Decoding of Polar Codes,” 2019 IEEE International Symposium on Information Theory (ISIT).

J. Chrisnata, H.M. Kiah, S. Rao, A. Vardy, E. Yaakobi and **H. Yao**, “On the Number of Distinct k-Decks: Enumeration and Bounds”, The 19th International Symposium on Communications and Information Technologies (ISCIT), 2019.

H. Yang, E. Liang, **H. Yao**, A. Vardy, D. Divsalar and R.D. Wesel, “A List-Decoding Approach to Low-Complexity Soft Maximum-Likelihood Decoding of Cyclic Codes,” 2019 IEEE Global Communications Conference (GLOBECOM).

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in Communication Theory and Systems

Advisor: Paul Siegel, Alexander Vardy

ABSTRACT OF THE DISSERTATION

Topics in Polar Coding: Structural Properties, Construction, and Decoding

by

Hanwen Yao

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

University of California San Diego, 2022

Professor Paul Siegel, Chair

The discovery of polar codes has been widely acknowledged as one of the most original and profound breakthroughs in coding theory in the recent two decades. Polar codes form the first explicit family of codes that provably achieves Shannon's capacities with efficient encoding and decoding for a wide range of channels. This solves one of the most fundamental problems in coding theory. At the beginning of its invention, polar code is more recognized as an intriguing theoretical topic due its mediocre performance at moderate block lengths. Later, with the invention of the list decoding algorithm and various other techniques, polar codes now show competitive, and in some cases, better performance as compared with turbo and LDPC codes. Due to this and other considerations, the 3rd Generation Partnership Project (3GPP) has selected polar codes for control and physical broadcast channels in the enhanced mobile broadband (eMBB) mode and the ultra-reliable low latency communications (URLLC) mode of the fifth generation (5G) wireless communications standard.

In this dissertation, we propose new theories on a wide range of topics in polar coding, including structural properties, construction methods, and decoding algorithms.

We begin by looking into the weight distribution of polar codes. As an important characteristic for an error correction code, weight distribution directly gives us estimations on the maximum-likelihood decoding performance of the code. In this dissertation, we present a deterministic algorithm for computing the entire weight distribution of polar codes. We first derive an efficient procedure to compute the weight distribution of *polar cosets*, and then show that any polar code can be represented as a disjoint union of such polar cosets. We further study the algebraic properties of polar codes as decreasing monomial codes to bound the complexity of our approach. Moreover, we show that this complexity can be drastically reduced using the automorphism group of decreasing monomial codes.

Next, we dive into the topic of large kernel polar codes. It has been shown that polar codes achieve capacity at a rather slow *speed*, where this *speed* can be measured by a parameter called scaling exponent. One way to improve the scaling exponent of polar codes, is by replacing their conventional 2×2 kernel with a larger polarization kernel. In this dissertation, we propose theories and a construction approach for a special type of large polarization kernels to construct polar codes with better scaling exponents. Our construction method gives us the first explicit family of codes with scaling exponent provably under 3. However, large kernel polar codes are known for their high decoding complexity. In that respect, we also propose a new decoding algorithm that can efficiently perform successive cancellation decoding for large kernel polar codes.

Moving on to the decoding algorithms, we focus ourselves on a new family of codes called *PAC codes*, recently introduced by Arıkan, that combines polar codes with convolutional precoding. At short block lengths such as 128, PAC codes show better performance under sequential decoding compared with conventional polar codes with CRC precoding. In this dissertation, we first show that we can achieve the same superior performance of PAC codes using list decoding with relatively large list sizes. Then we carry out a qualitative complexity comparison between sequential decoding and list decoding for PAC codes.

Lastly, we look into the subject of polar coded modulation. Bit-interleaved coded modulation (BICM) and multilevel coded modulation (MLC) are two ways commonly used to combine polar codes with high order modulation. In this dissertation, we propose a new hybrid polar coded modulation scheme that lies between BICM and MLC. For high order modulation, our hybrid scheme has a latency advantage compared with MLC. And by simulation we show that our hybrid scheme also achieves a considerable performance gain compared with BICM.

Chapter 1

Introduction

1.1 Background on Error Correction Codes

In 1984, Shannon [Sha48] established the fundamental limits of any communication system using the following mathematical analog shown in Figure 1.1.

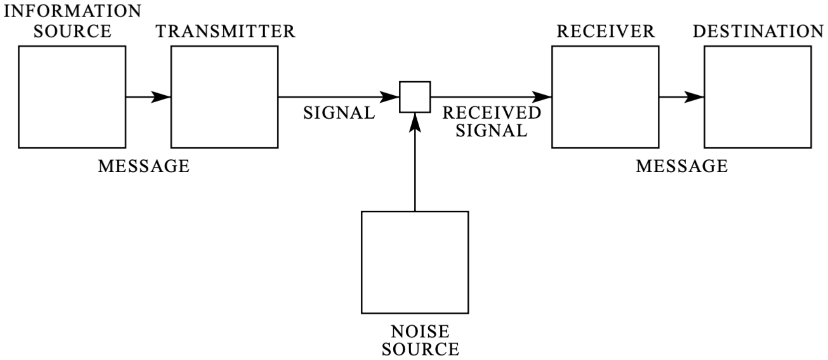

Figure 1.1. Schematic diagram of a general communication system [Sha48].

This mathematical model consists of five parts: 1) An information source that produces the message to be transmitted; 2) A transmitter that encodes the message into signals suitable for transmission; 3) An unreliable channel as the medium for signal transmission, where the input signals may suffer from noise distortion; 4) A receiver that takes the noise-distorted signals from the channel, and assigns a guess to the original message; 5) A destination where the message intends to arrive.

In this model, the communication channel $W : \mathcal{X} \to \mathcal{Y}$ is described with the input alphabet $\mathcal{X}$, the output alphabet $\mathcal{Y}$, and the collection of transition probabilities $P(y|x)$ of observing y as the channel output given the channel input x, for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

To protect the messages against noise, the transmitter typically encodes the messages with redundant information in the form of error correction codes. An (M,n) code for the channel consists of the following: 1) An index set $\{1,2,\cdots,M\}$ representing the M different messages that the information source can produce; 2) An encoding function $\mathcal{E} : \{1,2,\cdots,M\} \to \mathcal{X}^n$ for the transmitter, that takes the message, and yields a codeword consists of n symbols suitable for transmission; 3) A decoding function $\mathcal{D} : \mathcal{Y}^n \to \{1,2,\cdots,M\}$ for the receiver, that takes the error-distorted channel outputs, and assigns a guess to the original message. The rate of the code is defined to be

$$R = \frac{\log_{|\mathcal{X}|} M}{n},$$

and it measures the relative proportion out of those n channel uses that are non-redundant.

Let X and Y denote the random variables representing the channel input and the channel output, with the transition probabilities $P(y|x)$ inherited from channel W. The maximal probability of error for an error correction code with encoding function $\mathcal{E}$ and decoding function $\mathcal{D}$ can be defined as

$$P_e^{(n)} = \max_{m \in \{1,2,\cdots,M\}} \Pr\left(\mathcal{D}(Y^n) \neq m \mid X^n = \mathcal{E}(m)\right).$$

A code rate R is called *achievable* if there exists an error correction code with this code rate, that has asymptotically diminishing maximal probability of error $P_e^{(n)}$.

In Shannon's original 1946 paper [Sha48], he proves that the channel capacity C, defined as

$$C = \sup_X I(X;Y),$$

gives us the highest code rate that is achievable. Here, $I(X;Y)$ denotes the mutual information between the two random variables X and Y.

Theorem 1 (Channel coding theorem [Sha48])**.** *For a noisy channel W, all rates below its capacity C are achievable. Specifically, for every rate $R < C$, there exists a sequence of $(|\mathcal{X}|^{nR}, n)$ codes with maximum probability of error $P_e^{(n)} \to 0$. Conversely, any sequence of $(|\mathcal{X}|^{nR}, n)$ codes with $P_e^{(n)} \to 0$ must have $R \leqslant C$.*

If we want to fully recover the transmitted message with diminishing error probability, Shannon's channel coding theory provides us the limit of the achievable code rates. However, Shannon's proof for the theorem is based on a probabilistic method. It does not give us an explicit code construction that achieves this capacity. Ever since the publication of Shannon's ground-breaking paper [Sha48] in 1984, the holy grail of the coding theory society was to find explicit error correction codes that achieved Shannon's capacity. This problem remained unsolved for almost half a decade, until Erdal Arıkan presented the construction for polar codes in his seminal paper [Arı09] in 2009.

1.2 A Brief Review on Polar Codes

The invention of polar codes has been widely acknowledged as one of the most original and profound developments in coding theory in the recent two decades. Polar codes, introduced by Arıkan in [Arı09], form the first explicit family of codes that provably achieves Shannon's capacity for a wide range of channels with polynomial encoding and decoding complexity. The main idea for the construction of polar codes is based on a phenomenon called *channel polarization*.

Denote a binary memoryless symmetric (BMS) channel $W : \mathcal{X} \to \mathcal{Y}$ with input alphabet $\mathcal{X} = \{0, 1\}$ and output alphabet $\mathcal{Y}$. For a length-2 binary message $\boldsymbol{U}_2 = (U_0, U_1)$ where U_0 and U_1 follow independent Bernoulli distribution $\mathrm{Ber}(p)$ with $p = 1/2$, the following scheme is considered. First, a 2×2 matrix K_2 is used to perform a linear transformation for $\boldsymbol{U}_2$ as

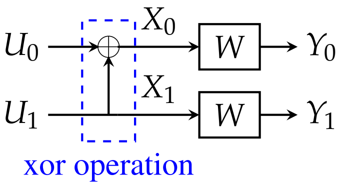

Figure 1.2. Channel polarization in a length-2 polar code.

$\boldsymbol{X}_2 = \boldsymbol{U}_2 \cdot K_2$, where

$$K_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{1.1}$$

Then, the binary vector $\boldsymbol{X}_2 = (X_0, X_1)$ is transmitted using two independent copies of W. The channel outputs are denoted as $\boldsymbol{Y}_2 = (Y_0, Y_1)$. The linear transformation by K_2 can be viewed as an xor operation on the two message bits U_0 and U_1, as illustrated in Figure 1.2.

On the decoder side, bits U_0 and U_1 are decoded sequentially. In the first stage, U_0 is decoded with the knowledge of $\boldsymbol{Y}_2$. Here, the channel "seen" by the first bit U_0 is a new BMS channel $W^- : \mathcal{X} \to \mathcal{Y}^2$ shown on the left of Figure 1.3. In the second stage, U_1 is decoded with the knowledge of both U_0 and $\boldsymbol{Y}_2$, assuming that U_0 has been recovered correctly. This is equivalent to another new BMS channel $W^+ : \mathcal{X} \to \mathcal{X} \times \mathcal{Y}^2$ shown on the right of Figure 1.3.

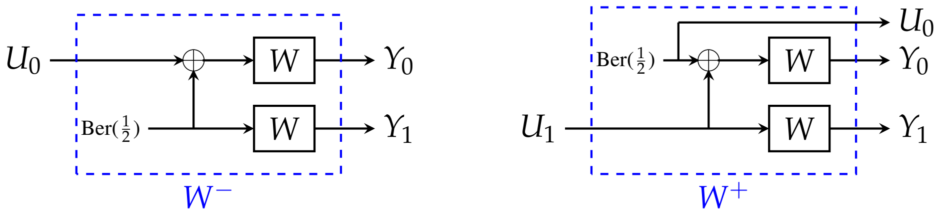

Figure 1.3. Two polarized channels W^- and W^+

In this scheme, by the xor operation on U_0 and U_1, two copies of W are effectively transformed into two new BMS channels W^- and W^+. While the sum of their capacities remains unchanged:

$$I(W^-) + I(W^+) = 2I(W),$$

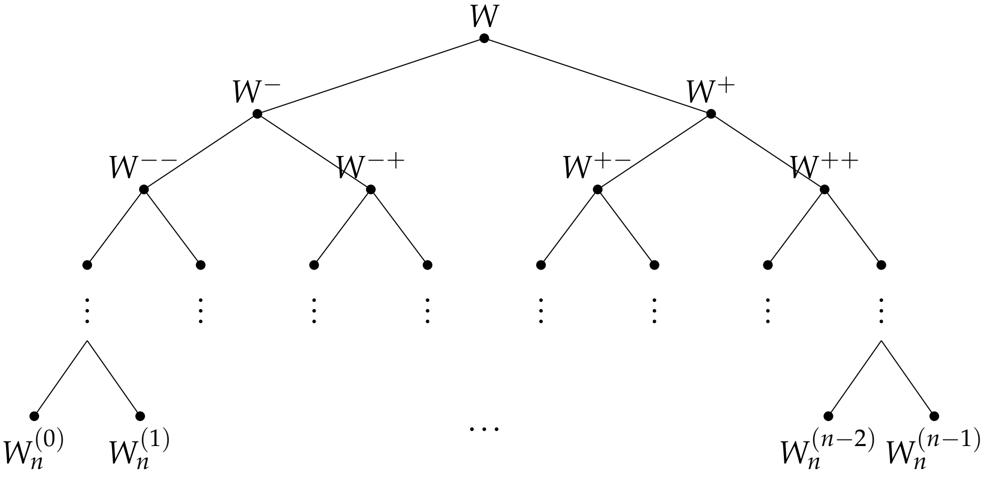

Figure 1.4. Binary tree for the channel polarization process.

it can be shown that [Arı09]:

$$I(W^-) \leqslant I(W) \leqslant I(W^+).$$

Thus one obtains a worse channel W^-, and a better channel W^+. This phenomenon is called *channel polarization*. The same xor operation can be applied to W^- to get two new channels W^{--} and W^{-+}, and applied to W^+ to get another two channels W^{+-} and W^{++}. By repetitively applying this xor operation following the binary tree shown in Figure 1.4, at depth m, a collection of $n = 2^m$ channels can be generated, denoted as $W_n^{(0)}, W_n^{(1)}, \cdots, W_n^{(n-1)}$. The key observation of [Arı09] is that, as n goes to infinity, those n channels will *polarize* in the sense that, almost all of them will have capacities either arbitrarily close to 0, or arbitrarily close to 1. For a small $\delta \in (0,1)$, if we call a channel $W_n^{(i)}$ *δ-good* if $I(W_n^{(i)}) > 1 - \delta$, and *$\delta$-bad* if $I(W_n^{(i)}) < \delta$, then the polarization theorem [Arı09] states the following

Theorem 2 (Polarization theorem [Arı09]). *For every $\delta \in (0,1)$, almost all n channels at depth $\log_2(n)$ of the binary tree in Figure 1.4 become either δ-good or δ-bad, as $n \to \infty$. In fact, since*

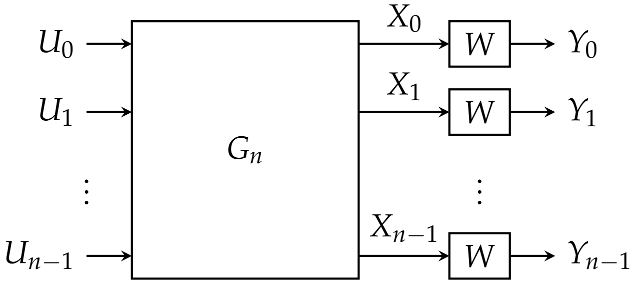

Figure 1.5. The scheme of a length-n polar code.

the sum of the capacities is preserved:

$$I(W_n^{(0)}) + I(W_n^{(1)}) + \ldots + I(W_n^{(n-1)}) = nI(W),$$

as $n \to \infty$, the fraction of δ-good channels approaches the capacity $I(W)$ of the underlying channel W, while the fraction of δ-bad channels approaches $1 - I(W)$.

For a length-n polar code, the combination of all the xor operations can be summarized using an $n \times n$ matrix G_n called the *polarization matrix*. This polarization matrix has the form $G_n = B_n \cdot K_2^{\otimes m}$, where B_n is the bit-reversal permutation matrix, and $K_2^{\otimes m}$ is the m-th Kronecker power of the K_2 in (1.1). Let $\boldsymbol{U} = (U_0, U_1, \cdots, U_{n-1})$ denotes a length-n binary message, where all U_i's follow independent Bernoulli distribution $\text{Ber}(p)$ with $p = 1/2$. Similar to the length-2 polar code, the following scheme is considered. On the transmitter side, the polarization matrix G_n is used to perform a linear transformation for $\boldsymbol{U}$ as $\boldsymbol{X} = \boldsymbol{U} \cdot G_n$. Then n independent copies of the underlying channel W are used to transmit the symbols in vector $\boldsymbol{X}$. The channel outputs are denoted as $\boldsymbol{Y} = (Y_0, Y_1, \ldots, Y_{n-1})$. This scheme is illustrated in Figure 1.5.

On the receiver side, the message bits are decoded one bit at a time following the order $U_0, U_1, \ldots, U_{n-1}$, assuming all previous bits have been decoded correctly. Specifically, at the i-th stage, U_i is decoded with the knowledge of both $U_0, U_1, \ldots, U_{i-1}$ and $\boldsymbol{Y}$. Thus the channel "seen" by the message bit U_i is a new channel $W_n^{(i)} : \mathcal{X} \to \mathcal{X}^i \times \mathcal{Y}^n$ illustrated in Figure 1.6. It is commonly referred as the i-th *bit channel*, and it is exactly the i-th channel at depth $m = \log_2(n)$

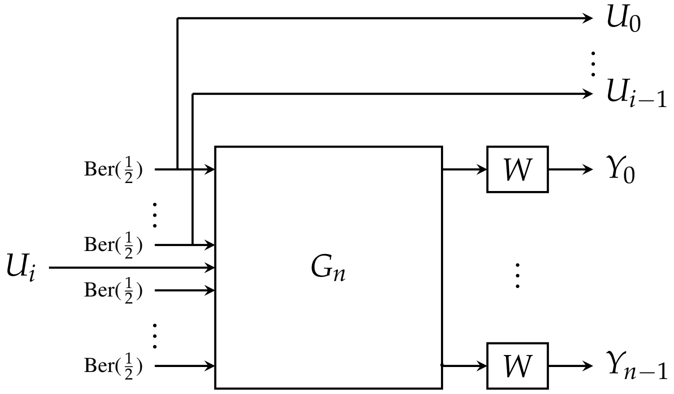

Figure 1.6. The i-th bit channel $W_n^{(i)}$.

of the binary tree in Figure 1.4. If the underlying W is a BMS channel, $W_n^{(i)}$ is also a BMS channel. Let $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\boldsymbol{y} = (y_0, y_1, \cdots, y_{n-1})$ denote the realizations of $\boldsymbol{U}$ and $\boldsymbol{Y}$, then $W_n^{(i)}$ has transition probability:

$$W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_{i-1} | u_i) = \frac{1}{2^{n-1}} \sum_{\boldsymbol{u}' \in \{0,1\}^{n-i-1}} W^n(\boldsymbol{u} | (\boldsymbol{u}_{i-1}, u_i, \boldsymbol{u}')G_n), \tag{1.2}$$

where $\boldsymbol{u}_{i-1} = (u_0, u_1, \ldots, u_{i-1})$ denotes the realization of the first i bits of $\boldsymbol{U}$, W^n denotes n independent uses of the underlying channel W, and $(\boldsymbol{u}_{i-1}, u_i, \boldsymbol{u}')$ represents the concatenation of $\boldsymbol{u}_{i-1}$, u_i, and $\boldsymbol{u}'$.

Following this scheme (Figure 1.5), the polarization theorem (Theorem 2) naturally leads to the construction of capacity-achieving polar codes. Specifically, for an (n,k) polar code, the inputs to the k bit channels in $W_n^{(0)}, W_n^{(1)}, \cdots, W_n^{(n-1)}$ with the highest channel capacities are selected to carry the message bits, while the inputs to rest of the $n-k$ bit channels are frozen to zeros. The size-k index set for the k best bit channels is denoted as $\mathcal{A} \subseteq \{0, 1, \ldots, n-1\}$.

On the receiver side, the bits in the message vector $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ are decoded sequentially following their orders. Let $\widehat{u}_i$ denotes the decoder's estimated value for u_i. At the i-th decoding stage, if $i \in \mathcal{A}$, naturally the decoder would set $\widehat{u}_i = 0$. If $i \notin \mathcal{A}$, following the channel model for W_i, the decoder should theoretically estimates u_i based on both

$\boldsymbol{u}_{i-1} = (u_0, u_1, \ldots, u_{i-1})$ and the channel output $\boldsymbol{y}$. A beautiful idea proposed by Arıkan was substituting the real values for $\boldsymbol{u}_{i-1}$ by the previously estimated $\widehat{\boldsymbol{u}}_{i-1} = (\widehat{u}_0, \widehat{u}_1, \ldots, \widehat{u}_{i-1})$. This substitution has no effect on the overall frame error probability for polar codes, and it converts this hypothetical decoding process into a real one. Thus if $i \notin \mathcal{A}$, the decoder estimates u_i as

$$\widehat{u}_i = \begin{cases} 0 & \text{if } W_n^{(i)}(\boldsymbol{y}, \widehat{\boldsymbol{u}}_{i-1}|0) \geqslant W_n^{(i)}(\boldsymbol{y}, \widehat{\boldsymbol{u}}_{i-1}|1), \\ 1 & \text{if } W_n^{(i)}(\boldsymbol{y}, \widehat{\boldsymbol{u}}_{i-1}|0) < W_n^{(i)}(\boldsymbol{y}, \widehat{\boldsymbol{u}}_{i-1}|1). \end{cases} \tag{1.3}$$

This is called *successive-cancellation decoding*. Arıkan further showed that due to the Kronecker product structure (FFT-like) for the polar transform matrix G, the probabilities $W_n^{(i)}(\boldsymbol{y}, \widehat{\boldsymbol{u}}_{i-1}|0)$ and $W_n^{(i)}(\boldsymbol{y}, \widehat{\boldsymbol{u}}_{i-1}|1)$ can be calculated recursively, resulting in an overall complexity $O(n \log n)$ and latency $O(n)$ for this decoding process. Later, many techniques were proposed that allows efficient implementation of this decoder [AYK11, SGV^{+}14a, HA17].

1.3 Dissertation Overview

After briefly reviewing the basic model of polar codes, we now give an overview on the topics covered by this dissertation.

In Chapter 2, we present a deterministic algorithm for computing the entire weight distribution of polar codes. As the first step, we derive an efficient recursive procedure to compute the weight distribution that arises in successive cancellation decoding of polar codes along any decoding path. This solves the open problem recently posed by Polyanskaya, Davletshin, and Polyanskii. Using this recursive procedure, at code length n, we can compute the entire weight distribution of certain *polar cosets* in time $O(n^2)$. We show that any polar code can be represented as a disjoint union of such polar cosets; moreover, this representation extends to polar codes with dynamically frozen bits. This implies that our methods can be also used to compute the weight distribution of general linear codes. However, the number of polar cosets in such representation scales exponentially with a parameter introduced herein, which we call the *mixing factor*. To upper bound the complexity of our algorithm, we identify polar codes as decreasing monomial codes, and study the range of their mixing factors. We prove that among all decreasing monomial codes with rates at most 1/2, self-dual Reed-Muller codes have the largest mixing factors. To further reduce the exponential complexity of our algorithm, we make use of the fact that, as decreasing monomial codes, polar codes have a large automorphism group, which includes the lower-triangular affine group $\mathrm{LTA}(m,2)$. We prove that $\mathrm{LTA}(m,2)$ acts transitively on certain subsets of the codes, thereby drastically reducing the number of polar cosets we need to evaluate. This complexity reduction makes it possible to compute the weight distribution of *any polar code* of length up to $n = 128$.

In Chapter 3, we dive into the topic of large kernel polar codes. In finite-length analysis, the scaling between code length and the gap to capacity is usually measured in terms of the *scaling exponent* μ. It is well known that the optimal scaling exponent, achieved by random binary codes, is $\mu = 2$. It is also well known that the scaling exponent of conventional polar codes on the binary erasure channel (BEC) is $\mu = 3.627$, which falls far short of the optimal value. On the other hand, it was recently shown that polar codes derived from $\ell \times \ell$ *binary polarization kernels* approach the optimal scaling exponent $\mu = 2$ on the BEC as $\ell \to \infty$, with high probability over a random choice of the kernel.

Herein, we focus on explicit constructions of $\ell \times \ell$ binary kernels with small scaling exponent for $\ell \leqslant 64$. In particular, we exhibit a sequence of binary linear codes that approaches capacity on the BEC with quasi-linear complexity and scaling exponent $\mu < 3$. To the best of our knowledge, such a sequence of codes was not previously known to exist. The principal challenges in establishing our results are twofold: how to construct such kernels and how to evaluate their scaling exponent. We begin with the fact that a given $\ell \times \ell$ kernel K_ℓ transforms ℓ copies of the underlying channels W into ℓ bit-channels $W_1, W_2, \ldots, W_\ell$. Notably, if W is a BEC with erasure probability z, then $W_1, W_2, \cdots, W_\ell$ are all BECs as well. The erasure probabilities of $W_1, W_2, \ldots, W_\ell$ are polynomials in z with integer coefficients and degree at most ℓ. We refer to the corresponding set of polynomials $\{p_1(z), p_2(z), \cdots, p_\ell(z)\}$ as the *polarization behavior* of K_ℓ; the scaling exponent of K_ℓ is completely determined by its polarization behavior.

For kernel construction, We first introduce a class of *self-dual* binary kernels and prove that their polarization behavior satisfies a strong symmetry property. This reduces the problem of constructing K_ℓ to that of producing a certain nested chain of only $\ell/2$ self-orthogonal codes. We use nested cyclic codes, whose distance is as high as possible subject to the orthogonality constraint, to construct the kernels K_{32} and K_{64}. In order to evaluate the polarization behavior of K_{32} and K_{64}, a proper trellis representations (which may be of independent interest) is proposed. Using the resulting trellises, we show that $\mu(K_{32}) = 3.122$ and explicitly compute over half of the polarization-behavior coefficients for K_{64}, at which point the complexity becomes prohibitive. To complete the computation, we introduce a Monte-Carlo interpolation method, which produces the estimate $\mu(K_{64}) \simeq 2.87$. We augment this estimate with a rigorous proof that $\mu(K_{64}) < 2.97$.

In Chapter 4, we continue our discussion on large kernel polar codes, and propose a decoding algorithm that can perform efficient successive cancellation decoding for large kernels. Polar codes constructed with large polarization kernels shown better scaling properties compared with conventional polar codes. However, straightforward decoding for large kernel polar codes introduces a complexity coefficient that is exponential to the kernel sizes, which makes such codes generally believed to be impractical. In Chapter 4, we present a new method that decodes large kernel polar codes with a complexity coefficient that is polynomial to the kernel sizes. This could facilitate the implementation of large kernel polar codes for practical use in the future.

Successive cancellation decoding for large kernel polar codes requires calculation on the probabilities of its bit channels. Similar to conventional polar codes, those bit channels follow a recursive relation, which make this calculation boil down to computing the probabilities for bit channels of a single polarization kernel. This kernel-level computation can be shown equivalent to soft-output maximum-likelihood (ML) decoding on a single bit of a linear block code. In our proposed method, we first use linear operations to represent the considered linear block code as a polar code with dynamically frozen bits, and then use a modified polar list decoder to get an approximate value on the soft-output of the desired bit. This method is motivated by the observation that at short block lengths, polar list decoding with a large enough list size can well-approximate ML decoding. The proposed low-complexity method allows us to decode polar codes constructed with a 64×64 polarization kernel with scaling exponent $\mu \approx 2.87$ for the first time.

In Chapter 5, we move on to the topic of PAC codes. Polar coding gives rise to the first explicit family of codes that provably achieve capacity with efficient encoding and decoding for a wide range of channels. However, its performance at short blocklengths under standard successive cancellation decoding is far from optimal. A well-known way to improve the performance of polar codes at short blocklengths is CRC precoding followed by successive-cancellation list decoding [TV15]. This approach, along with various refinements thereof, has largely remained the state of the art in polar coding since it was introduced in 2011. Recently, Arıkan presented a new polar coding scheme, which he called polarization-adjusted convolutional (PAC) codes [Arı19]. At short blocklengths, such codes offer a dramatic improvement in performance as compared to CRC-aided list decoding of conventional polar codes. PAC codes are based primarily upon the following main ideas: replacing CRC codes with convolutional precoding (under appropriate rate profiling) and replacing list decoding by sequential decoding. One of our primary goals in Chapter 5 is to answer the following question: is sequential decoding essential for the superior performance of PAC codes? We show that similar performance can be achieved using list decoding when the list size L is moderately large (say, $L \geqslant 128$). List decoding has distinct advantages over sequential decoding in certain scenarios, such as low-SNR regimes or situations where the worst-case complexity/latency is the primary constraint. Another objective is to provide some insights into the remarkable performance of PAC codes. We first observe that both sequential decoding and list decoding of PAC codes closely match ML decoding thereof. We then estimate the number of low weight codewords in PAC codes, and use these estimates to approximate the union bound on their performance. These results indicate that PAC codes are superior to both polar codes and Reed–Muller codes. We also consider random time-varying convolutional precoding for PAC codes, and observe that this scheme achieves the same superior performance with constraint length as low as $\nu = 2$.

Finally, Chapter 6 bring up a new scheme for polar codes on high order modulation. Bit-interleaved coded modulation (BICM) and multilevel coded modulation (MLC) are commonly used to combine polar codes with high order modulation. While BICM benefits from simple design and the separation of coding and modulation, MLC shows better performance under successive-cancellation decoding. In Chapter 6, we propose a hybrid polar coded modulation scheme that lies between BICM and MLC, wherein a fraction of bits are assigned to set-partition (SP) labeling and the remaining bits are assigned for Gray labeling. The SP labeled bits undergo sequential demodulation, using iterative demodulation and polar decoding similar to MLC, whereas the Gray labeled bits are first demodulated in parallel and then sent for decoding similar to BICM. Either polar codes or other channel codes (such as LDPC codes) can be used for the Gray labeled bits. For length 2048 rate 1/2 polar code on 256-QAM, the performance gap between BICM (Gray labeling only) and MLC (SP labeling only) can be almost fully closed by the hybrid scheme. Notably, the hybrid scheme has a significant latency advantage over MLC. These performance gains make the proposed scheme attractive for future communication systems.

Chapter 2

Compute the Weight Distribution of Polar Code

2.1 Introduction

The weight distribution of an error correction code counts the number of codewords in this code of any given Hamming weights. The weight distribution is one of the main characteristic of a code, useful for analysing its performance under maximum-likelihood decoding, and various other decoding algorithms. However, computing the weight distribution of a general linear code is known to be NP-hard [BMVT78]. And there are very few families of codes of which the weight distribution is known. Some families of codes that we do know the weight distributions are Hamming codes, Golay codes, Reed-Solomon codes, and double-error-correcting BCH codes. For Reed-Muller codes, we only know part of its weight distribution for codewords of weight less than 2.5 times the minimum distance [KTA76]. Polar codes, introduced by Arıkan [Arı09], form the first explicit family of codes that provably achieve capacity with efficient encoding and decoding for a wide range of channels. The weight distribution of polar codes is currently not known, and there are no algorithms that can efficiently compute it.

2.1.1 Related Prior Work

To the best of our knowledge, there are no prior results on how to efficiently compute the entire weight distribution of polar codes. For crude estimations, there are probabilistic methods discussed in [VY13] and [ZLP17]. Although we don't know the weight distribution of polar codes, we do know their minimal weight, and the number of codewords of that weight. In the work by Bardet, Dragoi, Otmani and Tillich [BDOT16], they study the automorphism group of polar codes from a polynomial formalism, and provide an explicit formula for the number of codewords of minimal weight. There are also ways to estimate the first few non-zero numbers in the weight distribution of polar codes. In the work by Li, Shen and Tse [LST12], they devise an experiment that evaluates the number of low-weight polar codewords, by transmitting an all-zero codeword in the extremely high SNR regime, and use polar list decoder [TV15] to decode the channel output. But still, with this experiment, only the first few non-zero numbers in the entire weight distribution can be estimated, and this procedure is non-deterministic.

2.1.2 Our Contributions

In this chapter, we present a deterministic algorithm that computes the entire weight distribution of polar codes. We first propose an efficient recursive procedure to compute the weight enumerating function of certain *polar cosets* to be defined later. Those polar cosets arise during the successive cancellation decoding process, and their weight distribution can be used to estimate the error probabilities of the bit channels. In two separate works by Niu, Li, and Wu [NLW19], and by Polyanskaya, Davletshin, and Polyanskii [PDP20], algorithms that compute the weight distribution of these polar cosets along the all-zero decoding path are proposed. However, efficiently computing the weight distribution of polar cosets along an arbitrary decoding path remained an open problem. In this work, we solve this problem by establishing a recursive relation followed by the weight enumerating functions of those polar cosets. Using this recursive relation, we can compute the weight distribution of polar cosets along arbitrary decoding path in time $O(n^2)$. This computation procedure has two applications: analyzing the successive cancellation (SC) decoding performance as explained in [PDP20] and computing the entire weight distribution of polar codes.

To make the connection between the weight enumerating functions of polar cosets and the weight distribution of polar codes, we show that we can represent any polar code as a disjoint union of certain polar cosets. In this way, we can obtain the weight distribution of the entire code by summing up the weight enumerating functions of those polar cosets. This representation extends to polar codes with dynamically frozen bits [TM13], so our method can be applied to polarization-adjusted convolutional (PAC) codes [Arı19], and other pre-transformed polar codes [LZG19]. Since any binary linear codes can be represented as polar codes with dynamically frozen bits [TM13, FVY20], our algorithm applies to general linear codes as well. However, the number of polar cosets in this representation scales exponentially with a code parameter introduced herein, which we call the *mixing factor*. The complexity of our algorithm is largely governed by the mixing factor of polar codes. To bound this mixing factor, and thus give a bound on the complexity of our algorithm, we identify polar codes as *decreasing monomial codes*, introduced in [BDOT16], and prove that self-dual Reed-Muller codes have the largest mixing factor among all decreasing monomial codes with rates at most 1/2.

Representing polar codes as disjoint unions of polar cosets works for polar codes in a general setting, where we can select any subsets of rows in the polar transformation matrix as generators. In a more restricted setting, where we only select the rows whose corresponding bit channels have the smallest Bhattacharyya parameters, polar codes fall into the category of decreasing monomial codes. Decreasing monomial codes, first studied by Bardet, Dragoi, Otmani, and Tillich [BDOT16], have a large automorphism group, which includes the lower triangular affine group $\mathrm{LTA}(m,2)$. We show that using $\mathrm{LTA}(m,2)$, we can largely reduce the complexity of our algorithm. We prove that $\mathrm{LTA}(m,2)$ acts transitively on certain subsets of polar codes, which implies that a lot of polar cosets in our representation share the same weight distribution. It allows us to drastically reduce the number of polar cosets we need to evaluate in our algorithm. This complexity reduction makes it possible to compute the weight distribution of any polar codes up to length 128. In particular, since Reed-Muller codes are also decreasing monomial codes, our complexity reduction applies to Reed-Muller codes as well. This makes it possible for our algorithm to compute the entire weight distribution of Reed-Muller codes for all rates and length up to 128 with reasonable complexity.

2.1.3 Notations

Here we specify some notation conventions we follow in this chapter. All the vectors in this paper are row vectors, unless otherwise specified. We use bold letters like $\boldsymbol{u}$ to denote vectors, and non-bold letters like u_i to denote symbols within that vector. We let the indices for the symbols within vectors start from zero. We use $\boldsymbol{u}_i$ to represent $(u_0, u_1, \cdots, u_i)$, a subvector of $\boldsymbol{u}$ with its first $(i+1)$ symbols. And we denote the concatenation of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ as $(\boldsymbol{u}, \boldsymbol{v})$.

2.2 Polar Cosets

Assuming $n = 2^m$, recall that an (n, k) polar code is a binary linear block code generated by k rows in the polar transformation matrix $G_n = B_n \cdot K_2^{\otimes m}$, where B_n is the bit-reversal permutation matrix, $K_2^{\otimes m}$ is the n-th Kronecker power of K_2, and

$$K_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

We first give the definition for polar cosets.

Definition 1. *For a vector $\boldsymbol{u}_i \in \{0,1\}^{i+1}$ with $0 \leqslant i \leqslant n-1$, we define the* ***polar coset*** *for path $\boldsymbol{u}_i$ as the affine space*

$$C_n(\boldsymbol{u}_i) \overset{\text{def}}{=} \left\{ (\boldsymbol{u}_i, \boldsymbol{u}') G_n \mid \boldsymbol{u}' \in \{0,1\}^{n-i-1} \right\}$$

where $(\boldsymbol{u}_i, \boldsymbol{u}')$ represents the concatenation of $\boldsymbol{u}$ and $\boldsymbol{u}'$, and G_n is the polar transformation matrix.

Example 1. Consider G_8 with its rows denoted by $g_0, g_1, \cdots, g_7$ as shown in Figure 2.1.

$$G_8 = \begin{array}{cc} \left[\begin{array}{cccccccc} 1&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&0&1&0&1&0&1&0\\ 1&1&0&0&0&0&0&0\\ 1&1&0&0&1&1&0&0\\ 1&1&1&1&0&0&0&0\\ 1&1&1&1&1&1&1&1 \end{array}\right] & \begin{array}{l} g_0\\ g_1\\ g_2\\ g_3\\ g_4\\ g_5\\ g_6\\ g_7 \end{array} \end{array}$$

Figure 2.1. Polar transformation matrix G_8

Let $\boldsymbol{u}_4 = (0,1,0,1,0)$, then the polar coset $C_8(\boldsymbol{u}_4)$ is the affine space generated by g_5, g_6, g_7, and shifted by g_1 and g_3:

$$C_8(\boldsymbol{u}_4) = g_1 + g_3 + \text{span}\{g_5, g_6, g_7\}$$

In Figure 2.1, those rows are highlighted in gray and in cyan, respectively.

We remark that using polar cosets, equation (1.2) that defines the bit channel can be equivalently written as

$$W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_{i-1} | u_i) = \frac{1}{2^{n-1}} \sum_{\boldsymbol{c} \in C_n(\boldsymbol{u}_i)} W^n(\boldsymbol{y}|\boldsymbol{c}). \tag{2.1}$$

In this section, we will mainly discuss the weight distribution of polar cosets, which can be described by their weight enumerator functions.

Definition 2. *For a vector $\boldsymbol{u}_i \in \{0,1\}^{i+1}$ with $0 \leqslant i \leqslant n-1$, we denote the weight enumerator function for polar coset $C_n(\boldsymbol{u}_i)$ as the polynomial*

$$A_n(\boldsymbol{u}_i)(X) \overset{\text{def}}{=} \sum_{w=0}^{n} A_w X^w,$$

where A_w is the number of vectors in $C_n(\boldsymbol{u}_i)$ with Hamming weight w.

In prior works, the weight distribution of polar coset $C_n(\boldsymbol{u}_i)$, where $\boldsymbol{u}_i = (0,0,\cdots,0,1)$

is a length-$(i+1)$ all-zero vector with a single 1 at the end, is also referred to as *polar spectrum* in [NLW19], and as the *weight distribution for SC decoding of polar codes* in [PDP20]. It has also been pointed out both in [NLW19, Sec.III.B] and in [PDP20, Sec.II.C] that the weight distribution of such polar coset $C_n(\boldsymbol{u}_i)$ can be used to analyze the error probability of the i-th bit channel $W_n^{(i)}$.

2.3 Computing the Weight Enumerating Function of Polar Cosets

In this section, we present the first key result of this chapter: a recursive procedure that computes the weight enumerating function for arbitrary polar cosets. Recently, both in [NLW19] and in [PDP20], the authors introduce their respective algorithms that compute the weight distribution for polar coset $C_n(\boldsymbol{u}_i)$ with $\boldsymbol{u}_i = (0,0,\cdots,0,0)$ and $\boldsymbol{u}_i = (0,0,\cdots,0,1)$. How to efficiently compute the weight distribution for $C_n(\boldsymbol{u}_i)$ with arbitrary path $\boldsymbol{u}_i$ remains open. Here we present a recursive computation procedure with time complexity $O(n^2)$ that solves this problem.

Let's first establish some notations. We use $\boldsymbol{u}_{\text{even}}$ and $\boldsymbol{u}_{\text{odd}}$ to denote the subvectors $(u_0, u_2, \cdots)$ and $(u_1, u_3, \cdots)$ of $\boldsymbol{u}$ with only even indices and only odd indices, respectively, and we use $\boldsymbol{u}_{i,\text{even}}$ and $\boldsymbol{u}_{i,\text{odd}}$ to denote the subvectors of $\boldsymbol{u}_i$ with only even indices and only odd indices, respectively. We also denote the concatenation of vector $\boldsymbol{u}_i$ with a single bit u_{i+1} as $(\boldsymbol{u}_i, u_{i+1})$.

Our algorithm for polar cosets is based on the following recursive relations.

Theorem 3. *Let $m \geqslant 0$, $n = 2^m$, and $0 \leqslant i \leqslant n-1$, then*

$$A_{2n}(\boldsymbol{u}_{2i})(X) = \sum_{u_{2i+1}\in\{0,1\}} A_n(\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}))(X) \cdot A_n(\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1})(X), \quad (2.2)$$

and

$$A_{2n}(\boldsymbol{u}_{2i+1})(X) = A_n(\boldsymbol{u}_{2i+1,\text{even}} \oplus \boldsymbol{u}_{2i+1,\text{odd}})(X) \cdot A_n(\boldsymbol{u}_{2i+1,\text{odd}})(X). \quad (2.3)$$

Proof. Let $m \geqslant 0$ and $n = 2^m$. For any $\boldsymbol{u} \in \{0,1\}^{2n}$ we have

$$\begin{aligned} \boldsymbol{u} \cdot G_{2n} = (\boldsymbol{u} \cdot B_{2n}) K_2^{\otimes(m+1)} &= (\boldsymbol{u}_{\text{even}} \cdot B_n, \boldsymbol{u}_{\text{odd}} \cdot B_n) \begin{bmatrix} K_2^{\otimes m} & 0 \\ K_2^{\otimes m} & K_2^{\otimes m} \end{bmatrix} \\ &= \Big((\boldsymbol{u}_{\text{even}} \oplus \boldsymbol{u}_{\text{odd}}) \cdot B_n K_2^{\otimes m}, \boldsymbol{u}_{\text{odd}} \cdot B_n K_2^{\otimes m} \Big) \\ &= \Big((\boldsymbol{u}_{\text{even}} \oplus \boldsymbol{u}_{\text{odd}}) \cdot G_n, \boldsymbol{u}_{\text{odd}} \cdot G_n \Big) \end{aligned} \tag{2.4}$$

We first prove equation (2.2). According to Definition 1, we have

$$C_{2n}(\boldsymbol{u}_{2i}) = \Big\{ (\boldsymbol{u}_{2i}, \boldsymbol{u}') G_{2n} \mid \boldsymbol{u}' \in \{0,1\}^{2n-2i-1} \Big\} \tag{2.5}$$

Denote $\boldsymbol{u}'$ as $\boldsymbol{u}' = (u_{2i+1}, \boldsymbol{v})$. According to (2.4), the expression $(\boldsymbol{u}_{2i}, \boldsymbol{u}')G_{2n}$ in (2.5) can be written as

$$(\boldsymbol{u}_{2i}, \boldsymbol{u}') G_{2n} = \Big((\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}), \boldsymbol{v}_{\text{even}} \oplus \boldsymbol{v}_{\text{odd}}) \cdot G_{2n}, (\boldsymbol{u}_{2i,\text{odd}}, \boldsymbol{v}_{\text{odd}}) \cdot G_{2n} \Big) \tag{2.6}$$

When $\boldsymbol{u}'$ ranges over $\{0,1\}^{2n-2i-1}$, both $\boldsymbol{v}_{\text{odd}}$ and $(\boldsymbol{v}_{\text{even}} \oplus \boldsymbol{v}_{\text{odd}})$ will range over $\{0,1\}^{n-i-1}$ separately. So we have

$$C_{2n}(\boldsymbol{u}_{2i}) = \bigcup_{u_{2i+1} \in \{0,1\}} \Big\{ (\boldsymbol{c}_1, \boldsymbol{c}_2) \mid \boldsymbol{c}_1 \in C_n(\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1})), \boldsymbol{c}_2 \in C_n(\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}) \Big\} \tag{2.7}$$

Therefore for each $u_{2i+1} \in \{0,1\}$, $C_{2n}(\boldsymbol{u}_{2i})$ can be expressed as the concatenation of two polar cosets. Since the weight enumerating function of the concatenation equals the product of the two individual weight distribution functions, we have

$$A_{2n}(\boldsymbol{u}_{2i})(X) = \sum_{u_{2i+1} \in \{0,1\}} A_n(\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}))(X) \cdot A_n(\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1})(X).$$

That proves equation (2.2). Next, we prove equation (2.3) in a similar way. According to Definition 1, we have

$$C_{2n}(\boldsymbol{u}_{2i+1}) = \left\{ (\boldsymbol{u}_{2i+1}, \boldsymbol{u}')G_{2n} \mid \boldsymbol{u}' \in \{0,1\}^{2n-2i-2} \right\} \tag{2.8}$$

According to (2.4), the expression $(\boldsymbol{u}_{2i+1}, \boldsymbol{u}')G_{2n}$ in (2.8) can be written as

$$(\boldsymbol{u}_{2i+1}, \boldsymbol{u}')G_{2n} = \Big((\boldsymbol{u}_{2i+1,\text{even}} \oplus \boldsymbol{u}_{2i+1,\text{odd}}, \boldsymbol{u}'_{\text{even}} \oplus \boldsymbol{u}'_{\text{odd}}) \cdot G_{2n}, (\boldsymbol{u}_{2i+1,\text{odd}}, \boldsymbol{u}'_{\text{odd}}) \cdot G_{2n} \Big) \tag{2.9}$$

When $\boldsymbol{u}'$ ranges over $\{0,1\}^{2n-2i-2}$, both $\boldsymbol{u}_{\text{odd}}$ and $(\boldsymbol{u}_{\text{even}} \oplus \boldsymbol{v}_{\text{odd}})$ will range over $\{0,1\}^{n-i-1}$ separately. So we have

$$C_{2n}(\boldsymbol{u}_{2i+1}) = \left\{ (\boldsymbol{c}_1, \boldsymbol{c}_2) \mid \boldsymbol{c}_1 \in C_n(\boldsymbol{u}_{2i+1,\text{even}} \oplus \boldsymbol{u}_{2i+1,\text{odd}}), \boldsymbol{c}_2 \in C_n(\boldsymbol{u}_{2i+1,\text{odd}}) \right\} \tag{2.10}$$

Thus $C_{2n}(\boldsymbol{u}_{2i+1})$ can also be expressed as the concatenation of two polar cosets. This gives us equation (2.3). □

In Theorem 3, equation (2.2) and equation (2.3) can also be written as

$$A_{2n}(\boldsymbol{u}_{2i-1}, u_{2i})(X) = \sum_{u_{2i+1} \in \{0,1\}} A_n(\boldsymbol{u}_{2i-1,\text{even}} \oplus \boldsymbol{u}_{2i-1,\text{odd}}, u_{2i} \oplus u_{2i+1}))(X) \cdot A_n(\boldsymbol{u}_{2i-1,\text{odd}}, u_{2i+1})(X) \tag{2.11}$$

and as

$$A_{2n}(\boldsymbol{u}_{2i}, u_{2i+1})(X) = A_n(\boldsymbol{u}_{2i-1,\text{even}} \oplus \boldsymbol{u}_{2i-1,\text{odd}}, u_{2i} \oplus u_{2i+1})(X) \cdot A_n(\boldsymbol{u}_{2i-1,\text{odd}}, u_{2i+1})(X) \tag{2.12}$$

respectively. In this way, equation (2.11) and equation (2.12) fall into forms similar to the

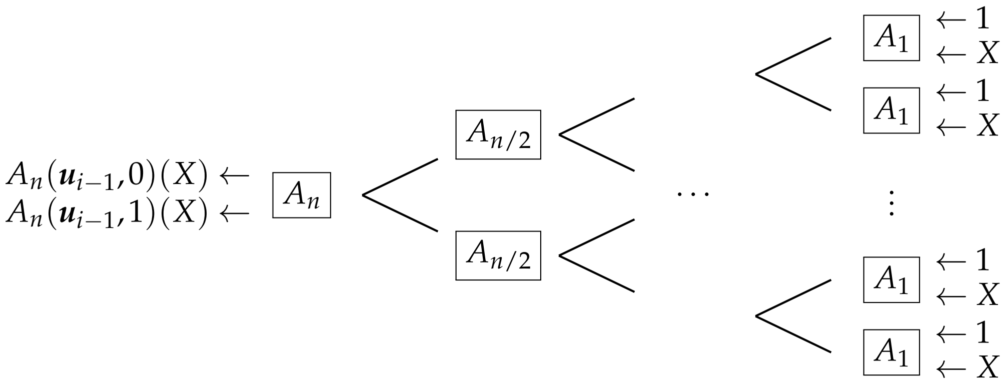

Figure 2.2. The recursive procedure that computes the weight enumerating function for polar cosets

recursive relations for the bit channels [Arı09, Equations (22) and (23)]. Therefore, similar to the recursive procedure that computes the probabilities for the bit channels, we can also compute the weight enumerating functions of polar cosets recursively with the stopping conditions:

$$A_1(0) = 1, \quad A_1(1) = X.$$

This recursive procedure is illustrated in Figure 2.2, and its steps are shown in Algorithm 1.

We make the following remarks for Algorithm 1:

- The object for recursion in Algorithm 1 is a pair of weight enumerating functions $A_n(\boldsymbol{u}_{i-1}, 0)(X)$ and $A_n(\boldsymbol{u}_{i-1}, 1)(X)$.
- If we want to compute the weight distribution for polar coset $C_n(\boldsymbol{u}_i)$, we should run Algorithm 1 with inputs n and $\boldsymbol{u}_{i-1}$, and select one of the two weight enumerating functions from the output corresponding to the desired u_i.

Next, we prove that Algorithm 1 has time complexity $O(n^2)$.

Theorem 4. *Algorithm 1 has time complexity $O(n^2)$.*

Proof. In Algorithm 1, depending on the inputs i and $\boldsymbol{u}_{i-1}$, we have the following three cases for the lines we need to run:

Algorithm 1: CalcA(n,$\boldsymbol{u}_{i-1}$)

```
   Input: block length $n$ and vector $\boldsymbol{u}_{i-1}$
   Output: a pair of polynomials
           $(A_n(\boldsymbol{u}_{i-1},0)(X), A_n(\boldsymbol{u}_{i-1},1)(X))$
1  if $n = 1$ then                                          // Stopping conditions
2      return $(1, X)$
3  else
4      if $i \mod 2 = 0$ then
5          $(f_0, f_1) \leftarrow$ CalcA($n/2, \boldsymbol{u}_{i-1,\text{even}} \oplus \boldsymbol{u}_{i-1,\text{odd}}$)
6          $(g_0, g_1) \leftarrow$ CalcA($n/2, \boldsymbol{u}_{i-1,\text{odd}}$)
7          return $(f_0 g_0 + f_1 g_1, f_0 g_1 + f_1 g_0)$ ;      // Use equation (2.2)
8      else
9          $(f_0, f_1) \leftarrow$ CalcA($n/2, \boldsymbol{u}_{i-2,\text{even}} \oplus \boldsymbol{u}_{i-2,\text{odd}}$)
10         $(g_0, g_1) \leftarrow$ CalcA($n/2, \boldsymbol{u}_{i-2,\text{odd}}$)
11         if $u_{i-1} = 0$ then
12             return $(f_0 g_0, f_1 g_1)$ ;                    // Use equation (2.3)
13         else
14             return $(f_1 g_0, f_0 g_1)$ ;                    // Use equation (2.3)
```

Case 1: When i is even, we run lines 5, 6 and 7.

Case 2: When i is odd and $u_i = 0$, we run lines 9, 10 and 12.

Case 3: When i is odd and $u_i = 1$, we run lines 9, 10 and 14.

First, line 5 is the same as line 9, and line 6 is the same as line 10. Then for line 7, we need to do 4 polynomial multiplications and 2 polynomial additions, while for line 12 or line 14, we only need to do 2 polynomial multiplications. So case 1 has the highest complexity among the above three cases. Thus, henceforth we only consider case 1.

Denote by $T(n)$ the time complexity of Algorithm 1. For the recursive part in the algorithm, line 5 and line 6 both take time $T(n/2)$. For the non-recursive part, in line 7 we need to do 4 polynomial multiplications and 2 polynomial additions. Since f_0, f_1, g_0, g_1 are weight enumerating functions of polar cosets with block length $n/2$, all of them have degrees at most $n/2$. Assume multiplication of two degree-n polynomials takes time $O(n^2)$, and addition of two degree-n polynomials takes time $O(n)$, it follows that

$$T(n) \leqslant 2T(n/2) + 4 \cdot O(n^2/4) + 2 \cdot O(n^2/4),$$

which gives us $T(n) = O(n^2)$. □

We also remark that the time complexity of Algorithm 1 may be improved assuming multiplication of two degree-n polynomials takes time $O(n \log n)$ with the Fast-Fourier Transform.

2.4 Computing the Entire Weight Distribution of Polar Codes

In this section, we present a deterministic algorithm that computes the entire weight distribution of polar codes. We first show that any polar code can be represented as a disjoint union of certain polar cosets. This allows us to obtain the weight distribution of the entire code by summing up the weight distributions of those polar cosets. However, the number of polar cosets in this representation scales exponentially with a new parameter that we introduce herein, called the mixing factor. Next, we show that our approach naturally extends to polar codes with dynamically frozen bits. Since any binary linear code can be represented as a polar code with dynamically frozen bits [TM13, FVY20], our method can be used to compute the weight distribution of any binary linear code, provided that its mixing factor is relatively small.

2.4.1 Representing Polar Codes with Polar Cosets

First, we introduce two new parameters of polar codes that we call the last frozen index and the mixing factor, respectively.

Definition 3. *Consider an (n,k) polar code $\mathbb{C} = \mathbb{C}_n(\mathcal{A})$ specified in terms of its information index set $\mathcal{A}$. With $\mathcal{F} = \{0,1,\ldots,n-1\} \backslash \mathcal{A}$, we define the* ***last frozen index*** *of $\mathbb{C}$ as*

$$\tau(\mathbb{C}) \overset{\text{def}}{=} \max\{\mathcal{F}\},$$

and define the ***mixing factor*** *of* $\mathbb{C}$ *as*

$$\mathrm{MF}(\mathbb{C}) \overset{\text{def}}{=} \big|\{i \in \mathcal{A} \mid i < \tau(\mathbb{C})\}\big|.$$

Loosely speaking, the mixing factor of $\mathbb{C}$ counts the number of information bits that are *mixed-in* among the frozen bits. It is easy to see that $\mathrm{MF}(\mathbb{C})$ can be computed from $\tau(\mathbb{C})$ as follows:

$$\mathrm{MF}(\mathbb{C}) = k - \big|\{i \in \mathcal{A} \mid i > \tau(\mathbb{C})\}\big| = \tau(\mathbb{C}) - (n-k) + 1 \tag{2.13}$$

Starting with an example, we now show that any polar code can be represented as a disjoint union of certain polar cosets.

Example 2. In this example, we denote the (16,11,4) extended Hamming code as $\mathbb{C}_H$. It can be generated by rows in the polar transformation matrix G_{16}. Thus we can view $\mathbb{C}_H$ as a polar code of length 16 with frozen index set $\mathcal{F} = \{0,1,2,4,8\}$. The polar transformation matrix G_{16} is shown in Figure 2.3.

$$\begin{array}{c}
u_0 \\ u_1 \\ u_2 \\ {\color{red}u_3} \\ u_4 \\ {\color{red}u_5} \\ {\color{red}u_6} \\ {\color{red}u_7} \\ u_8 \\ {\color{blue}u_9} \\ {\color{blue}u_{10}} \\ {\color{blue}u_{11}} \\ {\color{blue}u_{12}} \\ {\color{blue}u_{13}} \\ {\color{blue}u_{14}} \\ {\color{blue}u_{15}}
\end{array}
\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}$$

Figure 2.3. Polar transformation matrix G_{16} in Example 2

In Figure 2.3, the information bits of $\mathbb{C}_H$ are highlighted in red and in blue, and the frozen bits are black. We color the information bits that are mixed-in among the frozen bits in red, and color the rest of the information bits in blue. The last frozen bit of $\mathbb{C}_H$ is u_8, so the last frozen index of $\mathbb{C}_H$ is $\tau(\mathbb{C}_H) = 8$. The mixing factor of $\mathbb{C}_H$ counts the number of red bits, so the mixing factor of $\mathbb{C}_H$ is $\mathrm{MF}(\mathbb{C}_H) = 4$.

Consider the polar coset $C_{16}(\boldsymbol{u}_8)$. For any binary vector $\boldsymbol{u}_8$ with $u_0 = u_1 = u_2 = u_4 = u_8 = 0$, and $u_3, u_5, u_6, u_7 \in \{0,1\}$, the polar coset $C_{16}(\boldsymbol{u}_8)$ will be a subset of $\mathbb{C}_H$. In total we have $2^4 = 16$ options to assign the values for u_3, u_5, u_6, u_7. Hence there are 16 such disjoint polar cosets, and the union of them is the entire code $\mathbb{C}_H$:

$$\mathbb{C}_H = \bigcup_{\boldsymbol{u}_8 \in \{0,1\}^9 \colon u_0 = u_1 = u_2 = u_4 = u_8 = 0} C_{16}(\boldsymbol{u}_8)$$

Therefore, the entire weight distribution of $\mathbb{C}_H$ can be obtained by first computing the weight enumerating functions of all those 16 polar cosets, and then taking the sum.

This polar coset representation for general polar codes can be summarized by the following proposition.

Proposition 1. *Let $\mathbb{C}$ be a polar code with frozen index set $\mathcal{F}$, and last frozen index τ. Then $\mathbb{C}$ can be represented as a disjoint union of polar cosets as:*

$$\mathbb{C} = \bigcup_{\boldsymbol{u}_\tau \in \{0,1\}^{\tau+1} \colon u_i = 0 \textit{ for all } i \in \mathcal{F}} C_n(\boldsymbol{u}_\tau)$$

The number of polar cosets in this representation equals $2^{\mathrm{MF}(\mathbb{C})}$.

2.4.2 Representing Polar Codes with Dynamically Frozen Bits

We now show that our polar coset representation in Proposition 1 extends to polar codes with *dynamically frozen bits*. Polar codes with dynamically frozen bits, first introduced in [TM13], are polar codes where each of the frozen bits u_i is not fixed to be zero, but set to be a boolean function (usually, a linear function) of its previous bits as $u_i = f_i(\boldsymbol{u}_{i-1})$. For frozen bits with indices in $\mathcal{F}$, we refer to the set of those boolean functions $\{f_i \mid i \in \mathcal{F}\}$ as the *dynamic constraints* for the code. Examples of polar codes with dynamically frozen bits are polar codes with CRC precoding [TV15], polar subcodes [TM15], polarization-adjusted convolutional (PAC) codes [Arı19], etc. In fact, since any binary linear code can be represented as a polar code with dynamically frozen bits [TM13, FVY20], our representation extends to all binary linear codes, as well.

The concept of last frozen index and mixing factor in Definition 3 naturally extends to polar codes with dynamically frozen bits. We again illustrate our polar coset representation with an example, in which the Hamming code in Example 2 is slightly modified so its frozen bits become dynamically frozen.

Example 3. Denote by $\mathbb{C}'_H$ a $(16,11)$ polar code with frozen index set $\mathcal{F} = \{0,1,2,4,8\}$, where u_0, u_1, u_2 are frozen as 0, and u_4 and u_8 are dynamically frozen as $u_4 = u_3$ and $u_8 = u_5 + u_6$ respectively. We have $\tau(\mathbb{C}'_H) = 8$ and $\mathrm{MF}(\mathbb{C}'_H) = 4$ the same as in Example 2.

$$
\begin{array}{l}
u_0 \\
u_1 \\
u_2 \\
u_3 \\
u_4 = u_3 \\
u_5 \\
u_6 \\
u_7 \\
u_8 = u_5 + u_6 \\
u_9 \\
u_{10} \\
u_{11} \\
u_{12} \\
u_{13} \\
u_{14} \\
u_{15}
\end{array}
\left[\begin{array}{cccccccccccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0 \\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0 \\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 \\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{array}\right]
$$

Figure 2.4. Polar transformation matrix G_{16} in Example 3

Consider the polar coset $C_{16}(\boldsymbol{u}_8)$. For any binary vector $\boldsymbol{u}_8$ with $u_3, u_5, u_6, u_7 \in \{0,1\}$, if we let $u_0 = u_1 = u_2 = 0$, $u_4 = u_3$ and $u_8 = u_5 + u_6$, then the polar coset $C_{16}(\boldsymbol{u}_8)$ will be a subset of $\mathbb{C}'_H$. Thus similar to Example 2, with $2^4 = 16$ options to assign the values for u_3, u_5, u_6

Algorithm 2: Compute the weight enumerating functions of polar code with dynamic freezing

```
Input: block length $n$, frozen index set $\mathcal{F}$, and dynamic constraint $\{f_i \mid i \in \mathcal{F}\}$
Output: weight enumerating function $A_{\mathbb{C}}(X)$ of code $\mathbb{C}$
$\tau \leftarrow \max\{\mathcal{F}\}$
$A_{\mathbb{C}}(X) \leftarrow 0$
for $\boldsymbol{u}_\tau \in \{0,1\}^{\tau+1} : u_i = f_i(\boldsymbol{u}_{i-1})$ for all $i \in \mathcal{F}$ do
    $(f_0, f_1) \leftarrow$ CalcA($n, \boldsymbol{u}_{\tau-1}$) ;          // Use Algorithm 1
    $u_\tau \leftarrow f_\tau(\boldsymbol{u}_{\tau-1})$
    if $u_\tau = 0$ then
        $A_{\mathbb{C}}(X) \leftarrow A_{\mathbb{C}}(X) + f_0$
    else
        $A_{\mathbb{C}}(X) \leftarrow A_{\mathbb{C}}(X) + f_1$
return $A_{\mathbb{C}}(X)$
```

and u_7, $\mathbb{C}'_H$ can be represented as a disjoint union of 16 disjoint polar cosets as

$$\mathbb{C} = \bigcup_{\substack{\boldsymbol{u}_8 \in \{0,1\}^9 :\, u_0 = u_1 = u_2 = 0, u_4 = u_3, \\ u_8 = u_5 + u_6}} C_{16}(\boldsymbol{u}_8)$$

In general, Proposition 1 extends to polar codes with dynamically frozen bits as follows.

Proposition 2. *Let $\mathbb{C}$ be a polar code with dynamically frozen bits, with frozen index set $\mathcal{F}$, last frozen index τ, and the dynamic constraints $\{f_i \mid i \in \mathcal{F}\}$. Then $\mathbb{C}$ can be represented as a disjoint union of polar cosets as:*

$$\mathbb{C} = \bigcup_{\boldsymbol{u}_\tau \in \{0,1\}^{\tau+1} :\, u_i = f_i(\boldsymbol{u}_{i-1}) \text{ for all } i \in \mathcal{F}} C_n(\boldsymbol{u}_\tau) \tag{2.14}$$

The number of polar cosets in this representation equals $2^{\mathrm{MF}(\mathbb{C})}$.

2.4.3 Computing the Entire Weight Distribution

This polar coset representation directly gives us a way to compute the weight distribution of polar codes. We can compute the weight enumerating function of each polar coset in the representation using Algorithm 1, and then take their sum to obtain the weight distribution of the entire code. This procedure is shown in Algorithm 2, in which conventional polar codes are considered as special cases of polar codes with dynamically frozen bits.

For a polar code $\mathbb{C}$, the number of polar cosets in the representation equals 2 to the power of $\mathrm{MF}(\mathbb{C})$. Thus without parallel computation, Algorithm 2 has time complexity $O(2^{\mathrm{MF}(\mathbb{C})}n^2)$. It's clear that this complexity is largely governed by the mixing factor of polar codes. For reference, we list the mixing factor of rate 1/2 polar codes following the 5G standard [3GP18, BCL20] from length 8 to length 1024 in Table 2.1.

Table 2.1. Mixing factor of rate 1/2 polar codes in 5G

code length n	8	16	32	64	128	256	512	1024
mixing factor $\mathrm{MF}(\mathbb{C})$	1	2	9	17	34	73	161	385

2.5 Mixing Factor of Polar Code

In Section 2.3, we present Algorithm 2 that computes the weight distribution by representing any polar code as a disjoint union of polar cosets. The number of polar cosets in this representation equals two to the power of mixing factor of the code. Therefore, it is clear that for a polar code $\mathbb{C}$, the complexity of our approach is largely governed by its mixing factor $\mathrm{MF}(\mathbb{C})$. In a prior work [FVY21], it is shown that for a given polar code $\mathbb{C}$, ML decoding of polar code can be performed with a hybrid list decoder with list size $L = 2^{\mathrm{MF}(\mathbb{C})}$. In that sense, the mixing factor of polar code appears to be a rather fundamental parameter that provides us an exponential upper bound on the complexity of two important operations: computing the weight distribution of the code and performing ML decoding of the code.

In this section, we study the range of mixing factor of polar codes. We remark that our approach in Section 2.4 applies to polar codes in a general setting where: (1) the frozen bits can be dynamically frozen; (2) the information index set can be arbitrary. Hereforth, we focus on conventional polar codes where: (1) the frozen bits are all frozen to zero; (2) for code of dimension k, the information index set $\mathcal{A}$ is chosen such that the bit channels $W_n^{(i)}$'s for $i \in \mathcal{A}$ are the "best" k bit channels. In Arıkan's definition, the k bit channels with the smallest Bhattacharyya parameters are selected. Two alternative criteria for picking the best k bit channels are mutual information and error probability. If we follow either one of these three criteria, polar codes fall into the category of decreasing monomial codes, first introduced by Bardet, Dragoi, Otmani, and Tillich [BDOT16]. For the details of decreasing monomial codes, we refer the readers to [BDOT16].

Here, we first review properties of polar codes as decreasing monomial codes. Then, to upper bound the mixing factor of polar codes, and thus upper bound the complexity of Algorithm 2, we prove that self-dual Reed-Muller codes have the largest mixing factors among all polar codes with rates at most $1/2$.

2.5.1 Properties of Polar Codes as Decreasing Monomial Codes

We start by reviewing the definition of monomial codes. Let $n = 2^m$, and let the polynomial ring given by

$$\mathcal{R}_m = \mathbb{F}[x_0, x_1, \cdots, x_{m-1}]/(x_0^2 - x_0, x_1^2 - x_1, \cdots, x_{m-1}^2 - x_{m-1}).$$

Each polynomial $p \in \mathcal{R}_m$ can be associated with a binary vector in $\mathbb{F}_2^n$ as the evaluation of p in all the binary entries $\boldsymbol{x} = (x_0, \cdots, x_{m-1}) \in \mathbb{F}_2^m$. In other words, polynomial p is associated with $\mathrm{ev}(p) = (p(\boldsymbol{x}))_{\boldsymbol{x} \in \mathbb{F}_2^m}$ where $\mathrm{ev} : \mathcal{R}_m \to \mathbb{F}_2^n$ is a homomorphism from the polynomials to the binary n-tuples. In this work, we specify the order of $\boldsymbol{x}$ in vector $(p(\boldsymbol{x}))_{\boldsymbol{x} \in \mathbb{F}_2^m}$ such that from left to right, the number $\sum_{i=0}^{m-1} z_i 2^i$ is in ascending order from 0 to $2^m - 1$, where the binary vector $(z_0, z_1, \cdots, z_{m-1})$ is defined by:

$$(z_0, z_1, \cdots, z_{m-1}) = (1 - x_{m-1}, 1 - x_{m-2}, \cdots, 1 - x_0)$$

Denote the set of all the monomials in $\mathcal{R}_m$ as

$$\mathcal{M}_m = \left\{ x_0^{b_0} x_1^{b_1} \cdots x_{m-1}^{b_{m-1}} \,\middle|\, (b_0, b_1, \cdots, b_{m-1}) \in \mathbb{F}_2^m \right\}.$$

The monomial codes can be defined as follows.

Definition 4. *Let $n = 2^m$ and $\mathcal{I} \in \mathcal{M}_m$, the monomial code $\mathbb{C}(\mathcal{I})$ generated by $\mathcal{I}$ is the linear space*

$$\mathbb{C}(\mathcal{I}) \overset{\text{def}}{=} \text{span}\{\text{ev}(f) \mid f \in \mathcal{I}\}.$$

Since every row in the polar transformation matrix G_n can be obtained as $\text{ev}(f)$ with some $f \in \mathcal{M}_m$, polar codes can be viewed as monomial codes. For a monomial $f \in \mathcal{M}_m$ given by $f = x_{i_1} x_{i_2} \cdots x_{i_d}$, we write:

$$\begin{aligned} \deg f &= d \\ \text{ind}(f) &= \{i_1, i_2, \ldots, i_d\} \\ [f] &= (a_{m-1}, a_{m-2}, \ldots, a_0) \in \{0,1\}^m \\ [[f]] &= \sum_{i=0}^{m-1} a_i 2^i = \sum_{i=0}^{m-1} (1-b_i) 2^i \end{aligned}$$

where the binary vectors $(a_{m-1}, a_{m-2}, \ldots, a_0)$ and $(b_{m-1}, b_{m-2}, \ldots, b_0)$ are defined by:

$$f = x_0^{1-a_0} x_1^{1-a_1} \cdots x_{m-1}^{1-a_{m-1}} = x_0^{b_0} x_1^{b_1} \cdots x_{m-1}^{b_{m-1}}$$

Following this notation, if we label the rows in the polar transformation matrix G_n with indices from 0 to $n-1$, the evaluation $\text{ev}(f)$ for a monomial $f \in \mathcal{M}_m$ has row index $[[f]]$ in G_n, and $[f]$ contains the digits in the binary expansion of $[[f]]$. When the underlying G_n is clear from the context, we simply refer $[[f]]$ as the ***row index*** for f.

Example 4. Consider the polar transform matrix G_{16}. The monomials in $\mathcal{M}_4$ whose evaluations are rows in G_{16} are shown in Figure 2.5.

$$\begin{array}{cccc}
f & [f] & [\![f]\!] & \\
\begin{array}{c} x_0x_1x_2x_3 \\ x_1x_2x_3 \\ x_0x_2x_3 \\ x_2x_3 \\ x_0x_1x_3 \\ x_1x_3 \\ x_0x_3 \\ x_3 \\ x_0x_1x_2 \\ x_1x_2 \\ x_0x_2 \\ x_2 \\ x_0x_1 \\ x_1 \\ x_0 \\ 1 \end{array} &
\begin{array}{c} (0,0,0,0) \\ (0,0,0,1) \\ (0,0,1,0) \\ (0,0,1,1) \\ (0,1,0,0) \\ (0,1,0,1) \\ (0,1,1,0) \\ (0,1,1,1) \\ (1,0,0,0) \\ (1,0,0,1) \\ (1,0,1,0) \\ (1,0,1,1) \\ (1,1,0,0) \\ (1,1,0,1) \\ (1,1,1,0) \\ (1,1,1,1) \end{array} &
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{array} &
\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0 \\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0 \\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 \\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}
\end{array}$$

Figure 2.5. Polar transformation matrix G_{16} in Example 4

Henceforth, whenever we write a monomial as $f = x_{i_1}x_{i_2}\cdots x_{i_d}$ we assume that $i_1 < i_2 < \cdots < i_d$, unless stated otherwise. A partial order on the monomials in $\mathcal{M}_m$ is introduced in [BDOT16] as follows:

Definition 5. *If $f = x_{i_1}x_{i_2}\cdots x_{i_d}$ and $g = x_{j_1}x_{j_2}\cdots x_{j_d}$ are two monomials of the same degree d, we write $f \preccurlyeq g$ if*

$$i_1 \leqslant j_1, \quad i_2 \leqslant j_2, \quad \cdots, \quad i_d \leqslant j_d$$

If $\deg f < \deg g$, we write $f \preccurlyeq g$ if there exists a divisor g^* of g, such that g^* has the same degree as f and $f \preccurlyeq g^*$. If $f \preccurlyeq g$ and $f \neq g$, we write $f \prec g$.

As decreasing monomial codes, polar codes satisfy the following property, first established by Bardet, Dragoi, Otmani, and Tillich in [BDOT16], and by Schürch in [Sch16].

Theorem 5. *Let $\mathbb{C}_n(\mathcal{A})$ be a polar code, specified in terms of its information index set $\mathcal{A}$. If $[\![g]\!] \in \mathcal{A}$ and $f \preccurlyeq g$, then also $[\![f]\!] \in \mathcal{A}$. Equivalently, if $[\![f]\!] \in \mathcal{F}$ and $f \preccurlyeq g$, then also $[\![g]\!] \in \mathcal{F}$.*

A simple lemma about the partial order of two monomials, and their row indices that is easy to verify is the following:

Lemma 1. *If $g \preccurlyeq f$, then $[\![g]\!] \geqslant [\![f]\!]$.*

2.5.2 The Largest Mixing Factor of Polar Codes

Now we are ready to study the range of mixing factor of polar codes as decreasing monomial codes. Since by the MacWilliams identity [Mac63], one can easily obtain the weight distribution of a code from the weight distribution of its dual, if we want to compute the weight distribution of a given polar code, we have the options of applying Algorithm 2 to either the code itself, or to its dual. Since the dual code of a decreasing monomial code is also a decreasing monomial code [BDOT16], to get a complexity cap of our approach, it suffices to limit our space to polar codes of rates at most 1/2.

For conventional polar codes of rates at most 1/2, the following theorem states that their mixing factor is bounded by the mixing factor of self-dual Reed-Muller codes.

Theorem 6. *Let $\mathbb{C}$ be an (n,k) polar code with $n = 2^m$, $m = 2t+1$, and dimension $k \leqslant n/2$, then*

$$\mathrm{MF}(\mathbb{C}) \leqslant 2^{2t} - 2^{t+1} + 1 \tag{2.15}$$

Moreover, the equality holds only when $\mathbb{C}$ is the self-dual Reed-Muller code.

According to Theorem 6, the mixing factor of polar codes at length $n = 2^m$, where m is an odd number, is bounded by the mixing factor of self-dual RM codes. Here we list the mixing factor of self-dual RM codes at length 8, 32, 128, 512 and 2048 in Table 2.2.

Table 2.2. Mixing factor of self-dual RM codes

code length n	8	32	128	512	2048
mixing factor	1	9	49	225	961

For polar codes at length $n = 2^m$, where m is an even number, we make the following conjecture about their largest mixing factors. The conjectured upper bounds for polar codes at length 16, 64, 256, 1024 are listed in Table 2.3.

Conjecture 1. *Let $\mathbb{C}$ be an (n,k) polar code, with $n = 2^m$, $m = 2t$, and dimension $k \leqslant n/2$, then*

$$\mathrm{MF}(\mathbb{C}) \leqslant 2^{2t-1} - 2^{t+1} + 2$$

and the equality is achievable.

Table 2.3. Conjectured upper bounds for polar codes with rates $\leqslant$ 1/2

code length n	16	64	256	1024
mixing factor $\leqslant$	2	18	98	450

The rest of this section is devoted to the proof of Theorem 6.

Proof of Theorem 6. It can be verified by exhaustive search that Theorem 6 holds when $t = 1$ and $t = 2$. So hereforth, we focus on proving the theorem when $t \geqslant 3$. In this proof we use Table 2.4 to help illustrate our arguments. First we show self-dual Reed-Muller codes achieve the equality in (2.15).

Claim 1. Let $\mathbb{C}$ be the self-dual RM code of length 2^{2t+1}, then $\mathrm{MF}(\mathbb{C}) = 2^{2t} - 2^{t+1} + 1$.

Proof. Let $\mathcal{I}$ be set of monomials generating $\mathbb{C}$, Then $\mathcal{I}$ contains all monomials of degree less or equal to t. Referring to Table 2.4, we have $\tau(\mathbb{C}) = 2^{2t} - 2^{t+1}$. Thus from equation (2.13), we have

$$\mathrm{MF}(\mathbb{C}) = 2^{2t} - 2^{t+1} + 1.$$

Then we focus on the following claim, which states that if the mixing factor of the code is at least $2^{2t} - 2^{t+1} + 1$, then the code has to be the self-dual Reed-Muller code.

Claim 2. Let $\mathbb{C}$ be a polar code of length $n = 2^{2t+1}$ and dimension $k \leqslant n/2$. If

$$\mathrm{MF}(\mathbb{C}) \geqslant 2^{2t} - 2^{t+1} + 1,$$

Table 2.4. A table illustrating the positions of g and g' in the proof of Theorem 6

$[\![f]\!]$	$[f]$	f	
0	$(\underbrace{0,0,\cdots,0,0}_{2t+1})$	$x_0x_1x_2\cdots x_{2t}$	
1	$(0,0,\cdots,0,1)$	$x_1x_2\cdots x_{2t}$	
$\vdots$	$\vdots$	$\vdots$	
$2^{2t+1}-2^{t+1}$	$(\underbrace{1,\cdots,1}_{t},\underbrace{0,\cdots,0,0}_{t+1})$	$x_0x_1x_2\cdots x_t$	$\leftarrow g$
$2^{2t+1}-2^{t+1}+1$	$(\underbrace{1,\cdots,1}_{t},0,\cdots,0,1)$	$x_1x_2\cdots x_t$	$\leftarrow g'$
$\vdots$	$\vdots$	$\vdots$	
$2^{2t+1}-2$	$(1,1,\cdots,1,0)$	x_0	
$2^{2t+1}-1$	$(1,1,\cdots,1,1)$	1	

then $\mathbb{C}$ can only be the self-dual Reed-Muller code.

Now it suffices to prove Claim 2, since it is clear that Theorem 6 follows if we combine Claim 1 and Claim 2. Hereforth, we denote $g = x_0x_1x_2\cdots x_t$ as the monomial with $[\![g]\!] = 2^{2t+1}-2^{t+1}$, and denote $g' = x_1x_2\cdots x_t$ as the monomial with $[\![g']\!] = [\![g]\!]+1$. If we list out all the monomials in $\mathcal{M}_{2t+1}$ following their row indices, the positions of g and g' in this list are shown in Table 2.4.

Our proof for Claim 2 relies on the following three claims.

Claim 3. Let $\mathbb{C}$ be a polar code of length $n = 2^{2t+1}$, and frozen index set $\mathcal{F}$. If $\tau(\mathbb{C}) \geqslant [\![g]\!]$, then $[\![g]\!] \in \mathcal{F}$.

Proof. Observe from Table 2.4 that for any monomial h with $[\![h]\!] \geqslant [\![g]\!]$, h is a divisor of g, and thus $h \preccurlyeq g$. Therefore, if $\tau(\mathbb{C}) \geqslant [\![g]\!]$, it follows from Theorem 5 that $[\![g]\!] \in \mathcal{F}$.

Claim 4. Let $\mathbb{C}$ be a polar code of length $n = 2^{2t+1}$, and frozen index set $\mathcal{F}$. If $\tau(\mathbb{C}) \geqslant [\![g']\!]$, then $[\![g']\!] \in \mathcal{F}$.

Proof. Similar to the proof for Claim 3, we can observe from Table 2.4 that for any monomial h with $[\![h]\!] \geqslant [\![g']\!]$, we have $h \preccurlyeq g'$. Therefore if $\tau(\mathbb{C}) \geqslant [\![g']\!]$, it follows from Theorem 5 that $[\![g']\!] \in \mathcal{F}$.

Claim 5. Let $\mathbb{C}$ be a polar code of length $n = 2^{2t+1}$ and dimension $k \leqslant n/2$. If

$$\mathrm{MF}(\mathbb{C}) \geqslant 2^{2t} - 2^{t+1} + 1 \quad \text{and} \quad \tau(\mathbb{C}) = [\![g]\!],$$

then $\mathbb{C}$ is the self-dual Reed-Muller code.

Proof. Since for any monomial h with $\deg h \geqslant t+1$, we have $g \preccurlyeq h$, it follows from Theorem 5 that $[\![h]\!] \in \mathcal{F}$ for any monomial h with degree at least $t+1$. So $\mathbb{C}$ is a subcode of the self-dual Reed-Muller code. On the other hand, in view of equation (2.13), the dimension of the code is at least

$$k = \mathrm{MF}(\mathbb{C}) + (n-1) - \tau(\mathbb{C}) \geqslant n/2$$

Thus $\mathbb{C}$ can only be the self-dual Reed-Muller code itself.

At this point, we are ready to prove Claim 2. We will first show that given the conditions in Claim 2, g has to be frozen. Moreover, we will then show that the last frozen index of the code has to be exactly $[\![g]\!]$.

Proof for Claim 2. First from equation (2.13), we have

$$\tau(\mathbb{C}) = \mathrm{MF}(\mathbb{C}) + (n-k) - 1 \geqslant 2^{2t+1} - 2^{t+1}$$

So the last frozen index of $\mathbb{C}$ is at least $[\![g]\!]$. Then we show that $\tau(\mathbb{C}) > [\![g]\!]$ leads to a contradiction. Assuming $\tau(\mathbb{C}) > [\![g]\!]$, we have $[\![g]\!] \in \mathcal{F}$ and $[\![g']\!] \in \mathcal{F}$ following Claim 3 and Claim 4 respectively. Now we count the number of monomials having row indices in $\mathcal{F}$. First for any h with $\deg h \geqslant t+1$, we have $h \succcurlyeq g$. Thus it follows from Theorem 5 that $[\![h]\!] \in \mathcal{F}$ for all h with $\deg h \geqslant t+1$. The number of those monomials can be counted as

$$\sum_{i=t+1}^{2t+1} \binom{2t+1}{i} = 2^{2t}$$

Then for any degree-t monomial h without x_0, we have $h \succcurlyeq g'$, which gives us $[\![h]\!] \in \mathcal{F}$ following Theorem 5. The number of those monomials can be counted as $\binom{2t}{t}$. Therefore, the number of frozen indices of $\mathbb{C}$ is at least

$$|\mathcal{F}| \geqslant 2^{2t} + \binom{2t}{t}$$

This gives

$$|\mathcal{A}| = n - |\mathcal{F}| \leqslant 2^{2t} - \binom{2t}{t}$$

But that contradicts $\mathrm{MF}(\mathbb{C}) \geqslant 2^{2t} - 2^{t+1} + 1$, since

$$2^{2t} - \binom{2t}{t} < 2^{2t} - 2^{t+1} + 1$$

for all $t \geqslant 3$.

Since $\tau(\mathbb{C}) > [\![g]\!]$ leads to a contradiction, we can only have $\tau(\mathbb{C}) = [\![g]\!]$. Thus it follows from Claim 5 that $\mathbb{C}$ can only be the self-dual Reed-Muller code.

Theorem 6 follows if we combine Claim 1 and Claim 2. □

2.6 Reducing Computation Complexity using LTA(m,2)

As decreasing monomial codes, polar codes have a large automorphism group including the *lower triangular affine* group $\mathrm{LTA}(m,2)$ [BDOT16]. In this section, we look into the algebraic property of polar codes as decreasing monomial codes, and prove that $\mathrm{LTA}(m,2)$ acts transitively on certain subsets of the codes. This implies that those subsets share the same weight distribution. We will show that this allow us to drastically reduces the number of polar cosets we need to evaluate when we compute the weight distribution of polar codes. Since Reed-Muller codes are also decreasing monomial codes, the results in this section apply to Reed-Muller codes as well. This complexity reduction makes it possible to compute the weight distribution of polar codes and Reed-Muller codes for all rates of length up to $n = 128$ with reasonable complexity.

2.6.1 Lower Triangular Affine Groups and Their Group Action

We start by reviewing the definition for the lower triangular affine group, and how it acts on polynomials. Henceforth, binary $m \times m$ matrices are denoted by $\mathbb{F}_2^{m\times m}$, and m-tuples in $\mathbb{F}_2^m$ are treated as column vectors. Following the notation in [BDOT16], we denote the affine transformations $\boldsymbol{x} \mapsto A\boldsymbol{x} + \boldsymbol{b}$ over $\mathbb{F}_2^m$ by a pair $(A, \boldsymbol{b})$, where $A \in \mathbb{F}_2^{m\times m}$ and $\boldsymbol{b} \in \mathbb{F}_2^m$.

Definition 6. *The* ***lower triangular affine group****, denoted as* $\mathrm{LTA}(m,2)$*, consists of all affine* *transformations $(A,\boldsymbol{b})$, where $A \in \mathbb{F}_2^{m\times m}$ is a non-singular lower triangular square matrix, and $\boldsymbol{b} \in \mathbb{F}_2^m$.*

The group action of $\mathrm{LTA}(m,2)$ on $\mathcal{R}_m$ can be defined as follows. For an affine transformation $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)$ with $A = (a_{ij})$, and a polynomial $p \in \mathcal{R}_m$, denote by $(A,\boldsymbol{b})\cdot p$ the action of $(A,\boldsymbol{b})$ on p, where each monomial x_i in p is replaced by another monomial y_i defined as

$$y_i = \sum_{j=0}^{m-1} a_{ij}x_j + b_i.$$

We can observe the following for the group action of $\mathrm{LTA}(m,2)$ on monomials.

Proposition 3. *Let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)$ and $f \in \mathcal{M}_m$, then $(A,\boldsymbol{b})\cdot f$ can be written in the following form*

$$(A,\boldsymbol{b})\cdot f = f + \sum_{g\in\mathcal{M}_m:\, g\prec f} u_g \cdot g, \tag{2.16}$$

where $u_g \in \{0,1\}$ for all g.

Proof. Since $A = (a_{ij})$ is non-singular and lower triangular, the action by $(A,\boldsymbol{b})$ will replace each monomial x_i in f by

$$y_i = x_i + \sum_{j=0}^{i-1} a_{ij}x_j + b_i.$$

Therefore

$$(A,\boldsymbol{b})\cdot f = \prod_{i\in\mathrm{ind}(f)} y_i = \prod_{i\in\mathrm{ind}(f)} \left(x_i + \sum_{j=0}^{i-1} a_{ij}x_j + b_i \right)$$

Equation (2.16) follows by expanding this expression. □

Here is another way to view the action by the affine transformations. Recall that the evaluation $\mathrm{ev}(p)$ of a polynomial $p \in \mathcal{R}_m$ is a vector that consists of $p(\boldsymbol{x})$ over all $\boldsymbol{x} \in \mathbb{F}_2^m$. Since every affine transformation $(A,\boldsymbol{b})$ is a bijection on $\mathbb{F}_2^m$, the evaluation $\mathrm{ev}((A,\boldsymbol{b})\cdot p)$ can be obtained from $\mathrm{ev}(p)$ by permuting its coordinates. Denote the action of $(A,\boldsymbol{b})$ on a polynomial evaluation as

$$(A,\boldsymbol{b}) \cdot \mathrm{ev}(p) = \mathrm{ev}((A,\boldsymbol{b}) \cdot p),$$

we can then view this action as a permutation on the coordinates of $\mathrm{ev}(p)$. In particular, vector $(A,\boldsymbol{b}) \cdot \mathrm{ev}(p)$ and vector $\mathrm{ev}(p)$ have the same Hamming weight.

Bardet, Dragoi, Otmani, and Tillich first show that the automorphism group of decreasing monomial codes includes the lower triangular affine group [BDOT16, Theorem 2].

Theorem 7. *The automorphism group of decreasing monomial codes over m variables contains* $\mathrm{LTA}(m,2)$.

2.6.2 A Subgroup of LTA(m,2)

In the main theorem of this section, we consider a subgroup of $\mathrm{LTA}(m,2)$, denoted $\mathrm{LTA}(m,2)_f$, that we associate with a given monomial f. This subgroup was introduced in [BDOT16], where it was used to analyze and count the minimum weight codewords of decreasing monomial codes.

Definition 7. *Let* $f \in \mathcal{M}_m$. *The subgroup* $\mathrm{LTA}(m,2)_f$ *of* $\mathrm{LTA}(m,2)$ *associated with the monomial* f *is defined as*

$$\mathrm{LTA}(m,2)_f \overset{\mathrm{def}}{=} \{(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2) \mid A \in M_f, \boldsymbol{b} \in B_f\},$$

where

$$M_f = \{(a_{ij}) \in \mathbb{F}_2^{m \times m} \mid \forall\, i > j,\, a_{ij} = 0 \;\; \textit{if} \;\; i \notin \mathrm{ind}(f) \;\; \textit{or} \;\; j \in \mathrm{ind}(f)\}$$

and

$$B_f = \{\boldsymbol{b} \in \mathbb{F}_2^m \mid b_i = 0 \;\; \textit{if} \;\; i \notin \mathrm{ind}(f)\}$$

Example 5. Consider $\mathrm{LTA}(5,2)$. For $f = x_0x_3x_4 \in \mathcal{M}_5$, we have

$$M_f = \{(a_{ij}) \in \mathbb{F}_2^{5\times 5} \mid \forall\, i > j,\, a_{ij} = 0 \text{ if } i \notin \{0,3,4\} \text{ or } j \in \{0,3,4\}\}$$

and

$$B_f = \{\boldsymbol{b} \in \mathbb{F}_2^m \mid b_i = 0 \text{ if } i \notin \{0,3,4\}\}$$

Therefore, the affine transformations $(A, \boldsymbol{b})$ in the subgroup $\mathrm{LTA}(5,2)_f$ have the form:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & a_{3,1} & a_{3,2} & 1 & 0 \\ 0 & a_{4,1} & a_{4,2} & 0 & 1 \end{pmatrix}, \quad \text{and} \quad b = \begin{pmatrix} b_0 \\ 0 \\ 0 \\ b_3 \\ b_4 \end{pmatrix},$$

where $a_{3,1}, a_{3,2}, a_{4,1}, a_{4,2}, b_0, b_3, b_4$ can take any value in $\{0,1\}$. There are 2^7 such affine transformations, so the order of $\mathrm{LTA}(2,4)_f$ is $2^7 = 128$.

2.6.3 One-Variable Descendance Relation

We also introduce a new relation on the monomials for our main theorem of this section. Henceforth, whenever we write $f = gx_i$ for two monomials f and g, we assume $i \notin \mathrm{ind}(g)$.

Definition 8. *For $f, g \in \mathcal{M}_m$, we say g is a* ***one-variable descendant*** *of f, and write $g \prec_1 f$ if either one of the following holds:*

1. *$f = hx_i$ and $g = hx_j$ for some monomial h with $j < i$.*

2. *$f = gx_i$.*

Compared with the partial order in Definition 5, this one-variable descendance relation is a more restricted relation in the sense that, the two involved monomials can only differ by *one* variable. We remark that this one-variable descendance relation is only a relation, but not a partial order on the monomials. The following example shows that this new relation is not transitive.

Example 6. For monomials in $\mathcal{M}_4$, we have

$$x_0x_2 \prec_1 x_0x_1x_2, \quad \text{and} \quad x_0x_1x_2 \prec_1 x_0x_1x_3,$$

but x_0x_2 is not a one-variable descendant of $x_0x_1x_3$.

Nevertheless, in view of the following proposition, which can be easily checked, the partial order in Definition 5 can be viewed as a *refinement* of the the one-variable descendance relation.

Proposition 4. *For any $f, g \in \mathcal{M}_m$*

1) If $g \prec_1 f$, then $g \prec f$.

2) If $g \prec f$, then there exists a finite sequence of monomials $g_0, g_1, \cdots, g_t \in \mathcal{M}_m$ such that:

$$g = g_0 \prec_1 g_1 \prec_1 g_2 \prec_1 \cdots \prec_1 g_t = f$$

2.6.4 The Main Theorem: A Transitive Group Action

Now we are ready to present the main theorem of this section.

Theorem 8. *Let $\mathbb{C}(\mathcal{I})$ be a polar code generated by $\mathcal{I} \in \mathcal{M}_m$, and let f be the monomial in $\mathcal{I}$ with the smallest row index. Partition $\mathcal{I}$ into the following disjoint union*

$$\mathcal{I} = \{f\} \cup \mathcal{S} \cup \mathcal{T},$$

where $\mathcal{S}$ consists of all the one-variable descendants of f with row indices smaller than $\tau(\mathbb{C})$, and $\mathcal{T}$ contains the rest of the monomials in $\mathcal{I}$:

$$\mathcal{S} = \{h \in \mathcal{I} \mid h \prec_1 f \textit{ and } [\![h]\!] < \tau(\mathbb{C})\}, \qquad \textit{and} \qquad \mathcal{T} = \mathcal{I} \setminus (\{f\} \cup \mathcal{S}).$$

Then the group action of subgroup $\mathrm{LTA}(m,2)_f$ *on the set $\mathcal{X}$ is transitive, where $\mathcal{X}$ is the set consists of cosets of $\mathbb{C}(\mathcal{T})$ defined as follows*

$$\mathcal{X} = \left\{ \mathrm{ev}(f) + \sum_{h \in \mathcal{S}} u_h \cdot \mathrm{ev}(h) + \mathbb{C}(\mathcal{T}) \;\middle|\; \forall\, h \in \mathcal{S},\, u_h \in \{0,1\} \right\}$$

This implies that all the cosets of $\mathbb{C}(\mathcal{T})$ in $\mathcal{X}$ have the same weight distribution.

Before proving this theorem, we illustrate Theorem 8 with an example, and show how we can use this theorem to reduce the complexity when computing the weight distribution of polar codes.

Example 7. Consider a (32,24) polar code $\mathbb{C}$ specified by the frozen index set $\mathcal{F} = \{0,1,2,3,4,5,8,16\}$. The monomials corresponding to the rows in G_{32} are shown in Figure 2.6, where the information bits are highlighted in red, orange and blue, and the frozen bits are black. Code $\mathbb{C}$ has last frozen index $\tau(\mathbb{C}) = 16$, and mixing factor $\mathrm{MF}(\mathbb{C}) = 9$.

Illustrating Theorem 8

Let $f = x_0x_3x_4$ be the monomial with the smallest row index in $\mathcal{I}$. Then $\mathcal{I}$ can be partitioned as

$$\mathcal{I} = \{f\} \cup \mathcal{S} \cup \mathcal{T},$$

where $\mathcal{S}$ consists of four of the one-variable descendants of f with row indices smaller than

$$
\begin{array}{rcc|cccccccccccccccccccccccccccccccc|l}
 & x_0x_1x_2x_3x_4 & u_0 & 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1x_2x_3x_4 & u_1 & 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_0x_2x_3x_4 & u_2 & 1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_2x_3x_4 & u_3 & 1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 & \\
 & x_0x_1x_3x_4 & u_4 & 1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1x_3x_4 & u_5 & 1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 & \\
f \rightarrow & \boxed{x_0x_3x_4} & u_6 & 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
\rightarrow & x_3x_4 & u_7 & 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 & \\
 & x_0x_1x_2x_4 & u_8 & 1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1x_2x_4 & u_9 & 1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
\mathcal{S} \; \rightarrow & x_0x_2x_4 & u_{10} & 1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_2x_4 & u_{11} & 1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0 & \\
\rightarrow & x_0x_1x_4 & u_{12} & 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1x_4 & u_{13} & 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0 & \\
\rightarrow & x_0x_4 & u_{14} & 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_4 & u_{15} & 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 & \\
 & x_0x_1x_2x_3 & u_{16} & 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1x_2x_3 & u_{17} & 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_0x_2x_3 & u_{18} & 1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_2x_3 & u_{19} & 1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0 & \\
 & x_0x_1x_3 & u_{20} & 1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \mathbb{C}(\mathcal{T}) \\
 & x_1x_3 & u_{21} & 1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0 & \\
 & x_0x_3 & u_{22} & 1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_3 & u_{23} & 1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 & \\
 & x_0x_1x_2 & u_{24} & 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1x_2 & u_{25} & 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_0x_2 & u_{26} & 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_2 & u_{27} & 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 & \\
 & x_0x_1 & u_{28} & 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & x_1 & u_{29} & 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 & \\
 & x_0 & u_{30} & 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 & \\
 & 1 & u_{31} & 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 &
\end{array}
$$

Figure 2.6. Polar Transformation matrix G_{32} in Example 7

$\tau(\mathbb{C}) = 16$, and $\mathcal{T}$ consists of the rest of the monomials in $\mathcal{I}$:

$$\mathcal{S} = \{x_3x_4,\ x_0x_2x_4,\ x_0x_1x_4,\ x_0x_4\},$$

$$\mathcal{T} = \{x_1x_2x_4,\ x_2x_4,\ x_1x_4,\ x_4,\ x_1x_2x_3,\ x_0x_2x_3,\ \cdots,\ x_0,\ 1\}$$

As shown in Figure 2.6, the monomials in $\mathcal{S}$ are colored in red, the monomials in $\mathcal{T}$ are colored in orange and in blue, and the subcode $\mathbb{C}(\mathcal{T})$ is generated by the gray rows in G_{32}.

Then, set $\mathcal{X}$ is defined to consist of 16 cosets of $\mathbb{C}(\mathcal{T})$ in the form

$$\mathrm{ev}(f) + u_1 \cdot \mathrm{ev}(x_3x_4) + u_2 \cdot \mathrm{ev}(x_0x_2x_4) + u_3 \cdot \mathrm{ev}(x_0x_1x_4) + u_4 \cdot \mathrm{ev}(x_0x_4) + \mathbb{C}(\mathcal{T}),$$

where u_1, u_2, u_3, u_4 are four coefficients that can take any value in $\{0,1\}$.

According to Theorem 8, the subgroup $\text{LTA}(5,2)_f$ acts transitively on $\mathcal{X}$. Since the group action of the affine transformations in $\text{LTA}(5,2)_f$ can be viewed as permutations on the codeword coordinates, we can conclude that all 16 cosets in $\mathcal{X}$ have the same weight distribution.

Computing the Weight Distribution

If we directly apply Algorithm 2 to compute the weight distribution of $\mathbb{C}$, we need to compute the weight enumerating function of 2^9 polar cosets. Now we show how we can reduce this number using Theorem 8.

We start by partitioning code $\mathbb{C}$ into two parts according to u_6. Let $\mathbb{C}\{u_6 = 1\}$ denote the subset of $\mathbb{C}$ where u_6 is fixed to be 1, and let $\mathbb{C}\{u_6 = 0\}$ denotes the subcode of $\mathbb{C}$ where u_6 is fixed to be 0. Then

$$\mathbb{C} = \mathbb{C}\{u_6 = 1\} \cup \mathbb{C}\{u_6 = 0\},$$

and both $\mathbb{C}\{u_6 = 1\}$ and $\mathbb{C}\{u_6 = 0\}$ can be represented as disjoint unions of 2^8 polar cosets. Next, we compute the weight distribution of $\mathbb{C}\{u_6 = 1\}$ and $\mathbb{C}\{u_6 = 0\}$ separately.

For $\mathbb{C}\{u_6 = 1\}$, observe that

$$\mathbb{C}\{u_6 = 1\} = \bigcup_{X \in \mathcal{X}} X$$

By Theorem 8, all cosets in $\mathcal{X}$ have the same weight distribution. Thus to get the weight distribution for $\mathbb{C}\{u_6 = 1\}$, it suffices to first compute the weight distribution of a single coset in $\mathcal{X}$ using Algorithm 2, and then multiply it by $|\mathcal{X}| = 2^4$. This reduces the number of polar cosets we need to evaluate for $\mathbb{C}\{u_6 = 1\}$ from 2^8 down to 2^4.

After that we consider $\mathbb{C}\{u_6 = 0\}$. Since the subcode $\mathbb{C}\{u_6 = 0\}$ is also a decreasing monomial code itself, we can again partition $\mathbb{C}\{u_6=0\}$ into two parts according to u_7:

$$\mathbb{C}\{u_6=0\}=\mathbb{C}\{u_6=0,u_7=1\}\cup\mathbb{C}\{u_6=0,u_7=0\},$$

and apply Theorem 8 to reduce the complexity for $\mathbb{C}\{u_6=0,u_7=1\}$.

By repeating this procedure, code $\mathbb{C}$ can be unfolded as follows

$$\begin{aligned}\mathbb{C}=\;&\mathbb{C}\{u_6=1\}\\ &\cup\mathbb{C}\{u_6=0,u_7=1\}\\ &\cup\mathbb{C}\{u_6=0,u_7=0,u_9=1\}\\ &\cup\cdots\\ &\cup\mathbb{C}\{u_6=0,u_7=0,\cdots,u_{14}=0,u_{15}=1\}\\ &\cup\mathbb{C}\{u_6=0,u_7=0,\cdots,u_{14}=0,u_{15}=0\},\end{aligned}$$

and Theorem 8 allows us to reduce the number of polar cosets we need to evaluate for each of those components. The amount of complexity reduction for the components, and the total amount of complexity reduction for $\mathbb{C}$ are shown on the left of Table 2.5. We also show the computed weight distribution for $\mathbb{C}$ in this example on the right of Table 2.5.

2.6.5 Proof of Theorem 8

The rest of this section is devoted to the proof of Theorem 8. We first introduce more notations. For any polynomial $p\in\mathcal{R}_m$, we can expand it and express p as a sum of monomials in $\mathcal{M}_m$ as

$$p=\sum_{q\in\mathcal{M}_m}u_q\cdot q, \tag{2.17}$$

Table 2.5. Left: the complexity reduction amounts in Example 7. The second column shows the number of polar cosets in each component, and the third column shows the number of polar cosets we need to evaluate after applying Theorem 8. **Right**: Weight Distribution of $\mathbb{C}$ in Example 7, where the unlisted A_d equals to zero.

components	complexity	reduced complexity
$\mathbb{C}\{u_6 = 1\}$	$2^8 = 256$	$2^4 = 16$
$\mathbb{C}\{\cdots, u_7 = 1\}$	$2^7 = 128$	$2^3 = 8$
$\mathbb{C}\{\cdots, u_9 = 1\}$	$2^6 = 64$	$2^2 = 4$
$\mathbb{C}\{\cdots, u_{10} = 1\}$	$2^5 = 32$	$2^2 = 4$
$\mathbb{C}\{\cdots, u_{11} = 1\}$	$2^4 = 16$	$2^1 = 2$
$\mathbb{C}\{\cdots, u_{12} = 1\}$	$2^3 = 8$	$2^1 = 2$
$\mathbb{C}\{\cdots, u_{13} = 1\}$	$2^2 = 4$	$2^0 = 1$
$\mathbb{C}\{\cdots, u_{14} = 1\}$	$2^1 = 2$	$2^0 = 1$
$\mathbb{C}\{\cdots, u_{15} = 1\}$	$2^0 = 1$	$2^0 = 1$
$\mathbb{C}\{\cdots, u_{15} = 0\}$	$2^0 = 1$	$2^0 = 1$
$\mathbb{C}$	$2^9 = 512$	40

d	A_d
0	1
4	472
6	6272
8	83164
10	503424
12	1768424
14	3668224
16	4717254
18	3668224
20	1768424
22	503424
24	83164
26	6272
28	472
32	1

where $u_q \in \{0,1\}$ are the coefficients. For each monomial q, we denote the coefficient u_q in this expansion of p by $\langle p \rangle_q$. Using this notation, equation (2.17) can be written as

$$p = \sum_{q \in \mathcal{M}_m} \langle p \rangle_q \cdot q,$$

We start our proof by establishing a few lemmas. First, we consider the group action of an affine transformation $(A, \boldsymbol{b})$ in the subgroup $\mathrm{LTA}(m,2)_f$ on f itself. The following lemma states that the coefficient of a monomial $h \in \mathcal{S}$ in the expansion of $(A, \boldsymbol{b}) \cdot f$ can actually be determined by a single entry in $(A, \boldsymbol{b})$.

Lemma 2. *In Theorem 8, let $(A, \boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$ with $A = (a_{ij})$, and $h \in \mathcal{S}$, then the coefficient of h in the expansion of $(A, \boldsymbol{b}) \cdot f$ equals to a single entry in $(A, \boldsymbol{b})$. More precisely,*

- *if $f = qx_s$ and $h = qx_t$ for some monomial q with $t < s$, then $\langle (A, \boldsymbol{b}) \cdot f \rangle_h = a_{st}$;*

- *if $f = hx_s$, then $\langle (A,\boldsymbol{b}) \cdot f \rangle_h = b_s$.*

Proof. First, f can be written as

$$f = \prod_{i \in \mathrm{ind}(f)} x_i$$

Consider the action of $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)$ on f. According to Definition 7, each monomial x_i in f will be replaced by

$$y_i = x_i + \sum_{j<i\colon j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i$$

Therefore, $(A,\boldsymbol{b}) \cdot f$ can be written as a product of ℓ linear terms, where $\ell = \deg f$:

$$(A,\boldsymbol{b}) \cdot f = \prod_{i \in \mathrm{ind}(f)} \left(x_i + \sum_{j<i\colon j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i \right) \tag{2.18}$$

Given that $h \in \mathcal{S}$ is a one-variable descendant of f, we now verify this lemma by discussing the following two cases for h:

- Case 1: $f = qx_s$ and $h = qx_t$ for some monomial q with $t < s$.

  We can observe that when we expand the right hand side of equation (2.18), there is only one way to generate the term h, corresponding to choosing $a_{st}x_t$ from the linear term led by x_s, and choosing the leading x_i for the rest of the linear terms. Thus $\langle (A,\boldsymbol{b}) \cdot f \rangle_h = a_{st}$.

- Case 2: $f = hx_s$.

  Similarly, we can observe that when we expand the right hand side of equation (2.18), there is only one way to generate the term h, corresponding to choosing b_s from the linear term led by x_s, and choosing the leading x_i for the rest of the linear terms. Thus $\langle (A,\boldsymbol{b}) \cdot f \rangle_h = b_s$.

□

Next, we consider the group action of an $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$ on a monomial $g \in \mathcal{T}$. The following lemma states that, the coefficient of a monomial $h \in \mathcal{S}$ in the expansion of $(A,\boldsymbol{b}) \cdot g$ is always equal to zero.

Lemma 3. *In Theorem 8, let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$, $h \in \mathcal{S}$, and $g \in \mathcal{T}$, then the coefficient of h in the expansion of $(A,\boldsymbol{b}) \cdot g$ is zero. In other words, $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 0$.*

Proof. Consider the action of $(A,\boldsymbol{b})$ on g. According to Definition 7, the monomials in g will change as follows:

- Every x_i with $i \in \mathrm{ind}(g) \cap \mathrm{ind}(f)$ will be replaced by

$$y_i = x_i + \sum_{j<i,\, j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i$$

- Every x_i with $i \in \mathrm{ind}(g) \setminus \mathrm{ind}(f)$ will be replaced by $y_i = x_i$, and thus remain unchanged.

So after the action by $(A,\boldsymbol{b})$, we have

$$\begin{aligned} g &= \prod_{i \in \mathrm{ind}(g)} x_i \\ \Rightarrow \quad (A,\boldsymbol{b}) \cdot g &= \underbrace{\left(\prod_{i \in \mathrm{ind}(g) \setminus \mathrm{ind}(f)} x_i \right)}_{(a)} \underbrace{\left(\prod_{i \in \mathrm{ind}(g) \cap \mathrm{ind}(f)} \left(x_i + \sum_{j<i,\, j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i \right) \right)}_{(b)} \end{aligned} \tag{2.19}$$

If $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 1$, then h should appear if we expand the right hand side of (2.19). Since $h \in \mathcal{S}$ is a one-variable descendant of f, according to Definition 8, we have the following two possible cases for the relation between h and f. Next, we show that if $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 1$, a contradiction can be drawn in both cases.

- Case 1: $f = qx_s$ and $h = qx_t$ for some monomial q with $t < s$.

If h appears in the expansion of the right hand side of (2.19), then we break it into two cases depending on where the x_t in h comes from.

- If the x_t in h comes from (a), then we must have

  $$\mathrm{ind}(g)\backslash\mathrm{ind}(f) = t \qquad \text{and} \qquad \mathrm{ind}(q) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$$

  Since q is a divisor of f, for $\mathrm{ind}(q) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$ to be true, we can either have

  $$\begin{aligned} &\mathrm{ind}(q) = \mathrm{ind}(g) \cap \mathrm{ind}(f) \\ &\qquad\qquad\qquad \Rightarrow \quad g = h \quad \text{(contradiction since g and h are distinct)} \end{aligned}$$

  or

  $$\begin{aligned} &\mathrm{ind}(q) \cup \{s\} = \mathrm{ind}(g) \cap \mathrm{ind}(f) \\ &\qquad\qquad\qquad \Rightarrow \quad g = qx_tx_s \quad \text{(contradiction since $[\![g]\!] > [\![f]\!]$)} \end{aligned}$$

  and both of them lead to a contradiction.

- If the x_t in h comes from (b), then we must have

  $$\mathrm{ind}(g)\backslash\mathrm{ind}(f) = \emptyset \qquad \text{and} \qquad \mathrm{ind}(q) \cup \{s\} \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$$

  which gives us

  $$\mathrm{ind}(g) = \mathrm{ind}(f) \quad \Rightarrow \quad g = f \qquad \text{(contradiction since g and f are distinct)}$$

- Case 2: $f = hx_s$.

If h appears in the expansion of the right hand side of (2.19), then we must have

$$\mathrm{ind}(g) \backslash \mathrm{ind}(f) = \emptyset \qquad \text{and} \qquad \mathrm{ind}(h) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$$

Since h is a divisor of f, for $\mathrm{ind}(h) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$ to be true, we can either have

$$\mathrm{ind}(h) = \mathrm{ind}(g) \cap \mathrm{ind}(f) \quad \Rightarrow \quad g = h \quad \text{(contradiction since } g \text{ and } h \text{ are distinct)}$$

or

$$\begin{aligned} &\mathrm{ind}(h) \cup \{s\} = \mathrm{ind}(g) \cap \mathrm{ind}(f) \\ &\qquad\qquad\qquad\qquad \Rightarrow \quad g = f \quad \text{(contradiction since } g \text{ and } f \text{ are distinct)} \end{aligned}$$

and both of them lead to a contradiction.

Therefore, in both Case 1 and Case 2, $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 1$ leads to contradictions. So we can only have $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 0$. □

Using Lemma 3, we can prove that subcode $\mathbb{C}(\mathcal{T})$ is invariant under $\mathrm{LTA}(m,2)_f$, as stated in the following lemma.

Lemma 4. *In Theorem 8, subcode* $\mathbb{C}(\mathcal{T})$ *is invariant under* $\mathrm{LTA}(m,2)_f$.

Proof. Let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$. The group action by $(A,\boldsymbol{b})$ can be viewed as a permutation on the codeword coordinates, so $(A,\boldsymbol{b})$ acting on $\mathbb{C}(\mathcal{T})$ will generate another subspace with the same dimension as $\mathbb{C}(\mathcal{T})$. Since $\mathbb{C}(\mathcal{T})$ is generated by the monomials in $\mathcal{T}$, to prove this claim, it suffices to prove that for any $g \in \mathcal{T}$,

$$(A,\boldsymbol{b}) \cdot \mathrm{ev}(g) \in \mathbb{C}(\mathcal{T})$$

Let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$ and $g \in \mathcal{T}$. First, it follows from Proposition 3 that

$$(A,\boldsymbol{b}) \cdot g = g + \sum_{g' \in \mathcal{M}_m \colon g' \prec g} u'_g \cdot g', \tag{2.20}$$

where $u'_g \in \{0,1\}$ are coefficients for all g'. Then, since $\mathcal{I}$ is the generating set of a polar code, from Theorem 5 we know all g' with $g' \prec g$ lie in $\mathcal{I}$. Hence (2.20) can be written as

$$(A,\boldsymbol{b}) \cdot g = g + \sum_{g' \in \mathcal{I} \colon g' \prec g} u'_g \cdot g'. \tag{2.21}$$

Recall f is the monomial with the smallest row index in $\mathcal{I}$, so it follows from Lemma 1 that $f \not\preccurlyeq g$. Also, Claim 1 tells us that in equation (2.21), $u_h = 0$ for all $h \in \mathcal{S}$. Since $\mathcal{I} = \{f\} \cup \mathcal{S} \cup \mathcal{T}$, equation (2.21) becomes

$$(A,\boldsymbol{b}) \cdot g = g + \sum_{g' \in \mathcal{T} \colon g' \prec g} u'_g \cdot g'$$

Therefore, any $(A,\boldsymbol{b}) \cdot g$ with $g \in \mathcal{T}$ can be generated by the monomials in $\mathcal{T}$. This finishes the proof of this claim. □

At this point, we are ready to put everything together and prove Theorem 8. Take $X_0 = \mathrm{ev}(f) + \mathbb{C}(\mathcal{T})$ to be a coset in $\mathcal{X}$. To prove that the group action of $\mathrm{LTA}(m,2)_f$ on $\mathcal{X}$ is transitive, it suffices to prove that the orbit of X_0 is the entire $\mathcal{X}$.

Let $(A,\boldsymbol{b})$ be an affine transformation in $\mathrm{LTA}(m,2)_f$. If we consider the action of $(A,\boldsymbol{b})$ on f, it follows from Proposition 3 that

$$(A,\boldsymbol{b}) \cdot f = f + \sum_{h \in \mathcal{S}} u_h \cdot h + \sum_{g \in \mathcal{T}} u_g \cdot g$$

where $u_h = \langle (A,\boldsymbol{b}) \cdot f \rangle_h$ for each $h \in \mathcal{S}$, and $u_g = \langle (A,\boldsymbol{b}) \cdot f \rangle_g$ for each $g \in \mathcal{T}$. Therefore, if we look at the action of $(A,\boldsymbol{b})$ on X_0, we have

$$\begin{aligned}(A,\boldsymbol{b})\cdot \mathrm{X}_0 &= \mathrm{ev}(f) + \sum_{h\in\mathcal{S}} u_h \cdot \mathrm{ev}(h) + \sum_{g\in\mathcal{T}} u_g \cdot \mathrm{ev}(g) + (A,\boldsymbol{b})\cdot \mathbb{C}(\mathcal{T}) && (2.22)\\ &= \mathrm{ev}(f) + \sum_{h\in\mathcal{S}} u_h \cdot \mathrm{ev}(h) + \sum_{g\in\mathcal{T}} u_g \cdot \mathrm{ev}(g) + \mathbb{C}(\mathcal{T}) && (2.23)\\ &= \mathrm{ev}(f) + \sum_{h\in\mathcal{S}} u_h \cdot \mathrm{ev}(h) + \mathbb{C}(\mathcal{T}) && (2.24)\\ &= \mathrm{ev}(f) + \sum_{h\in\mathcal{S}} \langle (A,\boldsymbol{b})\cdot f\rangle_h \cdot \mathrm{ev}(h) + \mathbb{C}(\mathcal{T}) && (2.25)\end{aligned}$$

where

- in (2.22), we have

  $$(A,\boldsymbol{b})\cdot \mathbb{C}(\mathcal{T}) = \mathbb{C}(\mathcal{T}),$$

  since $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$, and $\mathbb{C}(\mathcal{T})$ is invariant under $\mathrm{LTA}(m,2)_f$, according to Lemma 4.

- in (2.23), we have

  $$\sum_{g\in\mathcal{T}} u_g \cdot \mathrm{ev}(g) + \mathbb{C}(\mathcal{T}) = \mathbb{C}(\mathcal{T}),$$

  since $\mathrm{ev}(g) \in \mathbb{C}(\mathcal{T})$ for all $g \in \mathcal{T}$.

In (2.25), according to Lemma 2, every $\langle (A,\boldsymbol{b})\cdot f\rangle_h$ equals to a single entry in $(A,\boldsymbol{b})$. Therefore, given any $X \in \mathcal{X}$, we can pick an $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$ whose entries are chosen such that X can be generated by $(A,\boldsymbol{b})\cdot \mathrm{X}_0$. This proves that the orbit of X_0 is the entire set $\mathcal{X}$, which means the group action of $\mathrm{LTA}(m,2)_f$ on the $\mathcal{X}$ is transitive. Since the action by affine transformations in $\mathrm{LTA}(m,2)_f$ can be viewed as permutations on the codeword coordinates, all the cosets in $\mathcal{X}$ thus have the same weight distribution. This completes the proof.

2.7 Our Approach on Polar Codes and Reed-Muller Codes at Length 128

In this section, we present the weight distribution of the $(128,64)$ 5G polar code computed with our algorithm, and discuss the complexity of our algorithm on the self-dual (128,64) Reed-Muller code.

First, we run our algorithms on the (128,64) 5G polar code specified in [3GP18]. This code has mixing factor 34, so if we directly use Algorithm 2, the number of polar cosets we need to evaluate equals 2^{34}. The (128,64) 5G polar code can be verified to be a decreasing monomial code. If we apply the complexity reduction using the automorphism group of polar codes in Section 2.6, the number of polar cosets we need to evaluate can be drastically reduced to $39257360 \approx 2^{25.23}$. The computed weight distribution of this code is shown in Table 2.6. For reference, computing this weight distribution takes less than two hours on a laptop computer.

Then, we look at the (128,64) Reed-Muller code, which according to Theorem 6, achieves the largest mixing factor for polar codes at length 128. This Reed-Muller code has mixing factor 49, so to compute its entire weight distribution, the number of polar cosets we need to evaluate in Algorithm 2 equals 2^{49}. If we apply the complexity reduction in Section 2.6, this number can be reduced to $49761365064 \approx 2^{35.53}$, which is a viable computation complexity that can be achieved. Since this self-dual (128,64) Reed-Muller code has the largest mixing factor among all polar codes of length 128. It is reasonable to expect that after we apply the complexity reduction in Section 2.6, the number of polar cosets we need to evaluate for other polar codes at length 128 will not be much larger than 2^{35}. Therefore, we believe that our approach allows us to compute the weight distribution of any polar codes and Reed-Muller codes up to length 128.

2.8 Acknowledgements

This chapter, in part, has been published at 2021 IEEE International Symposium on Information Theory (ISIT) and appeared as: Hanwen Yao, Arman Fazeli, and Alexander Vardy

Table 2.6. Weight distribution of the (128,64) 5G polar code

d	A_d
0	1
8	304
12	768
16	161528
20	4452096
24	166137744
28	8299319808
32	474588991516
36	19910428320256
40	555627871531568
44	9459383897458944
48	94101946507153608
52	550051775557674240
56	1920378732932218128
60	4051638142931561472
64	5194332067339587654
68	4051638142931561472
72	1920378732932218128
76	550051775557674240
80	94101946507153608
84	9459383897458944
88	555627871531568
92	19910428320256
96	474588991516
100	8299319808
104	166137744
108	4452096
112	161528
116	768
120	304
128	1

“A Deterministic Algorithm for Computing the Weight Distribution of Polar Codes” [YFV21a]. The dissertation author was the primary author of this conference paper.

Chapter 3

Construct Large Kernel Polar Codes with Small Scaling Exponent

3.1 Introduction

Polar coding, pioneered by Arıkan [Arı09], gives rise to the first family of codes that provably achieve capacity for all binary memoryless symmetric (BMS) channels with efficient encoding and decoding. This chapter is concerned with how fast can polar coding approach capacity as a function of the code length. In finite-length analysis [HAU14, FHMV20, MHU16, PU19, PPV10], the scaling between code length n and the gap to capacity is usually measured in terms of the scaling exponent μ. In coding theory, the three most important factors of any coding family are: rate R, block length n, and block error probability P_e. While characterization of the exact relationship of all three of them remains formidable and unknown, we can study the relationship (also called the "scaling") of two of them by fixing the third parameter. One such study leads to the discussion of the scaling exponent, where we fix P_e and consider the relationship between the block length n and the code rate R. On a channel W with capacity $I(W)$, the optimal scaling is of the form $n = O(1/\epsilon^{\mu})$ with a constant μ, where ϵ is the gap to capacity given by $\epsilon = I(W) - R$. This constant μ is referred as the scaling exponent. It is known that the best possible scaling exponent is $\mu = 2$ [Str62], and it can be achieved by random linear codes [PPV10, Hay09].

For polar code, a sequence of papers [GB14, HAU14, KMTU10, MHU16, WLVG22] have studied the upper and lower bounds on its scaling exponent. Those study shows that polar code has scaling exponent $\mu = 3.63$ on the binary erasure channel (BEC), and the best-known bounds valid for its scaling exponent on general BMS channel W are given by $3.579 \leqslant \mu \leqslant 4.63$. However, those values fall far short of the optimal scaling exponent $\mu = 2$. One way to improve the scaling exponent of polar coding is replacing the Arıkan's 2×2 polarization kernel with a larger kernel. Recently, plenty of polarization kernels have been proposed with good scaling properties [MT12, FV14, PSL$^+$15, BFS$^+$17a, TT18, Mor20]. Moreover, it was shown in [FHMV20] that polar coding derived from a random $\ell \times \ell$ polarization kernel approach optimal scaling $\mu = 2$ on BEC with high probability as $\ell \to \infty$,

Korada, Şaşoğlu, and Urbanke [KŞU10] were the first to show that polarization theorem (Theorem 2) still holds if one replaces the conventional 2×2 kernel K_2 of Arıkan [Arı09] with an $\ell \times \ell$ binary matrix K_ℓ, provided that this matrix is non-singular and not upper triangular under any column permutation. Moreover, [KŞU10] establishes a simple formula for the error exponent of the resulting polar codes in terms of the *partial distances* of the nested *kernel codes*. However, an explicit formulation for the scaling exponent is at present unknown, even for the simple case of the BEC. Just like Arıkan's 2×2 kernel K_2, which transforms the underlying channel W into two synthesized bit channels $\{W^+, W^-\}$, an $\ell \times \ell$ kernel K_ℓ transforms W into ℓ synthesized bit channels $W_0, W_1, \ldots, W_{\ell-1}$. If W is a BEC with erasure probability z, the bit channels $W_0, W_1, \ldots, W_{\ell-1}$ are also BECs and their erasure probabilities are given by integer polynomials $f_i(z)$ for $i = 0, 1, \ldots, \ell - 1$. We refer the corresponding set $\{f_0(z), f_1(z), \ldots, f_{\ell-1}(z)\}$ as the *polarization behavior of* K_ℓ, which completely determines the scaling exponent $\mu(K_\ell)$.

While smaller scaling exponent translates into better finite-length performance, the complexity of decoding can grow exponentially with the kernel size. There have been attempts to reduce the decoding complexity of large kernels [Tri14, BFS$^+$17b, MT12, TT19, Tri19b], however this problem remains to be considered as unsolved in general. We note that, although our constructions are explicit, issue of decoding our constructed kernels is deferred to the next chapter. In this chapter, our goal is to pursuit towards the following simple question: what is the smallest scaling exponent one can get with an $\ell \times \ell$ binary kernel? In particular, we construct a kernel K_{64} with $\mu(K_{64}) \simeq 2.87$. This gives rise to a sequence of binary linear codes that approaches capacity on the BEC with quasi-linear complexity and scaling exponent strictly less than 3. To the best of our knowledge, such a sequence of codes was not previously known to exist.

3.1.1 Related Prior Work

Scaling exponents of error-correcting codes have been a subject to an extensive amount of research. It is known since the work of Strassen [Str62] that random codes attain the optimal scaling exponent $\mu = 2$. It was furthermore shown by Polyanskiy, Poor and Verdu [PPV10] that random *linear* codes also achieve this optimal value. For polar codes, a heuristic method for computing the scaling exponent over the BEC under successive cancellation (SC) decoding was given in [KMTU10], which yielded $\mu \approx 3.627$. It was later shown in [GX14] that the block length, construction complexity, and decoding complexity are all bounded by a polynomial in $1/\epsilon$. This implies that for polar codes, μ is finite. The first attempt at bounding their scaling exponents were given in [HAU14], where the scaling exponent of polar codes for arbitrary channels were shown to be bounded by $3.579 \leqslant \mu \leqslant 6$. The upper bound was later improved to $\mu \leqslant 5.702$ in [GB14], further improved to $\mu \leqslant 4.714$ in [MHU16], and then refined to $\mu \leqslant 4.63$ in [WLVG22]. An upper bound on the scaling exponent of polar codes for non-stationary channels was also presented in [Mah17] as $\mu \leqslant 8.54$.

Authors in [HAU14] also introduced a method to explicitly calculate the scaling exponent of polar codes over BEC based on its polarization behavior. They showed that for Arıkan's kernel K_2, $\mu = 3.627$. Later on, there are 8×8 and 16×16 kernels [FV14] found with scaling exponents better than K_2. In particular, a kernel K_8 was found with $\mu = 3.577$ for BEC, which is optimal among all kernels with $\ell \leqslant 8$ [FV14]. It was accompanied with a heuristic construction to design larger polarizing kernels with smaller scaling exponents, which gave rise to a 16×16 kernel with $\mu = 3.356$. Later, another 16×16 kernel with $\mu = 3.356$ was proposed in [TT18], and plenty of other polarization kernels have been proposed with good scaling properties [PSL$^+$15, BFS$^+$17a, Mor20]. In [MT12], a 32×32 kernel F_{32} and a 64×64 kernel was constructed, which was shown (via simulations) to have a better *frame error rate* than the Arıkan's kernel K_2. The authors in [MT12] have also introduced an algorithm based on the *binary decision diagram* (BDD) to efficiently calculate the polarization behavior of large kernels. Attempts to achieve the optimal scaling exponent of 2 were first seen in [PU19], where it was shown that polar codes can achieve the near-optimal scaling exponent of $\mu = 2 + \epsilon$ over erasure channels by using explicit large kernels over large alphabets. The conjecture that it suffices to consider binary kernels was recently solved in [FHMV20], where it was shown that one can achieve the near-optimal scaling exponent on BECs via *almost any binary* $\ell \times \ell$ kernel given that ℓ is sufficiently large. The extension of this result into general BMS channels was given in [GRY20], and the extension into discrete memoryless channels was given in [WD20]. Now it remains to find the explicit constructions of such *optimal* kernels. Our results in this chapter can be viewed as another step towards the de-randomization of the proof in [FHMV20].

3.1.2 Our Contributions

In this chapter, we will focus on the case where the underlying channel W is a BEC, and propose a more comprehensive kernel construction approach given the size of the kernel ℓ. Constructing large polarizing kernels composes of two computationally complex problems: a) there are exponentially many large kernels, and b) the computational complexity of deriving scaling exponent for each such kernel grows exponentially with the kernel size as well. To narrow down the search size, we first introduce a special class of polarizing kernels called the *self-dual* kernels. For those self-dual kernels, we prove a duality theorem showing that their polarization behaviors are symmetric, which enables us to construct the kernel by only designing its bottom half. In our construction, we use a greedy approach for the bottom half of the kernel, where we push the value of $f_i(z)$ to be as close to 0 as possible for small z, in the order of $i = \ell - 1, \ell - 2, \ldots$. This heuristic approach let us construct kernels whose $f_i(z)$'s

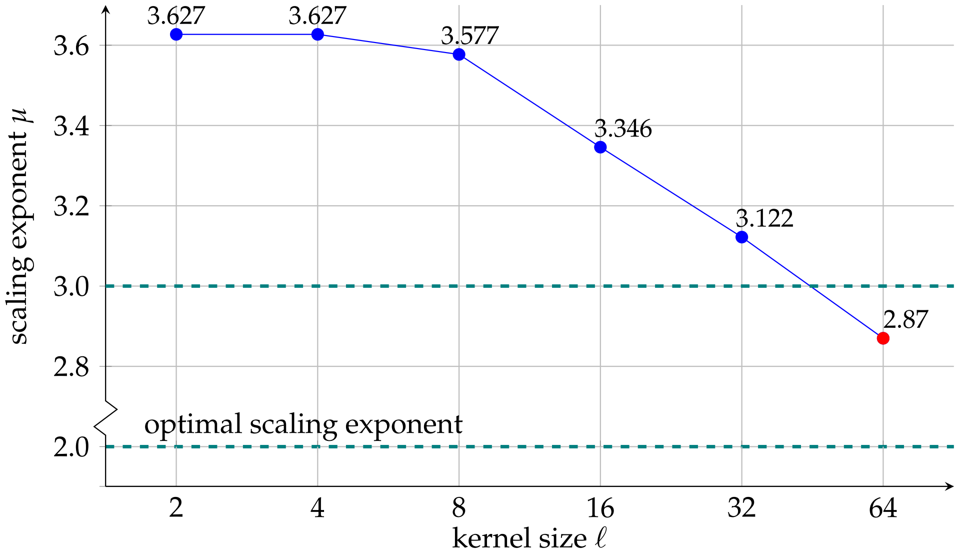

Figure 3.1. Scaling exponents of binary polarization kernels of size ℓ. The values for $\ell = 2,4,8$ are optimal [FV14]; the values for $\ell = 16,32,64$ are best known.

mimic the behavior of the polynomials in the polarization behavior of random large kernels, which intuitively gives us small scaling exponents. This construction gives the best previously found 16×16 kernel K_{16} provided in [TT18] with scaling exponent 3.346, a new 32×32 kernel K_{32} with $\mu(K_{32}) = 3.122$, and a new 64×64 kernel K_{64} with $\mu(K_{64}) \simeq 2.87$ as depicted in Figure 3.1. We also utilize some of the known partial distances of nested Reed-Muller (RM) codes and cyclic codes to further reduce the search size in the proposed construction algorithm.

To calculate the scaling exponent of our constructed kernels, we first calculate their polarization behaviors, and then invoke the method introduced in [HAU14]. It has been shown that for a specific bit channel, its $f_i(z)$ can be described by the weight distribution of its *uncorrectable erasure patterns* [FV14]. To calculate this weight distribution, we introduce a new trellis-based algorithm. Our algorithm is significantly faster than the BDD based algorithm proposed in [MT12]. It first builds a *proper trellis* for those uncorrectable erasure patterns, and then applies Viterbi algorithm to calculate its weight distribution. However, for a very large kernel, the complexity of our trellis algorithm gets prohibitively high for intermediate bit-channels. In particular, for our constructed K_{64}, we are only able to compute the first 15 and the last 15 polynomials in its polarization behavior. To complete the picture, we introduce an alternative Monte Carlo interpolation-based method to numerically estimate those polynomials in the middle, which gives us an estimation on the scaling exponent of K_{64} as $\mu(K_{64}) \simeq 2.87$. We support our estimation with a rigorous proof that our constructed K_{64} has scaling exponent $\mu(K_{64}) < 2.97$.

3.1.3 Notation

Here we specify some notation conventions we follow in this chapter. We use bold letters like $\boldsymbol{U}, \boldsymbol{u}$ to denote vectors, and nonbold letters like U_i, u_i to denote symbols within those vectors. We let the indices for the symbols within vectors start from zero. We use capital letters like $\boldsymbol{U}, U_i$ to denote random vectors or random variables, and lower-case letters like $\boldsymbol{u}, u_i$ to denote constant vectors or constants. We use $\boldsymbol{u}_i$ to represent $(u_0, u_1, \cdots, u_i)$, a subvector of $\boldsymbol{u}$ with its first $(i+1)$ symbols. And we denote the concatenation of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ as $(\boldsymbol{u}, \boldsymbol{v})$.

3.2 Background

In this section, we give a brief review on the background of large kernel polar codes. We first briefly recap the setting of polar codes constructed by large kernels, and explain how does the idea of channel polarization lead to their capacity-achieving property. Then we review the analysis of its performance in two common regimes: the *error-exponent* regime and the *scaling-exponent* regime. In this paper, we will focus on the scaling exponent regime over BECs.

3.2.1 Large Kernel Polar Codes

The channel polarization of larger kernel polar code is induced by a linear transformation by the $\ell \times \ell$ matrix $K^{\otimes m}$, where $\otimes m$ is the m-fold Kronecker product, and K is a binary square matrix, called the *polarization kernel*. Conventional polar codes introduced by Arıkan [Arı09]

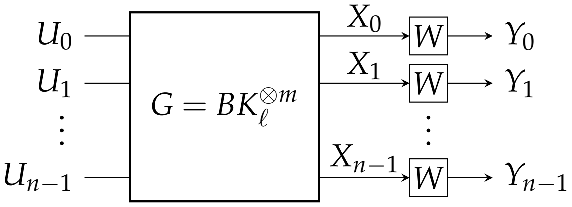

Figure 3.2. Block diagram of a polar coded communication scheme.

use the simple 2 × 2 kernel

$$K_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

It was later shown in [KŞU10] that we can construct polar codes from any $\ell \times \ell$ kernel K that is non-singular, and cannot be transformed into an upper triangular matrix under any column permutations. A length $n = \ell^m$, dimension k large kernel polar code is a linear code generated by k rows of the matrix $G = BK^{\otimes m}$, where B is an $n \times n$ base-ℓ digit reversal permutation matrix. To show that as the block length goes to infinity, large kernel polar codes achieve the capacity of a given BMS channel W, we can consider the *bit channels* introduced by G.

Let $W : \mathcal{X} \to \mathcal{Y}$ be a binary memoryless symmetric channel with input alphabet $\mathcal{X} = \{0,1\}$ and output alphabet $\mathcal{Y}$, characterized by its transition probabilities $W(y|x)$ for all $x \in \mathcal{X}, y \in \mathcal{Y}$. Let $\boldsymbol{U} = (U_0, U_1, \cdots, U_{n-1})$, drawn uniformly random from $\{0,1\}^n$, be the information carrier vector intended for transmission. The large kernel polar coding scheme will encode $\boldsymbol{U}$ as the codeword $\boldsymbol{X} = \boldsymbol{U}G$, and transmit $\boldsymbol{X}$ through n independent copies of W, as shown in Figure 3.2. Let $W^n : \mathcal{X}^n \to \mathcal{Y}^n$ be n independent channel uses of W. Similar to the conventional polar code, the bit channel $W_i : \{0,1\} \to \mathcal{Y}^n \times \{0,1\}^{i-1}$ for $i \in \{0, 2, \cdots, n-1\}$ for large kernel polar code is defined by

$$W_i(\boldsymbol{y}, \boldsymbol{u}_{i-1} | u_i) = \frac{1}{2^{n-1}} \sum_{\boldsymbol{u}' \in \{0,1\}^{n-i-1}} W^n(\boldsymbol{u} | (\boldsymbol{u}_{i-1}, u_i, \boldsymbol{u}')G) \tag{3.1}$$

Let $\boldsymbol{Y} \in \mathcal{Y}^n$ be the received vector. We can see that $W_i(\boldsymbol{y}, \boldsymbol{u}_{i-1} | u_i)$ is the conditional probability of the event $U_i = u_i$, given $(U_1, U_2, \cdots, U_{i-1}) = \boldsymbol{u}_{i-1}$ and $\boldsymbol{Y} = \boldsymbol{y}$.

A key observation by [Arı09] for K_2, and later shown to hold for any kernel K_ℓ that is non-singular and not upper triangular under any column permutation [KŞU10], is that as the block length n goes to infinity, those bit channels $W_0, W_1, \cdots, W_{n-1}$ will polarize in the sense that, most of them will have capacities either arbitrarily close to 0, or arbitrarily close to 1. Formally, the *capacity* and the *Bhattacharyya parameter* for a BMS channel W can be defined as

$$I(W) = \frac{1}{2} \sum_{y \in \mathcal{Y}} \sum_{x \in \{0,1\}} W(y|x) \log_2 \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)} \tag{3.2}$$

and as

$$Z(W) = \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)} \tag{3.3}$$

For a small $\delta \in (0,1)$, if we call a bit channel W_i *δ-good* if $Z(W_i) > 1 - \delta$, and *δ-bad* if $Z(W_i) < \delta$, then the polarization theorem [Arı09, KŞU10] states the following

Theorem 9 (Polarization theorem for large kernel)**.** *For every $\delta \in (0,1)$, almost all bit-channels become either δ-good or δ-bad as $n \to \infty$. In fact, as $n \to \infty$, the fraction of δ-good bit-channels approaches the capacity $I(W)$ of the underlying channel W, while the fraction of δ-bad bit-channels approaches* $1 - I(W)$.

With $\delta = o(1/n)$, this theorem leads to the construction of capacity-achieving polar codes by selecting k δ-good bit channels to carry the information, and freeze the rest of the $(n-k)$ bit channels to the constant zero.

3.2.2 The Scaling of Polar Codes

The performance of polar codes have been commonly analyzed in two regimes, the error-exponent regime and the scaling-exponent regime. In the error-exponent regime, we fix the rate $R < I(W)$, and study how does the error probability P_e scale as a function of the block length n. For conventional polar codes, it is shown in [AT09] that under successive cancellation (SC) decoding, P_e behaves roughly as $2^{-\sqrt{n}}$. A more refined results in this regime was shown later in [HMTU12]. For large kernel polar codes with kernel K, [KŞU10] showed that $P_e = o(1/2^{n^\beta})$ for any $\beta < E(K)$, where $E(K)$ is a constant called the *exponent* of the kernel, or the *rate of polarization*. It was shown in [KŞU10] that $E(K)$ can be explicitly expressed in terms of the *partial distances* of the *kernel codes*. We will give the definition for kernel codes later in this paper. Authors in [KŞU10] also proved that as the size of the kernel goes to infinity, there exists binary kernels with P_e scaling roughly as 2^{-n}. Some explicit constructions of kernels with large exponents are provided in [PSL$^+$15] and [LLAG15].

For this chapter, we focus on the scaling exponent regime, where we fix the error probability P_e, and study the scaling between the gap to capacity $\epsilon = I(W) - R$ and the block length n. If $n = O(1/\epsilon^\mu)$ with a constant μ, then we call μ the scaling exponent for this family of codes. For conventional polar codes, rigorous bounds on μ are provided in a series work of [HAU14, GB14, MHU16, WLVG22]. The currently best-known upper bounds on the scaling exponent are established in [MHU16] and in [WLVG22]: for any BMS channel, $\mu \leqslant 4.63$; and for BEC, $\mu \leqslant 3.639$. Authors in [HAU14] also introduced a method to explicitly calculate the scaling exponent of polar codes over BEC. It showed that for Arıkan's K_2, $\mu(K_2) = 3.627$. This calculation method is based on a *scaling assumption* which requires the existence of a certain limit, and that assumption remains open. We point out that in this paper, the values of the scaling exponents of our constructed kernels are calculated via the generalized version of this method, which is also based on a similar scaling assumption. But our proof that $\mu(K_{64}) < 2.97$ doesn't depend on this assumption.

3.2.3 Polarization Behavior and the Uncorrectable Erasure Patterns

Here we review the polarization behavior for a binary kernel as introduced in [FV14], which completely describes its channel polarization process over BECs, and thus also determines its scaling exponent. Let K be an $\ell \times \ell$ binary polarization kernel. To understand the channel polarization performed by K, we take the setting in Section 3.2.1 with $n = \ell$, or equivalently

$m = 1$. For a single step of polarization on a BEC, K will take ℓ independent copies of the underlying channel W, and transform them into ℓ bit channels $W_0, W_1, \cdots, W_{\ell-1}$, whose definitions are given in (3.1). When W is a BEC with erasure probability z, at the output of the bit channels we have $\boldsymbol{y} \in \mathcal{Y}^\ell = \{0,1,?\}^\ell$, where ? means erasure. An *erasure pattern* can be defined to be a vector $\boldsymbol{e} \in \{0,1\}^\ell$, where 1 corresponds to the erased positions possessed by ?, and 0 corresponds to the unerased positions. The probability of occurrence of a specific erasure pattern $\boldsymbol{e}$ will be

$$z^{\mathrm{wt}(\boldsymbol{e})}(1-z)^{\ell-\mathrm{wt}(\boldsymbol{e})}$$

where $\mathrm{wt}(\boldsymbol{e})$ stands for the Hamming weight of the erasure pattern $\boldsymbol{e}$.

It has been shown in [FV14] that if W is a BEC with erasure probability z, then the bit channels are all BECs as well. Moreover, the erasure probabilities of $W_0, W_1, \cdots, W_{\ell-1}$ will be polynomials of z with degree at most ℓ. To describe those polynomials, a concept called *uncorretable erasure pattern* can be introduced.

Definition 9. *We say an erasure pattern* $\mathbf{e}$ *is uncorrectable for a given bit-channel* W_i *if there exists two information vectors* $\boldsymbol{u}', \boldsymbol{u}''$ *such that* $u'_j = u''_j$ *for* $j < i$, $u'_i \neq u''_i$ *and* $(\boldsymbol{u}'K_\ell)_j = (\boldsymbol{u}''K_\ell)_j$ *for all unerased positions* $j \in \{k : e_k = 0\}$.

In other words, an erasure pattern is called *uncorrectable* for a bit-channel W_i if we are unable to determine the i-th information bit u_i given the channel output $\boldsymbol{y}$ and the first i bits in the information vector $u_0, u_1, \cdots, u_{i-1}$. We remark that the *uncorrectability* of an output $\boldsymbol{y}$ for a bit-channel depends solely on its erasure pattern, but not on the values of those unerased bits.

For the i-th bit-channel W_i, if we let $E_{i,w}$ denotes the number of its uncorrectable erasure patterns of weight w, then its erasure probability $f_i(z)$ can be represented as the polynomial

$$f_i(z) = \sum_{w=0}^{\ell} E_{i,w} z^w (1-z)^{(\ell-w)} \tag{3.4}$$

Therefore for W_i, if we can calculate the weight distribution of its uncorrectable erasure patterns $E_{i,0}, E_{i,1}, \ldots, E_{i,\ell}$, we can derive the polynomial $f_i(z)$. The entire set of those polynomials is called the *polarization behavior* of K_ℓ.

Definition 10. *The polarization behavior of a kernel K_ℓ is the set of the polynomials*

$$\{f_0(z), f_1(z), ..., f_{\ell-1}(z)\}$$

defined in (3.4).

3.2.4 Computing the Scaling Exponent

In the work by Hassani, Alishahi, and Urbanke [HAU14], a heuristic method was proposed to calculate the scaling exponent of polar codes based on a scaling assumption. They used this method to compute the scaling exponent of conventional polar codes constructed using Arıkan's 2 × 2 kernel K_2 as $\mu(K_2) = 3.626$. In this subsection, we show how their approach can be generalized to calculate the scaling exponents for large polarization kernels.

Let the underlying channel W be a BEC(z) with erasure probability z. Consider an $\ell \times \ell$ polarization kernel K_ℓ with polarization behavior $\{f_0(z), \ldots, f_{\ell-1}(z)\}$. The evolution of Bhattacharyya parameter for the polarized bit channels can be modeled as a random process Z_n formulated by

$$\begin{aligned} &Z_0 = z, \\ &Z_n = f_i(Z_{n-1}) \quad \text{w.p.} \quad \frac{1}{\ell} \quad \text{for all } i = 0, 1, \ldots, \ell - 1 \end{aligned}$$

For $0 < a < b < 1$, define

$$g_n(z, a, b) = \Pr(Z_n \in [a, b])$$

as the ratio of bit channels with Bhattacharyya parameters lying in the interval $[a, b]$ after n steps of polarization. This ratio can be viewed as the proportion of *unpolarized* bit channels when $a = \epsilon$ and $b = 1 - \epsilon$ for a small ϵ.

It can be noticed that $g_n(z,a,b)$ satisfies the following recursion

$$g_{n+1}(z,a,b) = \frac{1}{\ell}\sum_{i=0}^{\ell-1} g_n(f_i(z),a,b) \tag{3.5}$$

By numerical observation, a scaling assumption is proposed in [HAU14] for conventional polar codes. This scaling assumption can be generalized for large kernel polar codes as follows.

Assumption 1. *There exists $\mu \in (0,\infty)$ such that, for any $z,a,b \in (0,1)$ with $a < b$, The limit*

$$g_\infty(z,a,b) = \lim_{n\to\infty} \ell^{\frac{n}{\mu}} g_n(z,a,b)$$

exists in $(0,\infty)$.

Combining this scaling assumption with equation (3.5), we get

$$\ell^{-\frac{1}{\mu}} g_\infty(z,a,b) = \frac{1}{\ell}\sum_{i=0}^{\ell-1} g_\infty(f_i(z),a,b)$$

This allows us to numerically solve this limit $g_\infty(f_i(z),a,b)$ by iteration, following the steps described in [HAU14]. This numerical computation also gives us the scaling exponent μ.

3.3 Constructing Large Self-Dual Kernels

In this section, we propose a kernel construction method in pursuit of the following problem: given the size of the kernel ℓ, how can we construct a polarization kernel with smallest scaling exponent? It has been observed in [FV14] that any polarization kernel can be transformed into a lower-triangular matrix while preserving their polarization behaviors. However, the number of polarization kernel candidates still grows exponentially with the size ℓ, which makes exhausting search for all possible kernels infeasible for large ℓ. Therefore, instead of looking at all polarization kernels of size ℓ, we narrow our search space to a special class of kernels called the *self-dual* kernels. We prove that the polarization behaviors of self-dual kernels show a nice symmetry property, which not only simplifies our construction process, but also aids us in the analysis on their scaling exponents.

3.3.1 Kernels Codes and Uncorrectable Erasure Patterns

Before we introduce the self-dual kernels, we review the concept of kernel codes, first introduce in [KŞU10] to compute the error exponents for large polarization kernels. We then make connection between kernel codes and uncorrectable erasure patterns. This connection allows us to prove a symmetry property for self-dual kernels. In the next section, we will also use this connection to compute the weight distribution for uncorrectable erasure patterns on trellises.

Definition 11. *Let K_ℓ be an $\ell \times \ell$ kernel with rows $g_0, g_1, \cdots g_{\ell-1}$. Define the* ***kernel code*** *$\mathcal{C}_i$ to be the subspace*

$$\mathcal{C}_i = \text{span}\{g_i, \ldots, g_{\ell-1}\}$$

for all $0 \leqslant i < \ell$, and define $\mathcal{C}_\ell = \{\mathbf{0}\}$.

It follows from the above definition that those kernel codes form a nested chain:

$$\mathbb{F}_2^\ell = \mathcal{C}_0 \supseteq \mathcal{C}_1 \supseteq \mathcal{C}_2 \supseteq \cdots \supseteq \mathcal{C}_{\ell-1} \supseteq \mathcal{C}_\ell = \{\mathbf{0}\}$$

Here we also introduce the concept of *cover set*.

Definition 12. *Given two binary vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, we say $\boldsymbol{v}_1$* ***covers*** *$\boldsymbol{v}_2$ if the support of $\boldsymbol{v}_2$ is a subset of the support of $\boldsymbol{v}_1$:*

$$\text{supp}(\boldsymbol{v}_2) \subseteq \text{supp}(\boldsymbol{v}_1)$$

Given a set $\mathcal{S} \subseteq \mathbb{F}_2^\ell$, we define its ***cover set*** *$\Delta(\mathcal{S})$ as the set of vectors that covers at least one vector in $\mathcal{S}$:*

$$\Delta(\mathcal{S}) = \{\boldsymbol{u} \in \mathbb{F}_2^\ell \,:\, \exists\, \boldsymbol{v} \in \mathcal{S},\, \boldsymbol{u} \textit{ covers } \boldsymbol{v}\}$$

We prove that for a bit-channel W_i, its uncorrectable erasure patterns can be related to the cover set of the difference between two kernel codes.

Theorem 10. *An erasure pattern $\boldsymbol{e} \in \{0,1\}^{\ell}$ is uncorrectable for a bit-channel $\mathcal{W}_i$ if and only if*

$$\boldsymbol{e} \in \Delta(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$$

Proof. If $\boldsymbol{e}$ is uncorrectable, then there exists $\boldsymbol{u}', \boldsymbol{u}''$ as described in Definition 9, such that for the vector $(\boldsymbol{u}' - \boldsymbol{u}'')$, we have $(\boldsymbol{u}' - \boldsymbol{u}'')_j = 0$ for $j < i$ and $(\boldsymbol{u}' - \boldsymbol{u}'')_i = 1$. Consider the codeword encoded by this vector $\boldsymbol{c} = (\boldsymbol{u}' - \boldsymbol{u}'')K_{\ell}$, it is thus a codeword lies in the difference between two kernel codes:

$$\boldsymbol{c} \in (\mathcal{C}_i \backslash \mathcal{C}_{i+1})$$

On the other hand, since $\boldsymbol{u}'K_{\ell}$ and $\boldsymbol{u}''K_{\ell}$ have the same bits on all unerased positions, the codeword $\boldsymbol{c} = \boldsymbol{u}'K_{\ell} - \boldsymbol{u}''K_{\ell}$ is covered by the erasure pattern $\boldsymbol{e}$. Therefore, we have $\boldsymbol{e} \in \Delta(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$.

For the other direction, if $\boldsymbol{e} \in \Delta(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$, then $\boldsymbol{e}$ covers at least one codeword $\boldsymbol{c} \in (\mathcal{C}_{i-1} \backslash \mathcal{C}_i)$. Let $\boldsymbol{u}$ denote the information vector for $\boldsymbol{c}$ with $\boldsymbol{c} = \boldsymbol{u}K_{\ell}$, then it can be verified that $\boldsymbol{0}$ and $\boldsymbol{u}$ satisfiy the condition of $\boldsymbol{u}', \boldsymbol{u}''$ described in Definition 9, which proves that $\boldsymbol{e}$ is uncorrectable. □

3.3.2 Self-dual Kernel and the Duality Theorem

Now we introduce this special type of polarization kernels referred as the *self-dual kernels*. In our kernel construction, we are going to focus on kernels in this class.

Definition 13. *We call an $\ell \times \ell$ polarization kernel self-dual if for its kernel codes, we have*

$$\mathcal{C}_i = \mathcal{C}_{\ell-i}^{\perp}$$

for all $i = 0, \ldots, \ell$.

The following duality theorem shows that the polarization behaviors of self-dual kernels satisfy a nice symmetry property.

Theorem 11 (Duality theorem)**.** *If a kernel K_ℓ is self-dual, then we have*

$$f_i(z) + f_{\ell-1-i}(1-z) = 1$$

for all $i = 0, \dots, \ell - 1$.

Before proving this theorem, let's first illustrate this symmetry with the following example.

Example 8. Consider the self-dual kernel K_{32} in Figure 3.8 constructed following the the steps that we are going to explain later. In Figure 3.3 we show the curves of the 32 polynomials $f_0(z), f_1(z), \cdots, f_{31}(z)$ in its polarization behavior as functions of z. According to the duality theorem, we have $f_i(z) + f_j(1-z) = 1$ for every pair of i, j where $i + j = \ell - 1$. Two examples are $f_4(z) + f_{27}(1-z) = 1$ and $f_9(z) + f_{22}(1-z) = 1$, as highlighted in Figure 3.4 and Figure 3.5, respectively. We can observe that in Figure 3.4, the curves of $f_4(z)$ and $f_{27}(z)$ are indeed central symmetric with respect to the point $(0.5, 0.5)$ on the grid. Similarly in Figure 3.5, the curves of $f_9(z)$ and $f_{22}(z)$ are also central symmetric with respect to the point $(0.5, 0.5)$.

The rest of this subsection is devoted to the proof of Theorem 11. We start with the following lemma.

Lemma 5. *Let K_ℓ be a self-dual polarization kernel. If an erasure pattern $\boldsymbol{e}$ is uncorrectable for bit-channel W_i, then its complement $\overline{\boldsymbol{e}}$ has to be correctable for bit-channel $W_{\ell-1-i}$.*

Proof. We prove this lemma by contradiction. Assume we have an erasure pattern $\boldsymbol{e}$ that is uncorrectable for W_i, with its complement $\overline{\boldsymbol{e}}$ being uncorrectable for $W_{\ell-1-i}$ as well. Following this assumption, $\boldsymbol{e}$ covers a codeword, denoted as $\boldsymbol{c}_1$, that lies in $(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$, and $\overline{\boldsymbol{e}}$ covers a

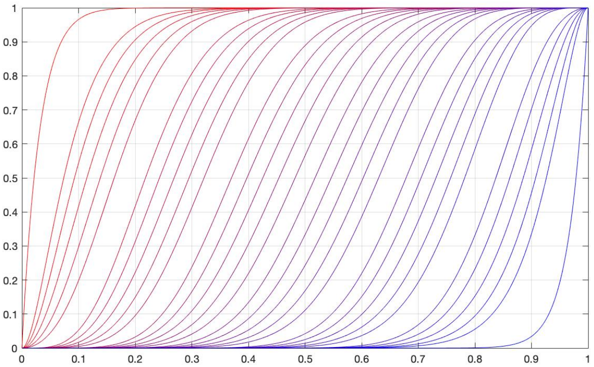

Figure 3.3. The polarization behavior $\{f_0(z), f_1(z), \cdots, f_{31}(z)\}$ of K_{32}

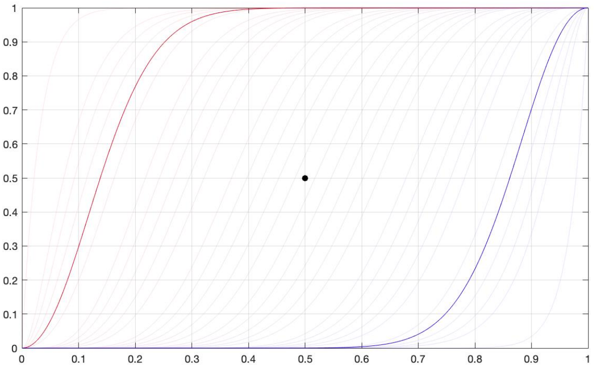

Figure 3.4. Two polynomials $f_4(z)$ and $f_{27}(z)$ in the polarization behavior of K_{32}

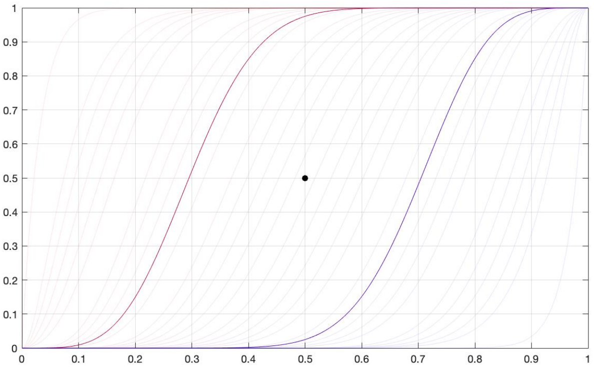

Figure 3.5. Two polynomials $f_9(z)$ and $f_{22}(z)$ in the polarization behavior of K_{32}

codeword, denoted as denoted as $\boldsymbol{c}_2$, that lies in $(\mathcal{C}_{\ell-1-i} \backslash \mathcal{C}_{\ell-i})$. Because K_ℓ is self-dual, we have

$$\boldsymbol{c}_2 \in (\mathcal{C}_{\ell-1-i} \backslash \mathcal{C}_{\ell-i}) = (\mathcal{C}_{i+1}^{\perp} \backslash \mathcal{C}_{i}^{\perp}) \quad \Rightarrow \quad \boldsymbol{c}_2 \perp \mathcal{C}_{i+1} \tag{3.6}$$

Since the support of $\boldsymbol{e}$ and the support of $\overline{\boldsymbol{e}}$ are disjoint, we have $\boldsymbol{c}_1 \perp \boldsymbol{c}_2$. Moreover, since $\mathcal{C}_i$ only has one more dimension than $\mathcal{C}_{i+1}$:

$$\mathcal{C}_i = \operatorname{span}\{\boldsymbol{c}_1 \cup \mathcal{C}_{i+1}\},$$

the fact that both $\boldsymbol{c}_2 \perp \mathcal{C}_{i+1}$ and $\boldsymbol{c}_2 \perp \boldsymbol{c}_1$ implies that $\boldsymbol{c}_2 \perp \mathcal{C}_i$. This contradict the fact that $\boldsymbol{c}_2$ doesn't lie in $\mathcal{C}_i^{\perp}$ as shown in (3.6). Therefore, if $\boldsymbol{e}$ is uncorrectable for W_i, its complement $\overline{\boldsymbol{e}}$ has to be correctable for $W_{\ell-1-i}$. □

With the help of Lemma 5, we can prove the following lemma. We make the remark that, later we are going to show the inequality in the following lemma actually holds as equality.

Lemma 6. *If K_ℓ is self-dual, then*

$$\forall_i \, \forall_w : \quad E_{i,w} + E_{\ell-1-i,\ell-w} \leqslant \binom{\ell}{w} \tag{3.7}$$

Proof. Recall that $E_{i,w}$ counts the number of uncorrectable erasure patterns with weight w for W_i, and $E_{\ell-1-i,\ell-w}$ counts the number of uncorrectable erasure patterns with weight $(\ell - w)$ for $W_{\ell-1-i}$. According to Lemma 5, every uncorrectable erasure pattern $\boldsymbol{e}$ with weight w for W_i gives us a correctable erasure pattern $\overline{\boldsymbol{e}}$ with weight $(\ell - w)$ for $W_{\ell-1-i}$. Thus among all $\binom{\ell}{\ell-w}$ erasure patterns of weight $(\ell - w)$, the number of correctable erasure patterns for $W_{\ell-1-i}$ is at least $E_{i,w}$, so the number of uncorrectable erasure patterns $W_{\ell-1-i}$ is at most $\binom{\ell}{\ell-w} - E_{i,w}$:

$$E_{\ell-1-i,\ell-w} \leqslant \binom{\ell}{\ell - w} - E_{i,w}$$

This gives us (3.7). □

Now, we are ready to prove the duality theorem (Theorem 11) using the result in Lemma 6.

Proof for Theorem 11. Let K_ℓ be a self-dual polarization kernel, and let $i \in \{0,1,\cdots,\ell-1\}$. We can expand the expression $f_i(z) + f_{\ell-1-i}(1-z)$ as follows:

$$\begin{aligned}
f_i(z) + f_{\ell-1-i}(1-z) &= \sum_{w=0}^{\ell} E_{i,w} z^w (1-z)^{(\ell-w)} + \sum_{w=0}^{\ell} E_{\ell-1-i,w} (1-z)^w z^{(\ell-w)} && (3.8)\\
&= \sum_{w=0}^{\ell} E_{i,w} z^w (1-z)^{(\ell-w)} + \sum_{w'=0}^{\ell} E_{\ell-1-i,\ell-w'} z^{w'} (1-z)^{(\ell-w')} && (3.9)\\
&= \sum_{w=0}^{\ell} (E_{i,w} + E_{\ell-1-i,\ell-w}) z^w (1-z)^{(\ell-w)} && (3.10)\\
&\leqslant \sum_{w=0}^{\ell} \binom{\ell}{w} z^w (1-z)^{(\ell-w)} && (3.11)\\
&= 1 && (3.12)
\end{aligned}$$

where

- in (3.8), we expand $f_i(z)$ and $f_{\ell-1-i}(1-z)$ according to equation (3.4).
- in (3.9), we replace w with $w' = \ell - w$ in the second summation.
- in (3.11), we invoke the result in Lemma 6.

Since the above inequality $f_i(z) + f_{\ell-1-i}(1-z) \leqslant 1$ holds for all $i = 0,1,\cdots,\ell-1$, we have

$$\sum_{i=0}^{\ell-1} \left(f_i(z) + f_{\ell-1-i}(1-z)\right) \leqslant \ell \tag{3.13}$$

On the other hand, by the definition of the bit-channels in (3.1), the mutual information is preserved under the polarization of kernel K_ℓ:

$$\sum_{i=0}^{\ell-1} I(W_i) = \ell \cdot I(W) \quad \Rightarrow \quad \sum_{i=0}^{\ell-1} f_i(z) = \ell z$$

Thus we have

$$\sum_{i=0}^{\ell-1} \left(f_i(z) + f_{\ell-1-i}(1-z)\right) = \ell z + \ell(1-z) = \ell$$

Therefore, the inequality in (3.13) must holds as equality, which implies that all the inequalities in (3.11) should hold equal as well. This improves the result in Lemma 6 to equality:

$$\forall_i \, \forall_w : \quad E_{i,w} + E_{\ell-1-i,\ell-w} = \binom{\ell}{w}$$

And it gives us

$$\forall_i : \quad f_i(z) + f_{\ell-1-i}(1-z) = 1,$$

which completes this proof. □

3.3.3 Constructing Self-dual Kernels with Small Scaling Exponents

In this subsection, we propose an approach to construct large polarization kernels with small scaling exponents. We start by explaining the intuition behind our approach. In a recent work by Fazeli, Hassani, Mondelli and Vardy [FHMV20], they use a probabilistic method to show that random large polarization kernel achieves near optimal scaling exponent with high probability as the size of the kernel $\ell \to \infty$. In their proof, one of the key step is to show that with high probability, for all $i = 0, 1, \cdots, \ell - 1$, the polynomial $f_i(z)$ in the polarization behavior of a random kernel K_ℓ has sharp transition from 0 to 1 in the vicinity of $z = i/\ell$. More precisely, as shown in Figure 3.6, the polynomial $f_i(z)$ in an asymptotically-optimal polarization behavior should exhibit the following behavior:

1) The value of $f_i(z)$ is close to zero when $z \leqslant i/\ell - \epsilon$ for a diminishing ϵ.

2) The sharp transition happens in the vicinity of $z = i/\ell$.

3) The value of $f_i(z)$ is close to one when $z \geqslant i/\ell + \epsilon$ for a diminishing ϵ.

Inspired by this result, we would like to imitate the polarization behavior for random large kernels in our kernel construction. Specifically, our intuitive goal is to produce $f_i(z)$'s with transitions as sharp as possible. This goal turns out to be consistent with symmetry property of self-dual

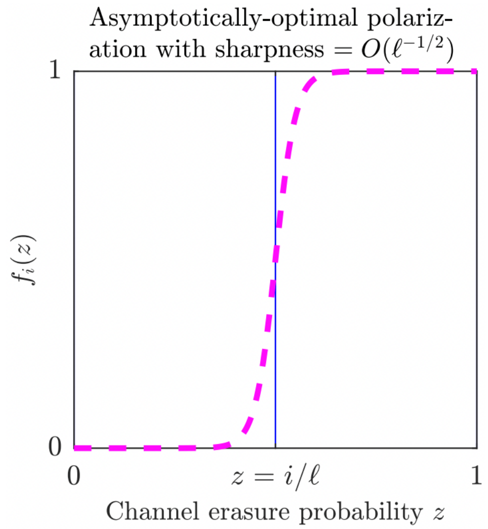

Figure 3.6. The sharp transition of the polynomial $f_i(z)$ in an asymptotically-optimal polarization behavior. (figure copied from [FHMV20])

kernels shown by the the duality theorem (Theorem 11). Consider a self-dual kernel K_ℓ, if one of its polynomial $f_i(z)$ has a sharp transition in the vicinity of i/ℓ, then by the duality theorem, we also know that $f_{\ell-1-i}(z)$ has a sharp transition in the vicinity of $(\ell - i)/\ell$ as desired. Therefore, if we focus on constructing a self-dual kernel, it suffices to design only half of its polynomials $f_0(z), f_1(z), \cdots, f_{\ell/2-1}(z)$ with sharp transitions. The other half will automatically have sharp transitions as well due to symmetry.

Following this idea, our strategy is to construct half of a self-dual kernel starting from the bottom. We propose a greedy construction that designs the kernel rows one at a time by maximizing the *partial distances* at each step. The definition for partial distances is given as follows.

Definition 14. *For an $\ell \times \ell$ polarization kernel K_ℓ with rows $g_0, g_1, \cdots, g_{\ell-1}$, the partial distance d_i between row g_i and the kernel code $\mathcal{C}_{i+1}$ is defined as*

$$d_i = d_H(g_i, \mathcal{C}_{i+1}),$$

for all $i = 0, \ldots, \ell - 1$, where d_H stands for Hamming distance.

For a polynomial $f_i(z)$ in the polarization behavior, if we expand it using equation (3.4), according to Theorem 10, the first non-zero coefficient will be E_{i,d_i}, with d_i being the partial distance between the i-th row and $\mathcal{C}_{i+1}$:

$$f_i(z) = \sum_{w=0}^{\ell} E_{i,w} z^w (1-z)^{(\ell-w)} = E_{i,d_i} z^{d_i} (1-z)^{(\ell-d_i)} + \cdots$$

Therefore, when z is close to 0, the value of $f_i(z)$ will be dominated by its first term $E_{i,d_i} z^{d_i} (1-z)^{(\ell-d_i)}$. So to push the value of $f_i(z)$ close to zero when z is small, we would like to maximize this partial distances d_i for every row. And we will do it greedily in our kernel design.

In our kernel construction, we build an $\ell \times \ell$ self-dual polarization kernel K_ℓ with the following steps. First, we focus on the bottom half of kernel, and we start from the last kernel row. For $i = \ell-1, \ell-2, \cdots, \ell/2$, we pick the i-th row of the kernel g_i to be a binary vector with $g_i \perp \mathcal{C}_{i+1}$ that maximizes the partial distance

$$d_i = d_H(g_i, \mathcal{C}_{i+1}).$$

We need the restriction $g_i \perp \mathcal{C}_{i+1}$ since this has to hold if we want K_ℓ to be self-dual. This requirement comes from the fact that, if K_ℓ is self-dual, when $i \geqslant \ell/2$, we have

$$\begin{cases} \mathcal{C}_{i+1} = \mathcal{C}_{\ell-i}^{\perp} \\ g_i \in \mathcal{C}_{\ell-i} \end{cases} \quad \Rightarrow \quad g_i \perp \mathcal{C}_{i+1}$$

Following this approach, we pick $g_{\ell-1}$ to be the all-one vector, pick $g_{\ell-2}$ to be the vector with Hamming weight $\ell/2$, so on and so forth. If at step i, there are multiple choices of g_i, we can pick any one of them to be the i-th row in the kernel. This construction procedure is illustrated in Figure 3.7. After we construct the bottom half of K_ℓ, its top half can be filled uniquely following its self-dual constraint.

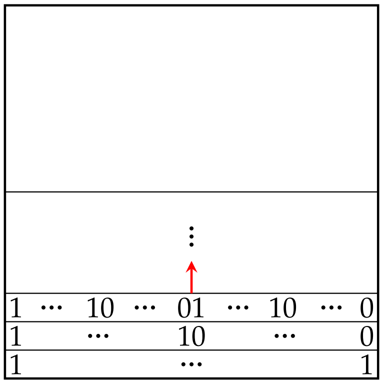

Figure 3.7. The bottom half of the self-dual kernel is constructed one row at a time, starting from the bottom.

Table 3.1. Kernel codes of K_{32} in its bottom half

rows	kernel codes	partial distances
31	$\mathcal{C}_{31} = \{\mathbf{0},\mathbf{1}\}$	32
27-30 26	subcodes of $\mathcal{C}_{26}$ $\mathcal{C}_{26} =$ RM(1,5)	16 16
22-25 21	subcodes of $\mathcal{C}_{11}$ $\mathcal{C}_{11} =$ extended BCH(31,11,11)	12 12
17-20 16	subcodes of $\mathcal{C}_{16}$ $\mathcal{C}_{16} =$ RM(2,5)	8 8

Example 9. In this example, we use the described kernel construction approach to design a 32×32 kernel K_{32}. The kernel codes that we pick for the bottom half of K_{32} are listed in Table 3.1. We start by picking the last row g_{31} to be the all-one vector. Then for rows 26-30, we pick codewords in RM(1,5) to obtain the maximum partial distance 16. After that, we carefully select codewords in the extended BCH codes BCH(31,11,11) for rows 21-25 to obtain the maximum partial distance 12. For rows 16-20, we pick the codewords in RM(2,5) to achieve the maximum partial distance 8. In the middle, the kernel code $\mathcal{C}_{17}$ turns out to be the self-dual code RM(2,5) exactly. We finish our construction by filling up the top half of the kernel following the self-dual constraint. The constructed K_{32} is shown in Figure 3.8.

Example 10. In the second example, we use the described kernel construction approach to design a 64×64 kernel K_{64}. Kernel K_{64} is constructed similar to K_{32} in our last example, except

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0\\
1&0&0&1&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&0\\
0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&1&0&0&1&0&0&0&0\\
1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&1&0\\
1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&0&0&0&1&1&1&1&0\\
1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&1&0&1&1&1&1&1&0\\
1&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&0&0&1&0&1&1&0&1&0&1&1&0&0\\
1&1&0&1&0&0&1&0&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&1&0&1&1&0&1&0&0\\
1&0&1&1&0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&0&1&1&0&1&0&0&1&0&0&1&0&0\\
0&0&0&1&1&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&1&1&0&1&1&0&0&0&1&1&1&0\\
1&1&1&1&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0&1&0&0&1&1&1&1&0&0\\
0&1&1&0&1&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&1&1&0&1&1&0\\
1&1&0&1&1&0&0&0&0&0&0&1&0&1&0&0&1&0&0&0&1&1&0&1&1&0&1&1&1&1&1&0\\
0&0&0&1&0&0&1&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&1&1&0&0&1&0&1&1&1&0\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}$$

Figure 3.8. Kernel K_{32}

Table 3.2. Kernel codes of K_{64} in its bottom half

rows	kernel codes	partial distances
63	$\mathcal{C}_{63} = \{\mathbf{0}, \mathbf{1}\}$	64
58-32	subcodes of $\mathcal{C}_{57}$	32
57	$\mathcal{C}_{57} =$ RM(1,6)	32
55-56	subcodes of $\mathcal{C}_{54}$	28
54	$\mathcal{C}_{54} =$ extended BCH(63,10,27)	28
49-53	subcodes of $\mathcal{C}_{48}$	24
48	$\mathcal{C}_{48} =$ extended BCH(63,16,23)	24
43-47	subcodes of $\mathcal{C}_{42}$	16
42	$\mathcal{C}_{42} =$ RM(2,6)	16
37-41	subcodes of $\mathcal{C}_{36}$	16
36	$\mathcal{C}_{36} =$ extended cyclic(63,28,15)	16
35	$\mathcal{C}_{35} =$ (64,29,14) linear code	14
34	$\mathcal{C}_{34} =$ (64,30,12) linear code	12
33	$\mathcal{C}_{33} =$ (64,31,12) linear code	12
32	$\mathcal{C}_{32} =$ (64,32,12) linear code	12

that row 32 to row 35 are picked via computer search. The kernel codes that we pick for the bottom half of K_{64} are listed in Table 3.2. The constructed K_{64} is shown in Figure 3.9.

$$\left[\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}\right]$$

Figure 3.9. Kernel K_{64}

3.4 Computing the Polarization Behavior

In this section, we address the last challenge: computing the scaling exponents for our constructed polarization kernels. Following the method described in Subsection 3.2.4, we can compute the scaling exponent for large kernels if we can obtain their polarization behaviors. However, it has been established in [FV14] that computing the polarization behavior for a general large kernel is NP-hard. In the work by Miloslavskaya and Trifonov [MT12], they introduce an algorithm based on the binary decision diagram (BDD) to compute the polarization behavior for larger kernels. In this section, we propose a trellis algorithm that has significantly less complexity compared with the BDD algorithm. Our trellis algorithm is still exponential, but it is efficient enough to let us compute the entire polarization behavior of K_{32}, and part of the polarization behavior of K_{64}.

3.4.1 The Proper Trellis Algorithm

A trellis T is a graphical representation of a block code, in which every path in this graph represents a codeword. It consists of a vertex set V, an edge set E, and a labeling function $L : E \to \{0,1\}$. The vertex set V can be partitioned as $V = V_0 \cup V_1 \cup \cdots \cup V_n$, with n being the block length of the code. Every edge denoted as $e = (v_j, v_{j+1}) \in E$ in the trellis connects two vertices from $v_j \in V_j$ to $v_{j+1} \in V_{j+1}$, and it is labeled with a single bit $L(e) \in \{0,1\}$ at coordinate j of a codeword. Hereforth, we assume the readers are familiar with the basic concept of trellis and the Viterbi algorithm. We refer the readers to [Var98] for a comprehensive introduction on the trellis theory that we are going to use in this section.

If we construct a trellis for a block code, we can then do maximum-likelihood decoding of the code by running the Viterbi algorithm on this trellis. Besides decoding, the Viterbi algorithm can also be adjusted to compute the weight distribution of the code, given that the trellis is *one-to-one*. A trellis is called one-to-one if all of its paths are labeled distinctively. Therefore, our idea for computing a polynomial $f_i(z)$ in the polarization behavior, is to first construct a one-to-one trellis for the cover set $\Delta(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$, and then use the Viterbi algorithm to compute the weight distribution $E_{i,0}, E_{i,1}, \cdots, E_{i,\ell}$ of this cover set on the trellis.

However, the cover set $\Delta(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$ that we are interested is typically both non-linear, and non-triangular, which makes it difficult to build a trellis representation directly. Instead, we first construct a trellis that is *not* one-to-one for the cover set, and then propose an algorithm that can convert it into a *proper* trellis. A trellis is called *proper* if its edges beginning at any given vertex are labeled distinctly, and it's known that a proper trellis is one-to-one. In general, our proposed algorithm can convert any non-proper trellis into a proper trellis, which might be of independent interests.

Let K_ℓ denote a polarization kernel with rows $g_0, g_1, \cdots, g_{\ell-1}$, our proper trellis algorithm has the following four steps:

1. First, we construct a canonical minimal trellis representing the kernel code $\mathcal{C}_{i+1}$, using one of several methods described in [Var98, Section 4.2].
2. Then, for every edge e connecting vertices from V_j to V_{j+1} with $j \in \mathrm{supp}(g_i)$, we flip its label $L(e)$. In this way, we get a minimal trellis representing
$$g_i + \mathcal{C}_{i+1} = \mathcal{C}_i \backslash \mathcal{C}_{i+1}$$
3. Next, for every edge e labeled with $L(e) = 0$, we add a parallel edge e' labeled with bit $L(e') = 1$. After this step, the trellis will represent the cover set $\Delta(\mathcal{C}_i \backslash \mathcal{C}_{i+1})$, except that it is typically not a one-to-one trellis.
4. Finally, let $T = (V, E, A)$ denote the trellis after the third step, we use Algorithm 10 to convert T into a proper trellis $T^* = (V^*, E^*, A^*)$. In Algorithm 10, for $j = 0, 1, 2, \cdots, \ell$, vertices in V_i^* are labeled uniquely by the subsets of V_i. After this step, we will get a one-to-one trellis T^* representing the cover set $\Delta(\mathcal{C}_{i-1} \backslash \mathcal{C}_i)$, where we can apply the Viterbi algorithm to compute its weight distribution $E_{i,0}, E_{i,1}, \cdots, E_{i,\ell}$.

Example 11. In this example, we illustrate the steps in our proper trellis algorithm for the cover set $\Delta(\mathcal{C}_1 \backslash \mathcal{C}_2)$ of kernel
$$K_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

```
Algorithm 3: Proper Trellis Conversion
  Input: trellis $T = (V, E, L)$ representing a block code of length $n$, with
         $V = V_0 \cup V_1 \cup \cdots \cup V_n$
  Output: proper trellis $T^* = (V^*, E^*, L^*)$ representing the same block code as $T$
  Initialize $V^* = V_0^* \cup V_1^* \cup \cdots \cup V_n^*$ with $V_0^* = \{V_0\}$ and
  $V_1^* = V_2^* = \cdots = V_n^* = \emptyset$
  for $i = 0$ to $n - 1$ do
      for every vertex $v_i^* \in V_i^*$ do
          for $a \in \{0, 1\}$ do
              calculate $s = \{v_{i+1} \in V_{i+1} : \exists v_i \in L(v_i^*), (v_i, v_{i+1}, a) \in E\}$
              if $\exists v_{i+1}^* \in V_{i+1}^*$ with $L(v_{i+1}^*) = s$ then
                  add an edge $(v_i^*, v_{i+1}^*, a)$ in $E^*$
              else
                  add a vertex $v_{i+1}^* \in V_{i+1}^*$ with $L(v_{i+1}^*) = s$
                  add an edge $(v_i^*, v_{i+1}^*, a)$ in $E^*$
```

In the first step, we build a trellis representing $\mathcal{C}_2$, as shown in the top left position of Figure 3.10. In the second step, for edges in the first two coordinates, we flip their labels to produce the trellis representing

$$g_2 + \mathcal{C}_2 = \mathcal{C}_1 \backslash \mathcal{C}_2,$$

as shown at the top right position of Figure 3.10. In the third step, for every edge in the trellis labeled with 0, we add a parallel edge labeled with 1 to produce the trellis for the cover set $\Delta(\mathcal{C}_1 \backslash \mathcal{C}_2)$. This is shown at the bottom left position of Figure 3.10. At this point, the trellis is not one-to-one, so we cannot use it to run the Viterbi algorithm. In the final step, we use Algorithm 10 to convert the trellis we obtained into a proper trellis, as shown at the bottom right position of Figure 3.10. After that, we can run the Viterbi algorithm to compute the weight distribution of $\Delta(\mathcal{C}_1 \backslash \mathcal{C}_2)$, and get

$$E_{1,0} = 0, \quad E_{1,1} = 0, \quad E_{1,2} = 4, \quad E_{1,3} = 4, \quad E_{1,4} = 1,$$

According to equation (3.4), the polynomial $f_1(z)$ in the polarization behavior of K_4 can be then

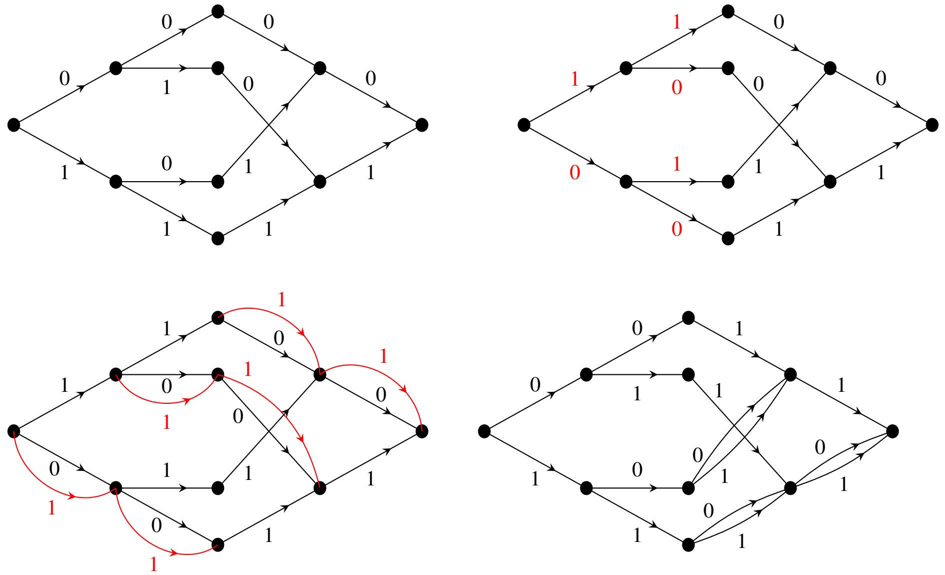

Figure 3.10. Trellises produced in four steps of the proper trellis algorithm in Example 11. The changes in the second step and the third step are colored in red. Top left: trellis for $\mathcal{C}_2$ in K_4; top right: trellis for $(\mathcal{C}_1 \backslash \mathcal{C}_2)$ in K_4; bottom left: trellis for $\Delta(\mathcal{C}_1 \backslash \mathcal{C}_2)$ after step 3, which is not one-to-one; bottom right: proper trellis for $\Delta(\mathcal{C}_1 \backslash \mathcal{C}_2)$ after applying Algorithm 10.

derived as

$$\begin{aligned} f_1(z) &= \sum_{w=0}^{4} E_{1,w} z^w (1-z)^{(4-w)} \\ &= 4z^2(1-z)^2 + 4z^3(1-z) + z^4 \\ &= 4z^2 - 4z^3 + z^4 \end{aligned}$$

3.4.2 Computing the Polarization Behaviors of K_{32} and K_{64}

For our constructed kernel K_{32} shown in Figure 3.8, the proper trellis algorithm allows us to compute its full polarization behavior, displayed in Figure 3.3. Following the method described in Subsection 3.2.4, the scaling exponent of K_{32} is calculated as $\mu(K_{32}) = 3.122$. For our constructed kernel K_{64} shown in Figure 3.9, the proper trellis algorithm allows us to compute the top 15 polynomials and the bottom 15 polynomials in its polarization behavior,

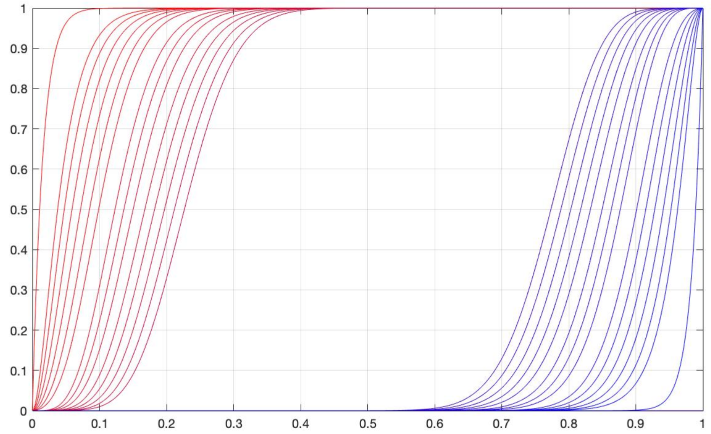

Figure 3.11. $f_0(z)$,$f_1(z)$,$\cdots$,$f_{14}(z)$ and $f_{49}(z)$,$f_1(z)$,$\cdots$,$f_{63}(z)$ in the polarization behavior of K_{64}

as shown in Figure 3.11. Note that since K_{64} is self-dual, computing the last 15 polynomials $f_{49}(z), f_1(z), \cdots, f_{63}(z)$ in its polarization behavior directly let us obtain the first 15 polynomials $f_0(z), f_1(z), \cdots, f_{14}(z)$ from symmetry (Theorem 11). For row 15 to row 48 in the middle of K_{64}, the complexity of our proper trellis algorithm becomes prohibitive, such that we are unable to obtain the full polarization behavior of K_{64}. In the next section, we introduce a Monte-Carlo interpolation method to complete Figure 3.11 by estimation. Using this Monte-Carlo interpolation method, we can get an estimate for the scaling exponent of K_{64}.

3.5 Monte-Carlo Interpolation Method

In this section, we introduce a Monte-Carlo algorithm to estimate the polynomials in the polarization behavior of large kernels. Using this method, we can complete Figure 3.11 for K_{64}.

Recall that $f_i(z)$ denotes the erasure probability for the i-th bit channel W_i, assuming that the underlying channel W is a BEC(z). A naive yet explicit approach to obtain the value of $f_i(z)$ is to cross check all 2^ℓ erasure patterns to discover the exact ratio of which become uncorrectable for W_i. Instead, we propose to take N samples of randomly generated erasure patterns, and estimate this ratio $f_i(z)$ accordingly. Recall that the computational complexity

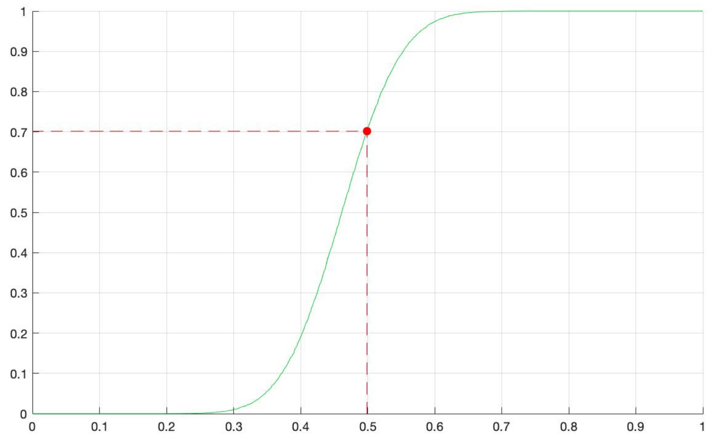

Figure 3.12. The estimated $\widehat{f}_{29}(z)$ in the polarization behavior of K_{64}

of determining "correctability" for a given erasure pattern is no more than the complexity of a MAP decoder for the BEC, which is bounded by $O(\ell^{\omega})$, where ω is the exponent of matrix multiplication. Therefore, to estimate the value of $f_i(z)$ for a single z, the overall complexity of the proposed approximation method can be bounded by $O(N\ell^{\omega})$.

Figure 3.12 shows the estimated $\widehat{f}_{29}(z)$ following this approach. To generate the curve in Figure 3.12, we first take 1000 z's evenly distributed within the interval $[0,1]$. Then, for every z, we estimate the value of $f_{29}(z)$ as $\widehat{f}_{29}(z)$ by taking $N = 10^6$ randomly generated erasure patterns from BEC(z), and check how many of them are correctable for W_i. The red dot in Figure 3.12 shows one of those estimated points in the curve. After that, we connect those 1000 estimated points with linear interpolation to obtain an entire curve for $\widehat{f}_{29}(z)$. We call it the Monte-Carlo interpolation method.

Using this method, we are able to estimate polynomials $f_{15}(z),\dots,f_{48}(z)$ in the polarization behavior of K_{64}, which completes Figure 3.11. The full polarization behavior of K_{64} is shown in Figure 3.13, where the green curves shows the estimated $\widehat{f}_{15}(z),\dots,\widehat{f}_{48}(z)$ obtained from the Monte-Carlo interpolation method.

While this approach adds some uncertainty to our derivations, the curves in Figure 3.13 suggest that those estimated $\widehat{f}_i(z)$'s become visibly smooth and stable at $N = 10^6$. Using

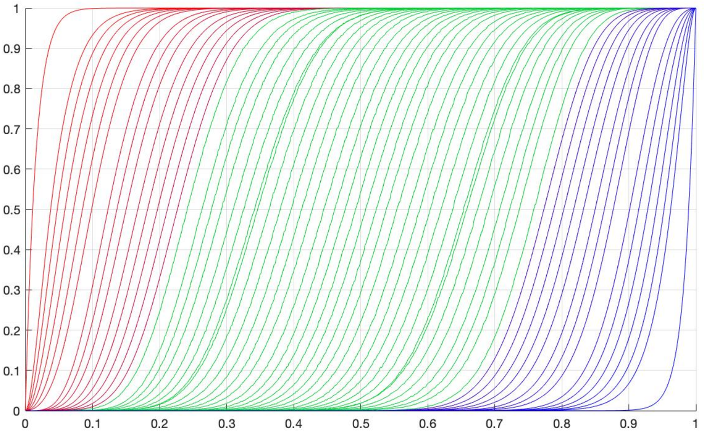

Figure 3.13. The polarization behavior of K_{64}, where the green curves are obtained from the Monte-Carlo interpolation method.

$\widehat{f}_{15}(z),\ldots,\widehat{f}_{48}(z)$ together with the polynomials that we have known as shown in Figure 3.13, we are able to invoking the method described in Subsection 3.2.4 to get $\mu(K_{64}) \simeq 2.87$.

3.6 A Proof that $\mu(K_{64}) < 3$

In this section, we support our estimation $\mu(K_{64}) \simeq 2.87$ with a rigorous proof that $\mu(K_{64}) \leqslant 2.9603$. This confirms that our constructed K_{64} has μ strictly under 3. We note that this proof doesn't depend on the scaling assumption described in Subsection 3.2.4. To the best of our knowledge, the large kernel polar codes constructed with our K_{64} form the first explicit family of codes with scaling exponent under 3.

3.6.1 An Upper Bound for the Scaling Exponent

Our approach of bounding $\mu(K_{64})$ follow the similar idea of the "first approach" described in Section IV of [HAU14]. Recall the evolution of Bhattacharyya parameter for the polarized bit channels can be modeled by a random process Z_n as described in Subsection 3.2.4. Our objective here is to find an upper bound $\overline{\mu}$ for the scaling exponent of K_{64}, by proving

$$\Pr(Z_n \in [a,b]) \leqslant O(\ell^{-\frac{n}{\overline{\mu}}}) \tag{3.14}$$

It has the following two steps:

1. Find a suitable test function g, and upper bound the speed of convergence for

$$g_n(z) = \mathbb{E}[g(Z_n) \mid Z_0 = z]$$

2. Turn this upper bound into a bound for the speed of convergence for $\Pr(Z_n \in [a,b])$.

For the first step, let g be any test function $g : [0,1] \to [0,1]$, define the sequence of functions $\{g_n\}_{n\in\mathbb{N}}$ as $g_n : [0,1] \to [0,1]$ such that

$$g_n(z) = \mathbb{E}[g(Z_n) \mid Z_0 = z]$$

This sequence of function satisfy the recursive relation:

$$g_0(z) = g(z), \qquad g_{n+1}(z) = \frac{1}{\ell}\sum_{i=0}^{\ell-1} g_n(f_i(z)) \tag{3.15}$$

Following the idea in Section IV of [HAU14], the speed of decay of sequence $\{g_n\}_{n\in\mathbb{N}}$ can be upper bounded through a sequence of numbers $\{b_n\}_{n\in\mathbb{N}}$, defined as

$$b_n = \sup_{z\in(0,1)} b_n(z), \qquad \text{where} \quad b_n(z) = \frac{g_{n+1}(z)}{g_n(z)} \tag{3.16}$$

From the definition of $\{b_n\}_{n\in\mathbb{N}}$ in (3.16), we have

$$g_n(z) \leqslant b_{n-1}g_{n-1}(z) \leqslant (b_{n-1}b_{n-2})g_{n-2}(z) \leqslant \cdots \leqslant (b_{n-1}b_{n-2}\cdots b_0)g_0(z) \tag{3.17}$$

Similar to the proof of Lemma 3 in [HAU14], it can be proved that $\{b_n\}_{n\in\mathbb{N}}$ is a decreasing sequence: $b_0 \geqslant b_1 \geqslant b_2 \geqslant \cdots$. Therefore, from (3.17) we have

$$g_n(z) \leqslant (b_{n-1}b_{n-2}\cdots b_0)g_0(z) \leqslant b_0^n g_0(z) = b_0^n g(z)$$

Therefore, the speed of decay of sequence $\{g_n\}_{n\in\mathbb{N}}$ can be upper bounded using b_0. For the second step, we turn this upper bound into a bound for the convergence of $\Pr(Z_n \in [a,b])$. This part is similar to the proof of Corollary 7 in [HAU14].

By Markov inequality:

$$\begin{aligned}\Pr(Z_n \in [a,b]) &\leqslant \Pr\Big(g(Z_n) \geqslant \min_{z\in[a,b]} g(z)\Big)\\ &\leqslant \frac{\mathbb{E}[g(Z_n)]}{\min_{z\in[a,b]} g(z)}\end{aligned}$$

Therefore,

$$\begin{aligned}\Pr(Z_n \in [a,b]) &\leqslant \frac{(b_0)^n g(z)}{\min_{z\in[a,b]} g(z)}\\ &\leqslant \ell^{n\log_\ell b_0 + O(1)}\\ &\leqslant O\left(\ell^{-\frac{n}{-1/(\log_\ell b_0)}}\right)\end{aligned}$$

Referring to (3.14), we thus have the upper bound

$$\overline{\mu} = -\frac{1}{\log_\ell b_0}$$

Now, all that remains is choosing a suitable test function g.

3.6.2 Constructing the Test Function

Recall the polynomial $f_i(z)$ in the polarization behavior can be expressed as

$$f_i(z) = \sum_{w=0}^{\ell} E_{i,w} z^w (1-z)^{(\ell-w)}$$

where $E_{i,w}$ denotes the number of uncorrectable erasure patterns of weight w for W_i. For K_{64}, using our trellis algorithm described in Section 3.4, we are able to explicitly compute the first and last 15 polynomials in its polarization behavior. There are 34 polynomials $f_{15}(z), \ldots, f_{48}(z)$ in the middle that we don't know. However, we can obtain the upper bounds and the lower bounds of those polynomials with the following Theorem in [MV20].

Theorem 12 (Theorem 1 in [MV20])**.** *Let $A_0, A_1, \cdots, A_\ell$ denote the weight enumerators for the coset $(C_i \backslash C_{i+1})$, then*

$$E_{i,w} \leqslant \min\left(\sum_{j=1}^{i} \binom{\ell-i}{j-i} A_i, \binom{\ell}{i}\right)$$

For $i = 15, 16, \ldots, 48$ and $w = 0, 1, \ldots, \ell$, define

$$\overline{E}_{i,w} = \min\left(\sum_{j=1}^{i} \binom{\ell-i}{j-i} A_i, \binom{\ell}{i}\right), \quad \text{and} \quad \underline{E}_{i,w} = \binom{\ell}{i} - \overline{E}_{\ell-1-i,\ell-w}$$

Then we have the upper bounds and the lower bounds for all the unknown $E_{i,w}$'s as

$$\overline{E}_{i,w} \geqslant E_{i,w} \geqslant \underline{E}_{i,w}$$

Define

$$\overline{f}_i(z) = \sum_{i=0}^{\ell} \overline{E}_{i,w} z^i (1-z)^{\ell-i}, \qquad \text{and} \qquad \underline{f}_i(z) = \sum_{i=0}^{\ell} \underline{E}_{i,w} z^i (1-z)^{\ell-i}$$

Then we have the upper bounds and the lower bounds for all the unknown polynomials as

$$\overline{f}_i(z) \geqslant f_i(z) \geqslant \underline{f}_i(z)$$

Therefore, by computing the weight distribution of $(C_i \backslash C_{i+1})$, we can get an upper bound and a lower bound for $f_i(z)$. An example is shown on the left of Figure 3.14.

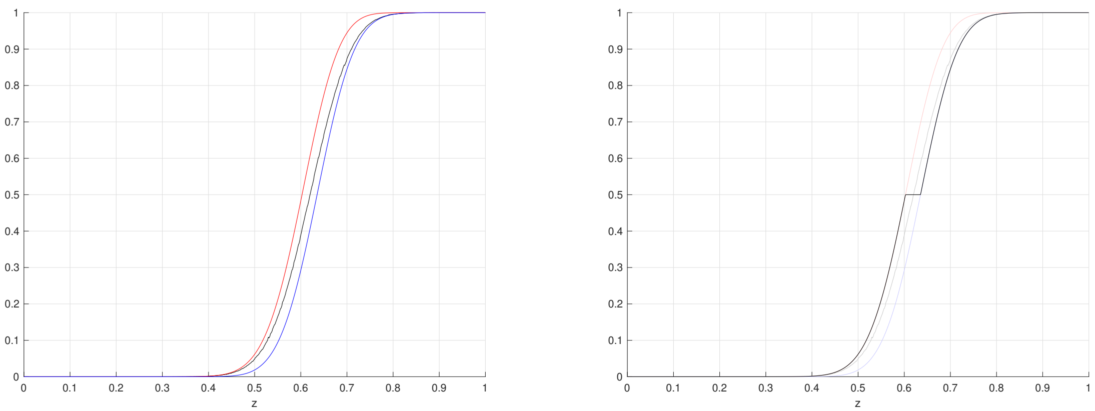

Figure 3.14. Left: $\overline{f}_{39}(z)$ (in red), $\widehat{f}_{39}(z)$, and $\underline{f}_{39}(z)$ (in blue). **Right**:$\tilde{f}_{39}(z)$ (highlighted)

Now, we give the definition for our carefully designed test function g:

$$g(z) = \frac{1}{64}\left(\sum_{i=0}^{14} g^*(f_i(z)) + \sum_{i=49}^{63} g^*(f_i(z)) + \sum_{i=15}^{48} g^*(\tilde{f}_i(z))\right), \qquad g^*(z) = z^{1/2}(1-z)^{1/2}$$

where

$$\tilde{f}_i(z) = \begin{cases} \overline{f}_i(z) & \overline{f}_i(z) \leqslant 0.5 \\ 0.5 & 0.5 \in (\overline{f}_i(z), \underline{f}_i(z)) \\ \underline{f}_i(z) & \underline{f}_i(z) \geqslant 0.5 \end{cases}$$

This test function is shown on the left of Figure 3.15. In the definition of g, we use the first and the last 15 polynomials that we have computed in the polarization behavior of K_{64}. For the $f_i(z)$ that we don't know, we design a new function $\tilde{f}_i(z)$ using its upper bound $\overline{f}_i(z)$ and lower bound $\underline{f}_i(z)$. An example $\tilde{f}_{39}(z)$ is shown on the right of Figure 3.14.

It can be shown that this explicitly defined test function g is increasing on $[0, 0.5]$, decreasing on $[0.5, 1]$, and reaches its maximum when $z = 0.5$. This allows us to upper bound

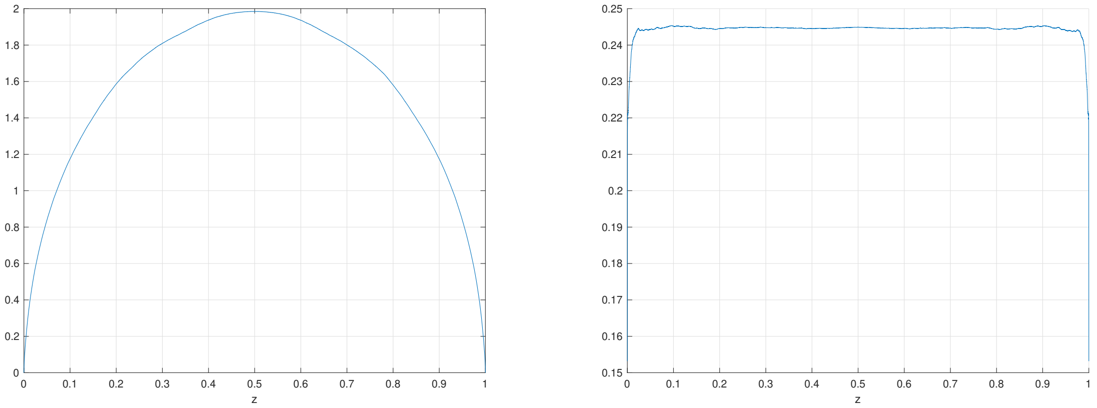

Figure 3.15. Left: test function $g(z)$. **Right**: upper bound $\overline{b}_0(z)$.

$g_1(z)$ using the recursive relation in (3.15):

$$g_1(z) = \frac{1}{64}\left(\sum_{i=1}^{64} g(f_i(z))\right) \leqslant \underbrace{\frac{1}{64}\left(\sum_{i=1}^{15} g(f_i(z)) + \sum_{i=50}^{64} g(f_i(z)) + \sum_{i=16}^{49} g(\tilde{f}_i(z))\right)}_{\text{defined as } \overline{g}_1(z)}$$

Therefore, a strict upper bound $\overline{b}_0(z)$ for $b_0(z)$ can be defined as:

$$b_0(z) = \frac{g_1(z)}{g(z)} \quad \leqslant \quad \overline{b}_0(z) = \frac{\overline{g}_1(z)}{g(z)}$$

This upper bound $\overline{b}_0(z)$ is shown on the right of Figure 3.15. The maximum value of $\overline{b}_0(z)$ can be calculated analytically up to any desired precision. Our calculation shows that:

$$\max_{z\in(0,1)} \overline{b}_0(z) = 0.2454$$

So

$$b_0 = \sup_{z\in(0,1)} b_0(z) \leqslant \max_{z\in(0,1)} \overline{b}_0(z) = 0.2454$$

This provides an upper bound for the scaling exponent of K_{64} as

$$\mu(K_{64}) \leqslant -\frac{1}{\log_{64} 0.2454} = 2.9603$$

This completes our proof.

3.7 Acknowledgements

This chapter, in part, has been published at 2019 IEEE International Symposium on Information Theory (ISIT) and appeared as: Hanwen Yao, Arman Fazeli, and Alexander Vardy "Explicit Polar Codes with Small Scaling Exponent" [YFV19]. The dissertation author was the primary author of this conference paper.

Chapter 4

Successive Cancellation Decoding for Large Kernel Polar Codes

4.1 Introduction

Polar codes, pioneered by Arıkan [Arı09], give rise to the first explicit family of codes that provably achieve capacity for a wide range of channels. However, the performance of polar codes at finite block lengths turns out to be mediocre. One reason is that polar codes approach channel capacity at a rather slow speed. This is reflected by its finite-length scaling properties. For a family of capacity-achieving codes, scaling exponent describes how the gap between the code rate and the channel capacity vanishes as a function of the block length. The smaller the scaling exponent, the faster this family of codes approaches channel capacity. It has been shown by a series of papers [GB14, HAU14, KMTU10, MHU16, WLVG22] that polar codes have scaling exponent $\mu = 3.627$ for binary erasure channels (BEC), and scaling exponent $3.579 \leqslant \mu \leqslant 4.63$ for general binary memoryless symmetric (BMS) channels. These numbers are far from the optimal scaling exponent $\mu = 2$ [Str62], which can be achieved by random linear codes [PPV10, Hay09].

Arıkan's polar codes are constructed based on the Kronecker product of a 2×2 binary matrix. As shown in the last chapter, one way to improve the scaling exponent for polar codes, and thus improving its finite-length performances, is by replacing Arıkan's size 2 matrix with some larger binary square matrices, called *polarization kernels*. Polar code with large kernels were shown to provide asymptotically optimal scaling exponents [FHMV20] as the size of the kernel goes to infinity. Recently, plenty of polarization kernels of size 16, size 32, and size 64 have been proposed with good scaling properties [FV14, PSL$^+$15, BFS$^+$17a, TT18, YFV19]. In particular, as shown in the last chapter, a 64×64 polarization kernel K_{64} is constructed in [YFV19], with $\mu \approx 2.87$, providing construction for the first explicit family of codes with scaling exponent under 3. However, decoding large kernel polar code is generally believed to be impractical due to its high computational complexity. Conventional polar code of length n admits successive cancellation (SC) decoding with complexity $O(n \log n)$. For a length-n polar code constructed with a $\ell \times \ell$ kernel, straightforward SC decoding requires $O(2^{\ell} n \log_{\ell} n)$ computational complexity. This means by employing a size ℓ polarization kernel, we can reduce the scaling exponent, but at the same time also introduce an extra 2^{ℓ} complexity coefficient on the decoding process. In the asymptotic regime, the coefficient 2^{ℓ} is just a constant. But in the finite-length regime, this coefficient turns out to be enormous for kernels of relatively large sizes.

4.1.1 Related Prior Works

Reducing the decoding complexity for large kernel polar codes have been a subject of investigation in multiple prior works. In most cases, specific polarization kernels are identified for which low-complexity decoding is possible. Peter Trifonov first proposed the window processing algorithm and used it to decode polar codes constructed with non-binary Reed-Solomon (RS) kernels [Tri14]. Later in [TT18, TT19], with further complexity reduction, the window processing algorithm was used to decode other selected polarization kernels of size 16 and 32. Recently in [AV20], the authors propose to perform column permutation on the polarization kernels to additionally reduce the complexity for the window processing algorithm. Independently, a class of polarization kernels called *permuted* kernels were introduced in [BFS$^+$17a, BFS$^+$17b]. Decoding permuted kernels can be viewed as SC decoding of conventional polar codes with look-aheads. Also, there are other decoding algorithms for large kernel polar codes based on trellises proposed in [Tri19b, MT20]. But in general, the complexity of those algorithms stays high for arbitrary kernels of size 32 and size 64.

In our proposed algorithm, we employ a linear transformation technique that allows one to view *any* linear code as a polar code with dynamically frozen bits. Polar codes with dynamically frozen bits were first introduced in [TM13]. It has been shown in [TM13, FVY20] that any linear code can be represented as a polar code with dynamically frozen bits. This allows us to use successive cancellation list (SCL) decoding for polar code to perform *approximate* maximum-likelihood (ML) decoding for the linear code. The required list size varies depending on the given linear code.

4.1.2 Our Contribution

In this paper, we propose a new method to perform SC decoding for large kernel polar codes. Our method is motivated by the observation that, by representing any linear code as a polar code with dynamically frozen bits, one can employ polar list decoding with large enough list size to get a *good approximate* to ML decoding.

SC decoding for large kernel polar codes requires calculation on the probabilities of its bit channels. Similar to conventional polar codes, those bit channels follow a recursive relation, so this calculation boils down to computing the probabilities for bit channels of a single polarization kernel. This kernel-level computation can be shown equivalent to soft-output ML decoding on a single bit of a linear block code. In our proposed method, we first employ linear transformation techniques to represent the considered linear block code as a polar code with dynamically frozen bits, and then use a modified polar list decoder with a large enough list size to get an approximate value on the soft-output of the desired bit.

Assuming we are using a size ℓ kernel, and list size L for the approximation, the complexity of our proposed approach for kernel-level computation is the same as the list decoding complexity for length-ℓ polar codes, which is $O(L\ell\log_2\ell)$. This complexity depends on the list size L, but it's polynomial in the kernel size ℓ. Our simulation result shows that one can obtain good approximation with a relatively small list size even for a kernel of size 64. The proposed method enables us to decode polar codes constructed with arbitrary polarization kernels of size 32 and 64. In particular, for the first time, we are able to decode polar code constructed with the 64×64 kernel K_{64} designed in [YFV19], which has a scaling exponent $\mu \approx 2.87$.

4.1.3 Notations

We use the following notations throughout this chapter. We use bold letters like $\boldsymbol{u}$ to denote vectors, and non-bold letters like u_i to denote symbols within that vector. We let the indices for the symbols start from zero. For $\boldsymbol{u} = (u_0, u_1, \cdots, u_{n-1})$, we denote its subvector consists of symbols with indices from a to b as $\boldsymbol{u}_a^b = (u_a, u_{a+1}, \cdots, u_b)$. And we denote the concatenation of vector $\boldsymbol{u}$ and vector $\boldsymbol{v}$ as $(\boldsymbol{u}, \boldsymbol{v})$.

4.2 Preliminaries

4.2.1 Large Kernel Polar Codes

Assuming $n = \ell^m$, an (n,k) large kernel polar code is a binary linear block code generated by k rows of the polar transform matrix $G = D_\ell K^{\otimes m}$, where $K^{\otimes m}$ is an m-fold Kronecker product of an $\ell \times \ell$ binary matrix K with itself, and D_ℓ is a base-ℓ digit-reversal permutation matrix. In this chapter, we use D_ℓ to denote the digit-reversal permutation matrix, and save the letter B for later use. For Arıkan's conventional polar codes, K would be the 2×2 matrix

$$K_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix},$$

and D_ℓ would be the bit-reversal permutation matrix.

Consider a BMS channel $W : \mathcal{X} \to \mathcal{Y}$ as base channel with input alphabet $\mathcal{X} = \{0,1\}$ and output alphabet $\mathcal{Y}$, characterized by its transition probabilities $W(y|x)$ for all $x \in \mathcal{X}$, $y \in \mathcal{Y}$. For a size ℓ polarization kernel K which is not upper triangular under any column permutation, it's shown in [KŞU10] that the polar transform matrix $G = D_\ell K^{\otimes m}$ gives rise to bit channels

$W_m^{(i)}(\boldsymbol{y}_0^{n-1}, \boldsymbol{u}_0^{i-1}|u_i)$ with capacities approaching 0 or 1 as $n \to \infty$. The definition for the bit channels is given by

$$W_m^{(i)}(\boldsymbol{y}_0^{n-1}, \boldsymbol{u}_0^{i-1}|u_i) = \frac{1}{2^{n-1}} \sum_{\boldsymbol{u}_{i+1}^{n-1}} W^n(\boldsymbol{y}_0^{n-1}|(\boldsymbol{u}_0^{i-1}, u_i, \boldsymbol{u}_{i+1}^{n-1})G), \tag{4.1}$$

where W^n denotes n independent uses of channel W.

The encoding scheme for large kernel polar codes is given by $\boldsymbol{c} = \boldsymbol{u}G$, where just like conventional polar codes, $\boldsymbol{u}$ is a length-n binary input vector carrying k information bits, and $\boldsymbol{c}$ is the codeword for transmission. The positions of those k information bits in $\boldsymbol{u}$ are specified by an index set $\mathcal{A} \subseteq \{0, 1, \cdots, n-1\}$. The index set $\mathcal{A}$ is chosen such that the bit channels $W_m^{(i)}$'s for $i \in \mathcal{A}$ are the k bit channels with the largest channel capacities. The rest of the $n-k$ bits in $\boldsymbol{u}$ are frozen to certain fixed values, usually zeros.

4.2.2 SC Decoding of Large Kernel Polar Codes

On the receiver side, for i goes from 0 to $n-1$, the SC decoding algorithm successively estimate the i-th bit u_i, and decode it as $\widehat{u}_i$ based on the earlier decoded path $\widehat{\boldsymbol{u}}_0^{i-1}$ in the following way:

$$\widehat{u}_i = \begin{cases} \arg\max_{u_i \in \{0,1\}} W_m^{(i)}(\boldsymbol{y}_0^{n-1}, \widehat{\boldsymbol{u}}_0^{i-1}|u_i) & i \in \mathcal{A} \\ \text{frozen value} & i \notin \mathcal{A} \end{cases} \tag{4.2}$$

At the end of this process, the SC decoder returns the length-n estimated vector $\widehat{\boldsymbol{u}}$ as the decoded vector for $\boldsymbol{u}$.

To perform SC decoding, one has to calculate the probability $W_m^{(i)}(\boldsymbol{y}_0^{n-1}, \boldsymbol{u}_0^{i-1}|u_i)$ with a given channel output vector $\boldsymbol{y}_0^{n-1}$. Due to the Kronecker product structure for the polar transform matrix G, this probability can be calculated recursively. Assuming $i \mod \ell = \phi$ with $i = s\ell + \phi$, this probability for the bit channel can be calculated as

$$W_m^{(i)}(\boldsymbol{y}_0^{n-1},\boldsymbol{u}_0^{i-1}|u_i) = W_1^{(\phi)}(\boldsymbol{z}_0^{\ell-1},\boldsymbol{u}_{s\ell}^{s\ell+\phi-1}|u_i) \tag{4.3}$$

Here for the expression $W_1^{(\phi)}(\boldsymbol{z}_0^{\ell-1},\boldsymbol{u}_{s\ell}^{s\ell+\phi-1}|u_i)$ in (4.3), we are considering the bit channel for a single polarization kernel K, and we let $\boldsymbol{z}_0^{\ell-1}$ denotes a length-ℓ channel output vector for this single kernel. For the symbols in $\boldsymbol{z}_0^{\ell-1}$, the transition probabilities of the base channels are given by

$$Pr(z_j|u) = W_{m-1}^{(s)}(\boldsymbol{y}_{(j+1)(n/\ell)-1}^{j(n/\ell)},\boldsymbol{v}(j)_0^{s-1}|u), \quad u \in \{0,1\} \tag{4.4}$$

where $\boldsymbol{v}(j)_0^{s-1}$ is a length-s binary vector with

$$\boldsymbol{v}(j)_k = (\boldsymbol{u}_{k\ell}^{(k+1)\ell-1}K)_j$$

Therefore, with the recursive relation in (4.3), calculating the bit channel probabilities for large kernel polar codes boils down to computing the probabilities $W_1^{(i)}(\boldsymbol{y}_0^{\ell-1},\boldsymbol{u}_0^{i-1}|u_i)$ for a single polarization kernel K. Moreover, for SC decoding in the LLR domain, it suffices to compute the ratio

$$R^{(i)}(\boldsymbol{u}_0^{i-1},\boldsymbol{y}_0^{\ell-1}) \overset{\text{def}}{=} \frac{W_1^{(i)}(\boldsymbol{y}_0^{\ell-1},\boldsymbol{u}_0^{i-1}|u_i=0)}{W_1^{(i)}(\boldsymbol{y}_0^{\ell-1},\boldsymbol{u}_0^{i-1}|u_i=1)}. \tag{4.5}$$

In the context of analyzing large kernel polar codes, instead of looking at the probability $W_m^{(i)}(\boldsymbol{y}_0^{n-1},\boldsymbol{u}_0^{i-1}|u_i)$, it's convenient to consider

$$W_m^{(i)}((\boldsymbol{u}_0^{i-1},u_i)|\boldsymbol{y}_0^{n-1}) = \frac{W_m^{(i)}(\boldsymbol{y}_0^{n-1},\boldsymbol{u}_0^{i-1}|u_i)}{2W(\boldsymbol{y}_0^{n-1})} = \sum_{\boldsymbol{u}_{i+1}^{n-1}\in\{0,1\}^{\ell-i-1}} W^n((\boldsymbol{u}_0^i,\boldsymbol{u}_{i+1}^{n-1})G|\boldsymbol{y}_0^{n-1}), \tag{4.6}$$

which is the probability for path $\boldsymbol{u}_0^i$ given the channel output $\boldsymbol{y}_0^{n-1}$. The ratio in (4.5) can also be derived using those probabilities for the paths as

$$R^{(i)}(\boldsymbol{u}_0^{i-1},\boldsymbol{y}_0^{\ell-1}) = \frac{W_1^{(i)}((\boldsymbol{u}_0^{i-1},0)|\boldsymbol{y}_0^{\ell-1})}{W_1^{(i)}((\boldsymbol{u}_0^{i-1},1)|\boldsymbol{y}_0^{\ell-1})} \tag{4.7}$$

In this way, the task for computing $W_1^{(i)}(\boldsymbol{y}_0^{\ell-1},\boldsymbol{u}_0^{i-1}|u_i)$ equivalently becomes the task for computing the ratio $R^{(i)}(\boldsymbol{u}_0^{i-1},\boldsymbol{y}_0^{\ell-1})$ for a single kernel K. This task is referred as *kernel processing* in [TT21], and as *kernel marginalization* in [BL18]. If this kernel-level computation task takes complexity $O(T)$, then the overall SC decoding complexity for a length-n large kernel polar code will be $O(T \cdot n \log_\ell n)$.

Straightforward calculation for $W_1^{(i)}(\boldsymbol{u}_0^i|\boldsymbol{y}_0^{\ell-1})$ as in (4.6) takes exponential complexity $O(2^\ell)$. This complexity can be slightly reduced by methods in [HZZ$^+$18] or by methods in [BL18]. But in general, direct calculation stays prohibitive for kernels with relative large sizes. More efficient algorithms such as window processing [Tri14] and recursive trellis processing [Tri19b] have been proposed for computing $W_1^{(i)}(\boldsymbol{u}_0^i|\boldsymbol{y}_0^{\ell-1})$. Those algorithms are still exponential with respect to the kernel size ℓ, but with various improvements [TT19, AV20, TT21], they are efficient enough to process kernels of size 16 and some designed low-complexity kernels of size 32.

4.3 SCL-Approximation Algorithm for Large Kernels

In this section, we propose a new algorithm that uses successive cancellation list (SCL) decoders for polar codes [TV15] to estimate the ratio in (4.7). We call it *SCL-Approximation Algorithm*. First, we show that the problem of computing the ratio $R^{(i)}(\boldsymbol{u}_0^{i-1},\boldsymbol{y}_0^{\ell-1})$ in (4.7) can be transformed into the problem of soft-output ML decoding on a single bit of a linear block code.

4.3.1 Cancelling the Effect of the Preceding Bits

Let K be an $\ell \times \ell$ polarization kernel with $\ell = 2^t$ being a power of 2, and let

$$K = \begin{bmatrix} A^{(i-1)} \\ B^{(i)} \end{bmatrix}$$

where $A^{(i-1)}$ is defined to be the submatrix of K consisting of its rows with indices from 0 to $i-1$, and $B^{(i)}$ is defined to be the submatrix of K consisting of its rows with indices from i to $\ell - 1$. We start by expressing $W_1^{(i)}(\boldsymbol{u}_0^i|\boldsymbol{y}_0^{\ell-1})$ as

$$\begin{aligned} & W_1^{(i)}((\boldsymbol{u}_0^{i-1}, u_i)|\boldsymbol{y}_0^{\ell-1}) \\ = & \sum_{\boldsymbol{u}_{i+1}^{\ell-1}} W^\ell((\boldsymbol{u}_0^{i-1}, u_i, \boldsymbol{u}_{i+1}^{\ell-1})K|\boldsymbol{y}_0^{\ell-1}) \\ = & \sum_{\boldsymbol{u}_{i+1}^{\ell-1}} W^\ell(\boldsymbol{u}_0^{i-1}A^{(i-1)} + (u_i, \boldsymbol{u}_{i+1}^{\ell-1})B^{(i)}|\boldsymbol{y}_0^{\ell-1}) \end{aligned}$$

Since the base channel W is a BMS channel, for any $y \in \mathcal{Y}$ with $W(0|y) = p_1$ and $W(1|y) = p_2$, there exists a $\bar{y} \in \mathcal{Y}$ such that $W(0|\bar{y}) = p_2$ and $W(1|\bar{y}) = p_1$. So in the above expression for $W_1^{(i)}(\boldsymbol{u}_0^i|\boldsymbol{y}_0^{\ell-1})$, we can cancel out the impact of the earlier path $\boldsymbol{u}_0^{i-1}$ on $\boldsymbol{y}_0^{\ell-1}$ by replacing $\boldsymbol{y}_0^{\ell-1}$ with a new channel output vector $\boldsymbol{z}_0^{\ell-1}$, where

$$z_j = \begin{cases} y_j, & (\boldsymbol{u}_0^{i-1}A^{(i-1)})_j = 0 \\ \bar{y}_j, & (\boldsymbol{u}_0^{i-1}A^{(i-1)})_j = 1 \end{cases}$$

We remark that this $\boldsymbol{z}_0^{\ell-1}$ is a new defined output vector different from the one in (4.3). By defining this new vector $\boldsymbol{z}_0^{\ell-1}$, we have

$$W_1^{(i)}((\boldsymbol{u}_0^{i-1}, u_i)|\boldsymbol{y}_0^{\ell-1}) = \sum_{\boldsymbol{u}_{i+1}^{\ell-1}} W^{\ell}((u_i, \boldsymbol{u}_{i+1}^{\ell-1})B^{(i)}|\boldsymbol{z}_0^{\ell-1})$$

So the ratio in (4.7) can be expressed as

$$R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1}) = \frac{\sum_{\boldsymbol{u}_{i+1}^{\ell-1}} W^{\ell}((0, \boldsymbol{u}_{i+1}^{\ell-1})B^{(i)}|\boldsymbol{z}_0^{\ell-1})}{\sum_{\boldsymbol{u}_{i+1}^{\ell-1}} W^{\ell}((1, \boldsymbol{u}_{i+1}^{\ell-1})B^{(i)}|\boldsymbol{z}_0^{\ell-1})} \tag{4.8}$$

If we denote the linear code generated by $B^{(i)}$ as $\mathbb{C}(B^{(i)})$, and view $\boldsymbol{z}_0^{\ell-1}$ as the channel output after transmitting a codeword in $\mathbb{C}(B^{(i)})$ through the base channels, then for the expression in (4.8), we basically have $Pr(u_i = 0|\boldsymbol{z}_0^{\ell-1})$ in the numerator, and $Pr(u_i = 1|\boldsymbol{z}_0^{\ell-1})$ in the denominator. Therefore, computing the ratio $R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$ can be achieved by soft-output ML decoding on the first bit of the code $\mathbb{C}(B^{(i)})$ generated by $B^{(i)}$.

Since the polarization kernel K has full-rank, its submatrix $B^{(i)}$ has rank $\ell - i$, and $\mathbb{C}(B^{(i)})$ is a linear block code with length ℓ and dimension $\ell - i$. To compute this ratio $R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$ directly, one needs to check the probabilities for all the codewords in $\mathbb{C}(B^{(i)})$. This straightforward approach requires complexity $O(2^{\ell})$, which is exponential in the kernel size ℓ. In our algorithm, we propose to perform polar list decoding for $\mathbb{C}(B^{(i)})$, and approximate the ratio $R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$ by only checking the probabilities for the codewords in the list. To perform polar list decoding, we first need to represent $\mathbb{C}(B^{(i)})$ as a polar code with dynamically frozen bits.

4.3.2 Representation as Polar Codes with Dynamic Freezing

Polar codes with dynamically frozen bits, first introduced in [TM13], are polar codes where each of the frozen bit u_j is not fixed to be zero, but set to be a linear function of its preceding bits as $u_j = f_i(u_0, u_1, \cdots, u_{j-1})$. It has been shown [TM13, FVY20] that with linear operation techniques, *any* linear code can be encoded as a polar code with dynamically frozen bits.

We now represent $\mathbb{C}(B^{(i)})$ as a polar code with dynamically frozen bits, so that we can perform polar list decoding on top of it. Let $K_{\mathrm{A}} = BK_2^{\otimes t}$ be the Arıkan's polar transform matrix of size ℓ. We first define an $(\ell - i) \times \ell$ precoder matrix M as

$$M = B^{(i)} K_{\mathrm{A}} \tag{4.9}$$

It can be observed that K_{A} is invertible with $(K_{\mathrm{A}})^{-1} = K_{\mathrm{A}}$. So

$$B^{(i)} = MK_{\mathrm{A}} \tag{4.10}$$

Then with elementary row operations, we can transform M into a matrix M^* in reduced row echelon form (RREF). The relation between M and M^* is given by $M^* = TM$, where T is some $(\ell - i) \times (\ell - i)$ invertible matrix. By mutiplying T on both sides of (4.10), we get

$$TB^{(i)} = M^* K_{\mathrm{A}} \tag{4.11}$$

Denote $B^{(i)*} = TB^{(i)}$ as a new generator matrix. Since row operations preserve the linear space spanned by $B^{(i)}$, $B^{(i)*}$ generates the same code as $B^{(i)}$. Let $\boldsymbol{v}$ be a length-$(\ell - i)$ binary vectors with $(\ell - i)$ information bits. The encoding of $\boldsymbol{v}$ with the generator matrix $B^{(i)*}$ is given by

$$\boldsymbol{v}B^{(i)*} = \underbrace{\boldsymbol{v}M^*}_{\boldsymbol{w}} K_{\mathrm{A}} \tag{4.12}$$

Here M^* is in RREF, so vector $\boldsymbol{w} = \boldsymbol{v}M^*$ is a length-ℓ vector with $(\ell - i)$ information bits, and i dynamically frozen bits. Therefore, the encoding $\boldsymbol{v}B^{(i)*}$ can also be achieved by multiplying $\boldsymbol{w}$ with dynamically frozen bits with the Arıkan's polar transform matrix K_{A}. In this way, we represent $\mathbb{C}(B^{(i)})$ as a polar code with dynamically frozen bits.

The linear code $\mathbb{C}(B^{(i)})$ can be encoded either by $\boldsymbol{u}_i^{\ell-1}B^{(i)}$, or by $\boldsymbol{v}B^{(i)*}$. For the same codeword, the relation between $\boldsymbol{u}_i^{\ell-1}$ and $\boldsymbol{v}$ can be established by

$$\boldsymbol{u}_i^{\ell-1}B^{(i)} = \boldsymbol{v}B^{(i)*} \quad \Rightarrow \quad \boldsymbol{u}_i^{\ell-1} = \boldsymbol{v}T \tag{4.13}$$

This shows u_i that equals to the first bit of $\boldsymbol{v}T$. In other words, $u_i = (\boldsymbol{v}T)_0$.

4.3.3 Ratio Estimation via Polar List Decoding

By viewing $\mathbb{C}(B^{(i)})$ as a polar code with dynamically frozen bits, we can now perform SCL decoding in [TV15] for $\mathbb{C}(B^{(i)})$ to decode the vector $\boldsymbol{v}$. Assuming the list size we are using is L, at the end of the SCL decoding process, we will get a list of L different paths for $\boldsymbol{v}$. We denoted the set of those L paths as $P = \{\boldsymbol{v}^{[1]}, \boldsymbol{v}^{[2]}, \cdots, \boldsymbol{v}^{[L]}\}$.

Let's re-examine the expression in (4.8) for the ratio $R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$. To compute this ratio directly, we need to divide all codewords in $\mathbb{C}(B^{(i)})$ into two sets. The first set contains codewords encoded by $\boldsymbol{u}_i^{\ell-1}$ with $u_i = 0$, and the second set contains codewords encoded by $\boldsymbol{u}_i^{\ell-1}$ with $u_i = 1$. Then $R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$ can be computed as the ratio of the sums of the probabilities for codewords in those two sets.

In our SCL-Approximation Algorithm, instead of checking all the codewords in $\mathbb{C}(B^{(i)})$, after the SCL decoding process, we propose to only check those L codewords generated by paths in the list. We divide those L codewords into two sets depending on the values of u_i, and get an approximate value for the ratio as

$$\widehat{R}^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1}) = \frac{\sum_{\boldsymbol{v} \in P:(\boldsymbol{v}T)_0 = 0} W^{\ell}(\boldsymbol{v}B^{(i)*} | \boldsymbol{z}_0^{\ell-1})}{\sum_{\boldsymbol{v} \in P:(\boldsymbol{v}T)_0 = 1} W^{\ell}(\boldsymbol{v}B^{(i)*} | \boldsymbol{z}_0^{\ell-1})} \tag{4.14}$$

With a large enough list size L, empirically the polar list decoder is a good approximation for the ML decoder. Therefore, it is reasonable to expect that those L codewords in the list will capture majority of the probabilities, and thus $\widehat{R}^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$ in (4.14) will give us a precise enough approximation for $R^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$.

In this approach, the computation for the precoder matrix M^*, and the new generator matrix $B^{(i)*}$ can all be performed offline. So the complexity of this kernel-level computation is $O(L\ell \log_2 \ell)$, the same as the complexity of polar list decoding for length-ℓ polar codes. This complexity is polynomial in the kernel size ℓ. If we apply the SCL-Approximation Algorithm, the overall complexity of SC decoding for length-n large kernel polar polar codes will be $O(L\ell \log_2 \ell \cdot n \log_\ell n)$. The pseudo code summarizing our SCL-Approximation Algorithm is given in Algorithm 4.

4.4 Simulation Results

In this section, we show some simulation results on SC decoding of large kernel polar codes using our SCL-Approximation Algorithm. We consider two polarization kernels K_{32} and K_{64} from [YFV19]. The 32×32 kernel K_{32} has scaling exponent $\mu(K_{32}) = 3.122$, and the 64×64 kernel K_{64} has scaling exponent $\mu(K_{64}) \approx 2.87$. The large kernel polar codes considered here are all constructed with the Monte-Carlo simulation.

Figure 3.8 shows the simulation results of SC decoding for several $(1024, 512)$ polar codes on AWGN channels. The black line shows the SC decoding performance for conventional polar codes, and the color lines show the SC decoding performance with SCL-Approximation Algorithm for large kernel polar codes constructed with K_{32}. We can observe that as we increase the list size L for the SCL-Approximation Algorithm, the overall SC decoding performance gets better as expected. Since larger L is expected to gives us better approximation for the ratio in (4.7) for kernel-level computations. For K_{32}, our simulation results show that $L = 32$ is large enough to get us close to the performance limit of our approach. Thus it is reasonable to believe that for K_{32}, the SCL-Approximation Algorithm with $L = 32$ gives us a pretty precise approximation for the ratio in (4.7).

Figure 3.9 shows the simulation results of SC decoding for several $(4096, 2048)$ polar

Algorithm 4: SCL-Approximation Algorithm

```
Input: size $\ell$ kernel $K = \begin{bmatrix} A^{(i-1)} \\ B^{(i)} \end{bmatrix}$, index $i$, base channel $W$, and channel output
    vector $\boldsymbol{y}_0^{\ell-1}$
Output: $\widehat{R}^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1})$
set $M \leftarrow B^{(i)} K_{\mathrm{A}}$, where $K_{\mathrm{A}}$ is Arıkan's polar transform matrix of size $\ell$
transform $M$ into $M^*$ in RREF by $M^* = TM$
set $B^{(i)*} \leftarrow M^* K_{\mathrm{A}}$
// The above part of the algorithm can be performed
   offline
$\boldsymbol{x} \leftarrow \boldsymbol{u}_0^{i-1} A^{(i-1)}$
$\boldsymbol{z} \leftarrow (z_0, z_1, \cdots, z_{\ell-1})$
for $i = 0, 1, ...., \ell - 1$ do
    if $x_i = 1$ then
        $z_i = y_i$
    else
        $z_i = \bar{y}_i$
perform SCL decoding with list size $L$ on $(\boldsymbol{v} M^*) K_{\mathrm{A}}$ with channel output $\boldsymbol{z}$ to get the
  set of $L$ paths $\{\boldsymbol{v}^{[1]}, \boldsymbol{v}^{[2]}, \cdots, \boldsymbol{v}^{[L]}\}$
$p_0 \leftarrow 0$
$p_1 \leftarrow 0$
for $i = 1, ...., L$ do
    if $(\boldsymbol{v}^{[i]} T)_0 = 0$ then
        $p_0 \leftarrow p_0 + Pr(\boldsymbol{v}^{[i]} B^{(i)*} | \boldsymbol{z})$
    else
        $p_1 \leftarrow p_1 + Pr(\boldsymbol{v}^{[i]} B^{(i)*} | \boldsymbol{z})$
return $\widehat{R}^{(i)}(\boldsymbol{u}_0^{i-1}, \boldsymbol{y}_0^{\ell-1}) = p_0 / p_1$
```

codes on AWGN channels. Similarly, the black line shows the SC decoding performance for conventional polar codes, and the color lines show the SC decoding performance with SCL-Approximation Algorithm for large kernel polar codes constructed with K_{64}. For K_{64}, $L = 64$ is large enough to get us close to the performance limit of our approach. This gives the evidence that for K_{64}, the SCL-Approximation Algorithm with $L = 64$ is likely to give us a pretty precise approximation for the ratio in (4.7).

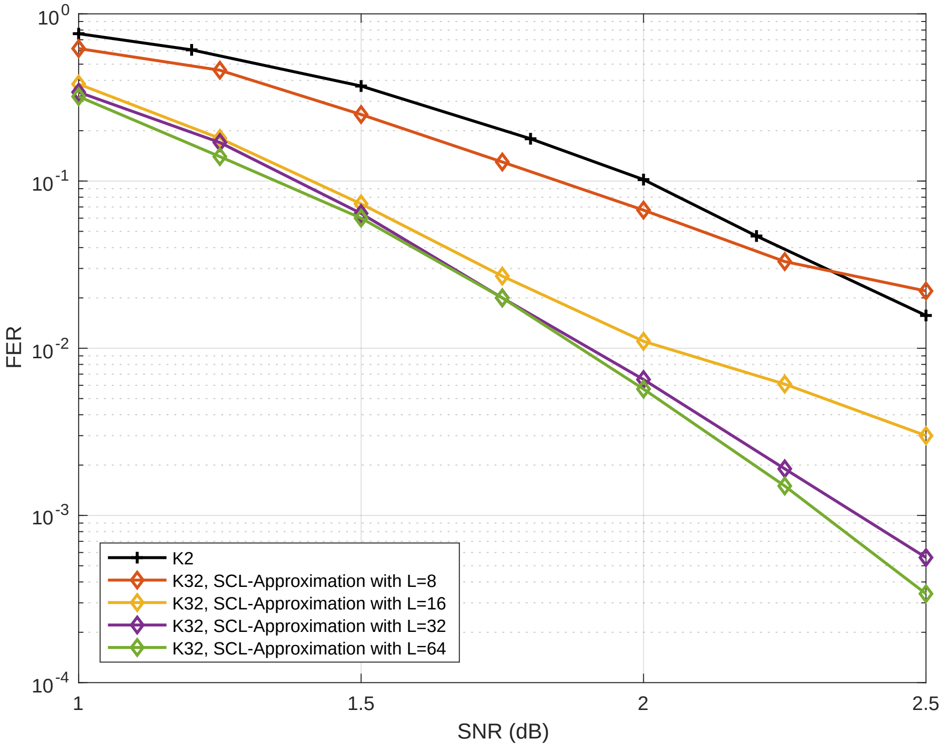

Figure 4.1. SC decoding performance for (1024,512) polar codes

4.5 Conclusion

In this chapter, we propose the SCL-Approximation Algorithm to perform kernel-level computation for SC decoding on large kernel polar codes. The SCL-Approximation Algorithm exploits the idea that polar list decoding with large enough list size can well-approximate ML decoding. This algorithm has computational complexity polynomial in the kernel size. With this low-complexity approach, we are able to SC decode polar codes constructed with a size 64 kernel for the first time.

4.6 Acknowledgements

This chapter, in part, has been published at 2021 IEEE Globecom Workshops (GC Wkshps) and appeared as: Bhaskar Gupta, Hanwen Yao, Arman Fazeli,and Alexander Vardy "Polar List Decoding for Large Polarization Kernels" [GYFV21]. The dissertation author was the primary author of this conference paper. Bhaskar Gupta contributed to the algorithm design

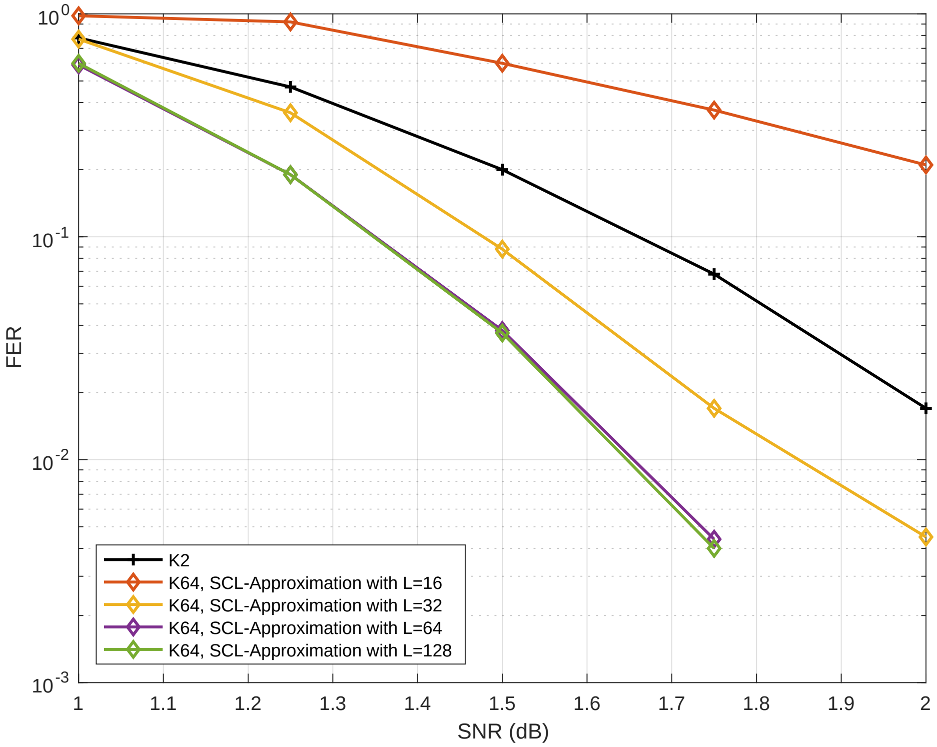

Figure 4.2. SC decoding performance for (4096,2048) polar codes

of this work.

Chapter 5

List Decoding of PAC Codes

5.1 Introduction

Polar coding, pioneered by Arıkan [Arı09], gives rise to the first explicit family of codes that provably achieve capacity for a wide range of channels with efficient encoding and decoding. However, it is well known that at short block lengths the performance of polar codes is far from optimal.

For example, the performance of a polar code of length 128 and rate $1/2$ on the binary-input AWGN channel under standard successive cancellation (SC) decoding is shown in Figure 5.1. Figure 5.1 largely reproduces the simulation results presented by Arıkan in [Arı19]. Codes of length 128 and rate $1/2$ serve as the running example throughout Arıkan's recent paper [Arı19], and we will also adopt this strategy herein. We make no attempt to optimize these codes; rather, our goal is to follow Arıkan [Arı19] as closely as possible. Also shown in Figure 5.1 is the BIAWGN dispersion bound approximation for such codes. This can be thought of as an estimate of the performance of random codes under ML decoding (see [PPV10]). Clearly, at length 128, there is a tremendous gap between polar codes under SC decoding and the best achievable performance.

As shown in [TV15] and other papers, the reasons for this gap are two-fold: the polar code itself is weak at such short lengths and SC decoding is weak in comparison with ML decoding. A well-known way to address both problems is CRC precoding followed by successive-cancellation

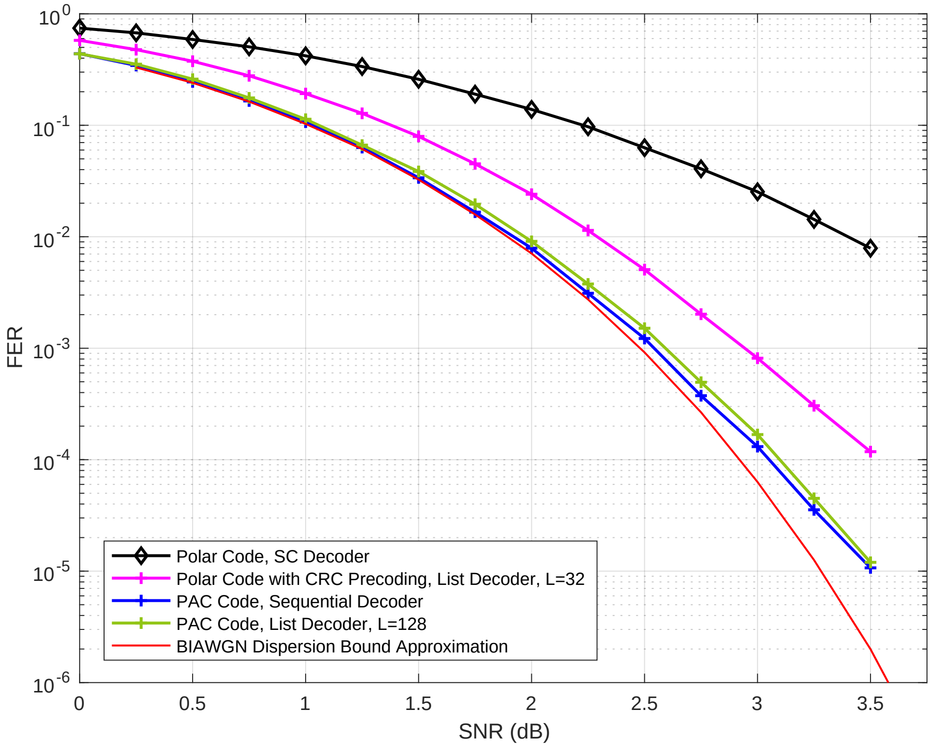

Figure 5.1. Performance of PAC codes versus polar codes.

list (SCL) decoding. Following [Arı19], the performance of CRC-aided polar codes (with 8-bit CRC) of rate $1/2$ under SCL decoding with list-size 32 is also shown in Figure 5.1. This approach, along with various refinements thereof (see [LST14, MT14, NC12] and other papers), has largely remained the state of the art in polar coding since it was first introduced in [TV15]. It is currently used as the coding scheme for control and physical broadcast channels in the enhanced mobile broadband (eMBB) mode and the ultra-reliable low latency communications (URLLC) mode of the fifth generation (5G) wireless communications standard [3GP18, BCL20].

In the Shannon Lecture at the ISIT in 2019, Erdal Arıkan presented a significant breakthrough in polar coding, which significantly boosts the performance of polar codes at short lengths. Specifically, Arıkan [Arı19] proposed a new polar coding scheme, which he calls polarization-adjusted convolutional (PAC) codes. Remarkably, under sequential decoding, the performance of PAC codes is very close to the BIAWGN dispersion bound approximation [PPV10, Ers16]. The performance of PAC codes of length 128 and rate $1/2$ is also shown (in blue and green) in Figure 5.1.

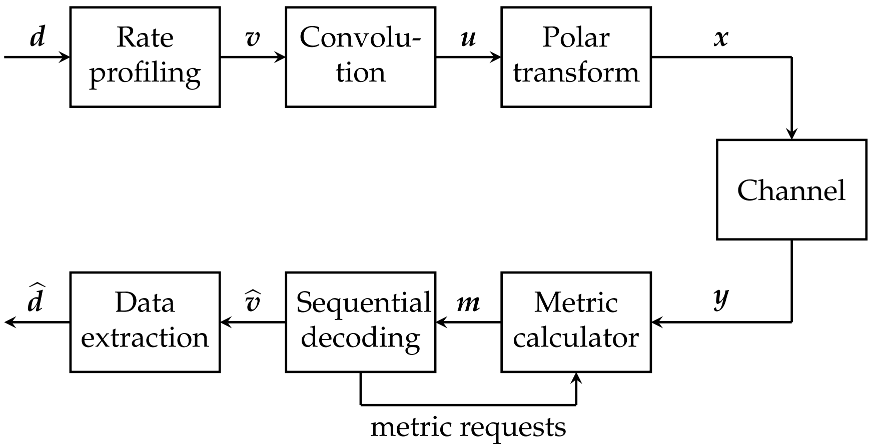

Figure 5.2. PAC coding scheme.

5.1.1 Brief Overview of PAC Codes

Arıkan's PAC codes are largely based upon the following two innovations: replacing CRC precoding with convolutional precoding (under appropriate rate-profiling, discussed later in this section) and replacing list decoding by sequential decoding. The encoding and decoding of PAC codes are shown schematically in Figure 5.2, which is reproduced from [Arı19].

Referring to Figure 5.2, let us consider an (n,k) PAC code. On the encoding side, Arıkan uses a rate-1 convolutional precoder concatenated with a standard polar encoder. Only k out of the n bits of the input $\boldsymbol{v}$ to the convolutional precoder carry the information (or data) vector $\boldsymbol{d}$. The remaining $n-k$ bits of $\boldsymbol{v}$ are set to 0. Just like for conventional polar codes, the overall performance of the resulting PAC code crucially depends upon which positions in $\boldsymbol{v}$ carry information and which are frozen to 0. This choice of frozen positions in $\boldsymbol{v}$, Arıkan has termed *rate-profiling*. Unlike conventional polar codes, the optimal rate-profiling choice is not known. In fact, it is not even clear what optimization criterion should govern this choice, although we hope to shed some light on this in Section 5.5.

The main operation on the decoder side is sequential decoding. Specifically, Arıkan employs Fano decoding (as described in [Fan63] and in Section 6.9 of [Gal68]) of the convolutional code to estimate its input $\boldsymbol{v}$. The path metrics used by this sequential decoder are obtained via repeated calls to the successive-cancellation decoder for the underlying polar code.

5.1.2 Our Contributions

One of our main goals in this chapter is to answer the following question: is sequential decoding essential for the superior performance of PAC codes? Is it possible, or perhaps advantageous, to replace the sequential decoder in Figure 5.2 by an alternative decoding method? We show that, indeed, similar performance can be achieved using list decoding, provided the list size L is moderately large. This conclusion is illustrated in Figure 5.1, where we use a list of size $L = 128$ to closely match the performance of the sequential decoder. It remains to be seen which of the two approaches is advantageous in terms of complexity. While a comprehensive answer to this question would require implementation in hardware, we carry out a qualitative complexity comparison in Section 5.4. This comparison indicates that list decoding has distinct advantages over sequential decoding in certain scenarios. In particular, list decoding is certainly advantageous in low-SNR regimes or in situations where the worst-case complexity/latency is the primary constraint.

Another objective of this chapter is to provide some insights into the remarkable performance of PAC codes observed in simulations. Although theoretical analysis of list decoding remains an open problem even for conventional polar codes, it has been observed in numerous studies that list decoding quickly approaches the performance of maximum-likelihood decoding with increasing list size L. As expected, we find this to be the case for PAC codes as well (see Figure 5.7). Fortunately, maximum-likelihood decoding of linear codes is reasonably well understood: its performance is governed by their weight distribution, and can be well approximated by the union bound, especially at high SNRs. Motivated by this observation, we use the method of [LST12] to estimate the number of low-weight codewords in PAC codes, under polar and RM rate profiles (introduced by Arıkan [Arı19]). We find that PAC codes with the RM rate-profile are superior to both polar codes (with or without CRC precoding) and the $(128, 64, 16)$ Reed-Muller code. For more on this, see Table 5.3 and Figure 5.9. We also introduce and study random time-varying convolutional precoding for PAC codes. We find that, as compared with the convolutional precoding introduced in [Arı19], time-varying convolutional precoding is much less sensitive to the constraint length. Arıkan uses in [Arı19] a convolutional code generated by $\boldsymbol{c} = (1,0,1,1,0,1,1)$, whose constraint length is $\nu = 6$. In Figure 5.12, we observe that under list decoding, random time-varying precoding achieves essentially the same performance with constraint length $\nu = 2$.

5.1.3 Related Work

Numerous attempts have been made to improve the performance of polar codes at short block lengths. Various approaches based on replacing successive-cancellation decoding with more advanced decoders include list decoding [TV15], adaptive list decoding [LST12], sequential decoding [MT14, Tri18], and stack decoding [NC12], among others. When concatenating a polar code with an outer code, most of the existing work still uses CRC outer codes and their variants, as originally proposed in [TV15]. However, many other modifications of the basic polar-coding paradigm have been extensively studied, including large polarization kernels [KŞU10, FV14, FHMV20, YFV19, MT20, Tri19b, Tri19a, TT18, TT19], polar subcodes [TM13, TM15, TT17, Tri19a, Tri17, MT19a, Tri20], “convolutional” polar codes [FP13, FHP17, Mor20, MT19a], and polarized Reed-Muller coding [AY20, LST14, MHU14, YA20] among others.

As shown later in this chapter, in Arıkan’s PAC codes, convolutional precoding combined with rate-profiling can be regarded as replacing traditional frozen bits with dynamically frozen bits. Polar coding with dynamically frozen bits was first introduced by Trifonov and Miloslavskaya in [TM13], and later studied in [TM13, TM15, TT17, Tri19a, Tri17, MT19b, YPB$^+$19, CNP20, MV20] and other papers. However, the dynamic freezing patterns in these papers are very different from [Arı19]. Prior to Arıkan’s work [Arı19], convolutional precoding of polar codes was proposed in [FTV17] and later studied in [FVY19].

Although quite recent, Arıkan’s PAC codes have already attracted considerable interest; see for example [Arı20, LZG19, MK20, Moz20, MMQA20, TG21, WJZL20]. While these papers investigate various aspects of PAC codes, none of them considers list decoding thereof. Finally, we note the work of [RBV20, RV21], which investigates both Fano decoding and list decoding of PAC codes. This work is apparently independent from and contemporaneous with our results herein. [1]

5.1.4 Chapter Outline

The rest of this chapter is organized as follows. We begin with an overview on Arıkan's PAC codes in Section 5.2, including both their encoding process and sequential decoding. In Section 5.3, we present our list-decoding algorithm. In Section 5.4, we compare it with sequential decoding, in terms of both performance and complexity. In Section 5.5, we endeavor to acquire some insight into the remarkable performance of PAC codes. First, we show empirically that both sequential decoding and list decoding thereof are extremely close to the ML decoding performance. To get a handle on the latter, we estimate the number of low-weight codewords in PAC codes (and polar codes) under different rate profiles. This makes it possible to approximate the performance of ML decoding with a union bound. In Section 5.6, we introduce and study random time-varying convolutional precoding for PAC codes, and show that it may be advantageous in terms of the constraint length. We conclude with a brief discussion in Section 5.7.

5.2 Overview of Arıkan's PAC Codes

For details on conventional polar codes under standard SC decoding, we refer the reader to Arıkan's seminal paper [Arı09]. Like polar codes, the block length n of a PAC code is also a power of 2. That is, $n = 2^m$ with $m \geqslant 1$. As shown in Figure 5.2, the encoding process for an (n,k) PAC code consists of the following three steps: rate-profiling, convolutional precoding, and polar encoding. In the first step, the k data (information) bits of the data vector $\boldsymbol{d}$ are

[1] The paper of Rowshan, Burg, and Viterbo [RBV20] was posted on `arxiv.org` in February 2020, while our work [YFV20] was submitted for review in January 2020. Our results became available on `arxiv.org` in May 2020. The Rowshan-Viterbo paper [RV21] was posted on `arxiv.org` in July 2020, after our results were presented in [YFV20].

embedded into a data-carrier vector $\boldsymbol{v}$ of length n, at k positions specified by an index set $\mathcal{A} \subseteq \{0,1,\ldots,n-1\}$ with $|\mathcal{A}| = k$. The remaining $n-k$ positions in $\boldsymbol{v}$ are frozen to zero. Arıkan [Arı19] used *rate-profiling* to refer to this step, along with the choice of the index set $\mathcal{A}$.

Just like for polar codes, a careful choice of the index set $\mathcal{A}$ is crucial to achieve good performance. Arıkan has proposed in [Arı19] two alternative approaches for selecting this set $\mathcal{A}$. The first approach, called *polar rate-profiling*, proceeds as follows. Let $W_0, W_1, \ldots, W_{n-1}$ be the n bit-channels, defined with respect to the conventional polar code of length n. In polar rate-profiling, $\mathcal{A}$ is chosen so that $\{W_i : i \in \mathcal{A}\}$ consists of the k best bit-channels in terms of their capacity. In other words, the capacities of the k bit-channels $\{W_i : i \in \mathcal{A}\}$ are the k highest values among $I(W_0), I(W_1), \ldots, I(W_{n-1})$. The second approach proposed in [Arı19] is called *RM rate-profiling*. Let $\mathrm{wt}(i)$ denote the Hamming weight of the binary expansion of an index i. In RM rate-profiling, we simply pick the k indices of the highest weight, with ties resolved arbitrarily. In other words, the set $\{\mathrm{wt}(i) : i \in \mathcal{A}\}$ consists of the k largest values among $\mathrm{wt}(0), \mathrm{wt}(1), \ldots, \mathrm{wt}(n-1)$. Notably, without convolutional precoding, this choice of $\mathcal{A}$ generates Reed-Muller codes (as subcodes of a rate-1 polar code).

In the second step, the data-carrier vector $\boldsymbol{v}$ resulting from the rate-profiling step is encoded using a rate-1 convolutional code generated by $\boldsymbol{c} = (c_0, c_1, \ldots, c_\nu)$, with $c_0 = c_\nu = 1$ (the latter can be assumed without loss of generality). This produces another vector $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ of length n, where

$$\begin{aligned} u_0 &= c_0 v_0, \\ u_1 &= c_0 v_1 + c_1 v_0, \\ u_2 &= c_0 v_2 + c_1 v_1 + c_2 v_0, \end{aligned}$$

and so on. In general, every bit in $\boldsymbol{u}$ is a linear combination of $(\nu+1)$ bits of $\boldsymbol{v}$ computed via the convolution operation:

$$u_i = \sum_{j=0}^{\nu} c_j v_{i-j} \tag{5.1}$$

where for $i - j < 0$, we set $v_{i-j} = 0$ by convention. Alternatively, this step can be viewed as a vector-matrix multiplication $\boldsymbol{u} = \boldsymbol{v}\mathbf{T}$, where $\mathbf{T}$ is the upper-triangular Toeplitz matrix:

$$\mathbf{T} = \begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_\nu & 0 & \cdots & 0 \\ 0 & c_0 & c_1 & c_2 & \cdots & c_\nu & & \vdots \\ 0 & 0 & c_0 & c_1 & \ddots & \cdots & c_\nu & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & & & \ddots & 0 & c_0 & c_1 & c_2 \\ \vdots & & & & 0 & 0 & c_0 & c_1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & c_0 \end{bmatrix} \tag{5.2}$$

In the third step, the vector $\boldsymbol{u}$ is finally encoded by a conventional polar encoder as the codeword $\boldsymbol{x} = \boldsymbol{u}\mathbf{P}_m$. Here

$$\mathbf{P}_m = \mathbf{B}_n \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes m} \tag{5.3}$$

where $\mathbf{B}_n$ is the $n \times n$ bit-reversal permutation matrix (as defined in Section VII of [Arı09]), and $\mathbf{P}_m$ is known as the polar transform matrix. Alternatively, the polar transform can be defined as in (5.3) but without the bit-reversal matrix $\mathbf{B}_n$; this has no effect on the performance of the resulting codes.

With reference to the foregoing discussion, the PAC code in Figure 5.1 is obtained via RM rate-profiling using the rate-1 convolutional code generated by $\boldsymbol{c} = (1,0,1,1,0,1,1)$. This produces the $(128,64)$ PAC code, which is the code studied by Arıkan in [Arı19]. This specific PAC code will serve as our primary running example throughout this chapter.

On the decoding side, Arıkan [Arı19] employs sequential decoding of the underlying convolutional code to decode the data-carrier vector $\boldsymbol{v}$. Under the frozen-bit constraints imposed by rate-profiling, the rate-1 convolutional code becomes an irregular tree code. There are many different variants of sequential decoding for irregular tree codes, varying in terms of both the decoding metric used and the algorithm itself. Arıkan [Arı19] uses the Fano sequential decoder, described in [Fan63, Gal68]. Notably, the path metrics at the input to the sequential decoder are obtained via repeated calls to the successive-cancellation decoder for the underlying polar code, as shown in Figure 5.2.

5.3 List Decoding of PAC Codes

One of our main objectives herein is to determine whether sequential decoding of PAC codes (cf. Figure 5.2) can be replaced by list decoding. In this section, we show how list decoding of PAC codes can be implemented efficiently (see Algorithms 5 and 6). In the next section, we will consider the performance and complexity of the resulting decoder, as compared to the sequential decoder of [Arı19].

5.3.1 PAC Codes as Polar Codes with Dynamically Frozen Bits

To achieve efficient list decoding of PAC codes, we use the list-decoding algorithm developed in [TV15]. The complexity of this algorithm is $O(Ln\log n)$, where L is the list size. However, the algorithm of [TV15] decodes conventional polar codes. In order to make it possible to decode PAC codes with (a modified version of) this algorithm, we first observe that PAC codes can be regarded as polar codes with dynamically frozen bits.

Polar coding with dynamically frozen bits was first introduced in [TM13] by Trifonov and Miloslavskaya, and later studied by the same authors in [TM15, TT17]. Let us briefly describe the general idea. In conventional polar coding, it is common practice to set all frozen bits to zero. That is, $u_i = 0$ for all $i \in \mathcal{F}$, where $\mathcal{F} \subset \{0,1,\ldots,n-1\}$ denotes the set of frozen indices. However, this choice is arbitrary: we can set $u_i = 1$ for some $i \in \mathcal{F}$ and $u_i = 0$ for other $i \in \mathcal{F}$. What matters is that the frozen bits are fixed and, therefore, known a priori to the decoder. In [TM13], it was further observed that in order to be known a priori to the decoder, the frozen bits do not have to be fixed. Given $i \in \mathcal{F}$, we can set

$$u_i = f_i(u_0, u_1, \ldots, u_{i-1}) \tag{5.4}$$

where f_i is a fixed Boolean function (usually, a linear function) known a priori to the decoder. For all $i \in \mathcal{F}$, the decoder can then decide as follows

$$\widehat{u}_i = f_i(\widehat{u}_0, \widehat{u}_1, \ldots, \widehat{u}_{i-1}) \tag{5.5}$$

where $\widehat{u}_0, \widehat{u}_1, \ldots, \widehat{u}_{i-1}$ are its earlier decisions. The encoding/decoding process in (5.4) and (5.5) is known as dynamic freezing.

In order to explain how Arıkan's PAC codes [Arı19] fit into the dynamic freezing framework, let us first introduce some notation. With reference to Section 5.2, for $i = 0, 1, \ldots, n-1$, let $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ denote the vectors $(u_0, u_1, \ldots, u_i)$ and $(v_0, v_1, \ldots, v_i)$, respectively. Further, let $\mathbf{T}_{i,j}$ denote the submatrix of the Toepliz matrix $\mathbf{T}$ in (5.2), consisting of the first (topmost) $i+1$ rows and the first (leftmost) $j+1$ columns. With this, it is easy to see that $\boldsymbol{u}_i = \boldsymbol{v}_i \mathbf{T}_{i,i}$ for all i. The matrix $\mathbf{T}_{i,i}$ is upper triangular with $\det \mathbf{T}_{i,i} = c_0 = 1$. Hence it is invertible, and we have $\boldsymbol{v}_i = \boldsymbol{u}_i \mathbf{T}_{i,i}^{-1}$ for all i. Now suppose that $i \in \mathcal{A}^c$, so that v_i is frozen to zero in the rate-profiling step. Then we have

$$\boldsymbol{u}_i = \boldsymbol{v}_{i-1} \mathbf{T}_{i-1,i} = \big(\boldsymbol{u}_{i-1} \mathbf{T}_{i-1,i-1}^{-1}\big) \mathbf{T}_{i-1,i} \tag{5.6}$$

In particular, this means that the last bit u_i of the vector $\boldsymbol{u}_i$ is an a priori fixed linear function of its first i bits, as follows:

$$u_i = (u_0, u_1, \ldots, u_{i-1})\, \mathbf{T}_{i-1,i-1}^{-1} \big(0, \ldots, 0, c_\nu, c_{\nu-1}, \ldots, c_1\big)^t$$

where $(0, \ldots, 0, c_\nu, c_{\nu-1}, \ldots, c_1)^t$ represents the last column of the matrix $\mathbf{T}_{i-1,i}$. Clearly, the above is a special case of dynamic freezing in (5.4).

Moreover, it follows that the set $\mathcal{F}$ of indices that are dynamically frozen is precisely the same as in the rate-profiling step, that is $\mathcal{F} = \mathcal{A}^c$.

If $i \in \mathcal{A}$, then v_i is an information bit, but the value of u_i is determined not only by v_i but by $v_{i-1}, v_{i-2}, \ldots, v_{i-\nu}$ as well. Thus, when representing PAC codes as polar codes, the information bits may be also regarded as dynamic.

Finally, note that in implementing the PAC decoder, there is no need to actually invert a matrix as in (5.6). Instead, we can successively compute the vector $\widehat{\boldsymbol{v}} = (\widehat{v}_0, \widehat{v}_1, \ldots, \widehat{v}_{n-1})$ as follows. If $i \in \mathcal{A}^c$, set $\widehat{v}_i = 0$. Otherwise, set

$$\widehat{v}_i = \widehat{u}_i - \sum_{j=1}^{\nu} c_j \widehat{v}_{i-j} \tag{5.7}$$

where the value of $\widehat{u}_i$ is provided by the polar decoder. Given $\widehat{v}_i, \widehat{v}_{i-1}, \ldots, \widehat{v}_{i-\nu}$, the values of the dynamically frozen bits $\widehat{u}_i$ for $i \in \mathcal{A}^c$ can be computed using (5.1). This computation, along with the one in (5.7), takes time linear in ν. All that is required is additional memory to store the vector $\widehat{\boldsymbol{v}} = (\widehat{v}_0, \widehat{v}_1, \ldots, \widehat{v}_{n-1})$.

5.3.2 List Decoding of PAC Codes

Representing PAC codes as polar codes with dynamically frozen bits makes it possible to adapt existing algorithms for successive-cancellation list decoding of polar codes to decode PAC codes.

There are, however, a few important differences. For example, for conventional polar codes, when the list decoder encounters a frozen index $i \in \mathcal{F}$, all the paths in the list-decoding tree are extended in the same way, by setting $\widehat{u}_i = 0$. For PAC codes, since freezing is dynamic, different paths are potentially extended differently, depending upon the previous decisions along the path.

In general, our list decoder for PAC codes maintains the same data structure as the successive-cancellation list decoder in [TV15]. In addition, for a list of size L, we introduce L

Algorithm 5: List Decoder for PAC Codes

```
Input: The received vector $\boldsymbol{y}$, the list size $L$, the generator $\boldsymbol{c} = (c_0, c_1, \dots, c_\nu)$ for the
       convolutional precoder as global
Output: Decoded codeword $\widehat{\boldsymbol{x}}$
// Initialization
$\cdots$ lines 2–5 of Algorithm 12 in [TV15]
shiftRegisters $\leftarrow$ new 2-D array of size $L \times (\nu + 1)$
for $\ell = 0, 1, \dots, L-1$ do
    shiftRegister[$\ell$] = $(0, 0, \dots, 0)$
// Main Loop
for $\phi = 0, 1, \dots, n-1$ do
    recursivelyCalcP($m, \phi$)
    if $u_\phi$ is frozen then
        for $\ell = 0, 1, \dots, L-1$ do
            if activePath[$\ell$] = false then
                continue
            left-shift shiftRegister[$\ell$] by one, with the rightmost position set to 0
            $C_m \leftarrow$ getArrayPointerC($m, \ell$)
            $(v_{\phi-\nu}, v_{\phi-\nu+1}, \dots, v_\phi) \leftarrow$ shiftRegister[$\ell$]
            // Set the frozen bit
            $C_m[0][\phi \bmod 2] \leftarrow \sum_{j=0}^{\nu} c_j v_{\phi-j}$
    else
        continuePaths_Unfzn($\phi$)
    if $\phi \bmod 2 = 1$ then
        recursivelyUpdateC($m, \phi$)
// Get the best codeword in the list
$\cdots$ lines 17–24 of Algorithm 12 in [TV15]
return $\widehat{x} = (C_0[\beta][0])_{\beta=0}^{n-1}$
```

auxiliary shift registers—one for each path. Each such shift register stores the last ν bits of the vector $\widehat{\boldsymbol{v}} = (\widehat{v}_0, \widehat{v}_1, \dots, \widehat{v}_{n-1})$, computed as in (5.7), for the corresponding path.

Algorithms 5 and 6 provide the full details of our list decoding algorithm for PAC codes. These algorithms fit into the same general mold as Algorithms 12 and 13 of [TV15], with the differences highlighted in blue.

Algorithm 6: continuePaths_Unfzn (PAC version)

```
Input: phase $\phi$
$\cdots$ lines 1–18 of Algorithm 13 in [TV15]
// Continue relevant paths
for $\ell = 0, 1, \ldots, L-1$ do
    if $contForks[\ell][0]$ = false and $contForks[\ell][1]$ = false then
        continue
    $C_m \leftarrow$ getArrayPointer_C$(m, \ell)$
    left-shift shiftRegister[$\ell$] by one, with the rightmost position set to 0
    $(v_{\phi-\nu}, v_{\phi-\nu+1}, \ldots, v_\phi) \leftarrow$ shiftRegister[$\ell$]
    if $contForks[\ell][0]$ = true and $contForks[\ell][1]$ = true then
        $C_m[0][\phi \mod 2] \leftarrow \sum_{j=0}^{\nu} c_j v_{\phi-j}$
        $\ell' \leftarrow$ clonePath$(\ell)$
        shiftRegister[$\ell'$] $\leftarrow$ shiftRegister[$\ell$]
        flip the rightmost bit of shiftRegister[$\ell'$]
        $C_m \leftarrow$ getArrayPointer_C$(m, \ell')$
        $(v'_{\phi-\nu}, v_{\phi-\nu+1}, \ldots, v'_\phi) \leftarrow$ shiftRegister[$\ell'$]
        $C_m[0][\phi \mod 2] \leftarrow \sum_{j=0}^{\nu} c_j v'_{\phi-j}$
    else
        if $contForks[\ell][0]$ = true then
            if $\sum_{j=0}^{\nu} c_j v_{\phi-j} = 1$ then
                flip the rightmost bit of shiftRegister$[\ell]$
            set $C_m[0][\phi \mod 2] \leftarrow 0$
        else
            if $\sum_{j=0}^{\nu} c_j v_{\phi-j} = 0$ then
                flip the rightmost bit of shiftRegister$[\ell]$
            set $C_m[0][\phi \mod 2] \leftarrow 1$
```

5.4 List Decoding versus Sequential Decoding

We now compare list decoding of PAC codes with sequential decoding, in terms of both performance and complexity. For list decoding, we use the algorithm of Section 5.3. For sequential decoding, we employ exactly the same Fano decoder that was used by Arıkan in [Arı19]. We are grateful to Erdal Arıkan for sharing the details of their decoding algorithm. We do not disclose these details here, instead referring the reader to [Arı19, Moz20, MMQA20].

We note that more efficient algorithms for sequential decoding of polar codes and their subcodes may be available; see in particular the work of Trifonov [Tri18, TT17]. However, in this chapter, we use the results of Arıkan [Arı19] as a benchmark, in terms of both performance and complexity.

Our main conclusion is that sequential decoding is not essential in order to achieve the remarkable performance of PAC codes: similar performance can be obtained with list decoding, providing the list size is sufficiently large. As far as complexity, sequential decoding is generally better at high SNRs and in terms of average complexity, while list decoding is advantageous in terms of worst-case complexity and at low SNRs.

5.4.1 Performance Comparison

Figure 5.3 summarizes simulation results comparing the performance of the Fano decoder from [Arı19] with our list decoding algorithm, as a function of the list size L. The underlying PAC code is the same as in Figure 5.1; it is the $(128, 64)$ PAC code obtained via RM rate-profiling (see Section 5.2). The underlying channel is the binary-input additive white Gaussian noise (BIAWGN) channel.

As expected, the performance of list decoding steadily improves with increasing list size. For $L = 128$, the list-decoding performance is very close to that of sequential decoding, while for $L = 256$ the two curves virtually coincide over the entire range of SNRs.

It should be pointed out that the frame error rate (FER) reported for sequential decoding in Figures 5.1 and 5.3 is due to two different mechanisms of error/failure. In some cases, the sequential decoder reaches the end of the search tree (see Figure 5.4) producing an incorrect codeword. These are decoding errors. In other cases, the end of the search tree is never reached; instead, the computation is aborted once it exceeds a predetermined cap on the number of cycles. These are decoding failures. As in [Arı19], the FER plotted in Figure 5.3 counts all the cases wherein the transmitted codeword is not produced by the decoder: thus it is the sum of the error rate and the failure rate. Table 5.1 below shows what fraction of such cases were due to decoding

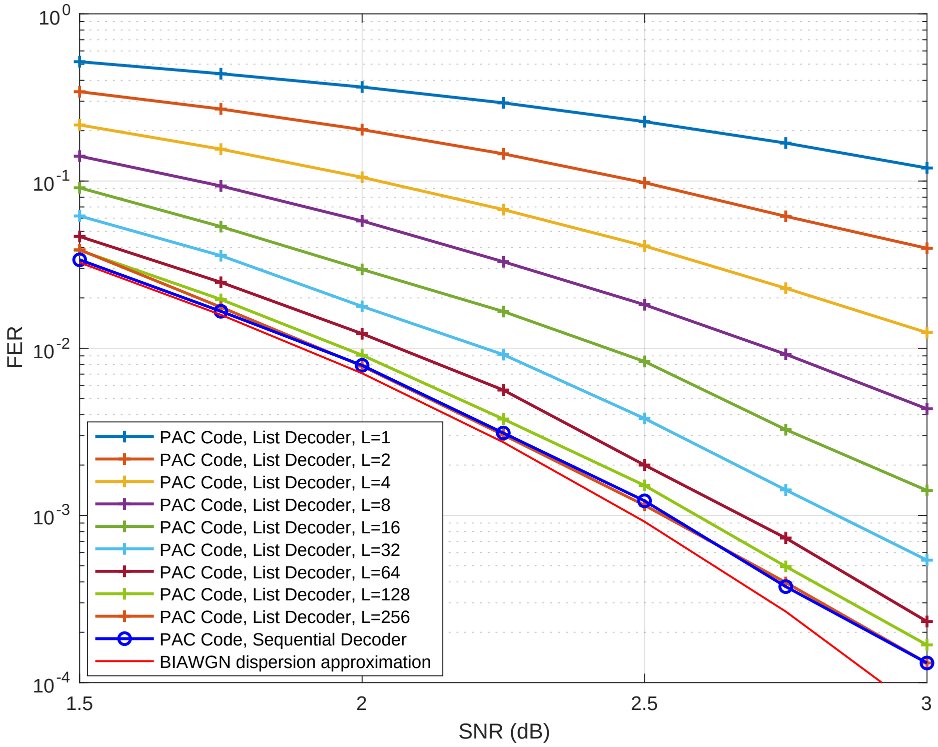

Figure 5.3. Performance of PAC codes under list decoding.

failures:

Table 5.1. Fraction of decoding failures as a function of SNR.

SNR [dB]	1.00	1.25	1.50	1.75	2.00	2.25
% of failures	4.53%	3.56%	1.86%	1.38%	1.01%	0.29%

A decoding failure was declared in our simulations whenever the number of cycles (loosely speaking, cycles count forward and backward movements along the search tree in the Fano decoder) exceeded $1,300,000$. This is exactly the same cap on the number of cycles that was used by Arıkan in [Arı19]. Overall, the foregoing table indicates that increasing this cap would not improve the performance significantly. In fact, we observe that decoding failures never dominate the overall FER of sequential decoding. Thus, it would be interesting to investigate how much this cap can be decreased without sacrificing the performance.

The FER for list decoding is also due to two distinct error mechanisms. In some cases, the transmitted codeword is not among the L codewords generated by our decoding algorithm.

In other cases, it is on the list of codewords generated, but it is not the most likely among them. Since the list decoder selects the most likely codeword on the list as its ultimate output, this leads to a decoding error. We refer to such instances as selection errors. Table 5.2 below shows the fraction of selection errors for lists of various sizes:

Table 5.2. Fraction of selection errors as a function of SNR.

SNR [dB]	1.50	1.75	2.00	2.25	2.50	2.75	3.00
$L = 64$	32.1%	32.2%	32.5%	32.3%	29.4%	36.7%	39.6%
$L = 128$	50.0%	51.6%	54.6%	53.6%	58.4%	60.4%	63.2%
$L = 256$	66.2%	71.0%	75.2%	78.0%	79.9%	83.6%	82.8%

This indicates that the performance of list decoding would further improve (at least, for $L \geqslant 64$) if we could somehow increase the minimum distance of the underlying code, or otherwise aid the decoder in selecting from the list (e.g., with CRC).

Finally, we also include in Figures 5.1 and 5.3 the BIAWGN dispersion-bound approximation for binary codes of rate $1/2$ and length 128. The specific curve plotted in Figure 5.1 and Figure 5.3 is the so-called saddlepoint approximation [VViFKL18] of the meta-converse dispersion bound of Polyanskiy, Poor, and Verdu [PPV10]. Our curve coincides with those given in Figure 1 of [CDJ+19] and Figure 6 of [GB19]. Note that a more accurate bound can be derived using the methods of Erseghe [Ers16], but this is not critical for our purposes. It is clear from Figures 5.1 and Figure 5.3 that the performance of the $(128, 64)$ PAC code, under both sequential decoding and list decoding with $L \geqslant 128$, is close to the best achievable performance.

5.4.2 Complexity Comparison

A comprehensive complexity analysis of list decoding versus sequential decoding of PAC codes in practical applications is likely to require algorithmic optimization and implementation in hardware. In the case of list decoding, this should be relatively easy based upon our representation of PAC codes as polar codes with dynamically frozen bits (see Section 5.3.1) in conjunction with existing work on efficient hardware implementation of polar list decoders (see [SGV+14b, SGV^{+}15], for example). On the other hand, we are not aware of any existing implementations of sequential decoding in hardware. Such implementation may be challenging due to variable running time, which depends on the channel noise, and complex control logic.

In this section, we provide a qualitative comparison of list decoding versus sequential decoding using two generic complexity metrics: the number of nodes visited in the polar search tree and the total number of floating-point operations performed by the decoder. The results we obtain for the two metrics, summarized in Figure 5.5 and Figure 5.6, are consistent with each other.

The polar search tree, shown schematically in Figure 5.4, represents all possible inputs $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ to the polar encoder. It is an irregular tree with $n+1$ levels containing 2^k paths. If $i \in \mathcal{A}^c$ then all nodes at level i have a single outgoing edge, as u_i is dynamically frozen in this case. In contrast with conventional polar codes, these edges may be labeled differently (cf. u_4 in Figure 5.4). If $i \in \mathcal{A}$ then all nodes at level i have two outgoing edges. In this framework, both list decoding and sequential decoding can be regarded as tree-search algorithms that try to identify the most likely path in the tree. The list decoder does so by following L paths in the tree, from the root to the leaves, and selecting the most likely one at the end. The Fano sequential decoder follows only one path, but has many back-and-forth movements during the decoding process.

For the sake of qualitative comparison, we take the total number of nodes the two algorithms visit in the tree as one reasonable proxy of their complexity. In doing so, we disregard the nodes at the frozen levels, counting only those nodes that have two outgoing edges (colored blue in Figure 5.4); we call them the decision nodes. Figure 5.5 shows the number of decision nodes visited by the two decoding algorithms as a function of SNR.

For sequential decoding, two phenomena are immediately apparent from Figure 5.5. First, there is a tremendous gap between worst-case complexity and average complexity. For most SNRs, the worst-case complexity is dominated by decoding failures, which trigger a computational timeout upon reaching the cap on the number of cycles (see Section 5.4.1).

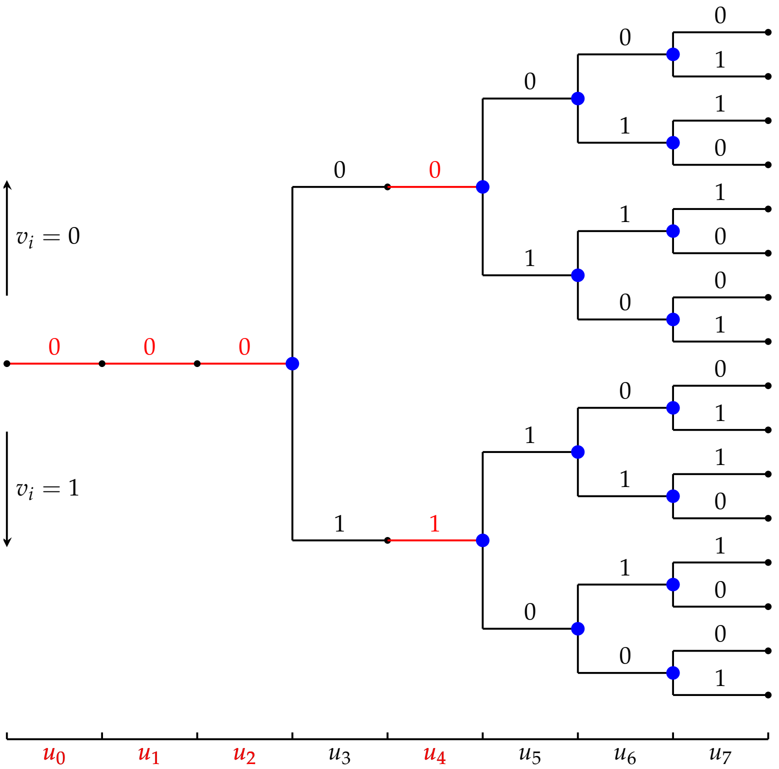

Figure 5.4. An example of the polar search tree, reproduced from [Arı19]

Clearly, reducing this cap would also reduce the worst-case complexity. On the other hand, for SNRs higher than 2.50 dB, decoding failures were not observed. Thus, beyond 2.50 dB, the worst-case complexity gradually decreases, as expected. Another phenomenon apparent from Figure 5.5 is that the average complexity is highly dependent on SNR. This is natural since the processing in the Fano sequential decoder depends on the channel noise. The less noise there is, the less likely is the sequential decoder to roll back in its search for a better path.

Neither of the two phenomena above is present for list decoding: the worst-case complexity is equal to the average complexity, and both are unaffected by SNR. The resulting curves in Figures 5.5 and 5.6 are flat, since the complexity of list decoding depends only on the list size L and the code dimension k.

In fact, the number of decision nodes visited by the list decoder in the polar search tree can be easily computed as follows. First assume, for simplicity, that L is a power of 2. As the

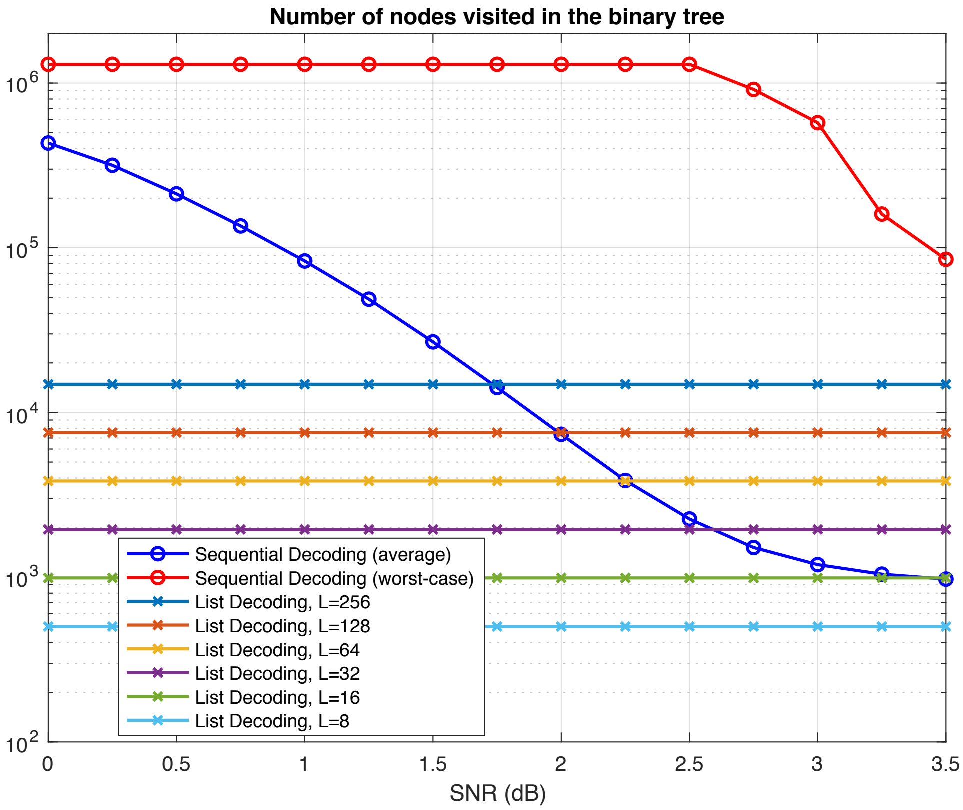

Figure 5.5. Sequential decoding vs. list decoding: Number of nodes visited in the polar search tree.

list decoder proceeds from the root to the leaves, the number of paths it traces doubles for every $i \in \mathcal{A}$ until it reaches L. The number of decision nodes it visits during this process is given by $1+2+4+\cdots+L = 2L-1$. After reaching L paths, the decoder visits L decision nodes at every one of the remaining $k - \log_2 L$ levels that are not frozen. Thus, the total number of decision nodes visited is $L(k+2-\log_2 L) - 1 = O(kL)$.

If L is not a power of 2, this counting argument readily generalizes, and the number of decision nodes visited is given by

$$L\big(k+1-\lceil \log_2 L \rceil\big) + 2^{\lceil \log_2 L \rceil} - 1 \;=\; O(kL) \tag{5.8}$$

As another qualitative metric of complexity of the two algorithms, we count the total number of additions, comparisons, and multiplications of floating-point numbers throughout the

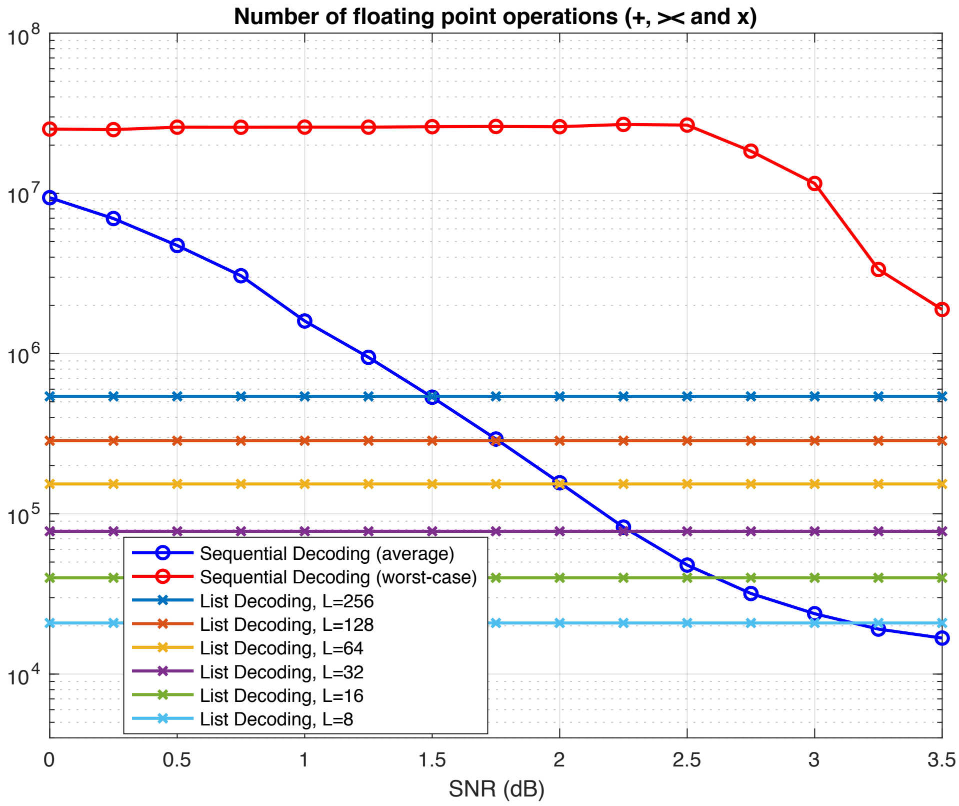

Figure 5.6. Sequential decoding vs. list decoding: Number of floating-point operations.

decoding process. The results of this comparison are compiled in Figure 5.6. The number of floating-point operations is a more precise measure of complexity than the number of decision nodes visited in the search tree. Yet we observe exactly the same pattern as in Figure 5.5. For list decoding, it is no longer possible to give a simple expression as in (5.8), but the complexity is still independent of SNR, resulting in flat curves. For sequential decoding, we again observe the same two phenomena discussed earlier in connection with Figure 5.5. In particular, the worst-case complexity remains prohibitive even at high SNRs.

In summary, our qualitative comparison suggests that, for a similar level of performance, sequential decoding is clearly advantageous in terms of average-case complexity at high SNRs. However, list decoding may have distinct advantages in low-SNR regimes or in situations where the worst-case complexity/latency is the primary constraint.

5.5 Performance Analysis for PAC Codes

In this section, we study the performance of PAC codes under the assumption of maximum-likelihood (ML) decoding. To this end, we estimate computationally the number of low-weight codewords in PAC codes (and other codes), then combine these estimates with the union bound. First, we explain why analysis of performance under ML decoding makes sense in our setting.

5.5.1 Sequential Decoding versus ML Decoding

It has been observed in several papers that for polar codes, list decoding rapidly approaches the performance of ML decoding with increasing list size L. In this section, as expected, we find this to be the case for Arıkan's $(128,64)$ PAC code as well.

Figure 5.7 shows a bound on the frame error-rate of ML decoding obtained in our simulations. This is an empirical lower bound, in the sense that the actual simulated performance of ML decoding could only be worse— even closer to the other two curves (for sequential decoding and list decoding) shown in Figure 5.7. The bound was generated using the Fano sequential decoder, as follows.

Every time the Fano decoder makes an error, we compare the likelihoods of the transmitted path and the path produced by the decoder. If the decoded path has a better path-metric (higher likelihood), then the ML decoder will surely make an error in this instance as well. We count such instances to generate the lower bound. This method of estimating ML performance in simulations is very similar to the one introduced in [TV15] for polar codes, except that [TV15] used list decoding.

Figure 5.7 provides strong evidence that it makes sense to study PAC codes under ML decoding in order to gain insights into their performance under sequential decoding, since the two are remarkably close. Figure 5.7 also reveals one of the reasons why Arıkan's PAC codes are so good at short blocklengths: they can be efficiently decoded with near-ML fidelity.

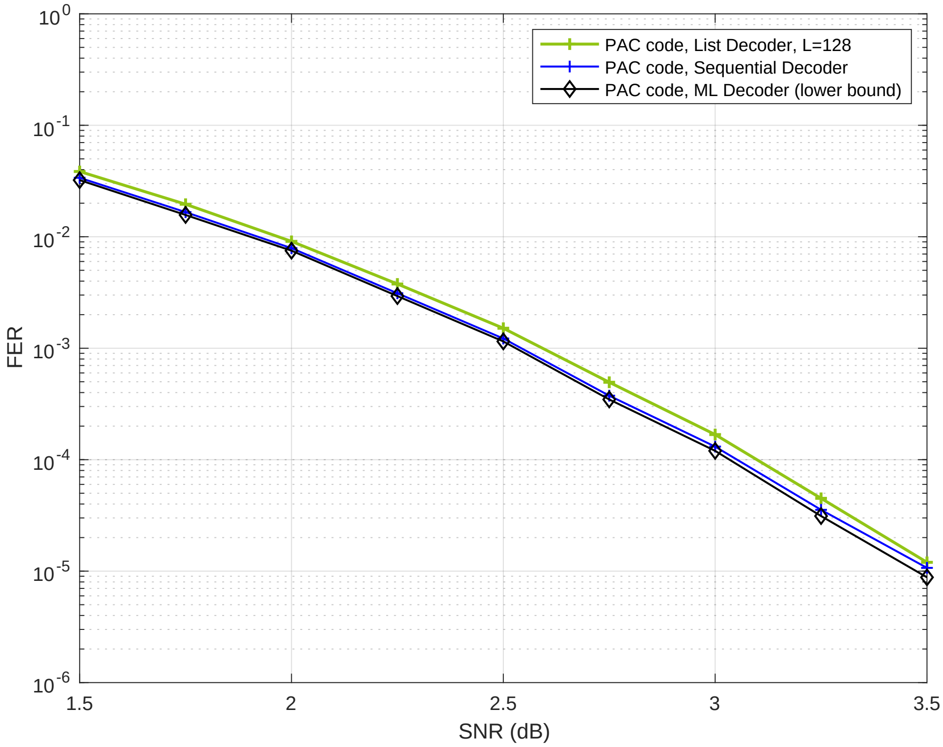

Figure 5.7. Performance of the PAC code under ML decoding.

5.5.2 Weight Distributions and Union Bounds

We now study the weight distribution of the $(128,64)$ PAC code in order to develop analytical understanding of its performance under ML decoding. Specifically, we use the method of [LST12] to estimate the number of low-weight codewords in this code.

Consider the following experiment, devised in [LST12]. Transmit the all-zero codeword in the extremely high SNR regime, and use list decoding to decode the channel output. It is reasonable to expect that in this situation, the list decoder will produce codewords of low weight. As L increases, since the decoder is forced to generate a list of size exactly L, more and more low-weight codewords emerge. The results of this experiment for the $(128,64)$ PAC code are shown in Figure 5.8 as a function of the list size. We can see that the only weights observed for L up to $400{,}000$ are $16,18,20,22$. Moreover, $A_{16} \geqslant 3120$, $A_{18} \geqslant 2696$, and $A_{20} \geqslant 95828$ (cf. Table 5.3). These numbers are lower bounds on the weight distribution of the code. However, the fact that the curves in Figure 5.8 saturate at these values provides strong evidence that these

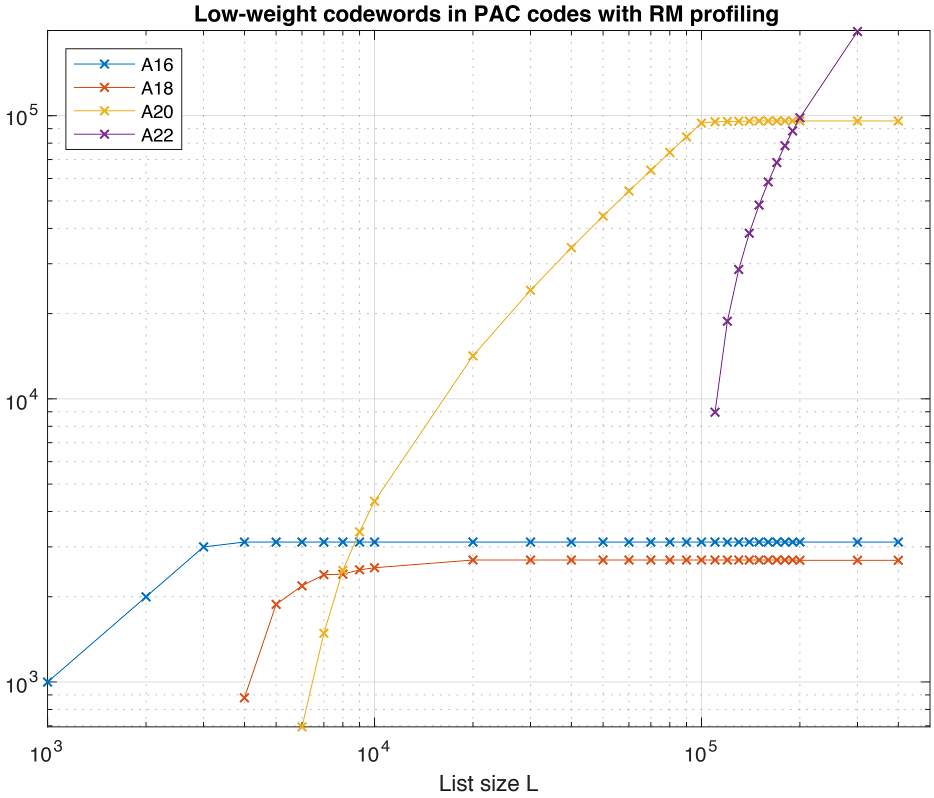

Figure 5.8. Low-weight codewords in the $(128,64)$ PAC code.

Table 5.3. Number of low-weight codewords in certain relevant codes.

	A_8	A_{12}	A_{16}	A_{18}	A_{20}	A_{22}
Polar code	48	0	68856	0	897024	0
Polar code, CRC8	20	173	$\geqslant 7069$	-	-	-
Reed-Muller	0	0	94488	0	0	0
PAC, polar profile	48	0	11032	6024	$> 10^5$	-
PAC, RM profile	0	0	3120	2696	95828	$> 10^5$

bounds are exact, and that codewords of other low weights do not exist.

We have used the same method to estimate the number of low-weight codewords in other relevant codes of rate $1/2$, including polar codes (with and without CRC precoding), the self-dual Reed–Muller code, and the PAC code with polar rate-profile. Our results are compiled in Table 5.3.

Again, the numbers in Table 5.3 should be regarded as lower bounds, which we conjecture

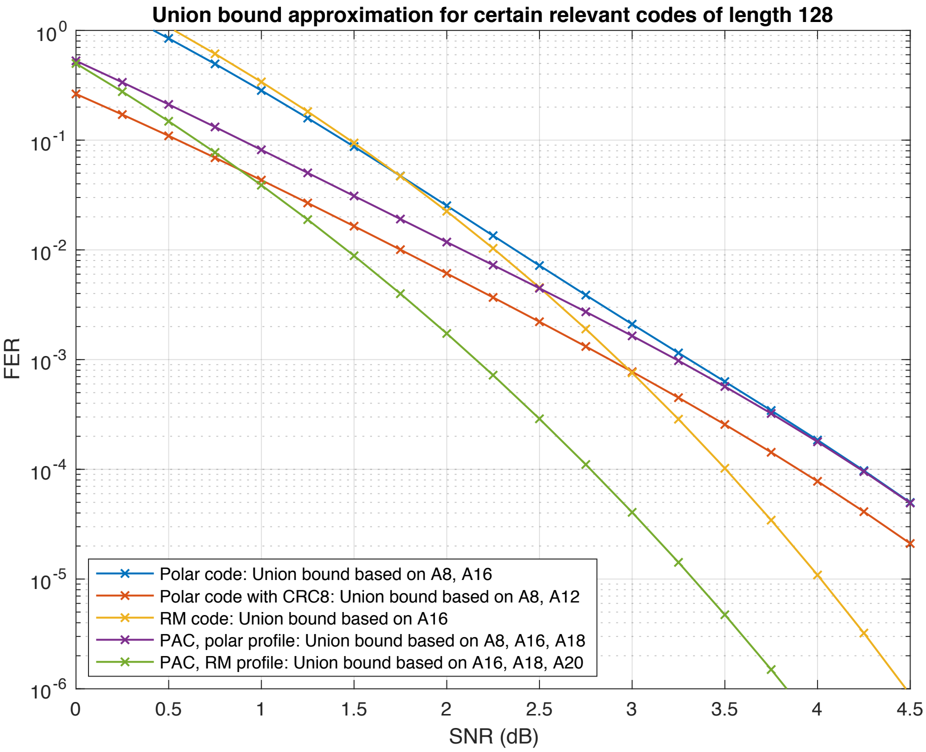

Figure 5.9. Truncated union bound for certain codes of length 128.

to be exact (except for the Reed–Muller code whose weight distribution is known [SITK71] and the polar code whose weight distribution can be computed using the methods of [YFV21a]). Assuming this conjecture, we expect the performance under ML decoding of the $(128,64)$ PAC code with RM rate-profile to be superior to all other polar and PAC codes in the table, since its minimum distance is twice as high. Interestingly, this code is also superior to the self-dual Reed–Muller code. The two codes have the same minimum distance, but the PAC code has significantly less codewords at this distance (by a factor of about 30). These observations are corroborated in Figures 5.9 and 5.10, where we plot the truncated union bound based on the partial weight distributions compiled in Table 5.3 (with all other terms set to zero). It is well known that the performance of a linear code under ML decoding is governed by its weight distribution, and can be well approximated by the union bound or variants thereof [SS$^+$06], especially at high SNRs. The "truncated union bound" is by far the simplest option, obtained by

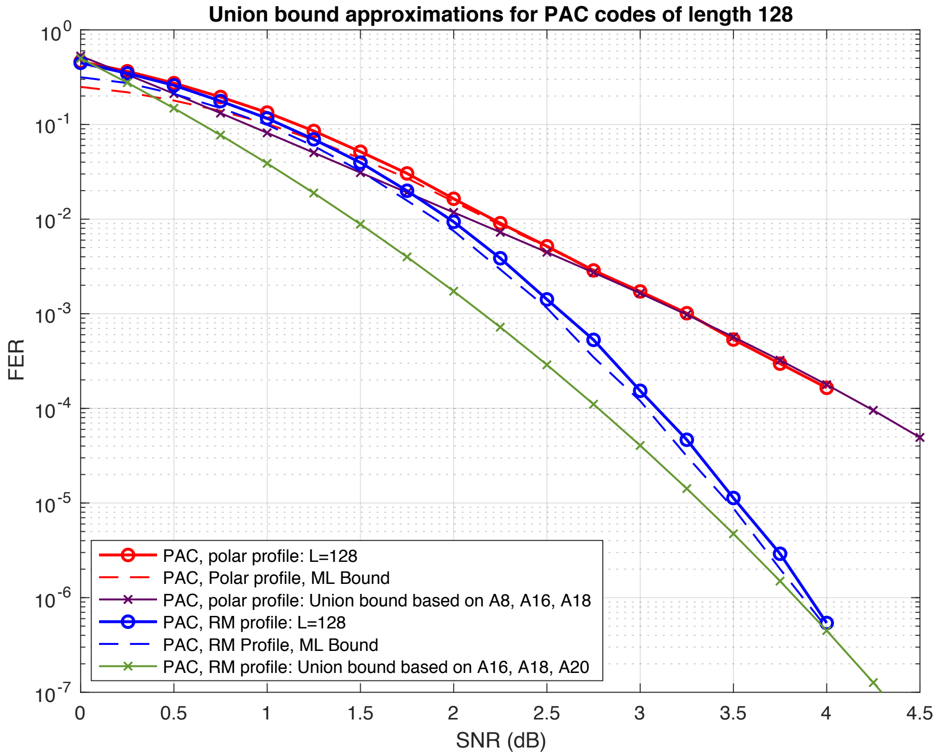

Figure 5.10. Truncated union bound vs. performance for two PAC codes.

simply ignoring those terms in the union bound for which the weight distribution is unknown. Consequently, it is neither an upper bound nor a lower bound. Nevertheless, we have found that in the high SNR regime, it provides a reasonable first-order approximation of performance under ML decoding for the codes at hand. For example, Figure 5.10 shows the truncated union bound for the two PAC codes in Table 5.3 along with upper and lower bounds on their performance (under ML decoding) obtained in simulations.

Our results in this section also provide potential guidance for the difficult problem of PAC code design. Since both sequential decoding and list decoding achieve near-ML performance, one important goal of rate-profiling should be to optimize the weight distribution at low weights. The same criterion applies for the choice of the convolutional precoder as well. A related problem is that of finding the best rate profile for a given list size, which does not necessarily approach ML decoding.

As we can see from Table 5.3, the $(128,64)$ PAC code with RM rate-profile succeeds at maintaining the minimum distance $d = 16$ of the self-dual Reed–Muller code, while "shifting" most of the codewords of weight 16 to higher weights. This is another reason for the remarkable performance of this code. The fact that the minimum distance of this PAC code is $d = 16$ also follows from Theorem 1 of [LZG19]. In fact, the work of Li, Zhang, and Gu [LZG19] shows that precoding with any nonsingular upper-triangular matrix, not necessarily a Toepliz matrix as in (5.2), cannot decrease the minimum distance. Moreover, there always exist such upper-triangular precoding matrices that strictly reduce the number of mimimum-weight codewords (see Theorem 2 of [LZG19]). Apparently, the Toepliz matrix generated by $\boldsymbol{c} = (1,0,1,1,0,1,1)$ is a particularly "nice" choice, reducing A_{16} from 94488 to only 3120. As we shall see in the next section, there are many such "nice" matrices, and it is possible to do even better.

5.6 PAC Codes with Random Time-Varying Convolutional Precoding

With reference to Section 5.2, the two main considerations when designing the rate-1 convolutional precoder are: the constraint length ν and the choice of the generator $\boldsymbol{c} = (c_0, c_1, \ldots, c_\nu)$. Arıkan [Arı19] refers to such generator $\boldsymbol{c}$ as the *impulse response* of the convolutional precoder. He furthermore writes in [Arı19] that:

> *As long as the constraint length of the convolution is sufficiently large, choosing* $\boldsymbol{c}$ *at random may be an acceptable design practice.*

The main question we wish to address herein is this: How large is "sufficiently large" in this context? It appears that if the impulse response $\boldsymbol{c}$ is fixed, then constraint length on the order of $\nu = 6$ is required. However, if we allow the impulse response to vary with time, then essentially the same performance can be achieved with constraint length as low as $\nu = 2$ (which is the minimum possible, since $c_0 = c_\nu = 1$ by assumption). This observation is of importance if trellis methods (such as list-Viterbi decoding, as suggested in [Arı19, RV21]) are used to decode PAC codes. Indeed, reducing the constraint length from $\nu = 6$ to $\nu = 2$ reduces the number of states in the resulting trellis from 64 to 4, respectively.

We also observe that under random time-varying convolutional precoding, the performance of PAC codes improves with constraint length but only slightly.

5.6.1 Random Time-Varying Convolutional Precoding

In time-varying convolutional precoding, the impulse response $\boldsymbol{c}$ is a function of time. Specifically, we keep the constraint length ν fixed, but use n potentially different impulse response vectors $\boldsymbol{c}^i = (c_0^i, c_1^i, \dots, c_\nu^i)$, where $c_0^i = c_\nu^i = 1$ for all i. Thus, each bit u_i of the input $\boldsymbol{u} = (u_0, u_1, \cdots, u_{n-1})$ to the polar encoder is computed via a potentially different convolution operation:

$$u_i = \sum_{j=0}^{\nu} c_j^{i-j} v_{i-j} \qquad \text{for } i = 0, 1, \cdots, n-1 \tag{5.9}$$

where $\boldsymbol{v}$ is the data-carrier vector resulting from the rate-profiling step, as in Section 5.2. As before, the convolution operations in (5.9) can be recast a vector-matrix multiplication $\boldsymbol{u} = \boldsymbol{v}\mathbf{T}$, where $\mathbf{T}$ is the following upper triangular matrix:

$$\mathbf{T} = \begin{bmatrix}
c_0^0 & c_1^0 & c_2^0 & \cdots & c_\nu^0 & 0 & \cdots & 0 \\
0 & c_0^1 & c_1^1 & c_2^1 & \cdots & c_\nu^1 & & \vdots \\
0 & 0 & c_0^2 & c_1^2 & \ddots & \cdots & c_\nu^2 & \vdots \\
\vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\
\vdots & & & \ddots & 0 & c_0^{n-3} & c_1^{n-3} & c_2^{n-3} \\
\vdots & & & & 0 & 0 & c_0^{n-2} & c_1^{n-2} \\
0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & c_0^{n-1}
\end{bmatrix} \tag{5.10}$$

In (5.10), the $2n - \nu$ bits shown in red, namely c_0^i and c_ν^i, are set to 1, whereas the $(2n-\nu)(\nu-1)/2$ bits shown in blue are unconstrained. In what follows, we consider random time-varying convolutional precoding, where these unconstrained bits are i.i.d. Bernoulli$(1/2)$

random variables. That is, each of these $(2n-\nu)(\nu-1)/2$ bits is set to 0 or 1 with probability $1/2$, independently of each other.

On the decoder side, we use a straightforward modification of the list-decoding algorithm introduced in Section 5.3. With reference to the pseudocode in Section 5.3, this modification consists of replacing c_j by $c_j^{\phi-j}$ at line 14 of Algorithm 5 as well as lines $9, 15, 18, 22$ of Algorithm 6, where $c_0^{\phi}, c_1^{\phi-1}, \ldots, c_\nu^{\phi-\nu}$ are as defined in (5.10). The complexity of such modified list-decoding algorithm is exactly the same as before; the only difference being that the decoder now needs to store the n impulse responses $\boldsymbol{c}^0, \boldsymbol{c}^1, \ldots, \boldsymbol{c}^{n-1}$. However, this storage requirement is still linear in n.

5.6.2 Performance of PAC Codes with Random Time-Varying Convolutional Precoding

We now assess the performance of random time-varying convolutional precoding using our running example: the $(128,64)$ PAC code with RM rate profile. As the comparison benchmark, we employ the convolutional precoder with $\nu = 6$ and $\boldsymbol{c} = (1,0,1,1,0,1,1)$ used by Arıkan in [Arı19].

Figure 5.11 summarizes our simulation results for the case where the constraint length is fixed at $\nu = 6$ while the list size ranges through $L = 1,4,16,128$. We can see from this figure that the performance of PAC codes under random time-varying convolutional precoding coincides with the list-decoding performance of the benchmark for all the relevant list sizes.

In Figure 5.12, we keep the list size constant at $L = 128$, but vary the constraint length ν. Note that setting $\nu = 0$ or $\nu = 1$ leads to degenerate cases. For $\nu = 0$, the matrix (5.10) reduces to the identity matrix and the PAC code reduces to the $(128,64)$ Reed–Muller code; the performance of this Reed–Muller code is also shown in Figure 5.12, for comparison. For $\nu = 1$, the precoding matrix in (5.10) is not time-varying and not random, with each row being a shift of the vector $\boldsymbol{c} = (1,1)$. Thus the smallest nontrivial constraint length is $\nu = 2$, which allows a single bit of randomness per row in (5.10). Surprisingly, this suffices to closely match

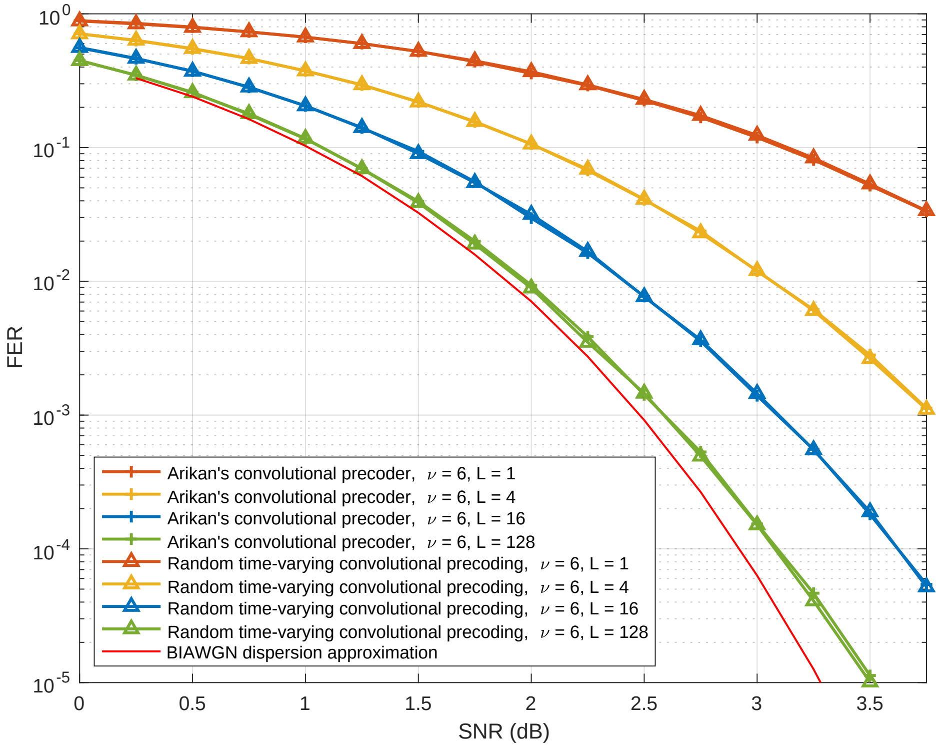

Figure 5.11. Performance of PAC codes for some specific realizations of random time-varying convolutional precoding with $\nu = 6$, as a function of the list size.

Table 5.4. Number of low-weight codewords in PAC codes for certain specific realizations of random time-varying convolutional precoding, as a function of the constraint length.

	A_8	A_{16}	A_{18}	A_{20}	A_{22}
Random precoding with $\nu = 2$	0	6424	7780	142,618	$> 10^5$
Arıkan's PAC code with $\nu = 6$	0	3120	2696	95,828	$> 10^5$
Random precoding with $\nu = 6$	0	2870	1526	88,250	$> 10^5$
Random precoding with $\nu = 10$	0	2969	412	81,026	$> 10^5$

the performance of Arıkan's PAC code [Arı19] with $\nu = 6$. As we increase the constraint length in (5.10) beyond $\nu = 2$, the performance further improves, but very slightly. Figure 5.12 shows that there is no significant gain even for $\nu = 127$, in which case the precoding matrix in (5.10) becomes a random nonsingular upper-triangular matrix. In Table 5.4, we compile (lower bounds on) the weight distribution for several typical realizations of the matrix in (5.10) which correspond to $\nu = 2, 6, 10$. These results corroborate the performance observed in simulations.

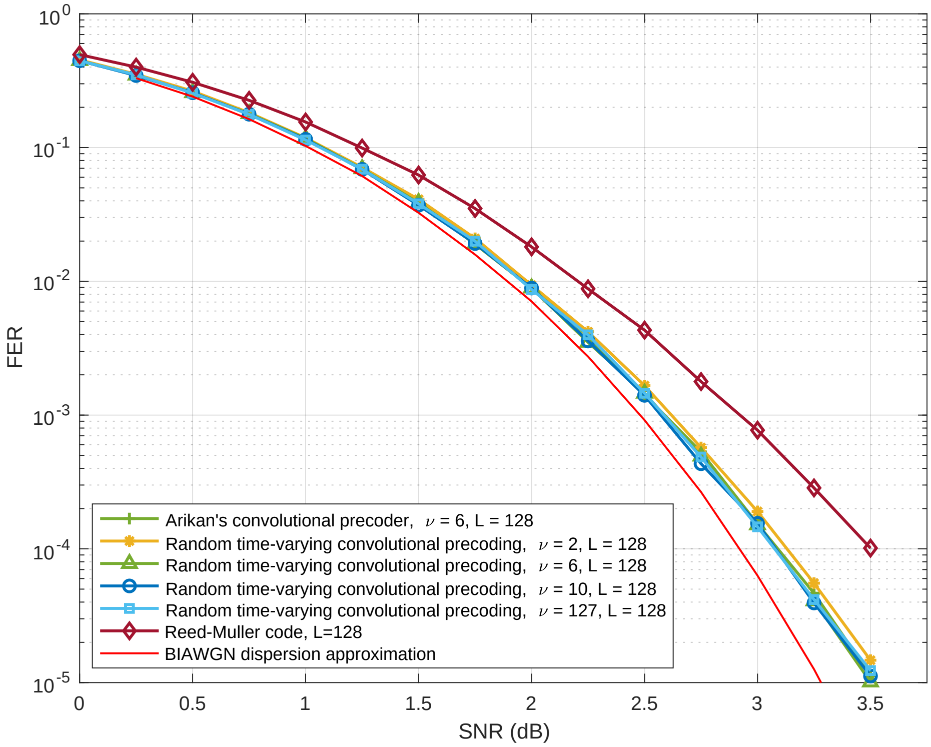

Figure 5.12. Performance of PAC codes for some specific realizations of random time-varying convolutional precoding for $L = 128$, as a function of the constraint length.

5.7 Conclusions and Discussion

In this chapter, we first observe that Arıkan's PAC codes can be regarded as polar codes with dynamically frozen bits and then, using this observation, propose an efficient list decoding algorithm for PAC codes. We show that replacing sequential decoding of PAC codes by list decoding does not lead to degradation in performance, providing the list size is sufficiently large. We then carry out a qualitative complexity analysis of the two approaches, which suggests that list decoding may be advantageous in terms of worst-case complexity. We also study the performance of PAC codes (and other codes) under ML decoding by estimating the first few terms in their weight distribution. The results of this study provide constructive insights into the remarkable performance of PAC codes at short blocklengths. We furthermore introduce random time-varying convolutional precoding for PAC codes, and observe that this makes it possible to achieve the same remarkable performance with much smaller constraint length.

Based upon our results in this chapter, we believe further complexity analysis of both sequential decoding and list decoding of PAC codes is warranted, including implementations in hardware. Some progress along these lines has been already reported in the recent paper [ZCZ$^+$20], which uses the list-decoding algorithm introduced herein as a starting point. Indeed, we hope that our work stimulates further research in this direction.

Finally, we would like to point out two important (and interdependent) but difficult questions regarding PAC codes that remain open. What is the best choice of the rate profile? What is the best choice of the precoder? We hope our results will contribute to further study of these problems. In turn, effective resolution of these problems should make it possible to replicate the success of PAC codes at length $n = 128$ for higher blocklengths.

5.8 Acknowledgements

This chapter, in part, has been published at Entropy 2021 and appeared as: Hanwen Yao, Arman Fazeli, and Alexander Vardy "List Decoding of Arıkan's PAC Codes" [YFV21b]. The dissertation author was the primary author of this journal paper.

Chapter 6

Hybrid Polar Coded Modulation

6.1 Introduction

Polar codes, pioneered by Erdal Arıkan [Arı09], form the first family of error-correcting codes that provably achieve capacity for a wide range of channels, with low encoding and decoding complexity. At short block lengths, concatenated with cyclic redundancy check (CRC) outer codes, polar codes under successive cancellation list decoding [TV15] show competitive, and in some cases, better performance as compared with turbo and LDPC codes [CDJ+19]. Thus polar codes were adopted as the error correcting code for control channels in the fifth generation (5G) wireless communications standard [3GP18, BCL20].

To achieve higher spectrum efficiency required by the next generation wireless networks, it is essential to combine polar coding with high order modulation. Two commonly used schemes that combine polar codes with channel modulation are bit-interleaved coded modulation (BICM) [CTB98, iFMC08], and multilevel coded modulation (MLC) [IH77, WFH99].

In bit-interleaved polar coded modulation (BI-PCM) [SSSH13], polar coding and modulation are connected by an interleaver, and Gray labeling is commonly used for mapping between the coded bits and the constellation symbols. At the receiver, on a constellation with 2^m symbols, the demodulator computes the soft information for all m bits of each received symbols in parallel, which are then de-interleaved and passed to the polar decoder. In BI-PCM, the 2^m-ary channel is effectively decomposed into m binary sub-channels, and decoded regardless of their dependency.

Benefits from its easiness of code design and the separation of coding and modulation, BI-PCM has been adopted for polar code in the 5G wireless communication standard [BCL20]. For constellation whose order m is not a power of 2, an additional polarization matrix can be used to connect polar codes with channel modulation [MEKLK15]. However, the major drawback of BICM is that it is unable to achieve the constellation-constrained capacity over additive white Gaussian noise (AWGN) channels [CTB98], due to loss of mutual information between the decomposed sub-channels.

It is known that MLC together with multi-stage decoding (MSD) can achieve the constellation constrained capacity over AWGN channels, provided that the code rate of each level is properly designed [WFH99]. It turns out that MSD is very similar to successive cancellation (SC) decoding of polar code on the conceptual level. Seidl *et al.* [SSSH13] first discuss the multilevel polar coded modulation (ML-PCM). They introduce a *channel parition* framework that unifies both the bit channel formation arise with SC decoding, and the channel decomposition in MSD. This framework makes it possible to assign the code rate, and design the polar code at each level of ML-PCM in a consistent way. In ML-PCM on a constellation with 2^m symbols, the 2^m-ary channel is decomposed into m binary sub-channels preserving their dependency. At the receiver, a multi-stage demodulator sequentially computes the soft information of those m sub-channels for each symbol. At each level, the computation is based on both the channel output and the hard values of all the previous sub-channels, where the hard values are obtained from the polar decoder. In this way, the mutual information between the sub-channels is preserved, and ML-PCM is expected to have a better performance compared with BI-PCM over AWGN channels.

However, there are multiple rounds of information exchanges between the demodulator and the decoder during its multi-stage decoding process in ML-PCM. At each level, the demodulator needs to send the evaluated soft information for a certain sub-channel over all received symbols to the decoder, and wait for the hard values from the decoder to proceed to the next stage. This frequent communication could introduce considerable latency for the ML-PCM receiver. To mitigate this latency issue, we introduce a hyrbid polar coded modulation design that lies between ML-PCM and BI-PCM, that is able to reduce the amount of communication between the demodulator and the decoder, while still maintaining a considerable performance gain over BI-PCM.

6.1.1 Our Contribution

In this chapter, we propose a new polar coded modulation scheme, referred as hybrid polar coded modulation (Hybrid-PCM) hereafter, that can be viewed as a comprehensive framework having both BI-PCM and ML-PCM as its special cases. In our Hybrid-PCM scheme, the 2^m-ary channel on a constellation with 2^m symbols is decomposed into m binary sub-channels following a new channel transformation that we refer as *hybrid binary partition*. This channel transformation has an integer-valued splitting parameter s that lies between 0 and m. At the receiver side, the demodulator computes the soft information for the the first s sub-channels sequentially, based on both the channel output and the hard value of their previous sub-channels. Then with the hard information of the first s levels, the demodulator estimates the rest of the $(m-s)$ sub-channels parallelly regardless of their dependency. Intuitively speaking, out of those m levels for every received symbol, the first s levels are sequentially decoded similar to ML-PCM, and the last $(m-s)$ levels are parallelly decoded similar to BI-PCM. In this way, only the mutual information of the first s sub-channels is preserved during the decoding process, and we are free to choose this splitting parameter s between 0 and m. We also propose a hybrid labeling rule to fit our scheme. This labeling rule lies between Gray labeling, commonly used in BICM, and set-partitioning (SP) labeling, commonly used in MLC, and it's governed by the same splitting parameter s.

If we choose s to be equal to 0, then our hybrid scheme becomes BI-PCM. And if we choose s to be m, our hybrid scheme becomes ML-PCM. For s lying between 0 and m, our hybrid scheme can reduce the amount of back-and-forth communication required in ML-PCM, while as we will show in Section V, still holding a considerable performance gain over BI-PCM.

6.1.2 Notations

Here are some notation conventions that we follow throughout this chapter. We use bold letters like $\boldsymbol{u}$ to denote vectors, and non-bold letters like u_i to denote symbols within that vector. For $\boldsymbol{u} = (u_1, u_2, \cdots, u_n)$, we denote its subvector of symbols with indices from a to b as $\boldsymbol{u}_a^b = (u_a, u_{a+1}, \cdots, u_b)$. And we use $(\boldsymbol{u}, \boldsymbol{v})$ to denote the concatenation of vector $\boldsymbol{u}$ and vector $\boldsymbol{v}$.

6.2 Preliminaries

In this section, we describe our system model, and give a brief review on the concepts of polar codes and polar coded modulation. This prepare us for the development of our proposed hybrid polar coded modulation scheme.

6.2.1 System Model

In this chapter, we consider memoryless AWGN channels with quadrature amplitude modulation (QAM) and pulse-amplitude modulation (PAM). Since any 2^{2m}-QAM constellation can be constructed from two independent 2^m-PAM constellations for the I-channel and Q-channel, henceforth we regard every QAM symbol as two independent PAM symbols.

For a PAM constellation with 2^m symbols, its signal points are given by

$$\mathcal{X} = \{\pm 1, \pm 3, \cdots, \pm(2^m - 1)\}.$$

Each symbol in the constellation is labeled by a binary m-tuple, and we say that symbols in this constellation have *m bit levels*. The input-output relation of the AWGN channel is given by $y = x + z$, with $x \in \mathcal{X}$ for each channel use, and z being a zero mean Gaussian noise with standard deviation σ_z. The quality of the channel is measured by the signal to noise ratio (SNR):

$$SNR = E[x^2]/\sigma_z^2.$$

6.2.2 Polar Codes

Assuming $n = 2^\ell$, recall that an (n,k) polar code is a binary linear block code generated by k rows of the polar transformation matrix $G_n = K_2^{\otimes \ell}$, where

$$K_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix},$$

and $K_2^{\otimes \ell}$ is the ℓ-th Kronecker power of K_2. The encoding scheme is given by $\boldsymbol{c} = \boldsymbol{u} G_n$, where $\boldsymbol{u}$ is a length-n binary input vector carrying k data bits, and c is the codeword for transmission. The positions of the data bits in $\boldsymbol{u}$ are specified by an information index set $\mathcal{A} \subseteq \{1,2,\cdots,n\}$ of size k, with the rest of the $n-k$ bits in $\boldsymbol{u}$ frozen to certain fixed values, usually zeros. The construction for polar codes usually refers to the selection of the information index set $\mathcal{A}$.

For decoding of polar code, in this paper we consider the conventional SC decoder, which is proven to be capacity achieving [Arı09]. For details of SC decoding for polar code, we refer the readers to Arıkan's seminal paper [Arı09].

6.2.3 Bit-Interleaved Polar Coded Modulation (BI-PCM)

Let $|\mathcal{X}| = 2^m$, and let N denotes the number of channel uses. In BI-PCM, the binary codeword generated by the polar encoder is permuted by an interleaver. Then, each block of m bits is mapped into a constellation symbol in $\mathcal{X}$ for channel transmission. At the receiver's side of BI-PCM, for each received symbol, the demodulator ignores the relation between bit levels, and computes the soft information for all bit levels solely based on the channel observation.

Let $W : \mathcal{X} \to \mathcal{Y}$ be a 2^m-ary channel with input symbol set $\mathcal{X}$ with $|\mathcal{X}| = 2^m$, and output alphabet $\mathcal{Y}$. In a BI-PCM scheme over this channel, W is decomposed into m binary sub-channels that are viewed as independent channels by the receiver. This channel transform is

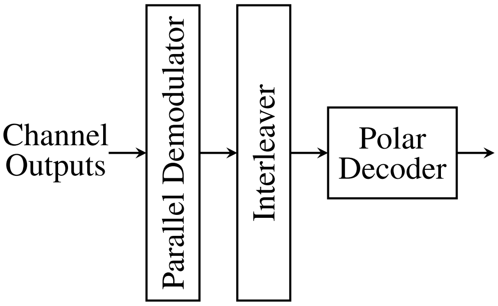

Figure 6.1. BI-PCM receiver

called *parallel binary partition* (PBP) in [SSSH13]. Here we denote it as

$$\varphi : W \to \{B_\varphi^{(1)}, B_\varphi^{(2)}, \cdots, B_\varphi^{(m)}\},$$

where $B_\varphi^{(j)} : \{0,1\} \to \mathcal{Y}$ denotes the binary sub-channel for the j-th bit level for $j = 1,2,\cdots,m$. In PBP, each sub-channel $B_\varphi^{(j)}$ only has the knowledge of the channel output $y \in \mathcal{Y}$. And Gray labeling is commonly used to generate sub-channels that are as independent as possible [SF07]. Let the bit-to-symbol labeling rule given by $\mathcal{L} : \{0,1\}^m \to \mathcal{X}$, then $B_\varphi^{(j)}$ has the transition probability

$$B_\varphi^{(j)}(y|b) = \frac{1}{2^{m-1}} \sum_{b_1^m \in \{0,1\}^m :\, b_j = b} W(y|\mathcal{L}(b_1^m)),$$

for $j = 1,2,\cdots,m$.

After demodulation, as shown in Figure 6.1, the soft information of all mN bits is deinterleaved, and fed to the decoder. Note that to use a single polar decoder for BI-PCM, the order m of the constellation has to be a power of 2.

6.2.4 Compound Polar Code

In BI-PCM, to handle constellation whose order m is not necessarily a power of 2, *compound polar code* is proposed in [MEKLK15] that uses an additional $m \times m$ polarization matrix to connect polar code with channel modulation. This structure is also mentioned in [SSSH13, Sec.V.D], and later used in [CNL13] on 64-QAM.

In BI-PCM with compound polar code, the 2^m-ary channel W is also decomposed into m binary sub-channels following PBP, but the decomposed channels are not decoded in parallel. In compound polar code, an $m \times m$ polarization matrix is used to further polarize those m decomposed sub-channels. And on the receiver's side, the polarized channels are decoded sequentially based on the hard information of their previous channels. For the details of compound polar code, we refer the readers to [MEKLK15].

With this additional polarization matrix, compound polar code shows better performance compared with plain BI-PCM under SC decoding [MEKLK15]. It inherits the benefit that demodulation and decoding are separated just like plain BI-PCM, but it also introduces extra decoding latency due to that additional polarization matrix. Since in this paper, we focus on reducing the iterative communication between the demodulator and the decoder in ML-PCM, we also include compound polar code in our simulation comparison in Section 6.4.

For our simulation in Section 6.4, we use K_3 as the additional polarization matrix for 8-PAM the same as in [MEKLK15, Sec.VII.A], and use K_4 as the additional polarization matrix for 16-PAM the same as in [SSSH13, Sec.V.D]:

$$K_3 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \qquad K_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

It has been shown in [BPYS17] that for the labeling in BI-PCM with compound polar code, the least significant bit Binary Reflected Gray Code (LSB-BRGC) shows a better performance compared with the Binary Reflected Gray Code (BRGC) under SC decoding. Thus we also adopt LSB-BRGC for the bit labeling for BI-PCM with compound polar code in our simulation.

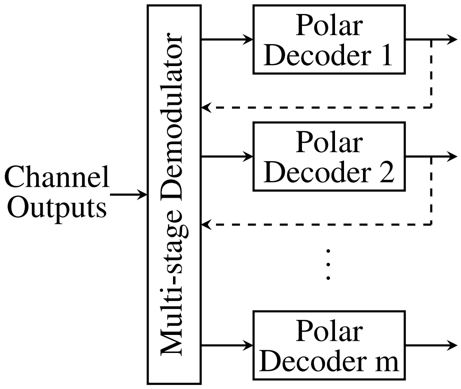

Figure 6.2. ML-PCM receiver for a 2^m-ary constellation.

6.2.5 Multilevel Polar Coded Modulation (ML-PCM)

Let $\mathcal{X}$ be the symbol set of a constellation of order 2^m, and let N denotes the number of channel uses. In an ML-PCM scheme, there are m component polar codes, each of length N. For encoding, a length mN binary vector carrying both the data bits and the frozen bits is split into m vectors of equal length, and encoded by those m component polar codes respectively. Let $\boldsymbol{c}_j = (c_{j1}, c_{j2}, \cdots, c_{jN})$ denote the encoder output of the j-th component polar code for $j = 1, 2, \cdots, m$. The modulator then map the m-tuple $(c_{1i}, c_{2i}, \cdots, c_{mi})$ into a constellation symbol for transmission for $i = 1, 2, \cdots, N$. In such a way, each component polar code only appears at a corresponding single bit level for every channel use.

At the receiver's side of ML-PCM, a multi-stage demodulator computes the soft information for those m bit levels sequentially, based on both the received symbols, and the hard values of the previous bit levels. More specifically, as shown in Figure 6.2, at stage j of the decoding process, the demodulator computes the soft information of the j-th bit level for every received symbols, and send it to the j-th polar decoder. Then, the demodulator waits for j-th decoder to send back its decoding result. After retrieving the hard values for the j-th bit level of every received symbol, the demodulator then proceeds to the next bit level.

Let $W : \mathcal{X} \rightarrow \mathcal{Y}$ be a 2^m-ary channel with input symbol set $\mathcal{X}$ with $|\mathcal{X}| = 2^m$, and output alphabet $\mathcal{Y}$. In a ML-PCM scheme over this channel, W is effectively decomposed into m

binary sub-channels preserving their mutual information. This channel decomposition is called *sequential binary partition* (SBP) in [SSSH13], here we denote it as

$$\psi: \; W \to \{B_{\psi}^{(1)}, B_{\psi}^{(2)}, \cdots, B_{\psi}^{(m)}\},$$

where $B_{\psi}^{(j)} : \{0,1\} \to \mathcal{Y} \times \{0,1\}^{j-1}$ denotes the binary sub-channel for the j-th bit level for $j = 1, 2, \cdots, m$. In SBP, each sub-channel $B_{\psi}^{(j)}$ has the knowledge of both the channel output $y \in \mathcal{Y}$, and their previous bit levels. And set-partitioning (SP) labeling is commonly used to generate widely separated bit level capacities [SSSH13]. Let the bit-to-symbol mapping rule given by $\mathcal{L} : \{0,1\}^m \to \mathcal{X}$, then $B_{\psi}^{(j)}$ has the transition probability

$$B_{\psi}^{(j)}(y, b_1^{j-1}|b) = \frac{1}{2^{m-j}} \sum_{b_j^m \in \{0,1\}^{m-j+1} : b_j = b} W(y|\mathcal{L}(b_1^m))$$

for $j = 1, \cdots, m$.

6.3 A Hybrid Scheme for Polar Coded Modulation

In this section, we propose a hybrid polar-coded modulation scheme that lies between BI-PCM and ML-PCM. Our hybrid scheme can be viewed as a comprehensive framework that has BI-PCM and ML-PCM as its two special cases. We begin by introducing a channel decomposition that we refer as *hybrid binary partition*.

6.3.1 Hybrid Binary Partitions

Let $W : \mathcal{X} \to \mathcal{Y}$ be a discrete memoryless channel with input symbol set $\mathcal{X}$ with $|\mathcal{X}| = 2^m$, and output symbol set $\mathcal{Y}$. We define the *hybrid binary partition* (HBP) with splitting parameter s as the channel transform

$$\psi_s: \; X \to \{B_{\psi_s}^{(1)}, B_{\psi_s}^{(2)}, \cdots, B_{\psi_s}^{(m)}\},$$

where s is an integer between 0 and m, and $B_{\psi_s}^{(j)}$ denotes the decomposed binary sub-channel for the j-th bit level for $j = 1,2,\cdots,m$. In this channel decomposition, the first s sub-channels have the knowledge of their previous bit levels and the rest of the $(m-s)$ sub-channels only have the knowledge of the first s bit levels.

Formally, for $1 \leqslant j \leqslant s$, we have

$$B_{\psi_s}^{(j)} : \{0,1\} \to \mathcal{Y} \times \{0,1\}^{j-1}$$

with transition probability

$$B_{\psi_s}^{(j)}(y, b_1^{j-1} | b) = \frac{1}{2^{m-j}} \sum_{b_j^m \in \{0,1\}^{m-j+1} : b_j = b} W(y | \mathcal{L}(b_1^m))$$

And for $s < j \leqslant m$, we have

$$B_{\psi_s}^{(j)} : \{0,1\} \to \mathcal{Y} \times \{0,1\}^{s}$$

with transition probability

$$B_{\psi_s}^{(j)}(y, b_1^{s} | b) = \frac{1}{2^{m-s-1}} \sum_{b_{s+1}^m \in \{0,1\}^{m-s} : b_j = b} W(y | \mathcal{L}(b_1^m)).$$

Following this definition, the first s sub-channels in HBP with splitting parameter s will be the same as the first s sub-channels in SBP on the same 2^m-ary channel W.

We make the remark that for a given 2^m-ary channel W, HBP with splitting parameter $s = 0$ will be the same as PBP, and HBP with splitting parameter $s = m$ will be the same as SBP. Therefore, PBP and SBP can be viewed as two special cases of HBP.

6.3.2 Hybrid Labeling

In polar coded modulation schemes, PBP in BI-PCM is commonly equipped with Gray labeling, and SBP in ML-PCM is commonly equipped with SP labeling [SSSH13]. Since HBP

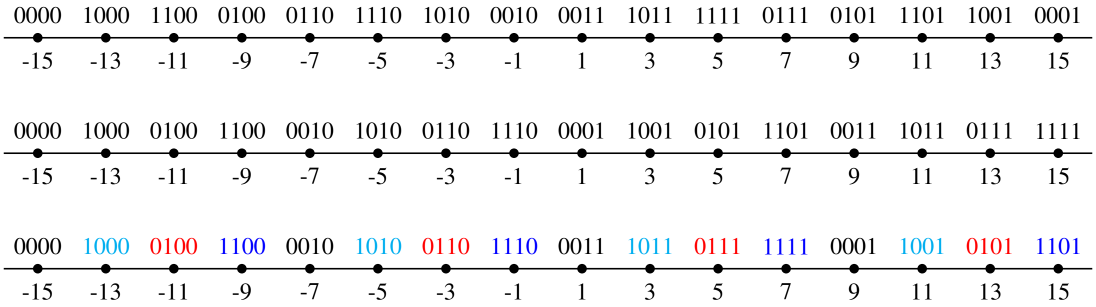

Figure 6.3. 16-ASK with Gray labeling (top), SP labeling (middle), and Hybrid labeling with splitting parameter 2 (bottom). The first bit level lies on the left.

is a hybrid channel transform that stands between PBP and SBP, we propose to equip it with a *hybrid labeling rule* that stands between Gray labeling and SP labeling.

Let $\mathcal{X}$ be the symbol set for a 2^m-ary constellation, we describe our hybrid labeling rule with splitting parameter s as follows:

1. For every symbol $x \in \mathcal{X}$, the first s bit levels are labeled the same as the SP labeling rule.

2. For the rest of the $(m-s)$ bit levels, we first partition $\mathcal{X}$ into subsets, such that symbols within each subset have the same bits on their first s bit levels. Then for each subset $\mathcal{Z} \subseteq \mathcal{X}$, we label the rest of the $(m-s)$ bit levels for symbols in $\mathcal{Z}$ following the Gray labeling rule for the 2^{m-s} sub-constellation $\mathcal{Z}$.

Example 12. We illustrate this hybrid labeling rule by taking the 16-PAM constellation as an example. Figure 6.3 shows examples of three labeling rules for 16-PAM, with Gray labeling at the top, SP labeling in the middle, and hybrid labeling with splitting parameter $s = 2$ at the bottom.

Denote the symbol set by $\mathcal{X} = \{-15, -13, \cdots, 15\}$. In the hybrid labeling with splitting parameter $s = 2$, the first two bit levels for every $x \in \mathcal{X}$ are labeled the same as in SP labeling.

Then $\mathcal{X}$ can be partitioned into four subsets according to the first two bit levels:

$$\mathcal{Z}_1 = \{-15, -7, 1, 9\}, \quad \mathcal{Z}_2 = \{-13, -5, 3, 11\},$$
$$\mathcal{Z}_3 = \{-11, -3, 5, 13\}, \quad \mathcal{Z}_4 = \{-9, -1, 7, 15\}.$$

Those four subsets are colored differently in Figure 6.3. Take $\mathcal{Z}_1$ for example, in the hybrid labeling, the last two bit levels for the symbols in $\mathcal{Z}_1$ are labeled following the Gray labeling rule viewing $\mathcal{Z}_1$ as a 4-PAM constellation.

6.3.3 Hybrid Polar Coded Modulation (Hybrid-PCM)

Now we describe our hybrid coded modulation scheme. Let $\mathcal{X}$ be the symbol set of a constellation of order 2^m, and let N denotes the number of channel uses. In our Hybrid-PCM with splitting parameter s, the 2^m-ary channel is decomposed by HBP with splitting parameter s into m binary sub-channels, where each of the first s sub-channels corresponds to a component polar code of length N, and the last $(m-s)$ sub-channels correspond to a single component code of length $(m-s)N$.

For encoding, a length mN binary vector carrying both the data and the frozen bits is split into s length-N vectors and one single vector of length $(m-s)N$. Those sub-vectors are encoded by the component codes respectively. Denote by $\boldsymbol{c}_j = (c_{j1}, c_{j2}, \cdots, c_{jN})$ the encoder output of the j-th component polar code for $j = 1, 2, \cdots, s$, and denote by

$$\boldsymbol{c}_{s+1} = (\boldsymbol{c}_{s+1,1},\ \boldsymbol{c}_{s+1,2},\ \cdots,\ \boldsymbol{c}_{s+1,N})$$

the encoder output of the $(s+1)$-th component code, where $\boldsymbol{c}_{s+1,i}$ is a length-$(m-s)$ vector for $i = 1, 2, \cdots, N$. The modulator then map the m-tuple

$$(c_{1i},\ c_{2i},\ \cdots,\ c_{si},\ \boldsymbol{c}_{s+1,i})$$

into a constellation symbol for transmission for $i = 1, 2, \cdots, N$.

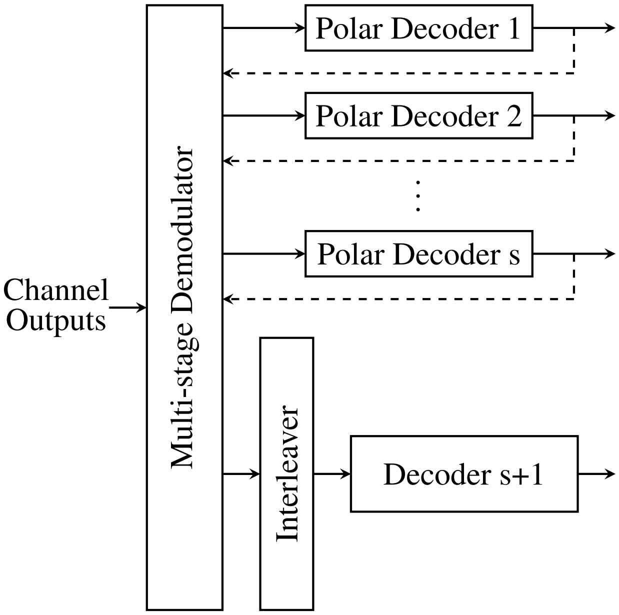

Figure 6.4. Hybrid-PCM receiver with splitting parameter s

At receiver's side of Hybrid-PCM, as shown in Figure 6.4, a multi-stage decoder first computes the soft information for the first s bit levels sequentially, based on both the received symbols, and the hard values of the previous bit levels. Then the demodulator computes the soft information for the rest of the $(m-s)$ bit levels in parallel, based on the received symbols, and the hard values of the first s bit levels. The soft information for the last $(m-s)$ bit levels is then de-interleaved, and then fed to the decoder for the last component code. In our hybrid scheme, we employ the hybrid labeling rule with the same splitting parameter s for the mapping between the coded bits and the constellation symbols.

Note that for a 2^m-ary channel W, this hybrid scheme with the splitting parameter s can be viewed as a general framework that includes both BI-PCM and ML-PCM as special cases by setting $s = 0$ and $s = m$, respectively.

6.4 Performance Evaluation

We present some simulation results of our hybrid scheme on 64-QAM and 256-QAM where all polar codes are constructed following the Monte Carlo construction.

Figure 6.5 shows the simulation results of SC decoding for $(1536,768)$ polar coded modulation on 64-QAM with three difference schemes: ML-PCM, Hybrid-PCM with splitting parameter $s = 1$, and BI-PCM with compound polar code. In our experiment, every 64-QAM is simulated by two independent 8-PAM symbols, and all the coded modulation schemes are applied on the 8-PAM constellation. We can observe that ML-PCM performs better than Hybrid-PCM, and BI-PCM with compound polar code as expected, and our hybrid scheme with splitting parameter $s = 1$ shows an approximated 0.2 dB performance gain over BI-PCM with compound polar code.

Figure 6.6 shows the simulation results of SC decoding for $(2048,1024)$ polar coded modulation on 256-QAM with four difference schemes: ML-PCM, Hybrid-PCM with splitting parameter $s = 2$, BI-PCM with compound polar code, and plain BI-PCM (Figure 6.2). In our experiment, every 256-QAM is simulated by two independent 16-PAM symbols, and all the coded modulation schemes are applied on the 16-PAM constellation. We can see that our hybrid scheme with splitting parameter $s = 2$ can close up majority of the performance gain of ML-PCM over BI-PCM with compound polar code, while reducing the number of required iterative information exchanges between the demodulator and the decoder in ML-PCM by half, thus reducing the overall decoding latency.

6.5 Conclusion and Discussion

In this chapter, we propose a new polar coded modulation scheme that we call hybrid polar coded modulation. This scheme uses hybrid bit partitions by assigning only a fraction of bits for *sequential binary partition* and subsequent iterative demodulation and decoding, whereas the remaining bits are subject to *parallel binary partition* and corresponding parallel demodulation and then decoding. Our hybrid scheme can be viewed as a comprehensive framework including both ML-PCM and BI-PCM as its two special cases, and our simulation results show that it can alleviate the latency in ML-PCM while still maintaining a considerable performance gain over

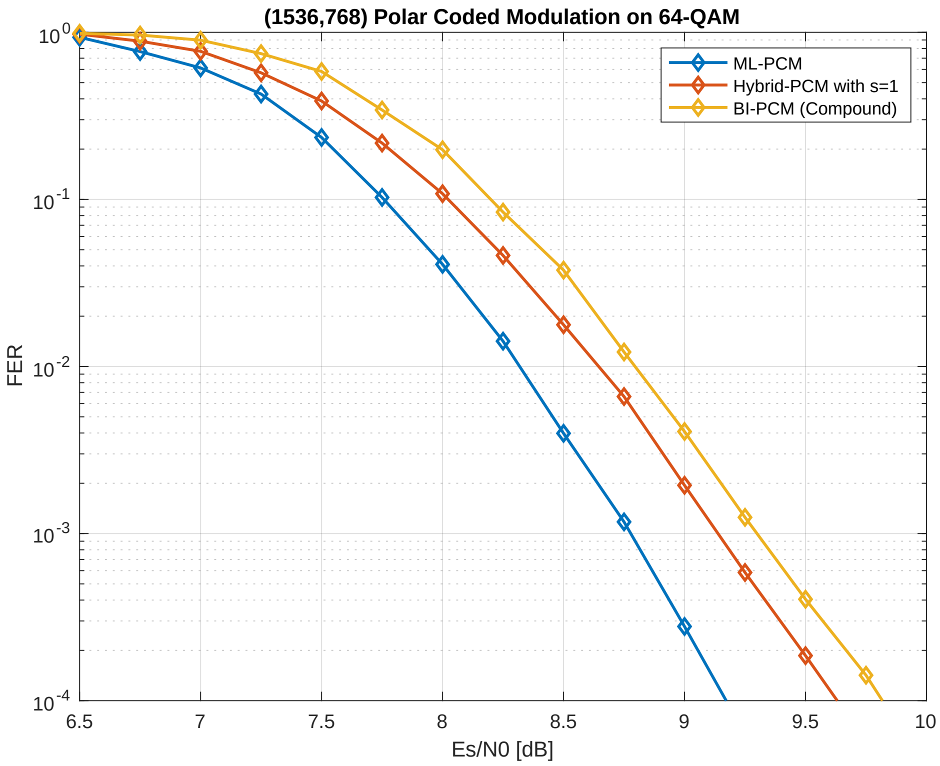

Figure 6.5. Performance comparison for ML-PCM, Hybrid-PCM and BI-PCM with compound polar code on 64-QAM.

BI-PCM.

Although we only discussed polar codes as component codes for our hybrid coded modulation, in principle, just like MLC and BICM, any other codes such as Turbo or LDPC codes can also be chosen as component codes in our hybrid scheme. The flexibility of working together with other codes, reduced latency compared to ML-PCM, and the large performance gain over BI-PCM on high order modulation make the proposed hybrid polar coded modulation scheme attractive for future communication systems such as 6G.

6.6 Acknowledgements

This chapter, in part, has been published at 2022 IEEE International Symposium on Information Theory (ISIT) and appeared as: Hanwen Yao, Jinfeng Du, and Alexander Vardy "Polar Coded Modulation via Hybrid Bit Labeling" [YDV22]. The dissertation author was the

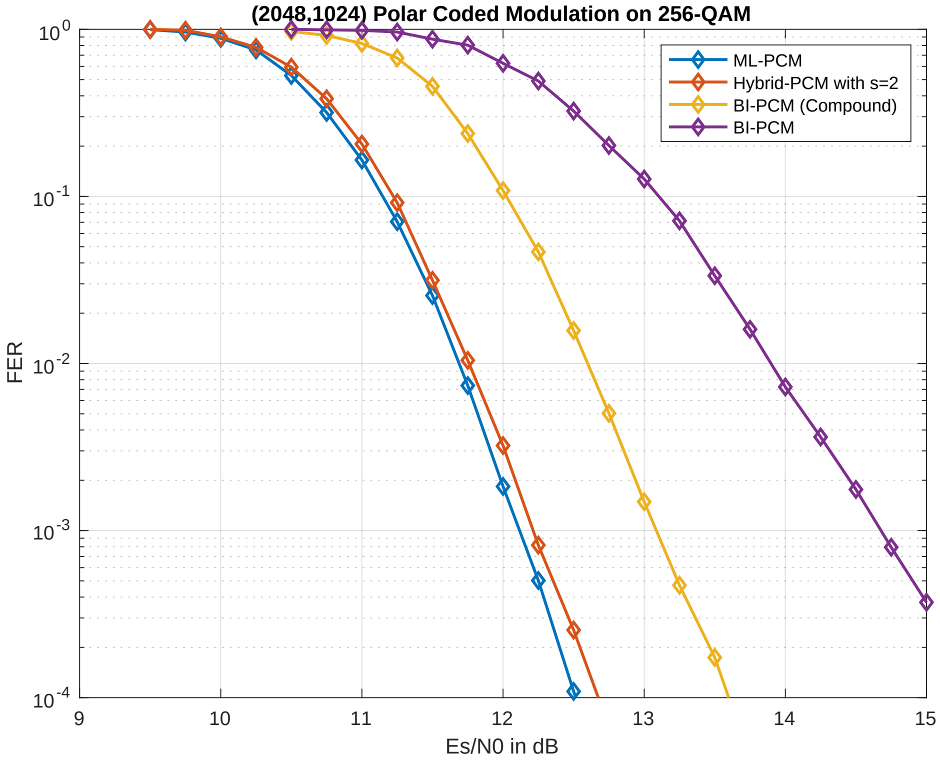

Figure 6.6. Performance comparison for ML-PCM, Hybrid-PCM, BI-PCM with compound polar code, and plain BI-PCM on 256-QAM

primary author of this conference paper.

Bibliography

[3GP18] ETSI TS 138.212 V15.2.0 LTE; multiplexing and channel coding (3GPP TS 38.212 version 15.2.0 release 15), July 2018.

[Arı09] Erdal Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7):3051–3073, 2009.

[Arı19] Erdal Arıkan. From sequential decoding to channel polarization and back again. *arXiv preprint arXiv:1908.09594*, 2019.

[Arı20] Erdal Arıkan. Systematic encoding and shortening of pac codes. *Entropy*, 22(11):1301, 2020.

[AT09] Erdal Arikan and Emre Telatar. On the rate of channel polarization. In *2009 IEEE International Symposium on Information Theory*, pages 1493–1495. IEEE, 2009.

[AV20] Fariba Abbasi and Emanuele Viterbo. Large kernel polar codes with efficient window decoding. *IEEE Transactions On Vehicular Technology*, 69(11):14031–14036, 2020.

[AY20] Emmanuel Abbe and Min Ye. Reed-muller codes polarize. *IEEE Transactions on Information Theory*, 66(12):7311–7332, 2020.

[AYK11] Amin Alamdar-Yazdi and Frank R Kschischang. A simplified successive-cancellation decoder for polar codes. *IEEE communications letters*, 15(12):1378–1380, 2011.

[BCL20] Valerio Bioglio, Carlo Condo, and Ingmar Land. Design of polar codes in 5g new radio. *IEEE Communications Surveys & Tutorials*, 23(1):29–40, 2020.

[BDOT16] Magali Bardet, Vlad Dragoi, Ayoub Otmani, and Jean-Pierre Tillich. Algebraic properties of polar codes from a new polynomial formalism. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 230–234. IEEE, 2016.

[BFS$^+$17a] Sarit Buzaglo, Arman Fazeli, Paul H Siegel, Veeresh Taranalli, and Alexander Vardy. On efficient decoding of polar codes with large kernels. In *2017 IEEE*

Wireless Communications and Networking Conference Workshops (WCNCW), pages 1–6. IEEE, 2017.

[BFS$^+$17b] Sarit Buzaglo, Arman Fazeli, Paul H Siegel, Veeresh Taranalli, and Alexander Vardy. Permuted successive cancellation decoding for polar codes. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2618–2622. IEEE, 2017.

[BL18] Valerio Bioglio and Ingmar Land. On the marginalization of polarizing kernels. In *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*, pages 1–5. IEEE, 2018.

[BMVT78] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[BPYS17] Georg Bocherer, Tobias Prinz, Peihong Yuan, and Fabian Steiner. Efficient polar code construction for higher-order modulation. In *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6. IEEE, 2017.

[CDJ$^+$19] Mustafa Cemil Coşkun, Giuseppe Durisi, Thomas Jerkovits, Gianluigi Liva, William Ryan, Brian Stein, and Fabian Steiner. Efficient error-correcting codes in the short blocklength regime. *Physical Communication*, 34:66–79, 2019.

[CNL13] Kai Chen, Kai Niu, and Jia-Ru Lin. An efficient design of bit-interleaved polar coded modulation. In *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 693–697. IEEE, 2013.

[CNP20] Mustafa Cemil Coşkun, Joachim Neu, and Henry D Pfister. Successive cancellation inactivation decoding for modified reed-muller and ebch codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 437–442. IEEE, 2020.

[CTB98] Giuseppe Caire, Giorgio Taricco, and Ezio Biglieri. Bit-interleaved coded modulation. *IEEE transactions on information theory*, 44(3):927–946, 1998.

[Ers16] Tomaso Erseghe. Coding in the finite-blocklength regime: Bounds based on laplace integrals and their asymptotic approximations. *IEEE Transactions on Information Theory*, 62(12):6854–6883, 2016.

[Fan63] Robert Fano. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, 9(2):64–74, 1963.

[FHMV20] Arman Fazeli, Hamed Hassani, Marco Mondelli, and Alexander Vardy. Binary linear codes with optimal scaling: Polar codes with large kernels. *IEEE Transactions on Information Theory*, 67(9):5693–5710, 2020.

[FHP17] Andrew James Ferris, Christoph Hirche, and David Poulin. Convolutional polar codes. *arXiv preprint arXiv:1704.00715*, 2017.

[FP13] Andrew J Ferris and David Poulin. Branching mera codes: a natural extension of polar codes. *arXiv preprint arXiv:1312.4575*, 2013.

[FTV17] Arman Fazeli, Kuangda Tian, and Alexander Vardy. Viterbi-aided successive-cancellation decoding of polar codes. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[FV14] Arman Fazeli and Alexander Vardy. On the scaling exponent of binary polarization kernels. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 797–804. IEEE, 2014.

[FVY19] Arman Fazeli, Alexander Vardy, and Hanwen Yao. Convolutional decoding of polar codes. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1397–1401. IEEE, 2019.

[FVY20] Arman Fazeli, Alexander Vardy, and Hanwen Yao. Hardness of successive-cancellation decoding of linear codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 455–460. IEEE, 2020.

[FVY21] Arman Fazeli, Alexander Vardy, and Hanwen Yao. List decoding of polar codes: How large should the list be to achieve ml decoding? In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1594–1599. IEEE, 2021.

[Gal68] Robert G Gallager. *Information theory and reliable communication*, volume 588. Springer, 1968.

[GB14] Dina Goldin and David Burshtein. Improved bounds on the finite length scaling of polar codes. *IEEE Transactions on Information Theory*, 60(11):6966–6978, 2014.

[GB19] Dina Goldin and David Burshtein. Performance bounds of concatenated polar coding schemes. *IEEE Transactions on Information Theory*, 65(11):7131–7148, 2019.

[GRY20] Venkatesan Guruswami, Andrii Riazanov, and Min Ye. Arikan meets shannon: Polar codes with near-optimal convergence to channel capacity. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 552–564, 2020.

[GX14] Venkatesan Guruswami and Patrick Xia. Polar codes: Speed of polarization and polynomial gap to capacity. *IEEE Transactions on Information Theory*, 61(1):3–16, 2014.

[GYFV21] Bhaskar Gupta, Hanwen Yao, Arman Fazeli, and Alexander Vardy. Polar list decoding for large polarization kernels. In *2021 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2021.

[HA17] Muhammad Hanif and Masoud Ardakani. Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes. *IEEE Communications Letters*, 21(11):2360–2363, 2017.

[HAU14] Seyed Hamed Hassani, Kasra Alishahi, and Rüdiger L Urbanke. Finite-length scaling for polar codes. *IEEE Transactions on Information Theory*, 60(10):5875–5898, 2014.

[Hay09] Masahito Hayashi. Information spectrum approach to second-order coding rate in channel coding. *IEEE Transactions on Information Theory*, 55(11):4947–4966, 2009.

[HMTU12] S Hamed Hassani, Ryuhei Mori, Toshiyuki Tanaka, and Rüdiger L Urbanke. Rate-dependent analysis of the asymptotic behavior of channel polarization. *IEEE Transactions on Information Theory*, 59(4):2267–2276, 2012.

[HZZ$^+$18] Zhiliang Huang, Shiyi Zhang, Feiyan Zhang, Chunjiang Duanmu, Farong Zhong, and Ming Chen. Simplified successive cancellation decoding of polar codes with medium-dimensional binary kernels. *IEEE Access*, 6:26707–26717, 2018.

[iFMC08] Albert Guillén i Fabregas, Alfonso Martinez, and Giuseppe Caire. Bit-interleaved coded modulation. 2008.

[IH77] Hideki Imai and Shuji Hirakawa. A new multilevel coding method using error-correcting codes. *IEEE Transactions on Information Theory*, 23(3):371–377, 1977.

[KMTU10] Satish Babu Korada, Andrea Montanari, Emre Telatar, and Rüdiger Urbanke. An empirical scaling law for polar codes. In *2010 IEEE International Symposium on Information Theory*, pages 884–888. IEEE, 2010.

[KŞU10] Satish Babu Korada, Eren Şaşoğlu, and Rüdiger Urbanke. Polar codes: Characterization of exponent, bounds, and constructions. *IEEE Transactions on Information Theory*, 56(12):6253–6264, 2010.

[KTA76] Tadao Kasami, Nobuki Tokura, and Saburo Azumi. On the weight enumeration of weights less than 2.5 d of reed—muller codes. *Information and control*, 30(4):380–395, 1976.

[LLAG15] Hsien-Ping Lin, Shu Lin, and Khaled AS Abdel-Ghaffar. Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents. *IEEE Transactions on Information Theory*, 61(10):5253–5270, 2015.

[LST12] Bin Li, Hui Shen, and David Tse. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Communications Letters*, 16(12):2044–2047, 2012.

[LST14] Bin Li, Hui Shen, and David Tse. A rm-polar codes. *arXiv preprint arXiv:1407.5483*, 2014.

[LZG19] Bin Li, Huazi Zhang, and Jiaqi Gu. On pre-transformed polar codes. *arXiv preprint arXiv:1912.06359*, 2019.

[Mac63] J. A. MacWilliams. A theorem on the distribution of weights in a systematic code. *Bell System Technical Journal*, 42(1):79–94, 1963.

[Mah17] Hessam Mahdavifar. Fast polarization for non-stationary channels. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 849–853. IEEE, 2017.

[MEKLK15] Hessam Mahdavifar, Mostafa El-Khamy, Jungwon Lee, and Inyup Kang. Polar coding for bit-interleaved coded modulation. *IEEE Transactions on Vehicular Technology*, 65(5):3115–3127, 2015.

[MHU14] Marco Mondelli, S Hamed Hassani, and Rüdiger L Urbanke. From polar to reed-muller codes: A technique to improve the finite-length performance. *IEEE Transactions on Communications*, 62(9):3084–3091, 2014.

[MHU16] Marco Mondelli, S Hamed Hassani, and Rüdiger L Urbanke. Unified scaling of polar codes: Error exponent, scaling exponent, moderate deviations, and error floors. *IEEE Transactions on Information Theory*, 62(12):6698–6712, 2016.

[MK20] Samir Kumar Mishra and KwangChul Kim. Selectively precoded polar codes. *arXiv preprint arXiv:2011.04930*, 2020.

[MMQA20] Mohsen Moradi, Amir Mozammel, Kangjian Qin, and Erdal Arikan. Performance and complexity of sequential decoding of pac codes. *arXiv preprint arXiv:2012.04990*, 2020.

[Mor20] Ruslan Alexandrovich Morozov. Convolutional polar kernels. *IEEE Transactions on Communications*, 68(12):7352–7361, 2020.

[Moz20] Amir Mozammel. Hardware implementation of fano decoder for PAC codes. *CoRR*, abs/2011.09819, 2020.

[MT12] Vera Miloslavskaya and Peter Trifonov. Design of binary polar codes with arbitrary kernel. In *2012 IEEE Information Theory Workshop*, pages 119–123. IEEE, 2012.

[MT14] Vera Miloslavskaya and Peter Trifonov. Sequential decoding of polar codes. *IEEE Communications Letters*, 18(7):1127–1130, 2014.

[MT19a] Ruslan Morozov and Peter Trifonov. On distance properties of convolutional polar codes. *IEEE Transactions on Communications*, 67(7):4585–4592, 2019.

[MT19b] Ruslan Morozov and Peter Trifonov. Successive and two-stage systematic encoding of polar subcodes. *IEEE Wireless Communications Letters*, 8(3):877–880, 2019.

[MT20] Elizaveta Moskovskaya and Peter Trifonov. Design of bch polarization kernels with reduced processing complexity. *IEEE Communications Letters*, 24(7):1383–1386, 2020.

[MV20] Vera Miloslavskaya and Branka Vucetic. Design of short polar codes for scl decoding. *IEEE Transactions on Communications*, 68(11):6657–6668, 2020.

[NC12] Kai Niu and Kai Chen. Crc-aided decoding of polar codes. *IEEE communications letters*, 16(10):1668–1671, 2012.

[NLW19] Kai Niu, Yan Li, and Weiling Wu. Polar codes: Analysis and construction based on polar spectrum. *arXiv preprint arXiv:1908.05889*, 2019.

[PDP20] Rina Polyanskaya, Mars Davletshin, and Nikita Polyanskii. Weight distributions for successive cancellation decoding of polar codes. *IEEE Transactions on Communications*, 68(12):7328–7336, 2020.

[PPV10] Yury Polyanskiy, H Vincent Poor, and Sergio Verdú. Channel coding rate in the finite blocklength regime. *IEEE Transactions on Information Theory*, 56(5):2307–2359, 2010.

[PSL$^+$15] Noam Presman, Ofer Shapira, Simon Litsyn, Tuvi Etzion, and Alexander Vardy. Binary polarization kernels from code decompositions. *IEEE Transactions on Information Theory*, 61(5):2227–2239, 2015.

[PU19] Henry D Pfister and Rüdiger L Urbanke. Near-optimal finite-length scaling for polar codes over large alphabets. *IEEE Transactions on Information Theory*, 65(9):5643–5655, 2019.

[RBV20] Mohammad Rowshan, Andreas Burg, and Emanuele Viterbo. Polarization-adjusted convolutional (pac) codes: Fano decoding vs list decoding. *arXiv preprint arXiv:2002.06805*, 2020.

[RV21] Mohammad Rowshan and Emanuele Viterbo. List viterbi decoding of pac codes. *IEEE Transactions on Vehicular Technology*, 70(3):2428–2435, 2021.

[Sch16] Christian Schürch. A partial order for the synthesized channels of a polar code. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 220–224. IEEE, 2016.

[SF07] Clemens Stierstorfer and Robert FH Fischer. (gray) mappings for bit-interleaved coded modulation. In *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pages 1703–1707. IEEE, 2007.

[SGV$^+$14a] Gabi Sarkis, Pascal Giard, Alexander Vardy, Claude Thibeault, and Warren J Gross. Fast polar decoders: Algorithm and implementation. *IEEE Journal on Selected Areas in Communications*, 32(5):946–957, 2014.

[SGV$^+$14b] Gabi Sarkis, Pascal Giard, Alexander Vardy, Claude Thibeault, and Warren J Gross. Increasing the speed of polar list decoders. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2014.

[SGV$^+$15] Gabi Sarkis, Pascal Giard, Alexander Vardy, Claude Thibeault, and Warren J Gross. Fast list decoders for polar codes. *IEEE Journal on Selected Areas in Communications*, 34(2):318–328, 2015.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[SITK71] M Sugino, Y Ienaga, Nobuki Tokura, and Tadao Kasami. Weight distribution of (128, 64) reed-muller code (corresp.). *IEEE Transactions on Information Theory*, 17(5):627–628, 1971.

[SS$^+$06] Igal Sason, Shlomo Shamai, et al. Performance analysis of linear codes under maximum-likelihood decoding: A tutorial. *Foundations and Trends® in Communications and Information Theory*, 3(1–2):1–222, 2006.

[SSSH13] Mathis Seidl, Andreas Schenk, Clemens Stierstorfer, and Johannes B Huber. Polar-coded modulation. *IEEE Transactions on Communications*, 61(10):4108–4119, 2013.

[Str62] Volker Strassen. Asymptotische abschatzugen in shannon's informationstheorie. In *Transactions of the Third Prague Conference on Information Theory etc, 1962. Czechoslovak Academy of Sciences, Prague*, pages 689–723, 1962.

[TG21] Thibaud Tonnellier and Warren J Gross. On systematic polarization-adjusted convolutional (pac) codes. *IEEE Communications Letters*, 25(7):2128–2132, 2021.

[TM13] Peter Trifonov and Vera Miloslavskaya. Polar codes with dynamic frozen symbols and their decoding by directed search. In *2013 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2013.

[TM15] Peter Trifonov and Vera Miloslavskaya. Polar subcodes. *IEEE Journal on Selected Areas in Communications*, 34(2):254–266, 2015.

[Tri14] Peter Trifonov. Binary successive cancellation decoding of polar codes with reed-solomon kernel. In *2014 IEEE International Symposium on Information Theory*, pages 2972–2976. IEEE, 2014.

[Tri17] Peter Trifonov. Star polar subcodes. In *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6, 2017.

[Tri18] Peter Trifonov. A score function for sequential decoding of polar codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1470–1474. IEEE, 2018.

[Tri19a] Peter Trifonov. On construction of polar subcodes with large kernels. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1932–1936. IEEE, 2019.

[Tri19b] Peter Trifonov. Trellis-based decoding techniques for polar codes with large kernels. In *2019 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2019.

[Tri20] Peter Trifonov. Randomized polar subcodes with optimized error coefficient. *IEEE Transactions on Communications*, 68(11):6714–6722, 2020.

[TT17] Peter Trifonov and Grigorii Trofimiuk. A randomized construction of polar subcodes. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1863–1867. IEEE, 2017.

[TT18] Grigorii Trofimiuk and Peter Trifonov. Efficient decoding of polar codes with some 16×16 kernels. In *2018 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2018.

[TT19] Grigorii Trofimiuk and Peter Trifonov. Reduced complexity window processing of binary polarization kernels. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1412–1416. IEEE, 2019.

[TT21] Grigorii Trofimiuk and Peter Trifonov. Window processing of binary polarization kernels. *IEEE Transactions on Communications*, 2021.

[TV15] Ido Tal and Alexander Vardy. List decoding of polar codes. *IEEE Transactions on Information Theory*, 61(5):2213–2226, 2015.

[Var98] A. Vardy. *Handbook of Coding Theory*, chapter Trellis structure of codes. Elsevier, 1998.

[VViFKL18] Gonzalo Vazquez-Vilar, Albert Guillen i Fabregas, Tobias Koch, and Alejandro Lancho. Saddlepoint approximation of the error probability of binary hypothesis testing. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2306–2310. IEEE, 2018.

[VY13] Mehrdad Valipour and Shahram Yousefi. On probabilistic weight distribution of polar codes. *IEEE communications letters*, 17(11):2120–2123, 2013.

[WD20] Hsin-Po Wang and Iwan M Duursma. Polar codes' simplicity, random codes' durability. *IEEE Transactions on Information Theory*, 67(3):1478–1508, 2020.

[WFH99] Udo Wachsmann, Robert FH Fischer, and Johannes B Huber. Multilevel codes: Theoretical concepts and practical design rules. *IEEE Transactions on Information Theory*, 45(5):1361–1391, 1999.

[WJZL20] Liyang Wang, Ming Jiang, Chunming Zhao, and Zhengyi Li. Genetic optimization of short block-length pac codes for high capacity phz communications. In *International Conference on Optoelectronic and Microelectronic Technology and Application*, volume 11617, pages 637–642. SPIE, 2020.

[WLVG22] Hsin-Po Wang, Ting-Chun Lin, Alexander Vardy, and Ryan Gabrys. Sub-4.7 scaling exponent of polar codes. *arXiv preprint arXiv:2204.11683*, 2022.

[YA20] Min Ye and Emmanuel Abbe. Recursive projection-aggregation decoding of reed-muller codes. *IEEE Transactions on Information Theory*, 66(8):4948–4965, 2020.

[YDV22] Hanwen Yao, Jinfeng Du, and Alexander Vardy. Polar coded modulation via hybrid bit labeling. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 980–985, 2022.

[YFV19] Hanwen Yao, Arman Fazeli, and Alexander Vardy. Explicit polar codes with small scaling exponent. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1757–1761. IEEE, 2019.

[YFV20] Hanwen Yao, Arman Fazeli, and Alexander Vardy. List decoding of arıkan's pac codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 443–448, 2020.

[YFV21a] Hanwen Yao, Arman Fazeli, and Alexander Vardy. A deterministic algorithm for computing the weight distribution of polar codes. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1218–1223. IEEE, 2021.

[YFV21b] Hanwen Yao, Arman Fazeli, and Alexander Vardy. List decoding of arıkan's pac codes. *Entropy*, 23(7):841, 2021.

[YPB$^+$19] Peihong Yuan, Tobias Prinz, Georg Böcherer, Onurcan Iscan, Ronald Boehnke, and Wen Xu. Polar code construction for list decoding. In *SCC 2019; 12th International ITG Conference on Systems, Communications and Coding*, pages 1–6. VDE, 2019.

[ZCZ$^+$20] Hongfei Zhu, Zhiwei Cao, Yuping Zhao, Dou Li, and Yanjun Yang. Fast list decoders for polarization-adjusted convolutional (pac) codes. *arXiv preprint arXiv:2012.09425*, 2020.

[ZLP17] Qingshuang Zhang, Aijun Liu, and Xiaofei Pan. An enhanced probabilistic computation method for the weight distribution of polar codes. *IEEE Communications Letters*, 21(12):2562–2565, 2017.