PERFORMANCE AND COMPUTATIONAL ANALYSIS OF POLARIZATION-ADJUSTED CONVOLUTIONAL (PAC) CODES

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

By

Mohsen Moradi

June 2022

Performance and Computational Analysis of Polarization-Adjusted Convolutional (PAC) Codes

By Mohsen Moradi

June 2022

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Erdal Arıkan(Advisor)

Tolga M. Duman

Ahmed Hareedy

Orhan Arıkan

Barış Nakiboğlu

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

PERFORMANCE AND COMPUTATIONAL ANALYSIS OF POLARIZATION-ADJUSTED CONVOLUTIONAL (PAC) CODES

Mohsen Moradi
Ph.D. in Electrical and Electronics Engineering
Advisor: Erdal Arıkan
June 2022

We study the performance of sequential decoding of polarization-adjusted convolutional (PAC) codes. We present a metric function that employs bit-channel mutual information and cutoff rate values as the bias values and significantly reduces the computational complexity while retaining the excellent error-correction performance of PAC codes. With the proposed metric function, the computational complexity of sequential decoding of PAC codes is equivalent to that of conventional convolutional codes.

Our results indicate that the upper bound on the sequential decoding computational complexity of PAC codes follows a Pareto distribution. We also employ guessing technique to derive a lower bound on the computational complexity of sequential decoding of PAC codes. To reduce the PAC sequential decoder's worst-case latency, we restrict the number of searches executed by the sequential decoder.

We introduce an improvement to the successive-cancellation list (SCL) decoding for polarized channels that reduces the number of sorting operations without degrading the code's error-correction performance. In an SCL decoding with an optimum metric function, we show that, on average, the correct branch's bit-metric value must be equal to the bit-channel capacity. On the other hand, the average bit-metric value of a wrong branch can be at most 0. This implies that a wrong path's partial path metric value deviates from the bit-channel capacity's partial summation. This enables the decoder to identify incorrect branches and exclude them from the list of metrics to be sorted. We employ a similar technique to the stack algorithm, resulting in a considerable reduction in the stack size.

Additionally, we propose a technique for constructing a rate profile for PAC codes of arbitrary length and rate which is capable of balancing the error-correction performance and decoding complexity of PAC codes. For signal-to-noise ratio (SNR) values larger than a target SNR value, the proposed approach

can significantly enhance the error-correction performance of PAC codes while retaining a low mean sequential decoding complexity.

Finally, we examine the weight distribution of PAC codes with the goal of providing a new demonstration that PAC codes surpass polar codes in terms of weight distribution.

Keywords: polarization-adjusted convolutional codes, polar codes, convolutional codes, sequential decoding, successive cancellation decoding, list decoding, weight distribution.

ÖZET

KUTUPSAL VE POLARİZAYSON AYARLI EVRİŞİMLİ (PAC) KODLARININ PERFORMANS VE HESAPLAMA ANALİZİ

Mohsen Moradi
Elektrik ve Elektronik Mühendisliği, Doktora
Tez Danışmanı: Akademik Ünvansız isim X
Haziran 2022

Kutupsal ve polarizasyon ayarlı evrişimli (PAC) kodların sıralı kod çözme performansını inceliyoruz. Sapma değerleri olarak bit kanalı karşılıklı bilgi ve kesme oranı değerlerini kullanan ve PAC kodlarının mükemmel hata düzeltme performansını korurken hesaplama karmaşıklığını önemli ölçüde azaltan bir metrik sunuyoruz. Önerilen metrik fonksiyonu sayesinde, PAC kodlarının sıralı kod çözme hesaplama karmaşıklığı, geleneksel evrişimli kodlarınkine eşdeğer olmaktadır.

Sonuçlarımız, PAC kodlarının sıralı kod çözme hesaplama karmaşıklığının üst sınırının bir Pareto dağılımını izlediğini göstermektedir. Ayrıca, PAC kodlarının sıralı kod çözme hesaplama karmaşıklığına ilişkin bir alt sınır elde etmek için tahmin tekniğini kullanıyoruz. PAC sıralı kod çözücünün gecikmesinin en kötü durumda alabileceği değeri azaltmak için sıralı kod çözücü tarafından yürütülen aramaların sayısını kısıtlıyoruz.

Polarize kanallar için sıralı elemeli ve listeli (SCL) kod çözme işleminde, kodun hata düzeltme performansını düşürmeden sıralama işlemlerinin sayısını azaltan bir iyileştirme sunuyoruz. Optimum metrik fonksiyona sahip bir SCL kod çözmede, ortalama olarak doğru dalın bit metrik değerinin bit kanalı kapasitesine eşit olması gerektiğini gösteriyoruz. Öte yandan, yanlış bir dalın ortalama bit metrik değeri en fazla sıfır olabilir. Bu durum, yanlış bir yolun kısmi yol metrik değerinin bit kanalı kapasitesinin kısmi toplamından saptığı anlamına gelir. Bu, kod çözücünün yanlış dalları tanımlamasını ve bunları sıralanacak metrikler listesinden çıkarmasını sağlar. Burada, yığın boyutunda önemli bir azalmayla sonuçlanan yığın algoritmasına benzer bir teknik kullanıyoruz.

Ek olarak, herhangi bir uzunluk ve hızdaki PAC kodları için, PAC kodlarının hata düzeltme performansını ve kod çözme karmaşıklığını dengeleyebilen bir oran profili oluşturmak adına bir teknik öneriyoruz. Önerilen yaklaşım, hedeflenen

sinyal-ses oranı (SNR) değerinden daha büyük SNR değerleri için, düşük bir ortalama sıralı kod çözme karmaşıklığını korumaktadır. Bununla birlikte, PAC kodlarının hata düzeltme performansını da önemli ölçüde artırabilir.

Son olarak, PAC kodlarının ağırlık dağılımı açısından polar kodları geride bıraktığının yeni bir gösterimini sağlamak amacıyla PAC kodlarının ağırlık dağılımını inceliyoruz.

Anahtar sözcükler: Kutupsal ve polarizasyon ayarlı evrişimli kodlar, kutupsal kodlar, evrişimli kodlar, sıralı kod çözme, sıralı elemeli kod çözme, liste kod çözme, ağırlık dağılımı..

Acknowledgement

I would like to thank my adviser, Professor Erdal Arıkan, for the guidance and support that he has provided. I am thankful to Professors Tolga M. Duman, Ahmed Hareedy, Orhan Arıkan, and Barış Nakiboğlu for reading my dissertation and providing helpful comments and suggestions during my defense. I was fortunate to begin and complete my doctorate at the same time as Amir Mozammel. This has made this stage of my life more bearable; our discussions contribute to the advancement of our projects. I also want to express my appreciation to Bilkent University for all of the provided support .

I would want to continue by expressing my deepest thanks and gratitude to my parents and the rest of my family for the unwavering support and encouragement they have provided to me during all of these years. Without their help, I never would have been able to get to this point in my life. Many thanks to everyone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The fundamental objective of channel coding is to develop systems that enable *reliable* and *efficient* data transmission. A basic model of a communication system is best explained by the block diagram in Figure 1.1 [1]. In this research, we treat both the source encoder outputs and the source decoder inputs as binary sequences, and we only focus on the channel coding problem depicted inside the dashed contour in this figure.

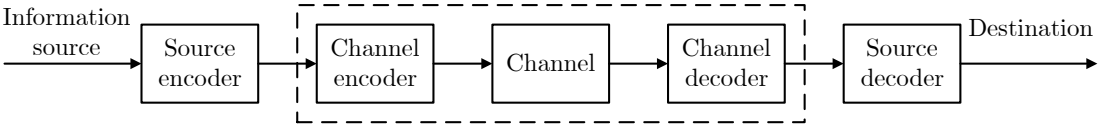

Figure 1.1: Block diagram of a digital data transmission system

The main result of Shannon's work [1] is that if the code *rate* is less than a certain value set by the *channel capacity* (a fundamental characteristic of the channel), there are encoding and decoding operations that can *asymptotically* lead to reliable reconstruction of the encoder input sequence. Shannon established that selecting a random element from a properly specified ensemble could create a reliable system with a high probability. The issue comes when we want low-complexity encoding and decoding techniques (efficient) that satisfy the requirements mentioned above.

Polar codes [2] are the first class of linear block codes that, based on channel polarization, provably achieve the capacity of certain classes of channels with practical encoding and decoding algorithms and explicit code construction. In particular, under low-complexity successive cancellation (SC) decoder, it is proved that the error probability of polar codes vanishes as code blocklength increases [2].

Earlier research on the performance of polar codes with SC decoder indicated that their performance was inferior to that of the state-of-the-art codes such as low-density parity-check (LDPC) [3–7] and turbo [8] codes at low-to-moderate block lengths [9]. Tal and Vardy soon demonstrated that by using an SC list (SCL) decoder, the performance of polar codes could be considerably improved [9]. An SCL decoder initially established for Reed-Muller (RM) codes [10, 11], with list size L maintains a list of L candidate codewords and chooses the most likely one as its choice at the end of the decoding process. Additionally, the SCL decoding of polar codes permits the concatenating of a polar code with a cyclic redundancy check (CRC) code as an outer code to further improve polar codes' block-error probability [12]. Polar codes are comparable to the state-of-the-art turbo and LDPC codes when decoded using a CRC-aided SCL decoding [9].

Recently, Arıkan proposed a novel coding method termed polarization-adjusted convolutional (PAC) coding that utilizes pretransformed convolutional codes (CCs) to polar codes [13]. PAC codes have a significant performance gain compared to the CRC-aided SCL decoding of conventional polar codes for small blocklengths. The experimental findings demonstrate that the PAC codes approach the theoretical limits for finite blocklength codes [13–15].

We investigate both the SCL and sequential decoding of PAC codes. Wozencraft [16] introduced sequential decoding, and the Fano [17] and stack [18,19] algorithms are two of the most well-known sequential decoding algorithms. The stack algorithm visits each node at most once during a decoding session but needs more storage capacity, while the Fano algorithm may visit each node several times but requires less memory. However, Fano decoding is more advantageous for practical hardware implementations of PAC codes due to its low memory requirements [20]. Since both algorithms ultimately pick the same pathways through the decoding

tree, the set of nodes traversed by the Fano and stack algorithms is identical [21]. In this dissertation, we study both Fano and stack decoding algorithms. The purpose of this dissertation is to investigate the error-correction performance and the computational complexity of PAC codes. The structure of the dissertation and my contributions are explained in detail in the following section.

1.1 Organization of the Dissertation

Chapter 2 provides a review of prior art on polar, convolutional, and PAC codes and their decoding methods. We discuss SCL and sequential (stack and Fano) decoding algorithms. This chapter also thoroughly explores several code construction techniques necessary for the subsequent chapters.

Chapter 3 provides a metric function for sequential decoding of PAC codes that takes advantage of the channel polarization. We investigate the relationship between the metric function in an ensemble of PAC codes and the polarized channels capacities. The fundamental disadvantage of sequential decoding, which contributes to its decline, is that its computational complexity is random. To overcome this, in this chapter, we also consider imposing a strict complexity constraint on the sequential decoding method, analyze its performance under such a constraint, and compare it to CRC-aided SCL decoding of polar codes.

Additionally, in this chapter, in collaboration with Amir Mozammel, we investigate a decoding technique for PAC codes that manages information using a binary heap data structure. Heap data structure is used in sequential decoding in [22]. The heap data structure is an array object that serves as a more efficient priority queue than the stack. Our proposed algorithm employs a vector of length $N/2$ to store intermediate check-sum values, which is half of the needed space reported in the literature. The operations in the heap data structure are in place, and simulation results demonstrate that even with a moderate heap size, PAC codes can achieve good error-correction performance.

Chapter 4 investigates the complexity bounds associated with sequential decoding of PAC codes. We employ guessing in this chapter to derive a lower bound on the computational complexity of sequential decoding of PAC code. By taking advantage of the channel polarization, we prove that the computational cutoff rate for sequential decoding of PAC codes polarizes.

Using channel polarization, we also propose a code construction approach in this chapter that allows PAC codes to function within theoretical limits for various code rates and lengths.

Chapter 5 analyzes the optimal metric function for the SCL decoding algorithm for the polarized channels. In this chapter, we propose an advancement to the SCL decoding for polarized channels that decreases the number of sorting operations without sacrificing the code's error-correction performance. We prove that, in an SCL decoding with an optimal metric function, the average bit-metric value of the correct branch must be equal to the bit-channel capacity and that, on the other hand, the average bit-metric value of a wrong branch can be at most zero. Therefore, the partial path metric value of an incorrect (wrong) path deviates from the partial summation of the bit-channel capacities. Also, in the case of a noiseless bit channel, we prove that the bit metric of a correct branch is 1, whereas the bit metric of an incorrect branch is $-\infty$. In this manner, we would be able to detect incorrect branches in the polarized channels and exclude them from the list of metrics that are needed to be sorted. Our results indicate that, when a list size of 4 is employed, our proposed technique helps the decoder to do almost 92 percent less sorting operations than the standard SCL algorithm while decoding a $(1024, 512)$ polar code at a 3 dB SNR value. We also apply this to the stack decoding algorithm, and the results show that an average stack size reduction of more than 90 percent is attainable for a PAC$(128, 64)$ code when the SNR is set to 3.5 dB. We prune the potential incorrect pathways using a threshold. We prove that when the threshold is less than the bit-channel cutoff rate, the chance of pruning the correct path from that bit of the decoding tree exponentially reduces by the given threshold.

Chapter 6 investigates the weight distribution of PAC codes and gives a

new proof to the result of [23]. This proof aims to demonstrate that PAC codes outperform polar codes in terms of weight distribution. In this chapter, we prove that adding an odd number of clockwise cyclic shifts to any row of the polar code generator matrix G added with some rows below it does not reduce the row's weight. To provide the proof, we use an algebraic equation of [24] which is $D^{N/4}G = GC^{N/4}$ (the notation is defined in the chapter). We generalize this result as $D^m G = GC^m$ for $1 \leq m < N$. This proves that summing a row of the polar code generator matrix with a row below it equals some clockwise cyclic shifts of that row. We use this to prove that the $d_{\min}$ for PAC codes is higher than or equal to the polar code's $d_{\min}$. The maximum likelihood (ML) decoding performance is dictated by the weight distribution of linear codes, which can be confidently predicted by the union bound, especially at high signal-to-noise ratios (SNRs). In terms of error-correction performance, this means that PAC codes outperform polar codes. This chapter

Additionally, in Section 6.2, we look at the performance of PAC codes when various *connection polynomials* are used. This section is written in collaboration with Amir Mozammel.

1.2 Notation

Throughout this dissertation, all the codes are over the binary Galois field $\mathbb{F}_2 = \{0, 1\}$ (GF(2)). We use boldface notation for vectors and for a vector $\mathbf{u} = (u_1, u_2, ..., u_N) \in \mathbb{F}_2^N$, $\mathbf{u}^i$ denotes the subvector $(u_1, u_2, ..., u_i)$ and $\mathbf{u}_i^j$ denotes the subvector $(u_i, ..., u_j)$ for $i \leq j$. For a vector $\mathbf{u}$, $(\mathbf{u})_i$ denotes its ith element u_i. For any subset of indices $\mathcal{A} \subset \{1, 2, ..., N\}$, $\mathcal{A}^c$ denotes the complement of $\mathcal{A}$ and $\mathbf{u}_\mathcal{A}$ represents the subvector $(u_i : i \in \mathcal{A})$. For a matrix G, $G_{\mathcal{A},\mathcal{B}}$ denotes a submatrix of G that rows are selected by set $\mathcal{A}$, and columns are selected by set $\mathcal{B}$.

Papers

1. Moradi, Mohsen. (2021). "On Sequential Decoding Metric Function of Polarization-Adjusted Convolutional (PAC) Codes." IEEE Transactions on Communications (DOI 10.1109/TCOMM.2021.3111018).

2. Moradi, M., Mozammel, A. (2022). "A Tree Pruning Technique for Decoding Complexity Reduction of Polar Codes and PAC Codes." Submitted to the IEEE Transactions on Communications.

3. Moradi, M., Mozammel, A., Qin, K., and Arıkan, E. (2020). "Performance and Complexity of Sequential Decoding of PAC Codes." arXiv preprint arXiv:2012.04990.

4. Moradi, M., Mozammel, A. (2021). "A Monte-Carlo Based Construction of Polarization-Adjusted Convolutional (PAC) Codes." arXiv preprint arXiv:2106.08118.

5. Moradi, M., Mozammel, A. (2022). "Concatenated Reed-Solomon and Polarization-Adjusted Convolutional (PAC) Codes." Accepted by IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom) [25].

6. Moradi, Mohsen. (2022). "Bit-Flipping for Stack Decoding of Polarization-Adjusted Convolutional (PAC) Codes." Accepted by IEEE 10th International Workshop on Signal Design and its Applications in Communications (IWSDA).

Chapter 2

Polar and PAC Codes

This chapter reviews convolutional, polar, and PAC codes and explains the essential prerequisites for the subsequent chapters. In particular, we discuss channel polarization in detail and some well-known techniques for constructing polar and PAC codes. We briefly study the SC [2] and SCL [9, 26, 27] decoding of polar and PAC codes. We also briefly review the sequential decoding of convolutional [17–19] and PAC codes [13, 14, 28].

2.1 Polar Codes and SC Decoding

Polar codes are the first family of codes that have been proved to achieve Shannon channel capacity. Polar codes have a low encoding and decoding complexity, with both encoding and decoding complexity equal to $\mathcal{O}(N \log N)$ for a code of length $N = 2^n$ [2].

This section describes polar codes and discusses their encoding and decoding. A discrete memoryless channel (DMC) $W : \mathcal{X} \to \mathcal{Y}$ has a discrete input alphabet $\mathcal{X}$, a discrete output alphabet $\mathcal{Y}$, and a set of conditional probabilities for the outputs given the inputs. We make the assumption that the input alphabet $\mathcal{X}$ is always binary. The given conditional probabilities are denoted by $W(y|x)$

for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Each channel output letter is solely dependent on the associated input (memoryless condition), thus the conditional probability of a corresponding output sequence, indicated by $\mathbf{y} = (y_1, y_2, \cdots, y_N)$, may be written as

$$W^N(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} W(y_i|x_i) \tag{2.1}$$

for an input sequence of length N, represented by $\mathbf{x} = (x_1, x_2, \cdots, x_N)$.

Given a DMC W, let

$$I(X;Y) \triangleq \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} q(x)W(y|x) \log \frac{W(y|x)}{\sum_{x'\in\mathcal{X}} q(x')W(y|x')} \tag{2.2}$$

denote the average mutual information between the DMC's inputs and outputs.

The channel input and the conditional channel probabilities determine the average mutual information $I(X;Y)$. It should be noted that the channel input probability is independent of the DMC. Then, with regard to the input probability distribution $\mathbf{q} = \{q(x) : x \in \mathcal{X}\}$, we can maximize $I(X;Y)$.

A DMC's channel capacity C is defined as

$$C \triangleq \max_{\mathbf{q}} I(X;Y). \tag{2.3}$$

Symmetric capacity, $I(W)$, is $I(X;Y)$ with $q(.)$ uniform and is a *rate* measure that represents the highest rate at which reliable communication across the channel W is possible while using W's inputs uniformly (channel input probability is as $q(0) = q(1) = \frac{1}{2}$ in the binary input case).

Another fundamental channel parameter is the Bhattacharyya parameter

$$Z(W) \triangleq \sum_{y\in\mathcal{Y}} \sqrt{W(y|0)W(y|1)} \tag{2.4}$$

and is a measure of the channel *reliability*, which is an upper bound on the probability of ML decision error when the channel W is used just once to transmit a bit.

Assume that we use two independent copies of a binary input DMC (BI-DMC) W to send u_1 and u_2 bits, and the corresponding channel outputs are y_1 and y_2. Let $W^{'}$ and $W^{''}$ denote the first and the second copies of the channel W. We have $I(W^{'}) = I(W) = I(W^{''})$ since each transmission utilizes the same channel. Hence, both usages of the channel are equally reliable in this case.

Now let us send $x_1 = u_1 \oplus u_2$ bit over the first copy of the channel W and $x_2 = u_2$ over the second copy of the channel W, where $\oplus$ denotes addition in $\mathbb{F}_2$. Let $W_2^{(1)} : \mathcal{X} \to \mathcal{Y}^2$ represent a synthetic (artificial) channel with an input of u_1 and an output of (y_1, y_2). Also, let $W_2^{(2)} : \mathcal{X} \to \mathcal{Y}^2 \times \mathcal{X}$ represent another synthetic channel with an input of u_2 and an output of (y_1, y_2, u_1). The channels' outputs are regarded to be vector outputs. Also, as Figure 2.1 shows, assume that W_2 is the overal channel with input (u_1, u_2) and output (y_1, y_2).

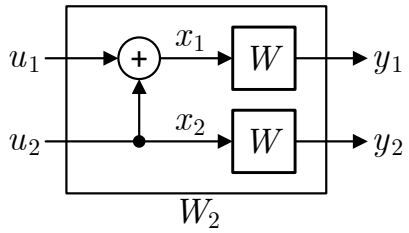

Figure 2.1: The channel W_2

Notice that u_2 has an effect on both y_1 and y_2, providing us with somewhat more information about u_2 than we have about u_1. Furthermore, because $u_1 \oplus u_2$ scrambles u_1 by u_2, there is somewhat lesser information about u_1 than there was in the initial transmission setup. As a result, with further precise descriptions to be established later,

$$I(W_2^{(1)}) \leq I(W) \leq I(W_2^{(2)}). \tag{2.5}$$

By using this fundamental coding method, a synthetic channel $W_2^{(2)}$ that is superior (more reliable) to the channel W is constructed. On the other hand, simultaneously, another channel $W_2^{(1)}$ that is inferior (less reliable) to the channel W is created.

For $N = 2$, $\mathbf{x} = (x_1, x_2)$ are obtained from $\mathbf{u} = (u_1, u_2)$ by

$$\begin{aligned} x_1 &= u_1 \oplus u_2, \\ x_2 &= u_2. \end{aligned} \tag{2.6}$$

This mapping from the input of W_2 to the input of W^2 can be expressed as $\mathbf{x} = \mathbf{u}F$ with

$$F \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.7}$$

$W_2(y_1, y_2|u_1, u_2)$ channel transition probability can be used to obtain the pair of synthetic channel transition probabilities $(W_2^{(1)}, W_2^{(2)})$. The transition probability of this joint channel is indicated by

$$\begin{aligned} W_2(y_1, y_2|u_1, u_2) &= W^2(y_1, y_2|(u_1, u_2)F) = W^2(y_1, y_2|(u_1 \oplus u_2, u_2)) \\ &= W(y_1|u_1 \oplus u_2)W(y_2|u_2), \end{aligned} \tag{2.8}$$

where $W(.|.)$ represents the channel W's transition probability, and where the final equality is true since the channel W is a memoryless channel. The synthetic channel transitions $W_2^{(1)}(y_1, y_2|u_1)$ and $W_2^{(2)}(y_1, y_2, u_1|u_2)$ are calculated using probability rules (assume uniformly distributed input bits) using

$$\begin{aligned} W_2^{(1)}(y_1, y_2|u_1) &= \sum_{u_2 \in \mathcal{X}} W_2(y_1, y_2|u_1, u_2)q(u_2) = \frac{1}{2}\sum_{u_2 \in \mathcal{X}} W_2(y_1, y_2|u_1, u_2) \\ W_2^{(2)}(y_1, y_2, u_1|u_2) &= W_2(y_1, y_2|u_1, u_2)q(u_1) = \frac{1}{2}W_2(y_1, y_2|u_1, u_2), \end{aligned} \tag{2.9}$$

which may be represented as

$$\begin{aligned} W_2^{(1)}(y_1, y_2|u_1) &= \frac{1}{2}\sum_{u_2 \in \mathcal{X}} W(y_1|u_1 \oplus u_2)W(y_2|u_2) \\ W_2^{(2)}(y_1, y_2, u_1|u_2) &= \frac{1}{2}W(y_1|u_1 \oplus u_2)W(y_2|u_2). \end{aligned} \tag{2.10}$$

In general, these equations have converted a pair of channels (W, W) into a pair of new synthesized channels $(W_2^{(1)}, W_2^{(2)})$.

The fundamental concept underlying SC decoding is that bits are estimated in a specific order, and formerly decoded bits are utilized to compute the likelihood

of following bits. The likelihood ratios for the $W_2^{(1)}$ and $W_2^{(2)}$ synthesized channels are used in the decoding process. As a function of the received values (y_1, y_2), let

$$\lambda_2^{(1)}(y_1, y_2) \triangleq \frac{W_2^{(1)}(y_1, y_2|u_1 = 0)}{W_2^{(1)}(y_1, y_2|u_1 = 1)} \tag{2.11}$$

indicate a likelihood ratio for the bit u_1, and

$$z_2^{(1)}(y_1, y_2) \triangleq \log_2 \lambda_2^{(1)}(y_1, y_2) = \log_2 \frac{W_2^{(1)}(y_1, y_2|u_1 = 0)}{W_2^{(1)}(y_1, y_2|u_1 = 1)} \tag{2.12}$$

indicates the log-likelihood ratio (LLR) for this bit. The SC decoder decides that the estimate $\hat{u}_1$ to be 0 if and only if $W_2^{(1)}(y_1, y_2|u_1 = 0) \geq W_2^{(1)}(y_1, y_2|u_1 = 1)$; otherwise the decoder decides that the estimate $\hat{u}_1$ to be 1. This decision rule may be expressed mathematically as

$$\hat{u}_1 = \begin{cases} 0, & \text{if } z_2^{(1)}(y_1, y_2) \geq 0, \\ 1, & \text{if } z_2^{(1)}(y_1, y_2) < 0. \end{cases} \tag{2.13}$$

Similarly,

$$z_2^{(2)}(y_1, y_2, \hat{u}_1) \triangleq \log \frac{W_2^{(2)}(y_1, y_2, \hat{u}_1|u_2 = 0)}{W_2^{(2)}(y_1, y_2, \hat{u}_1|u_2 = 1)} \tag{2.14}$$

indicates the LLR for the bit u_2 and the decision rule to estimate this bit can be formulated as

$$\hat{u}_2 = \begin{cases} 0, & \text{if } z_2^{(2)}(y_1, y_2, \hat{u}_1) \geq 0, \\ 1, & \text{if } z_2^{(2)}(y_1, y_2, \hat{u}_1) < 0. \end{cases} \tag{2.15}$$

Note that to decode the first bit, the channel output vector (y_1, y_2) is needed, and the SC decoder uses the estimated value of the first bit in addition to the channel outputs to decode the second bit.

Let us define the LLRs for the copies of the W channels as

$$L_i \triangleq \frac{W(y_i|x_i = 0)}{W(y_i|x_i = 1)}, \tag{2.16}$$

for $i = 1, 2$. Notice that from (2.6) we have $u_1 = x_1 \oplus x_2$ and using the known single parity check discussion in the LDPC codes context (as an example see [29]),

the LLR value for u_1 can be obtained from the check-node operation like for the LLR values for x_1 and x_2 as

$$\begin{aligned} z_2^{(1)}(y_1, y_2) &= L_1 \boxplus L_2 \\ &= 2\tanh^{-1}(\tanh(L_1/2)\tanh(L_2/2)) \\ &\approx \text{sign}(L_1)\text{sign}(L_2)\min(|L_1|, |L_2|). \end{aligned} \tag{2.17}$$

Also from (2.6) we have $u_2 = x_2$ and by estimating u_1 from $z_2^{(1)}(y_1, y_2)$, we can write $u_1 = x_1 \oplus x_2$ as $\hat{u}_1 = x_1 \oplus u_2$, which equivalently we have

$$\begin{aligned} u_2 &= \hat{u}_1 \oplus x_1 \\ u_2 &= x_2. \end{aligned} \tag{2.18}$$

If $\hat{u}_1 = 0$, using the repetition code discussion of the LDPC codes and (2.18) we have $z_2^{(2)}(y_1, y_2, \hat{u}_1 = 0) = L_2 + L_1$; otherwise, if $\hat{u}_1 = 1$, u_2 is the inverse of x_1 and we can have $z_2^{(2)}(y_1, y_2, \hat{u}_1 = 1) = L_2 - L_1$. Therefore, we can write the LLR value of u_2 as

$$z_2^{(2)}(y_1, y_2, \hat{u}_1) = L_2 + (-1)^{\hat{u}_1} L_1. \tag{2.19}$$

Note that for a binary-input additive white Gaussian noise (BI-AWGN) channel, $L_i = 2y_i/\sigma^2$, where y_i is the ith channel output and σ^2 is the noise power.

It is conventional to show the *check* operation of (2.17) by $f(L_1, L_2)$ known as f-function, and the *node* operation of (2.19) by $g(L_1, L_2, \hat{u}_1)$ known as g-function.

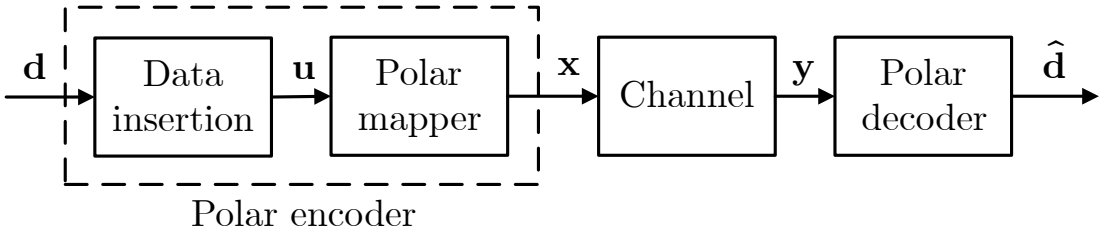

Figure 2.2: Polar coding scheme.

The general polar coding scheme is shown in block diagram form in Figure 2.2. In general, three parameters $(N, K, \mathcal{A})$ can be used to specify a polar code, where $N = 2^n$, $n \geq 1$ is the codeword length, K is the data word length and can be any integer number from 1 to N, and set $\mathcal{A}$, known as the data index set, is a

subset of $\{1, 2, \cdots, N\}$ of size $|\mathcal{A}| = K$. In the following subsections, we will go through some of the most common approaches for acquiring the data index set $\mathcal{A}$.

For an arbitrary blocklength $N = 2^n$, the encoding procedure may be expressed in terms of the generator matrix $F^{\otimes n}$ defined recursively as

$$F^{\otimes n} \triangleq \begin{bmatrix} F^{\otimes n-1} & \mathbf{0} \\ F^{\otimes n-1} & F^{\otimes n-1} \end{bmatrix}, \tag{2.20}$$

where

$$F^{\otimes 1} = F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{2.21}$$

is the kernel matrix of polar codes.

Data insertion is accomplished by inserting the bits of the data vector $\mathbf{d} = (d_1, \cdots, d_K)$ into the bits of a data-carrier vector $\mathbf{u} = (u_1, u_2, \cdots, u_N)$ as $\mathbf{u}_{\mathcal{A}} = \mathbf{d}$ and $\mathbf{u}_{\mathcal{A}^c} = \mathbf{0}$, resulting in an encoding rate of $R = K/N$, and the conventional polar transformation $F^{\otimes n}$ is applied to encode the vector $\mathbf{u}$. Finally, we may recover the data vector estimates $\hat{\mathbf{d}}$ from the channel output $\mathbf{y}$ using a polar decoder such as the SC decoding algorithm.

Example 1. *Consider a* $(8, 4, \{4, 6, 7, 8\})$ *polar code. The data insertion block maps the data vector* $\mathbf{d} = (d_1, d_2, d_3, d_4)$ *to* $\mathbf{u} = (0, 0, 0, d_1, 0, d_2, d_3, d_4)$*. Finally, the polar mapper block performs as*

$$\mathbf{u}F^{\otimes 3} = (u_1, u_2, \cdots, u_8) \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} u_1 \oplus u_2 \oplus \cdots \oplus u_8 \\ u_2 \oplus u_4 \oplus u_6 \oplus u_8 \\ u_3 \oplus u_4 \oplus u_7 \oplus u_8 \\ u_4 \oplus u_8 \\ u_5 \oplus u_6 \oplus u_7 \oplus u_8 \\ u_6 \oplus u_8 \\ u_7 \oplus u_8 \\ u_8 \end{bmatrix}^T$$

to obtain the channel input $\mathbf{x} = \mathbf{u}F^{\otimes 3}$.

To generalize the SC decoding for an arbitrary length, we use the conventional $\mathbf{f}$ and $\mathbf{g}$ vector functions on the LLR vectors $\mathbf{L}^{'} = (a_1, a_2, \cdots, a_M)$ and $\mathbf{L}^{"} = (b_1, b_2, \cdots, b_M)$ and a binary vector $\mathbf{u} = (u_1, u_2, \cdots, u_M)$ defined as

$$\begin{aligned} &\mathbf{f}(\mathbf{L}^{'}, \mathbf{L}^{"}) \triangleq (f(a_1, b_1), f(a_2, b_2), \cdots, f(a_M, b_M)), \\ &\mathbf{g}(\mathbf{L}^{'}, \mathbf{L}^{"}, \mathbf{u}) \triangleq (g(a_1, b_1, u_1), g(a_2, b_2, u_2), \cdots, g(a_M, b_M, u_M)), \end{aligned} \tag{2.22}$$

where the f and g functions are defined as (2.17) and (2.19), respectively.

For an arbitrary blocklength N, expressing SC decoding as a message passing through a tree would be more straightforward. The following discusses SC decoding with a blocklength of $N = 4$. It will be straightforward to generalize it to any blocklength.

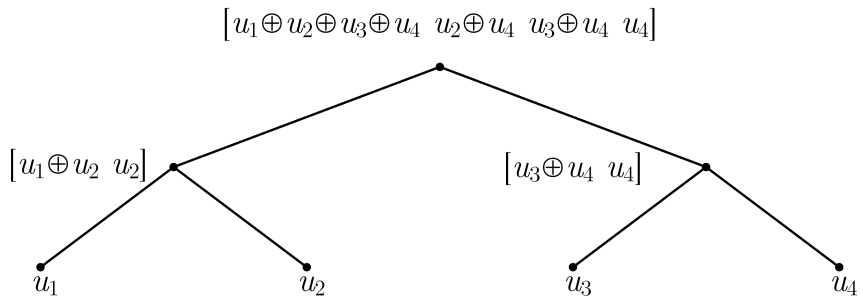

Figure 2.3: Polar encoding tree for $N = 4$.

Figure 2.3 represnts the polar mapper block operation for $N = 4$. The leaf nodes are labeled with the bits of the data-carrier vector $\mathbf{u}$ and the root node is labeled with the vector $\mathbf{x} = \mathbf{u}F^{\otimes 2}$. The output of the channel $\mathbf{y}$ would be the noisy version of $(x_1, x_2, x_3, x_4) = (u_1 \oplus u_2 \oplus u_3 \oplus u_4, \; u_2 \oplus u_4, \; u_3 \oplus u_4, \; u_4)$. Let (L_1, L_2, L_3, L_4) be the output LLR values obtained as (2.16).

Notice that in the encoding tree, the left child of the root node (in general notation: first node of depth 1) is equal to $(x_1 \oplus x_3, \; x_2 \oplus x_4)$; therefore, the LLR values corresponding to this node (with a same justification for $N = 2$) can be obtained as

$$\mathbf{f}(L_1, L_2, L_3, L_4) = (f(L_1, L_3), f(L_2, L_4)). \tag{2.23}$$

In decoding tree, each level of the tree has $N = 4$ LLR values and we

use $\mathbf{L}^d = (L_1^d, L_2^d, L_3^d, L_4^d)$ notation to present the corresponding LLR values for depth d of the tree, which the root node has depth 0 and the leaf nodes are in depth $d = \log_2 N$. Therefore, the channel output LLR values are represented as $(L_1, L_2, L_3, L_4) = (L_1^0, L_2^0, L_3^0, L_4^0)$, and from (2.23), $(L_1^1, L_2^1) = (f(L_1^0, L_3^0), f(L_2^0, L_4^0))$.

Similarly, in the encoding tree, notice that the left child of $(u_1 \oplus u_2, u_2)$ node is the binary summation of these two bits, and its corresponding LLR value can be obtained as $L_1^2 = f(L_1^1, L_2^1)$. Finally we have the LLR value corresponding to the first bit u_1 and the SC decoder based on the sign value of L_1^2 finds $\hat{u}_1$ (if $L_1^2 \geq 0$, $\hat{u}_1 = 0$; otherwise, $\hat{u}_1 = 1$). After this, the SC decoder checks if the first bit is frozen. If the bit is frozen, despite its LLR value, it will always make the estimated bit equal to 0.

The second step of SC decoding is to find the value $\hat{u}_2$. Similar to $N = 2$, its corresponding LLR value L_2^2 can be obtained as $L_2^2 = g(L_1^1, L_2^1, \hat{u}_1)$, and based on this obtained LLR value and checking if it is a frozen bit or not second bit can be estimated.

The third step of the algorithm is to obtain $\hat{u}_3$. With knowing $\hat{u}_1$ and $\hat{u}_2$, the estimated bits of the first node of depth one in the decoding tree can be computed as $(\hat{u}_1 \oplus \hat{u}_2, \hat{u}_2)$. Similar to $N = 2$, the LLR values of the right child of the root node can be calculated as

$$(L_3^1, L_4^1) = \mathbf{g}(L_1^0, L_2^0, L_3^0, L_4^0, (\hat{u}_1 \oplus \hat{u}_2, \hat{u}_2)) = (g(L_1^0, L_3^0, \hat{u}_1 \oplus \hat{u}_2), g(L_2^0, L_4^0, \hat{u}_2)).$$

Now, the LLR value corresponding to the third leaf node can be calculated as $L_3^2 = f(L_3^1, L_4^1)$. Therefore, based on L_3^2 and checking if it is a frozen bit or not, the third bit can be estimated as $\hat{u}_3$.

Finally, the L_4^2 can be obtained as $g(L_3^1, L_4^1, \hat{u}_3)$, and $\hat{u}_4$ can be obtained from L_4^2. This finishes the SC decoding. The LLR values of nodes on decoding tree for $N = 4$ is shown in Figure 2.4.

If the blocklength were longer, we would need to obtain the estimated bits corresponding to the root node of Figure 2.3, which is $(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4)F^{\otimes 2}$, and

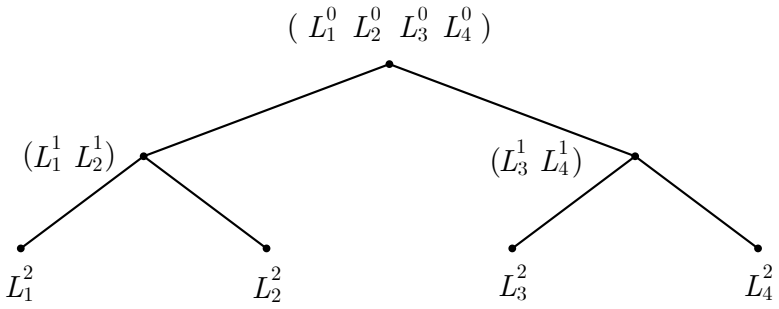

Figure 2.4: SC decoding tree for $N = 4$.

pass them to the parent node, where the decoding process would continue.

In summary, the SC decoder starts from the root node of the decoding tree. On each non-leaf node, it first operates an **f** function towards the left child, and after obtaining the estimated bits corresponding to that node, the SC decoder operates **g** function towards the right child. When reaching a leaf node, the associated bit is estimated depending on its LLR value and whether it is a frozen bit.

The SC decoding described above needs a memory size of $N \log_2 N$ to store the intermediate LLR values and $N \log_2 N$ to store the intermediate check-sum values. This implementation offers a computational benefit in sequential decoding algorithms that need backtracking (as discussed in the next chapter). If no backtracking is required (for sequential decoding algorithms such as the stack algorithm), as discussed in Section 3.5, a memory-efficient SC decoder stores intermediate LLR and check-sum values in a vector of lengths $N - 1$ and $N/2$, respectively.

2.2 On the Construction of Polar Codes

This section reviews some well-known polar code construction techniques for identifying subchannels suitable for transmitting data. The reliabilities of binary erasure channels (BECs) can be efficiently determined by recursively using the Bhattacharyya parameters [2]. Arıkan suggested, as a heuristic method, using the same recursion of Bhattacharyya parameters as for the BEC channel for the

other BI-DMCs [30]. Other well-known polar code constructions are the density evolution (DE) [31], Gaussian approximation (GA) [32], and RM-polar [33] code construction methods. We use the GA algorithm to obtain the polarized Bhattacharyya parameters, cutoff rates, and mutual information in our implementations in this dissertation, as detailed in [14].

Example 2. *For a* $(128, 64)$ *code with an SNR of* 2.5 *dB, Figure 2.5 illustrates the bit-channel mutual information. Data bit locations are determined by the bit channels with the highest bit-channel mutual information.*

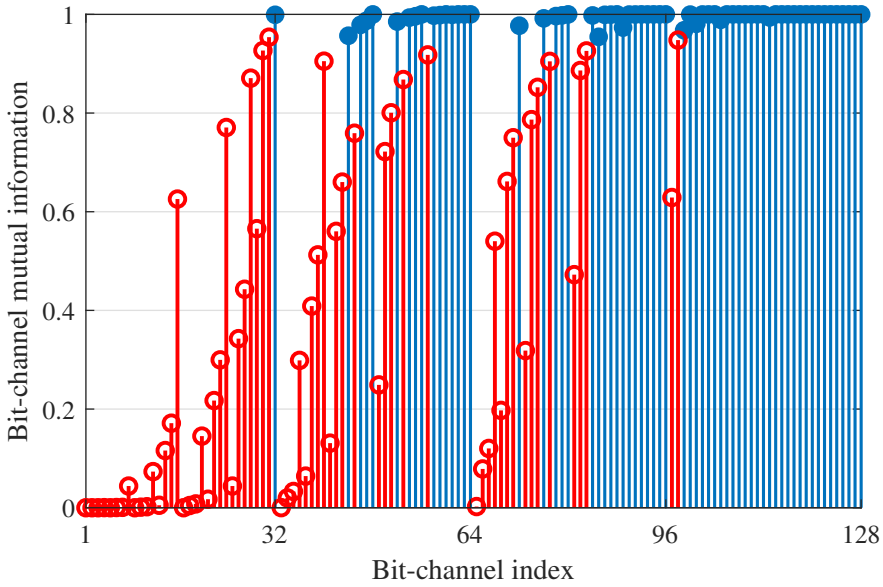

Figure 2.5: Bit-channel capacities for information bits (solid circles) and frozen bits (hollow circles) at 2.5 dB SNR.

Example 3. *For a* $(128, 64)$ *code with SNR* $= 2.5$ *dB, Figure 2.6 depicts the values of bit-channel Bhattacharyya parameters. The bit channels with the lowest values of bit-channel Bhattacharyya parameters determine data bit positions.*

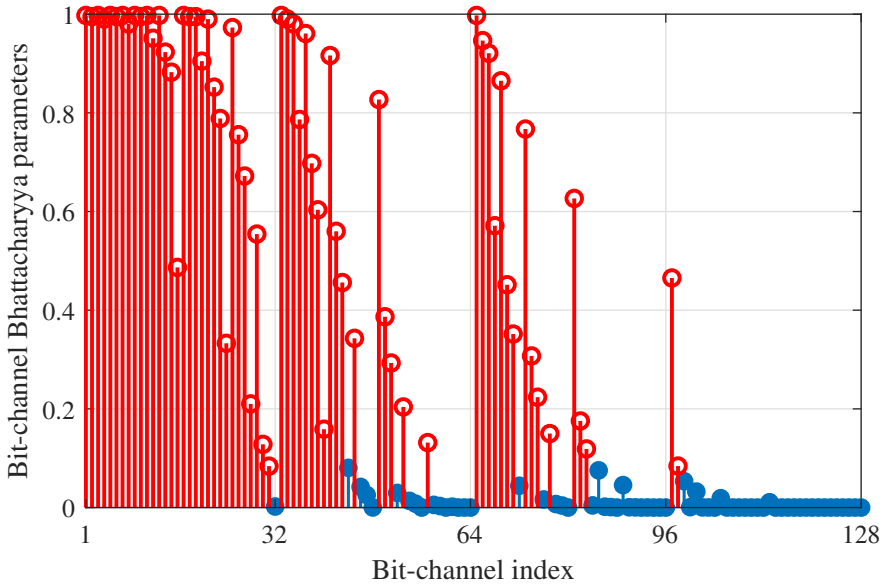

Figure 2.6: Bit-channel Bhattacharyya parameters for information bits (solid circles) and frozen bits (hollow circles) at 2.5 dB SNR.

Bit-channel cutoff rate

The Gallager's function of a channel W of probabilities $q(x)$ on the input data is described as

$$E_0(\rho, W) = -\log_2 \sum_{y \in \mathcal{Y}} \left[\sum_{x \in \mathcal{X}} q(x) W(y|x)^{\frac{1}{1+\rho}} \right]^{1+\rho}, \tag{2.24}$$

for a given BI-DMC W and $\rho \geq 0$ value [34].

We can also acquire the channel cutoff rate as

$$R_0(W, q) \triangleq E_0(1, W), \tag{2.25}$$

by simply replacing $\rho = 1$ in (2.24).

If the distribution of the inputs is uniform, the channel cutoff rate is therefor

$$E_0(1, W) = \log_2 \frac{2}{1 + Z(W)}, \tag{2.26}$$

and this is a lower bound on the value of symmetric capacity $I(W)$.

Based on the (2.26), the polarization of $Z(W)$ yields in the polarization of $E_0(1, W)$, and we denote the bit-channel cutoff rates by $E_0(1, W_N^{(i)})$. The bit-channel cutoff rates are compared to the bit-channel mutual information in Figure 2.7.

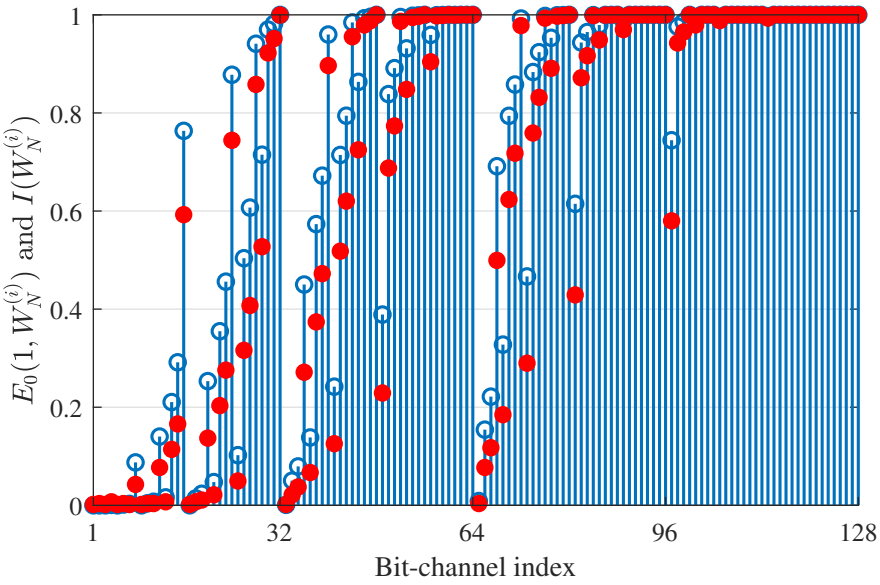

Figure 2.7: Bit-channel cutoff rates (solid circles) and bit-channel mutual information (hollow circles) at an SNR value of 2.5 dB.

RM-polar

In this subsection, we review a hybrid code construction called *RM-polar* code construction. This construction is created by merging Reed-Muller and polar code constructions. Not only does it have a better weight distribution than polar codes (construction only based on the bit-channels reliabilities), but it can also be decoded as polar codes by using sequential or SCL decoders. The RM-polar code construction outperforms polar code construction in terms of error-rate performance [33]. To construct a $(N, K, \mathcal{A})$ code s.t. $N = 2^n$:

1. Select r in a way that

$$k \triangleq \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{r} \leq K. \tag{2.27}$$

2. Select k row indices of $\mathrm{RM}(n,n) \triangleq F^{\otimes n}$ matrix corresponding to the rows with the highest weights and include them in the set $\mathcal{A}$.

3. Select the remaining $K-k$ row indices out of $N-k$ indices that are not yet in $\mathcal{A}$ with the highest reliability (equivalently, indices corresponding to the lowest Bhattacharyya parameters) and include them in the set $\mathcal{A}$.

Example 4. *To have a* $(8,6)$ *code,* $r=1$ *and* $k=\binom{3}{0}+\binom{3}{1}=4$*. In*

$$RM(3,3)=\begin{bmatrix}1&0&0&0&0&0&0&0\\1&1&0&0&0&0&0&0\\1&0&1&0&0&0&0&0\\1&1&1&1&0&0&0&0\\1&0&0&0&1&0&0&0\\1&1&0&0&1&1&0&0\\1&0&1&0&1&0&1&0\\1&1&1&1&1&1&1&1\end{bmatrix}$$

matrix, the $k=4$ *rows with indices 4, 6, 7, and 8 have the highest weights. So, in the step 2,* $\mathcal{A}=\{4,6,7,8\}$*. For the remaining row indices, the corresponding* $Z(W_N^{(i)})$ *values are as*

$$(0.9560, 0.6805, 0.5842, -, 0.4630, -, -, -).$$

The 3rd and 5th rows have the $K-k=2$ *lowest* $Z(W_N^{(i)})$ *values and as a result* $\mathcal{A}=\{3,4,5,6,7,8\}$*.*

2.3 PAC codes

Under SC decoding, polar code performance for short and medium blocklengths falls significantly short of theoretical boundaries. One issue may be related to the code's poor weight distribution, which may be addressed by including parity checks (CRC) in the data. Another issue arises from suboptimal SC decoding, which may be remedied by utilizing an SCL decoder. Since its introduction, these methods, along with other numerous improvements, have generally

remained state of the art in polar coding [9]. Polar codes' weight distribution also can be improved by utilizing a convolutional pre-transformation [23]. At low blocklengths, a novel polar coding method named polarization-adjusted convolutional (PAC) codes significantly outperforms standard polar codes by benefiting from a convolutional pre-transformation [13]. This section will go through PAC codes in detail.

Four parameters $(N, K, \mathcal{A}, T)$ can be used to specify a PAC code:

- N is the length of the blocks in PAC codes, which, like polar codes, is a power of two, i.e. $N = 2^n$, $n \geq 1$.
- K is the length of the data bits, which can be any integer from 1 to N.
- $\mathcal{A}$ is the data index set; it is a subset of $\{1, 2, \cdots, N\}$ with a size of $|\mathcal{A}| = K$.
- T is an upper-triangular Toeplitz matrix made using a connection polynomial $\mathbf{c}(t) = c_m t^m + \cdots + c_1 t + c_0$, with $c_0 = c_m = 1$ represented as

$$T = \begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_m & 0 & \cdots & 0 \\ 0 & c_0 & c_1 & c_2 & \cdots & c_m & & \vdots \\ 0 & 0 & c_0 & c_1 & \ddots & \cdots & c_m & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & 0 & c_0 & c_1 & c_2 \\ \vdots & & & & 0 & 0 & c_0 & c_1 \\ \vdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & c_0 \end{bmatrix}. \tag{2.28}$$

Figure 2.8 illustrates the PAC encoding scheme in block diagram form. Three components comprise the PAC encoding process: data insertion, convolutional encoder, and polar mapper.

- The data insertion block embeds the data vector $\mathbf{d} = (d_1, \cdots, d_K)$'s bits into a data-carrier vector $\mathbf{v} = (v_1, v_2, \cdots, c_N)$ as $\mathbf{v}_{\mathcal{A}} = \mathbf{d}$ and $\mathbf{v}_{\mathcal{A}^c} = \mathbf{0}$,

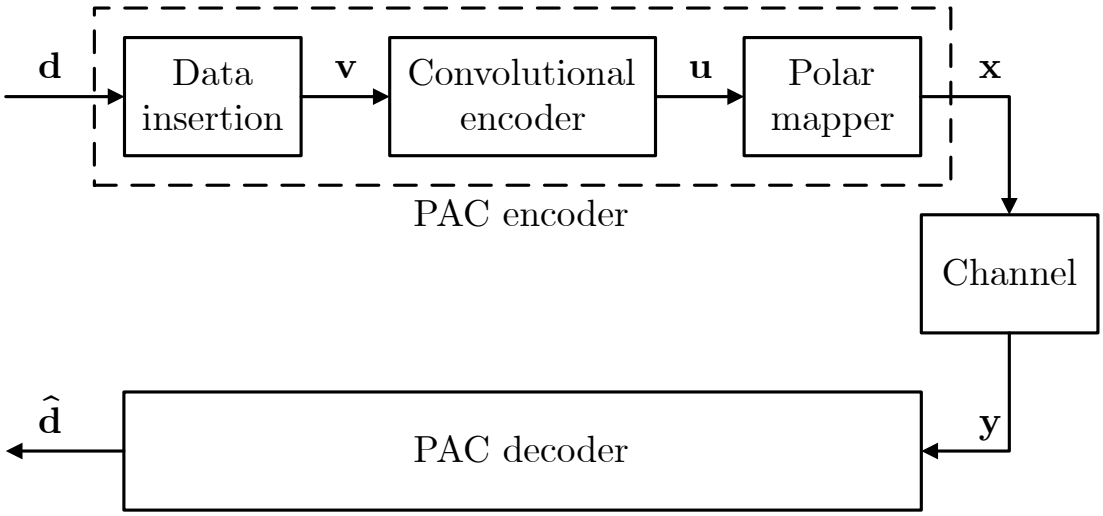

Figure 2.8: PAC coding scheme

thus resulting in a coding rate of $R = K/N$. This procedure, alongside with the selection of the index set $\mathcal{A}$, is referred to as *rate-profiling*. Polar rate-profiling and RM-polar rate-profiling are two construction methods for obtaining this set $\mathcal{A}$ (using the same set $\mathcal{A}$ as explained in the preceding subsection).

- In the convolutional encoder block, vector $\mathbf{u} = (u_1, u_2 \cdots, u_N)$ can be obtained as $\mathbf{u} = \mathbf{v}T$ for an upper-triangular Toeplitz matrix T. This generates the vector $\mathbf{u}$, each bit of which is a linear combination of at most $(m+1)$ bits of the vector $\mathbf{v}$ calculated by the convolution operation.

- Finally, the vector $\mathbf{u}$ is encoded using the standard polar transformation $F^{\otimes n}$.

For a specified matrix T and the index set $\mathcal{A}$, we use PAC(N, K) notation to show a PAC code with $(N, K, \mathcal{A}, T)$ parameters.

Example 5. *As a small instance, consider a PAC*$(8, 4)$ *code using polar code construction as a rate-profiling and the matrix* T *made using the connection polynomial* $\mathbf{c}(t) = t^2 + t + 1$. *The rate-profiling inserts data vector* $\mathbf{d} = (d_1, d_2, d_3, d_4)$ *into vector* $\mathbf{v} = (v_1, v_2, \cdots, v_8)$ *s.t.* $\mathbf{v} = (0, 0, 0, d_1, 0, d_2, d_3, d_4)$. *The convolutional encoder block generates vector* $\mathbf{u} = (0, 0, 0, d_1, d_1, d_1 + d_2, d_2 + d_3, d_2 + d_3, d_4)$ *from*

the vector $\mathbf{v}$ *as* $\mathbf{u} = \mathbf{v}T$*, i.e.* $u_i = v_i + v_{i-1} + v_{i-2}$ *for* $i = 3, \cdots, 8$*,* $u_2 = v_2 + v_1$*, and* $u_1 = v_1$*. As will be explained in the following sections, convolutional encoder can be implemented in a linear order complexity. Finally, the polar transform* $\mathbf{x} = \mathbf{u}F^{\otimes n}$ *is computed to finish the encoding process.*

2.3.1 List Decoding

List Decoding of Polar Codes

Even though polar codes achieve capacity asymptotically, empirical results indicate that SC decoding of polar codes performs worse than the well-known turbo and LDPC codes for short to moderate block lengths [9]. When we think about why this is happening, we can come up with two probable explanations: either the codes themselves are poor at these lengths, or there is a considerable performance difference between SC and ML decoding. This section discusses a modification to the SC decoder, specifically, an SCL decoder [9]. The SCL decoder is controlled by a single integer parameter L, which represents the size of the list. Like an SC decoder, the SCL decoder also decodes the input bits one by one. On the other hand, the SCL decoder considers L paths at the same time at each decoding step. In particular, the SCL decoder duplicates the number of potential decoding paths for each information bit u_i by two (pursuing both $u_i = 0$ and $u_i = 1$). It then applies a pruning procedure to exclude all but the L most likely paths. Finally, the most probable of the L decoding paths is chosen as the decoder output at the completion of the decoding procedure. For several list sizes of SCL decoding, Figure 2.9 displays the performance of a $(128, 64)$ polar code with RM code construction. As seen in this figure, increasing the list size improves the frame error rate (FER) performance of polar code.

There is a distinction between SC and SCL decoding, where the SCL decoder requires the selection of the L paths from a group of $2L$ children (only data bits are bifurcated). What is required is a path metric in the decoding process. The SCL decoder then selects the L paths that have the lowest partial path metric

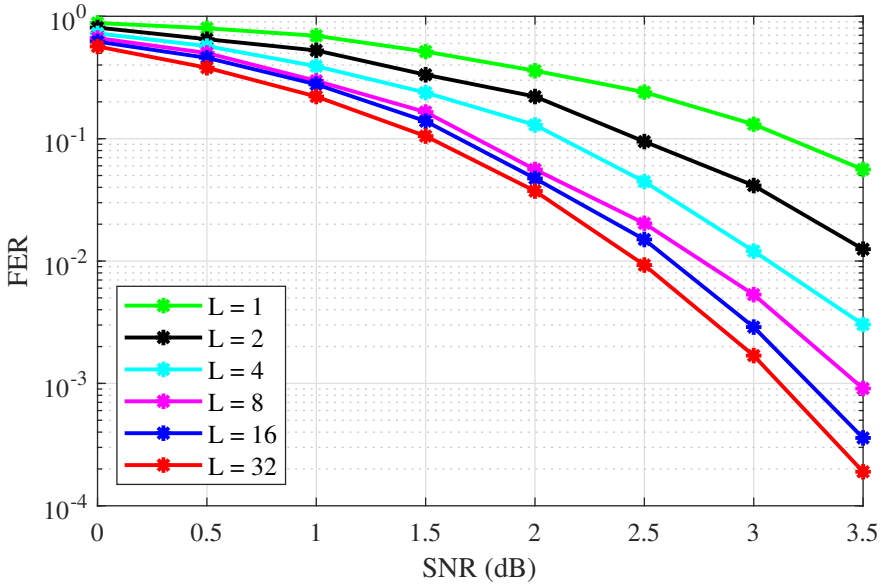

Figure 2.9: Performance of polar codes under list decoding.

values [26]

$$\mathrm{PM}^{(i)} = \sum_{j=1}^{i} \log_2 \left(1 + 2^{-L_j \cdot (-1)^{u_j}}\right). \tag{2.29}$$

CRC-aided SCL decoding

Furthermore, the performance of polar codes is worse than that of LDPC and turbo codes of equivalent length, even when ML decoding is used for decoding. In simulations, it is noticed that the sent codeword is on the created list with a high possibility, but it is not the most probable codeword on the list [9]. As a result, it is not chosen as the decoder output. In this scenario, precoding the transmitted codeword using a cyclic redundancy check (CRC) may assist in identifying it. Polar codes' performance under list decoding with CRC has been shown to be superior to that of the state-of-the-art turbo and LDPC codes in simulations. When considering a target FER of 10^{-4}, experimental findings reveal that polar coding with CRC-aided list decoding performs better than any other code presently known for lengths 512 and 1024 [9].

List Decoding of PAC Codes

The idea of substituting CRC with CC has given rise to the development of PAC codes. In this part, we demonstrate PAC codes' performance by using a list decoding algorithm [27]. The findings indicate that PAC codes outperform polar codes. One benefit of the list decoding algorithm is that its complexity is constant.

Similar to the SCL decoding of polar codes, the metric function for the SCL decoding of PAC codes can be represented as

$$\mathrm{PM}^{(i)} = \log_2\left(1 + 2^{-L_i \cdot (-1)^{u_i}}\right), \tag{2.30}$$

where

$$L_i = \log_2\left(\frac{P(\mathbf{y}, \mathbf{u}^{i-1}|u_i = 0)}{P(\mathbf{y}, \mathbf{u}^{i-1}|u_i = 1)}\right) \tag{2.31}$$

is the bit-channels likelihoods, and as depicted in Figure 2.8, vector $\mathbf{u}$ is the input of the polar mapper. One distinction from polar codes is that for each path of the list, L vectors with a size equal to the constraint length of the CC must be created as the content of the memory registers of L auxiliary shift registers and used to compute u_i. To acquire L u_is for (2.30), a 0 should be placed into each shift register if it corresponds to a frozen bit or zero branch; otherwise, a 1 should be loaded into each shift register. Figure 2.10 illustrates the FER performance of PAC$(128, 64)$ code constructed using RM rate profile.

2.4 Sequential Decoding of Convolutional Codes

This section discusses convolutional codes and their decoding through stack and Fano sequential decoding.

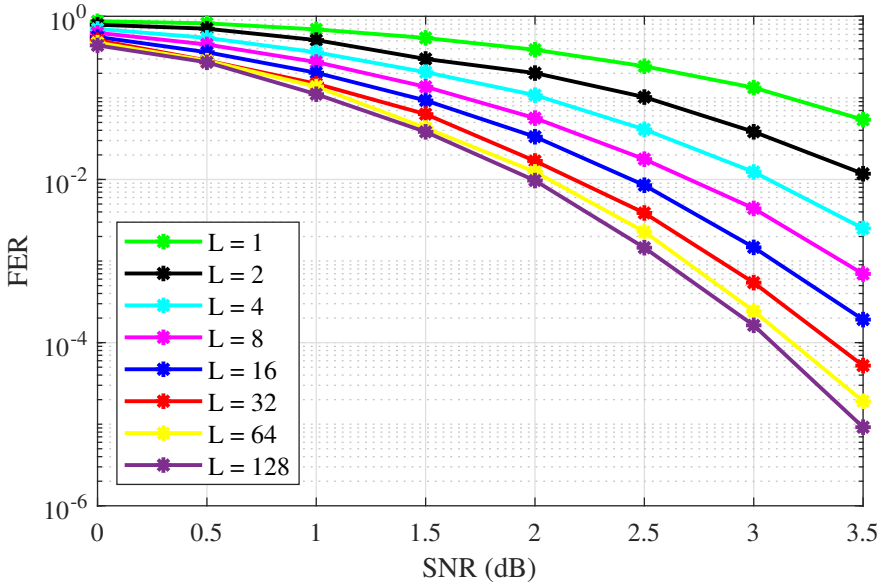

Figure 2.10: Performance of PAC codes under list decoding.

2.4.1 Convolutional Codes

A binary convolutional code (CC) is represented by the three-tuple (n, k, m), corresponding to the encoder that produces n bits for every k received input bits. Each current output vector of length n is a linear combination of the current input vector of length k and the preceding $m \times k$ input bits. As m is the amount of prior k-bit input blocks that the encoder have remember, m is referred to as the CC's memory order. A binary convolutional encoder can be easily constructed as a combination of shift registers and modulo-2 adders. The output bits are modular-two additions of the contents of selected shift registers and the current input bits. The encoder comprises k shift registers, with m being their maximum length.

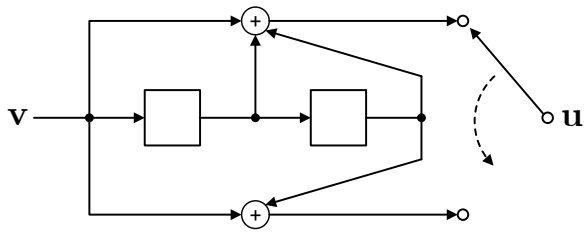

Figure 2.11: Convolutional encoder example.

Example 6. *The encoder of a binary* $(2, 1, 2)$ *CC is shown in Figure 2.11. The encoder's shift registers are initialized to zero in the encoding operation. In accordance with the shift-register structure, the* $k = 1$ *input bit from the input data is supplied to the encoder, resulting in* $n = 2$ *outputs being generated.* $m = 2$ *zeros are typically appended at the tail of every input data to set up the shift register entire content at the end of each run of input blocks.*

As shown in Figure 2.11, from a single input sequence $\mathbf{v} = (v_1, v_2, v_3, v_4, v_5) =$ (10111) *the CC's encoder generates two output sequences* (1100101) *and* (1001011)*, where* v_1 *is supplied into the encoder first. After that, the encoder interleaves two output sequences to produce*

$$\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_7) = (11\ 10\ 00\ 01\ 10\ 01\ 11). \tag{2.32}$$

Figure 2.12 shows a basic model of a communication system with CC and the Viterbi decoding. Viterbi algorithm [35, 36] is an ML decoder for CCs.

Figure 2.12: Flowchart of the convolutional coding with Viterbi decoding.

Assume that the information sequence $\mathbf{v} = (v_1, v_2, \cdots, v_K)$ of length K is encoded into code sequence $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{K+m})$ of length $N = (K + m) \times n$. Let the received sequence be $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{K+m})$, where the lth received block is $\mathbf{y}_l = (y_l^{(1)}, y_l^{(2)}, \cdots, y_l^{(n)})$.

An ML decoder determines the optimal path across the trellis based on the path conditional probability

$$P(\mathbf{y}|\mathbf{u}) = \prod_{l=1}^{K+m} P(\mathbf{y}_l|\mathbf{u}_l), \tag{2.33}$$

where

$$P(\mathbf{y}_l|\mathbf{u}_l) = \prod_{i=1}^{n} P(y_l^{(i)}|u_l^{(i)}) \tag{2.34}$$

is the branch-conditional probability. The bit-conditional probabilities represented by $P(y_l^{(i)}|u_l^{(i)})$ are the channel transition probabilities.

Since logarithm is an increasing function and preserves maximality, maximizing the conditional probability $P(\mathbf{y}|\mathbf{u})$ is equivalent to maximizing path metric

$$\Gamma(\mathbf{u};\mathbf{y}) \triangleq \log_2 P(\mathbf{y}|\mathbf{u}). \tag{2.35}$$

The path metric can be expressed as a summation of branch metrics:

$$\Gamma(\mathbf{u};\mathbf{y}) = \sum_{l=0}^{K+m} \log_2 P(\mathbf{y}_l|\mathbf{u}_l) = \sum_{l=0}^{K+m} \Gamma(\mathbf{u}_l;\mathbf{y}_l) \quad \text{(branch metric)}, \tag{2.36}$$

and sum of bit metrics can be used to calculate the branch metric:

$$\Gamma(\mathbf{u}_l;\mathbf{y}_l) = \sum_{i=1}^{n} \log_2 P(y_l^{(i)}|u_l^{(i)}) =: \sum_{i=1}^{n} \gamma(u_l^{(i)};y_l^{(i)}) \quad \text{(bit metric)}. \tag{2.37}$$

The chance of an error reduces exponentially as the length of the code constraint increases, but the computational complexity of Viterbi decoding increases exponentially [35,37]. As a result, the chance of an error reduces only algebraically as computational complexity increases. This scenario might be improved if we use a method to forgo calculating the metric of each path in the trellis and instead focus on those with higher metrics, which are more likely to contain the correct path.

On the other hand, a sequential decoder uses

$$\gamma(u_i;y_i) = \log_2 \frac{P(y_i|u_i)}{P(y_i)} - R \tag{2.38}$$

as the bit metric function, where $P(y_i)$ is the channel output probability and R is the code rate [38]. This metric function has a bias that is essential for improving the performance of any algorithm that does not examine every potential path but must choose amongst paths of different lengths. Sequential decoding algorithms, such as stack or Fano, compute the metric of paths by extending a previously investigated path by just one branch, and the choice on which path to extend is based only on the metrics of previously investigated paths. We explain both stack and Fano algorithms in depth in the following subsections.

2.4.2 Stack Algorithm

The fundamental premise of sequential decoding is that we should consider just the most promising pathways throughout the decoding process. If a path to a node seems poor, we may reject all pathways originating from that node without incurring any significant performance loss relative to an ML decode. The path metric function guides a decoder to search for the most probable path to investigate. This naturally leads us to the stack algorithm [18,19]. The stack algorithm is perhaps the most fundamental sequential decoding algorithm and definitely the easiest to explain. Due to the stack usage in the search for the optimum codeword, the algorithm is referred to as the stack algorithm. The following steps describe the stack algorithm for a rate b/n CC.

1. Calculate the path metric associated with the origin node. Stack its associated path metric value (initialize with 0 metric).
2. Calculate the path metric values of all 2^b immediate successor paths to the top path in the stack. Delete the stack's top path.
3. After adding the remaining successor paths to the stack, reorder the stack in ascending order of partial path metric values.
4. The algorithm terminates if the top path in the stack terminates at a leaf node in the decoding tree; otherwise, the algorithm proceeds to Step 2.

The algorithm generates a stack of previously explored paths of varying lengths that are ordered descendingly by their metric values. At each step, the top-of-the-stack's path is replaced by its 2^b successors that have been extended by one branch, and the partial path metric is augmented with the corresponding branch metrics. The algorithm continues in this manner until the decoding tree reaches its end. For a message sequence of length K and CC with memory m, in step 2 of the algorithm, it is just necessary to branch the tree for the first K bits. For the last m bits, it is necessary to compute only one new metric at step 2. As a result, the tree code has only 2^{bK} branches for a b/n CC. Similarly, the stack size

increases by $b-1$ for the first K bits but remains unchanged for the last m zero bits.

Example 7. *Assume that the CC has a coding rate of* 0.5 *and the BSC has a crossover probability of* $p = 0.045$. *We have* $p(y_i) = 1/2$ *and*

$$\gamma(u_i; y_i) = \begin{cases} \log_2(1-p) + 1 - R = 0.4486, & \text{for } y_i = u_i, \\ \log_2(p) + 1 - R = -4.3365, & \text{for } y_i \neq u_i, \end{cases} \tag{2.39}$$

As a consequence, the bit metric has just two potential values: 0.4336 *and* -3.9739. *To keep things simple, we scale and round the bit metrics to get*

$$\gamma(u_i; y_i) = \begin{cases} +0.5, & \text{for } y_i = u_i, \\ -4.5, & \text{for } y_i \neq u_i, \end{cases} \tag{2.40}$$

Assume that the receiving vector is $\mathbf{r} = (11\ 11\ 00\ 01\ 11\ 01\ 11)$.

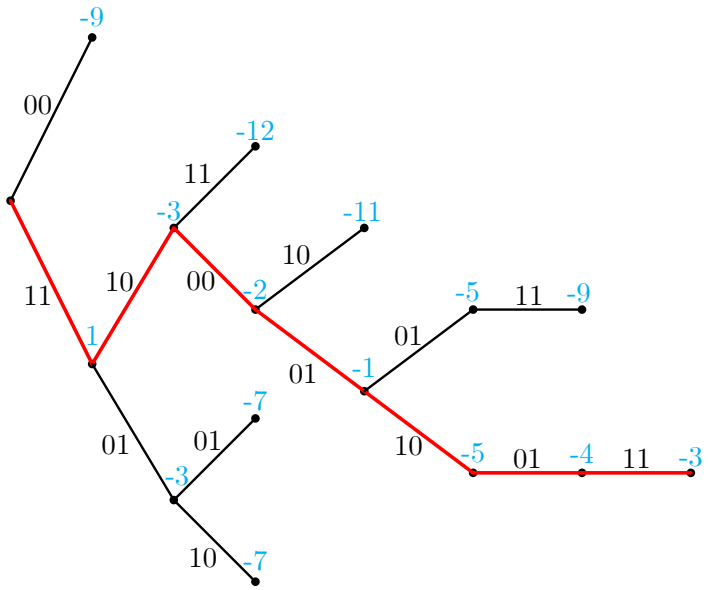

Figure 2.13: Partially explored binary tree by stack decoding algorithm.

In Figure 2.13, the partly investigated tree is shown with the partial path metrics. Each branch of the tree is labeled with the output bits that correspond to it. At time 0, the decoder begins at the root and traverses the tree from left to right. Whenever the corresponding information bit is 0, it selects the upper branch; when it is 1, it chooses the lower branch. Also, the contents of the stack are shown in

Table 2.1 after each partial path metric reordering. Both the input bits and the metrics associated with each path are stored in the stack. The algorithm is ended after the ninth cycle. Note that the stack size is not increased at steps $7, 8,$ *and* 9 *since these steps correspond to the* 0 *terminating bits, and the code tree does not branch for them. The algorithm's decision is the path* 1011100 *or the information sequence* 10111*. This is indicated by a bold red path in the code tree. Notice that the stack decoding algorithm's significant challenges include needing a large stack size in low SNR regimes and sorting at each cycle.*

Table 2.1: The stack content and its partial path metric values after the ordering in each decoding iteration.

Step	0	1	2	3
Stack Content	- (0)	1 $(0.5+0.5=1)$ 0 $(-4.5-4.5=-9)$	10 $(1+0.5-4.5=-3)$ 11 $(1-4.5+0.5=-3)$ 0 (-9)	101 $(-3+0.5+0.5=-2)$ 11 (-3) 0 (-9) 100 $(-3-4.5-4.5=-12)$

Step	4	5
Stack Content	1011 $(-2+0.5+0.5=-1)$ 11 (-3) 0 (-9) 1010 $(-2-4.5-4.5=-11)$ 101 (-12)	11 (-3) 10110 $(-1-4.5+0.5=-5)$ 10111 $(-1+0.5-4.5=-5)$ 0 (-9) 1010 (-11) 101 (-12)

Step	6	7
Stack Content	10110 (-5) 10111 (-5) 110 $(-3=4.5+0.5=-7)$ 111 $(-3+0.5-4.5=-7)$ 0 (-9) 1010 (-11) 101 (-12)	10111 (-5) 110 (-7) 111 (-7) 101100 $(-5-4.5+0.5=-9)$ 0 (-9) 1010 (-11) 101 (-12)

When the channel noise is low, the sequential decoding algorithm has an advantage over the Viterbi algorithm with constant computational complexity. Sequential decoding usually requires more computational complexity when the received vector $\mathbf{r}$ is very noisy. For the input sequence $\mathbf{u} = (10111)$, over the BSC, the received sequence in the previous example has 2 bits in error, and the stack algorithm requires nine steps to decode it. For the same input, in the following example, 3 number of bits are in error, and the stack algorithm requires 16 steps to decode this. Notice that the proportion of errors in the receiving sequence $\mathbf{r}$ is $\frac{3}{14} = 0.2143$ which is much higher than the channel crossover probability of $p = 0.045$. As a result, excessively noisy received sequences need a large amount

Step	8	9
Stack Content	101110 ($-5+0.5+0.5=-4$) 110 (-7) 111 (-7) 101100 (-9) 101101 (-9) 0 (-9) 1010 (-11) 101 (-12)	1011100 ($-4+0.5+0.5=-3$) 110 (-7) 111 (-7) 101100 (-9) 101101 (-9) 0 (-9) 1010 (-11) 101 (-12

of computation using a sequential decoder (this is recognized as the Pareto distribution, and it will be described in further detail in the following chapter). Also, since very noisy received sequences are uncommon, the average number of computations executed by a sequential decoder is often substantially lower than the fixed amount performed by the Viterbi decoding.

Example 8. *Consider having the same code, channel crossover probability, and metric function as in the preceding example, with* $\mathbf{r} = (10\ 11\ 00\ 01\ 11\ 01\ 11)$ *as the receiving vector. The stack's contents after every algorithm iterations are presented in Table 2.2. After* 16 *decoding cycles, the algorithm terminates, and the decoded information sequence is* $\mathbf{v} = (10111)$.

Table 2.2: The stack content and its partial path metric values after the ordering in each decoding iteration.

Step	0	1	2	3	4	5	6	7
Stack Content	-, (0)	0 (-4) 1 (-4)	01 (-3) 1 (-4) 00 (-13)	1 (-4) 010 (-7) 011 (-7) 00 (-13)	010 (-7) 011 (-7) 10 (-8) 11 (-8) 00 (-13)	011 (-7) 10 (-8) 11 (-8) 0100 (-11) 0101 (-11) 00 (-13)	0110 (-6) 10 (-8) 11 (-8) 0100 (-11) 0101 (-11) 00 (-13) 0111 (-16)	01100 (-5) 10 (-8) 11 (-8) 0100 (-11) 0101 (-11) 00 (-13) 01101 (-15) 0111 (-16)

Step	8	9	10	11	12
Stack Content	10 (-8) 11 (-8) 011000 (-9) 0100 (-11) 0101 (-11) 00 (-13) 01101 (-15) 0111 (-16)	101 (-7) 11 (-8) 011000 (-9) 0100 (-11) 0101 (-11) 00 (-13) 01101 (-15) 0111 (-16) 100 (-17)	1011 (-6) 11 (-8) 011000 (-9) 0100 (-11) 0101 (-11) 00 (-13) 01101 (-15) 1010 (-16) 0111 (-16) 100 (-17)	11 (-8) 011000 (-9) 10110 (-10) 10111 (-10) 0100 (-11) 0101 (-11) 00 (-13) 01101 (-15) 1010 (-16) 0111 (-16) 100 (-17)	011000 (-9) 10110 (-10) 10111 (-10) 0100 (-11) 0101 (-11) 110 (-12) 111 (-12) 00 (-13) 01101 (-15) 1010 (-16) 0111 (-16) 100 (-17)

Stack algorithm can also operate on a trellis; in this instance, the algorithm checks whether any successor paths merge with an existing path in the stack.

Step	13	14	15	16
Stack Content	10110 (-10)	10111 (-10)	101110 (-9)	1011100 (-8)
	10111 (-10)	0100 (-11)	0100 (-11)	0100 (-11)
	0100 (-11)	0101 (-11)	0101 (-11)	0101 (-11)
	0101 (-11)	110 (-12)	110 (-12)	110 (-12)
	110 (-12)	111 (-12)	111 (-12)	111 (-12)
	111 (-12)	00 (-13)	00 (-13)	00 (-13)
	00 (-13)	101100 (-14)	101100 (-14)	101100 (-14)
	01101 (-15)	01101 (-15)	01101 (-15)	01101 (-15)
	1010 (-16)	1010 (-16)	1010 (-16)	1010 (-16)
	0111 (-16)	0111 (-16)	0111 (-16)	0111 (-16)
	100 (-17)	100 (-17)	100 (-17)	100 (-17)
	010000 (-18)	010000 (-18)	010000 (-18)	010000 (-18)

Suppose it does, and the successor path's partial path metric value is greater than the partial path metric value of the existing path in the stack that traverses different nodes but terminates at the merged node. In this case, the algorithm modifies the existing path in the stack by substituting the successor path for its sub-path and updates the newly modified path's partial path metric value. Then, the algorithm removes the successor path that overlaps with parts of the stack's paths.

Despite its simplicity, there are significant implementation issues with the stack sequential decoding technique that restrict its broad adoption. The first issue is that the amount of computation is a random variable and is similar to that of other sequential decoding algorithms; we will examine it in detail for the Fano algorithm. Since the decoder traverses the decoding tree somewhat randomly, leaping from one node to another node, the decoder should have a buffer to hold incoming received sequences when they wait to be decoded. Long searches would cause the buffer to overflow, resulting in data loss or *erasure* depending on the speed factor of the decoder (the ratio of the speed at which calculations are conducted to the speed of the incoming data) is processed. One way to address this issue is to restrict the amount of searching that the decoder can perform, and we will study this for the Fano algorithm. In addition, certain systems can tolerate a certain level of data erasure. Since erasure occurs most often when the received sequence is extremely noisy, even if decoding is finished and the ML path is acquired, there is a rather substantial possibility that the estimate will be wrong. In many applications, erasing such a frame is preferable to erroneously decoding it. A decoder such as the Viterbi algorithm will always decode a frame even if it is likely to be decoded wrongly, but a sequential decoder swaps errors

for erasures by detecting noisy frames and then erasing them. This sequential decoding capability can be used to decide when an automatic repeat request (ARQ) transmission system should retransmit a frame.

A second issue is that any practical implementation of the stack decoding must have a limited stack size. This means that there is always a chance that the stack may become too large before decoding a particular frame is finished. The next chapter will address this issue when we present a sequential decoding algorithm based on the heap data structure. One approach is to simply enable the path with the smallest path metric to be taken from the heap during the subsequent decoding stage. This path is then permanently gone and cannot be rejoined to the heap. Results show that with a sufficiently large heap size, the chance that a path with the smallest path metric value would ever recover and reach the root of the heap and be expanded is so low that the performance loss is negligible.

The third issue is with the stack reordering following each decoding step. This may be rather time-intensive as the stack's size increases, severely limiting the decoding speed achievable using this simple algorithm. When a heap data structure, a maximally efficient priority queue, is used, the path with the highest path metric is extracted in logarithmic time. Furthermore, we attempt to address this problem in Section 5.3 for the stack decoding of polarized channels and the study reveals that for the high SNR values, average stack size is 1.

2.4.3 Fano Algorithm

Because the Fano algorithm is unable to analyze path merging, it can only work on a code tree. The Fano algorithm is an incredibly intelligent sequential decoding technique and is widely regarded as the most practical sequential decoding algorithm available [17]. It employs a sequence of thresholds separated by Δ increments. Its most appealing aspect is that it analyzes a single path at a time, obviating the need to store everything except one path and its metric. Essentially, the algorithm will continue to explore farther along a particular path so long as

the metric value of the path increases. When the metric starts to reduce considerably, it returns to earlier nodes on the previously traveled paths and searches for alternate pathways that originate from them. This is done by varying the size of a comparison threshold, T, in Δ-step amounts. When the metric grows enough in a forward-searching, the threshold is tightened (increased by Δ) and relaxed (reduced by Δ) throughout backward searching. This is performed in such a manner that no node would ever be searched forward more than once with the equal threshold setting; the threshold must always be smaller than the value used before.

The Fano algorithm's exact details can be understood by studying the flowchart in Figure 2.14 (modified from Figure 6.6 in [37]). Initially, the decoder begins at the origin node with the threshold T and the metric value Γ both equal to zero. Looking ahead on the better node is defined in the first block as calculating both bit metrics and provisionally supplementing the current node partial path metric by the larger of these two calculated metrics. If the superior node was already searched and the running threshold, T, has been violated, the forward look should be directed to the inferior node (this happens if the initial block is reached from point **A**). In either instance, the node's metric is compared to T, and if it is more than T (the metric is satisfied), the searching pointer is advanced to that node. The tests that follow determine if this node is reached for its first time during the sequential decoding process. This is also where we must check for the tree tail's end and stop if it is reached. If that is true (first visit), the metric of the preceding node violates $T + \Delta$ [34]. In this case, we may try to tighten the threshold by raising the running threshold T by the integer multiples of Δ until $\Gamma < T + \Delta$, and then looking forward once again. Suppose the node has even been explored previously. In that case, it is critical that the algorithm does not tighten the threshold before exploring again since this would cause the system to create a closed-loop and repeatedly do the same movements.

If the new node has metric $\Gamma < T$ obtained via forward search in the initial block, we must switch to backward search mode. Taking the previous bit metric and subtracting it from the current node metric is also accomplished in this step. After determining if this meets T's conditions, the pointer is pushed back one

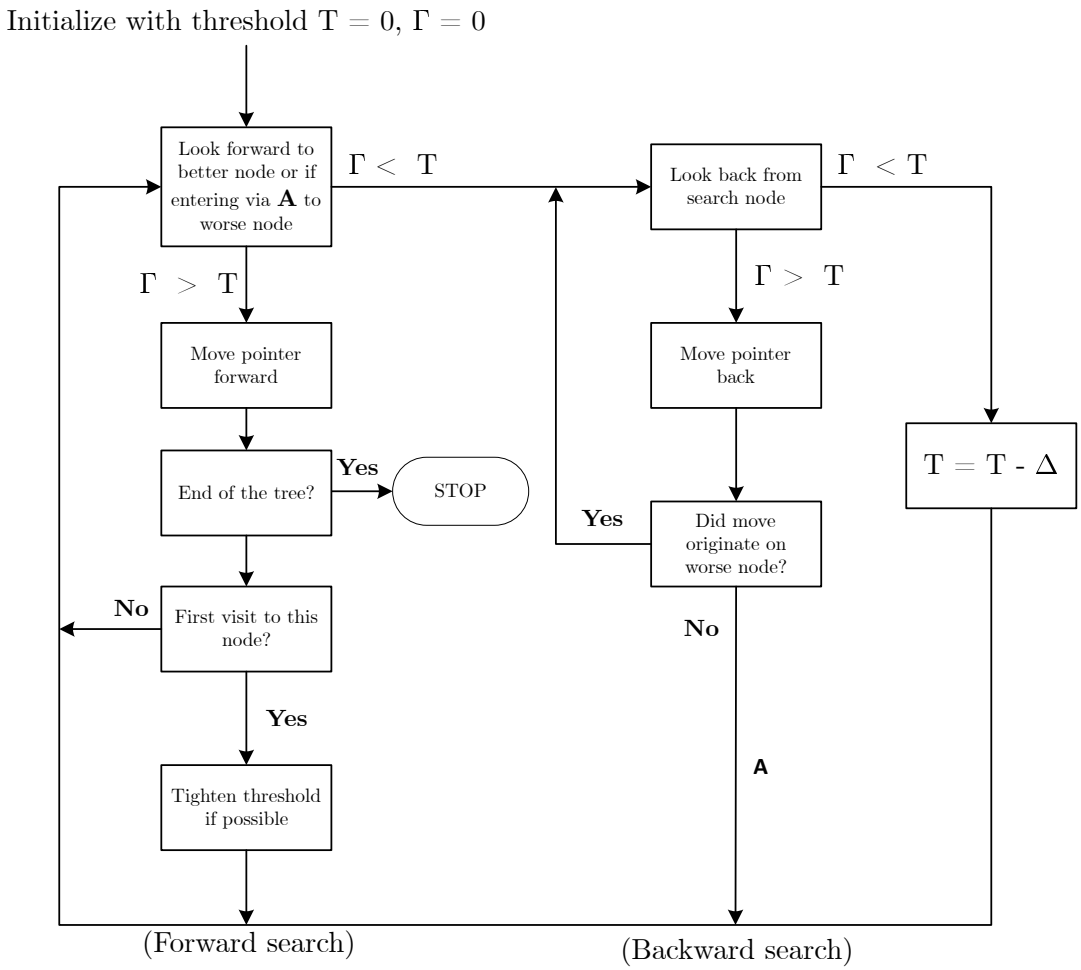

Figure 2.14: Flowchart of the Fano sequential decoding algorithm for a binary tree.

level; if the branch on which the backward moving was performed had the better metric of two branches originating from the node that just reached, the inferior node still needs to be explored. As a result, we are returned to the forward exploration through **A**. If this was from the worse branch, there would be no further forward branches from this node, and hence the algorithm should keep continuing the backward search. We cannot go back if the current threshold T is violated during a backward movement. When this happens, all paths reachable from this node with the given T in operation have been examined and determined to violate T. Thus, the threshold will be lowered by one Δ ($T = T - \Delta$), and forward exploration is again tried. It should be noted that in the event of a lookback from the origin node, we suppose that the previous node of origin has a metric value of $-\infty$, and the threshold is always dropped by Δ. It is worth noting that if a node is checked multiple times, every time it is examined with a reduced current threshold, avoiding endless cycles.

In [38], an example of the Fano algorithm with a detailed description of its steps can be found.

2.5 Sequential Decoding of PAC Codes

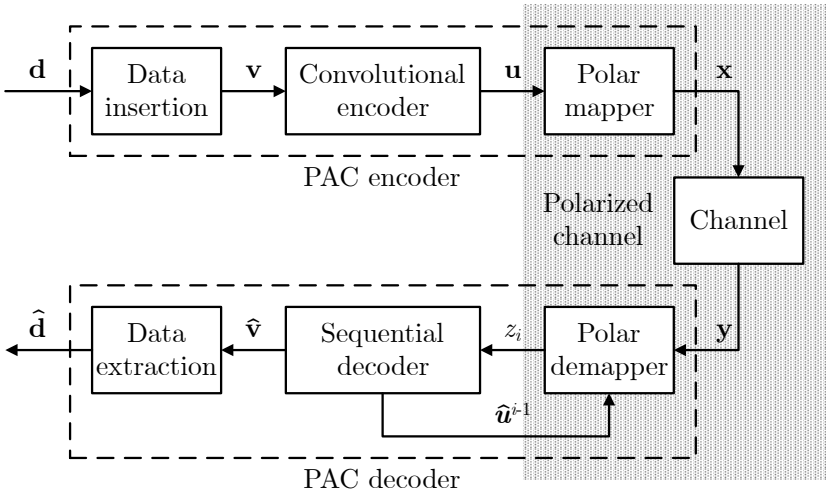

Figure 2.15: Flowchart of the sequential decoding algorithm for a binary tree.

We will investigate the sequential decoding of PAC codes in this section, and the specifics may be understood by viewing the flowchart in Figure 2.15. As this figure shows, a PAC decoder that employs a sequential algorithm is composed of polar demapper and sequential decoder blocks. We will describe how to adopt a sequential decoding algorithm to decode a PAC code.

Consider that the sequential decoder is proceeding to the ith node in a forward direction (the stack algorithm can only have a forward move, while the Fano algorithm can also have a backward move). The polar demapper, similar to the SC decoder, receives channel output $\mathbf{y}$ and uses the hard decisions $\hat{\mathbf{u}}^{i-1}$ vector provided by sequential decoder to calculate

$$z_i \triangleq \log_2 \left(\frac{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 0)}{P(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 1)} \right), \tag{2.41}$$

as a soft output. It is important to note that the polar demapper does not approximate u_i but rather gives the soft z_i values to the sequential decoder. Sequential decoding yields $\hat{v}_i$ by using z_i in the metric function described in the next chapter. Besides this, the sequential decoder retrieves $\hat{u}_i$ from $\hat{v}_i$ and returns it to the polar demapper through an encoder replica. The polar demapper then derives z_{i+1} employing $\hat{\mathbf{u}}^i$, and the decoding procedure proceeds in this manner until $\hat{v}_N$ is found or a predetermined stopping rule terminates the decoding process.

As previously indicated, in a Fano algorithm, if the tentative path metrics of both children are less than the running threshold T and the path metric of the preceding node is greater than T, the Fano decoder needs to move backward. Assume now that the Fano decoder is located at the ith node, moves back to the $(i-1)$th node, and feeds $\hat{u}^{i-1}$ to the polar demapper through feedback. It is necessary to save all intermediate LLR values to avoid the polar demapper from restarting the demapping procedure from scratch when determining z_{i-1}. Thereby, the polar demapper would attempt its working from the common ancestor of the $(i-1)$th and ith nodes in pursuance of deriving z_{i-1}. Similarly, to move backward from the ith node to the jth node when $j < i$, the polar demapper just has to start from the common ancestor of the jth and ith leaf nodes of the polar demapper tree [14]. As a result, the polar demapper used in this research saves all intermediate LLRs and has a memory of $N \log_2 N$. A trade-off

exists between delay and the memory consumption of polar demapper. When storing $N-1$ intermediate LLR values, due to the backtracking nature of the Fano algorithm, there is a significant amount of latency increment [14].

Unless otherwise stated, we use a memory of $N \log_2 N$ size to save all the intermediate LLRs for Fano decoding of PAC codes in all the computational complexity plots in this dissertation. The SCL decoding and other sequential decoding algorithms use a memory of $N-1$ to store the intermediate LLRs.

A sequential decoder (e.g., Fano or stack algorithms) searches the code tree (tree corresponding to the PAC code) for the correct path. The complexity of this search is a random variable that varies mainly based on the amount of noise. The following chapters will quantify the complexity of sequential decoding using a random variable Θ that counts the number of nodes in the decoding tree that the sequential algorithm visits during a decoding process. We are particularly concerned with the expectation of this random variable per bit (i.e., $E(\Theta)/N$), which we refer to as the average number of visits (ANV). We evaluate this expectation by calculating the empirical mean over a sufficiently large number of simulation runs (and to obtain the value of ANV, we divide it by the blocklength N). The stack algorithm can visit each node of the code tree just once; however, since the Fano decoding algorithm has a backtracking feature, it may visit certain nodes many times, which Θ counts each of these visits. It should be noted that since both algorithms essentially pick the same paths through the PAC decoding tree, the set of nodes traversed by the Fano and stack algorithms are identical [21].

Chapter 3

On the Metric Function and Complexity of Sequential Decoding

This chapter addresses a metric function for sequential decoding of PAC codes [14]. Additionally, we examine the relationship between the metric function in an ensemble of PAC codes and the bit-channel capacities. Our empirical findings indicate that when sequential decoding is used, the computational complexity of decoding PAC codes is comparable to that of decoding CCs, with a significant improvement in error-correction performance.

As with sequential decoding of CCs, our findings demonstrate that the computational complexity of PAC codes follows the Pareto distribution and that for high SNR values, the average number of computations per decoded codeword converges to one. We limit the number of searches conducted by the sequential decoder in order to lower the worst-case latency of the PAC sequential decoder. The results demonstrate that, for PAC codes with a length of 128, search-limited sequential decoding can obtain an error-correction performance comparable to that of polar codes with SCL decoding with a list size of 64 and a CRC length of 11 while requiring significantly less computational complexity [28].

Additionally, this chapter adopts the heap data structure for sequential decoding of PAC codes and discusses the memory and computational complexity requirements. Given that heap data structures perform operations in logarithmic order and there is no backtracking, we provide memory-efficient algorithms for storing intermediate LLR and check-sum values.

3.1 Metric Function

Since each output bit of the CC in a PAC code sees a synthesized channel created by the polar transform, using a different metric function for sequential decoding of PAC codes is required. In [28], fixed bias values for different code rates were used in the metric function. In [39], because the CC in a PAC code is a one-to-one transform, a fixed bias value equal to the CC rate was used, and an heuristic metric function was introduced. Similar to the heuristic path metric function of [39], an heuristic path metric function for decoding polar codes was introduced in [40] with a different initial value. In [41], a metric function for stack decoding of polar codes was introduced that only updates the path metric function if the extended branch is an information bit; this update is obtained by adding the logarithm of the probability of the bit-channel output for a given bit-channel input. In [42], a path metric function for sequential decoding of polar codes was introduced, which maximizes the most probable continuation of a partially explored path. For this metric, the expected value of the logarithm of the probability of the partially decoded correct path (obtained based on the cumulative function of the evolving log-likelihood ratios (LLRs)) is subtracted from the metric function, which makes the expectation of the metric (expectation over the partially explored correct path) equal to zero.

In PAC codes, because of the channel polarization effect, the bit-channel cut-off rates $E_0(1, W_N^{(i)})$ are boosted close to the bit-channel capacities $I(W_N^{(i)})$ as shown in Figure 2.7. We propose a suboptimal metric function, which, by using $E_0(1, W_N^{(i)})$ or $I(W_N^{(i)})$ as bias values, maintains the superior error-correction

performance of PAC codes while requiring lower computational complexity compared to the fixed bias values used in [13, 28, 39]. Our proposed metric function is based on the known metric function for sequential decoding of CCs; however it is adopted for polarized channels [17, 34]. Additionally, we adopt the behavior of the metric function in the ensemble of codes for polarized channels [34, 37]. Using our proposed metric function, the computational complexity of PAC sequential decoding is comparable to the computational complexity of sequential decoding of CCs.

Using metric bias values less than or equal to the bit-channel cutoff rates, we then empirically obtain an upper bound with Pareto distribution for the computational complexity of the PAC sequential decoder. Since the computational complexity has a Pareto distribution, the PAC sequential decoder requires a high computational complexity to decode a small fraction of the codewords. Search-limited PAC sequential decoding can be employed [28] to address this drawback.

A sequential decoding algorithm's purpose is to efficiently explore through the nodes of the code tree, that is, without inspecting an excessive number of nodes, in order to discover the optimal path. Each node investigated corresponds to a path traversing a part of the code tree. The metric value associated with a path determines whether it is likely to be included in the optimal path, where the metric value measures the path's proximity to the received data. The Fano metric is the most often used metric for a sequential decoding algorithm in a BI-DMC. As explained in the previous chapter, the metric function is described as

$$\gamma(x_i; y_i) = \log_2 \frac{P(y_i|x_i)}{P(y_i)} - b, \tag{3.1}$$

where $P(y_i)$ represents the channel output probability, $P(y_i|x_i)$ represents the channel transition probability, and b represents a constant bias. Initially found using simulations, Massey [43] proved that the Fano metric is a locally optimal metric for examining paths of various lengths for a BI-DMC.

Because demapping each bit on a polarized channel needs the results of all formerly decoded bits, the polarized channel in a PAC code is a channel with

memory. As a result, the Fano metric must be changed to be befitting for decoding on a tree code for a channel with memory. Furthermore, due to the frozen bits, the tree code is an irregular tree, and the metric function should also consider this. For the first i branches, the partial path metric is as follows:

$$\Gamma(\mathbf{u}^i;\mathbf{y}) = \log_2\left(\frac{P(\mathbf{y}|\mathbf{u}^i)}{P(\mathbf{y})}\right) - B_i, \tag{3.2}$$

where the vector $\mathbf{y}$ is the channel output, the data vector $\mathbf{u}^i$ corresponds to a path from the origin of the tree to the ith level, and B_i expresses the partial path bias value up to the ith bit.

It would be more simple to obtain the bit metric instead of the partial path metric of (3.2) for every time a new branch in the tree is inspected. When decoding u_j, the decoder is aware of the channel output $\mathbf{y}$ as well as the prior bits $\mathbf{u}^{j-1}$, which correspond to the input and output of the bit-channel $W_N^{(j)} : \mathcal{X} \longrightarrow \mathcal{Y} \times \mathcal{X}^{j-1}$. By referring to $\gamma(u_j;\mathbf{y},\mathbf{u}^{j-1})$ as the jth bit metric, (3.2) would be represented as

$$\Gamma(\mathbf{u}^i;\mathbf{y}) = \sum_{j=1}^{i} \gamma(u_j;\mathbf{y},\mathbf{u}^{j-1}). \tag{3.3}$$

Consequently,

$$\begin{aligned}
\gamma(u_j;\mathbf{y},\mathbf{u}^{j-1}) &= \Gamma(\mathbf{u}^j;\mathbf{y}) - \Gamma(\mathbf{u}^{j-1};\mathbf{y}) \\
&= \left[\log_2\left(\frac{P(\mathbf{y}|\mathbf{u}^j)}{P(\mathbf{y})}\right) - B_j\right] - \left[\log_2\left(\frac{P(\mathbf{y}|\mathbf{u}^{j-1})}{P(\mathbf{y})}\right) - B_{j-1}\right] \\
&= \log_2\left(\frac{P(\mathbf{y}|\mathbf{u}^j)}{P(\mathbf{y}|\mathbf{u}^{j-1})}\right) - (B_j - B_{j-1}) \\
&= \log_2\left(\frac{P(\mathbf{y}|\mathbf{u}^j)}{P(\mathbf{y}|\mathbf{u}^{j-1})}\right) - b_j = \log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{j-1}|u_j)}{P(\mathbf{y},\mathbf{u}^{j-1})}\right) - b_j,
\end{aligned} \tag{3.4}$$

where $b_j \triangleq B_j - B_{j-1}$ is the jth bit-channel bias value. The bit metric can be

represented as

$$\begin{aligned}
\gamma(u_j; \mathbf{y}, \mathbf{u}^{j-1}) &= \log_2 \left(\frac{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j)}{P(\mathbf{y}, \mathbf{u}^{j-1})} \right) - b_j \\
&= \log_2 \left(\frac{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j)}{\frac{1}{2}\left[P(\mathbf{y}, \mathbf{u}^{j-1}|u_j) + P(\mathbf{y}, \mathbf{u}^{j-1}|u_j \oplus 1)\right]} \right) - b_j \\
&= 1 - \log_2 \left(1 + \frac{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j \oplus 1)}{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j)} \right) - b_j \\
&= 1 - \log_2 \left(1 + 2^{-\log_2\left(\frac{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j)}{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j \oplus 1)}\right)} \right) - b_j \\
&= 1 - \log_2 \left(1 + 2^{-z_j \cdot (-1)^{u_j}} \right) - b_j
\end{aligned} \tag{3.5}$$

for a binary input channel with a uniform input distribution, where

$$z_j = \log_2 \left(\frac{P(\mathbf{y}, \hat{\mathbf{u}}^{j-1}|u_j = 0)}{P(\mathbf{y}, \hat{\mathbf{u}}^{j-1}|u_j = 1)} \right). \tag{3.6}$$

The metric function in the sequential decoding algorithm indicates which path across the decoding tree should be followed. By intuition, the bit-metric function value must always increase for every u_j bit on the correct path to determine the correct data vector $\mathbf{u}$ in a tree. As a result, the bit-metric value on a correct bit u_j at level j of the tree should be greater than the bit metric value on an incorrect bit $\tilde{u}_j$ at that level of the tree. It is challenging to quantify this perception for a particular code. Alternatively, we examine an ensemble of PAC codes with the specified channel and code length, simplifying the issue.

In the case of the ensemble of input bits u_j and the ensemble of bit-channels with output-input $(\mathbf{y}, \mathbf{u}^{j-1}; u_j)$ pair, the expectation of (3.4) is

$$\begin{aligned}
&\mathbb{E}_{U_j, (\mathbf{Y}, \mathbf{U}^{j-1})} \left[\gamma(U_j; \mathbf{Y}, \mathbf{U}^{j-1}) \right] \\
&= \sum_{u_j} q(u_j) \sum_{(\mathbf{y}, \mathbf{u}^{j-1})} P(\mathbf{y}, \mathbf{u}^{j-1}|u_j) \gamma(u_j; \mathbf{y}, \mathbf{u}^{j-1}) \\
&= \sum_{u_j} \sum_{(\mathbf{y}, \mathbf{u}^{j-1})} q(u_j) P(\mathbf{y}, \mathbf{u}^{j-1}|u_j) \left[\log_2 \left(\frac{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j)}{P(\mathbf{y}, \mathbf{u}^{j-1})} \right) - b_j \right] \\
&= I(W_N^{(j)}) - b_j,
\end{aligned} \tag{3.7}$$

where $I(W_N^{(j)})$ denotes the symmetric capacity of the bit-channel $W_N^{(j)}$. As a consequence, the heuristic result is that for bit-channel bias values less than the

bit-channel capacities, the average metric increment per bit of the correct path all the time is positive.

Suppose now that $\tilde{u}_j$ is an incorrect bit with a metric of $\gamma(\tilde{u}_j; \mathbf{y}, \mathbf{u}^{j-1})$ at level j. By averaging this bit metric value over the correct and incorrect branches, we derive

$$\begin{aligned}
&\mathbb{E}_{U_j, \tilde{U}_j, (\mathbf{Y}, \mathbf{U}^{j-1})} \left[\gamma(\tilde{U}_j; \mathbf{Y}, \mathbf{U}^{j-1}) \right] \\
&= \sum_{\tilde{u}_j} q(\tilde{u}_j) \sum_{u_j} q(u_j) \sum_{(\mathbf{y}, \mathbf{u}^{j-1})} P(\mathbf{y}, \mathbf{u}^{j-1} | u_j) \gamma(\tilde{u}_j; \mathbf{y}, \mathbf{u}^{j-1}) \\
&= \sum_{\tilde{u}_j} q(\tilde{u}_j) \sum_{(\mathbf{y}, \mathbf{u}^{j-1})} P(\mathbf{y}, \mathbf{u}^{j-1}) \gamma(\tilde{u}_j; \mathbf{y}, \mathbf{u}^{j-1}) \\
&= \sum_{\tilde{u}_j} \sum_{(\mathbf{y}, \mathbf{u}^{j-1})} q(\tilde{u}_j) P(\mathbf{y}, \mathbf{u}^{j-1}) \left[\log_2 \left(\frac{P(\mathbf{y}, \mathbf{u}^{j-1} | \tilde{u}_j)}{P(\mathbf{y}, \mathbf{u}^{j-1})} \right) - b_j \right] \\
&\leq \sum_{\tilde{u}_j} \sum_{(\mathbf{y}, \mathbf{u}^{j-1})} q(\tilde{u}_j) P(\mathbf{y}, \mathbf{u}^{j-1}) \left[\frac{P(\mathbf{y}, \mathbf{u}^{j-1} | \tilde{u}_j)}{P(\mathbf{y}, \mathbf{u}^{j-1})} - 1 \right] - b_j \\
&= -b_j.
\end{aligned} \tag{3.8}$$

Note that we utilized $\log_2(x) \leq x - 1$ inequality to derive (3.8). Because the bias value is positive, the above expectation implies that the bit metric reduces by a constant amount on average for each incorrect branch. When the decoder traverses in the incorrect direction for multiple branches, based on the decrease of the path-metric value, the decoder recognizes this and switches to another path until the correct path is found.

Given the channel output $\mathbf{y}$, the optimal metric (in terms of error probability) at the ith level of decoding is the one that selects $\hat{\mathbf{u}}^i$ as the vector $\mathbf{u}^i$ by which $p(\mathbf{u}^i|\mathbf{y})$ is maximum. This probability may be determined by utilizing Bayes' rule as

$$p(\mathbf{u}^i|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{u}^i)}{p(\mathbf{y})} p(\mathbf{u}^i). \tag{3.9}$$

Because the monotonically increasing $\log_2(.)$ function maintains maximality,

an optimum metric maximizes

$$\begin{aligned}\log_2 p(\mathbf{u}^i|\mathbf{y}) &= \log_2 \frac{p(\mathbf{y}|\mathbf{u}^i)}{p(\mathbf{y})} - \log_2 \frac{1}{p(\mathbf{u}^i)} \\ &= \log_2 \frac{p(\mathbf{y}|\mathbf{u}^i)}{p(\mathbf{y})} - \sum_{j=1}^{i} \log_2 \frac{1}{p(u_j)},\end{aligned} \tag{3.10}$$

as well, where the final equality is attained by virtue of convolution block in the PAC code being a one-to-one and deterministic operation.

Given that $\log_2 \frac{1}{p(u_j)}$ is indeed the self-information provided regarding bit u_j, which then its average equals $I(W_N^{(j)})$, the path metric function proposed as

$$\Gamma(\mathbf{u}^i; \mathbf{y}) = \log_2 \frac{p(\mathbf{y}|\mathbf{u}^i)}{p(\mathbf{y})} - \sum_{j=1}^{i} I(W_N^{(j)}) \tag{3.11}$$

is a suitable estimate for maximizing the likelihood of error correction, and the jth bit metric function becomes

$$\begin{aligned}\gamma(u_j; \mathbf{y}, \mathbf{u}^{j-1}) &= \Gamma(\mathbf{u}^j; \mathbf{y}) - \Gamma(\mathbf{u}^{j-1}; \mathbf{y}) \\ &= \log_2 \left(\frac{P(\mathbf{y}, \mathbf{u}^{j-1}|u_j)}{P(\mathbf{y}, \mathbf{u}^{j-1})} \right) - I(W_N^{(j)}).\end{aligned} \tag{3.12}$$

For any bias value b_j and bit-channel output $(\mathbf{y}, \mathbf{u}^{j-1})$, notice that $\gamma(u_j; \mathbf{y}, \mathbf{u}^{j-1}) \geq \gamma(\tilde{u}_j; \mathbf{y}, \mathbf{u}^{j-1})$ if and only if $P(\mathbf{y}, \mathbf{u}^{j-1}|u_j) \geq P(\mathbf{y}, \mathbf{u}^{j-1}|\tilde{u}_j)$. This occurs if and only if $h_j(\mathbf{y}, \mathbf{u}^{j-1}) = u_j$, where

$$h_j(\mathbf{y}, \mathbf{u}^{j-1}) = \begin{cases} 0, & P(\mathbf{y}, \mathbf{u}^{j-1}|0) \geq P(\mathbf{y}, \mathbf{u}^{j-1}|1), \\ 1, & \text{otherwise}, \end{cases} \tag{3.13}$$

denotes that for any bias value on a bit channel $W_N^{(j)}$, the bit-metric function satisfies the bit-channel ML rule. This implies that advancing the path with a greater bit metric enhances the likelihood that the expanded path is a component of the PAC code's optimum path.

Furthermore, when the input distribution is uniform, the bit metric is upper

bounded by

$$\begin{aligned}\gamma(u_j;\mathbf{y},\mathbf{u}^{j-1}) &= \log_2 \frac{P(\mathbf{y},\mathbf{u}^{j-1}|u_j)}{P(\mathbf{y},\mathbf{u}^{j-1})} - b_j \\ &= \log_2 \frac{P(\mathbf{y},\mathbf{u}^{j-1},u_j)}{P(u_j)P(\mathbf{y},\mathbf{u}^{j-1})} - b_j \\ &= \log_2 P(u_j|\mathbf{y},\mathbf{u}^{j-1}) + \log_2 \frac{1}{P(u_j)} - b_j \leq 1 - b_j.\end{aligned} \tag{3.14}$$

This results in a locally optimal (ML) decision criterion with a positive bit metric whenever the bias value is less than 1. Additionally, having a bias value less than or equal to $I(W_N^{(j)})$ yields an average positive bit metric. The purpose of the next section is to practically show the channel polarization effect on the computational complexity and error-correction performance by devising a design rule for b_j.

3.2 Results for Computational Complexity and Error-Correcting Performance

As previously explained, we obtain

$$\begin{aligned} &\textit{Correct path:} && b_i \leq I(W_N^{(i)}), \\ &\textit{Wrong path:} && -b_i \leq 0, \end{aligned} \tag{3.15}$$

conditions by averaging on the correct and incorrect paths.

It is important to keep in mind that using $b_i = E_0(1, W_N^{(i)})$ for the bit-channel bias values strictly satisfies both (3.15) inequalities. As a result, for these bias values, the partial path metric of the correct path should, on average, trend positive. For $K = 64$ and $N = 128$, Figure 3.1 supports the latter statement and compares it with the metric growth when using $b_i = I(W_N^{(j)})$ as the bias value. As a direct consequence, the computational complexity associated with bias $b_i = E_0(1, W_N^{(i)})$ should be less than that associated with bias $b_i = I(W_N^{(i)})$, with negligible degradation in error performance. This reduction in computational complexity is due to the polarization effect on both $E_0(1, W_N^{(i)})$ and $I(W_N^{(i)})$, where the distinction between the two situations is based on the polarization

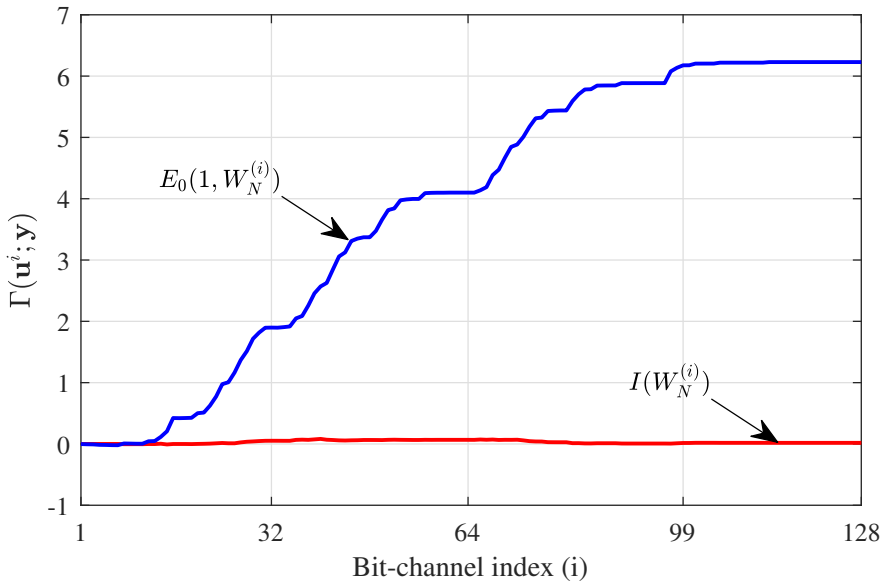

Figure 3.1: Partial path metrics v. bit indices.

difference between the channel cutoff rate and channel capacity, as illustrated in Figure 2.7. The bit-channel capacities and cutoff rates can be obtained before the decoding process (offline) in the sequential decoding of PAC codes. A one-bit quantization of these bias values may be used in a hardware implementation [20] of sequential decoding of PAC code where

$$q(x,\delta) := \begin{cases} 1, & \text{if } x \geq \delta, \\ 0, & \text{otherwise,} \end{cases}$$

is the definition of a 1-bit quantization function $q(x,\delta)$.

Using the bias values $b_i = I(W_N^{(i)})$ and $b_i = E_0(1,W_N^{(i)})$, Figure 3.2 compares the FER and ANV of a PAC$(128,64)$ code. This figure corroborates our prior assertion that, while $E_0(1,W_N^{(i)})$ is somewhat less than $I(W_N^{(i)})$, the computational complexity slightly increases when $I(W_N^{(i)})$ is used. The slight difference in the FER performances happens only at very low SNR levels (very noisy channels). Figure 3.3 also plots the ANV and FER of a quantized bit-channel cutoff rate bias values with $\delta = 0.5$, i.e. if $E_0(1,W_N^{(i)})$ is more than or equal to 0.5, then $b_i = 1$; otherwise $b_i = 0$. The FER of the dispersion approximation is also provided in this figure [44].

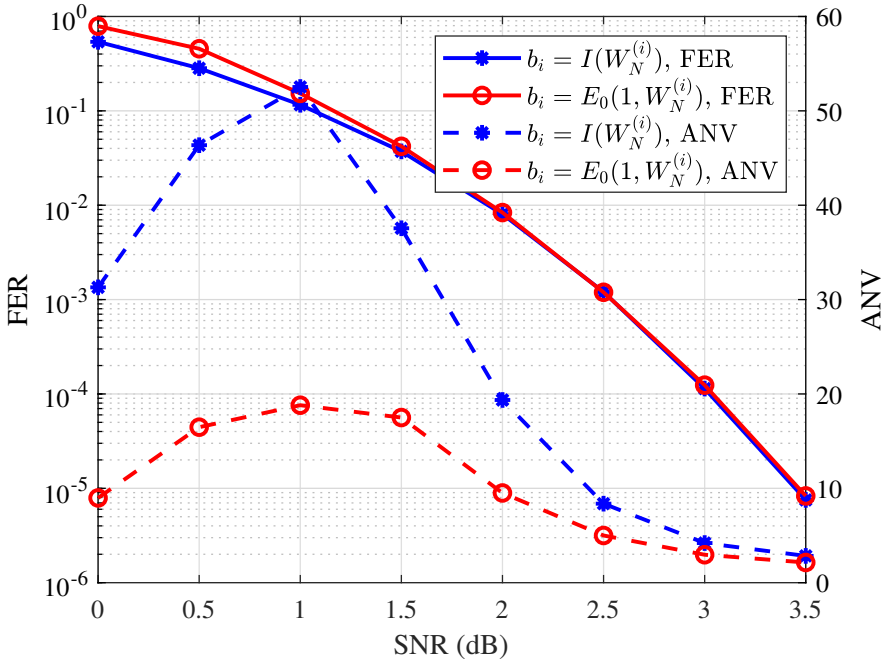

Figure 3.2: Performance of PAC codes with $I(W_N^{(i)})$ and $E_0(1, W_N^{(i)})$ biases in terms of FER and ANV.

3.3 Bounded Complexity Sequential Decoder

Although a sequential decoding algorithm has a variable complexity, it is often desired to have a fixed- or bounded-complexity decoding approach in many applications. We investigate setting a strict limitation on the complexity of the sequential decoding algorithm in this section and examine its performance under such a bound [28]. This requires examining the distribution of the computational complexity measured by Θ (Θ is defined as the number of branches that the algorithm searches divided by the blocklength N). For our purposes, we are mainly concerned with its mean $E(\Theta)$, which we will refer to as the average number of visits (ANV). We will estimate the ANV by calculating the empirical mean of Θ over a suitably large number of simulation runs.

We conduct a simulation experiment and document the observed Θ values for sequential decoding of the PAC$(128, 64)$ code. Table 3.1 summarizes the conclusions for SNR values ranging from 1 to 3.5 dB. The statistics in Table 3.1 demonstrate that the distribution of Θ is highly skewed (only a tiny percentage of

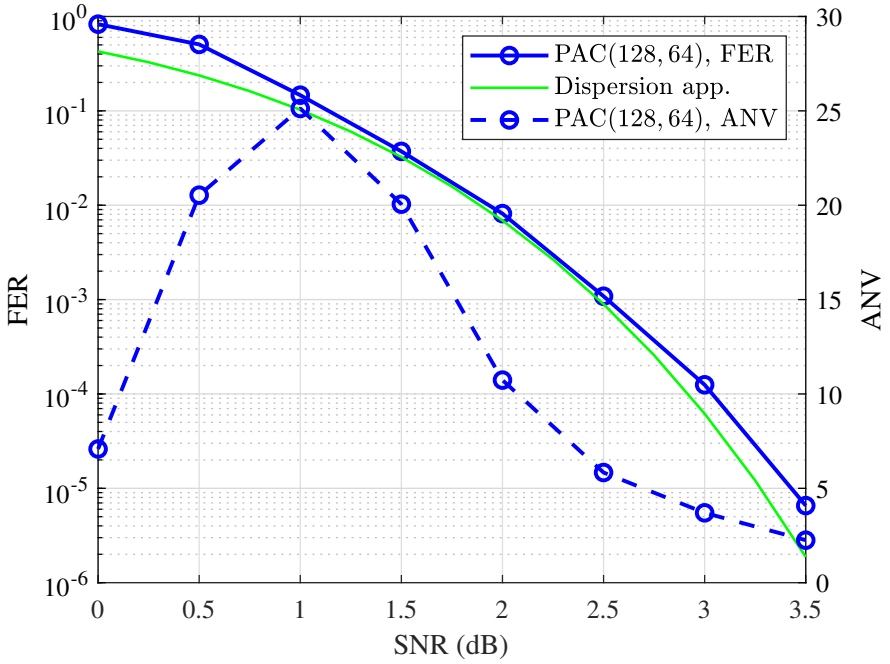

Figure 3.3: Performance of PAC codes with quantized bit-channel cutoff rate bias values in terms of FER and ANV.

decoding occurrences demand a very high level of decoding complexity). Because of this kind of distribution, when the number of visits reaches a certain maximum value, we may consider terminating the sequential decoder and announcing a decoder failure.

Table 3.1: Frequency distribution (number of visits of Θ)

Number of visits	1.0 dB (%)	1.5 dB (%)	2.0 dB (%)	2.5 dB (%)	3.0 dB (%)	3.5 dB (%)
$\Theta \leq 2^3$	64.1110	77.6720	87.6890	92.6490	95.6780	97.9090
$2^3 < \Theta \leq 2^4$	11.2100	7.3660	5.0010	3.6240	2.7070	1.5640
$2^4 < \Theta \leq 2^5$	9.0970	5.0290	2.9210	1.8100	0.9820	0.3690
$2^5 < \Theta \leq 2^6$	7.2300	3.6270	1.7890	0.9680	0.3790	0.1210
$2^6 < \Theta \leq 2^7$	5.2400	2.8520	1.1620	0.4940	0.1550	0.0280
$2^7 < \Theta$	3.1120	3.4540	1.4380	0.4550	0.0990	0.0090

A further term used to represent the maximum number of visits (MNV) allowed per codeword is the maximum number of nodes that the decoder is permitted to visit within a single decoding session to have a search-limited PAC code.

With the search-limited PAC$(128, 64)$ code with MNV $= N \times 2^7$ in Figure 3.4, we can see how well it performs when compared to the SCL decoding of the 5G polar code with a list size of 64 and CRC length of 11. As shown in this figure,

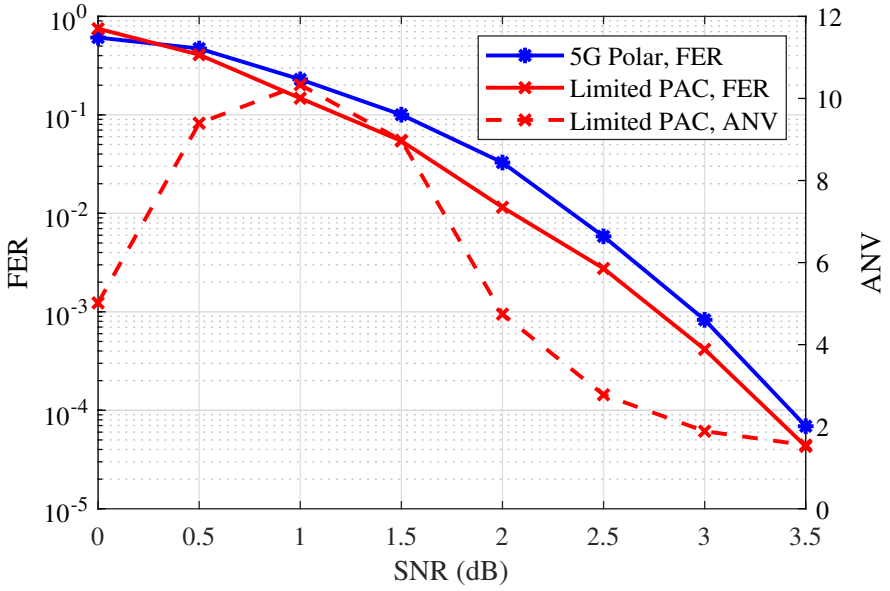

Figure 3.4: With MNV = 2^{14}, search-limited PAC code is compared to polar code with SCL decoding, with list size 64 and CRC length of 11.

the FER performance of PAC codes is nearly equal to that of polar codes, and the ANV values of PAC codes are significantly smaller than the fixed polar code complexity of list size of 64 for all SNR levels.

3.4 Distribution of Computational Complexity

This section examines the distribution of visits during the decoding of PAC codes. As shown in [38], for sequential decoding of CCs, the distribution of computation needed to advance each level in the tree is upper limited as

$$P(C_i > L) < AL^{-\beta}, \tag{3.16}$$

at rates below the cutoff rate, where $A > 0$ and $\beta > 0$ are constants and C_i is the average number of visits per decoded bit. This means that the upper limit on the complementary cumulative distribution function (CCDF) of the computation of sequential decoding of CC is Pareto distributed. Smaller quantities of β correspond to the Pareto distribution's wider tails, and the distribution has a finite mean for values of $\beta > 1$ and a finite variance for values of $\beta > 2$.

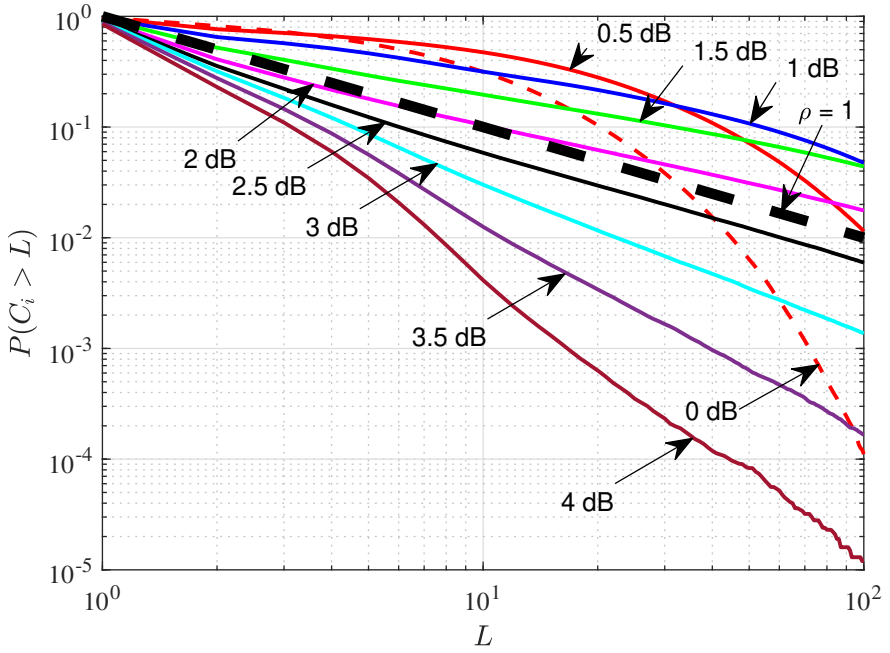

Figure 3.5: CCDF of the number of node visits during sequential decoding of PAC codes.

As an example, in a PAC code case, the CCDF of the number of nodes traversed for all decoded codewords is presented in Figure 3.5 for decoding a PAC$(128, 64)$ code at various SNR levels. The findings indicate that: 1) $P(C_i > L) < L^{-1}$ for the SNR values greater than 2 dB, and as a result, the mean of the upper bound distribution is finite; 2) For SNR values 3.5 dB and 4 dB, we have that $\beta > 2$, which results in a finite variance for the upper bound distribution. Additionally, Figure 3.5 depicts the trade-off involving outage probability (the likelihood that C_i exceeds some limit L) and sequential decoding's computational complexity. At an SNR of 3.5 dB, as an example, there is a 1 percent chance that a decoded codeword will take more than ten visits per branch, and the likelihood that the number of visits for a decoded codeword will exceed L decreases as L grows. This figure, in combination with Table 3.1, is a useful tool for determining how much we may restrict the number of visits depending on the desired performance.

3.5 Sequential Decoding of PAC Codes Using Heap Data Structure

This section presents a memory-efficient polar demapper and a sequential decoding technique for PAC codes based on the heap [45] data structure. Heap data structure is used in sequential decoding in [22, 46]. We investigate the suggested algorithm's error-correction performance, memory requirements, and complexity. We employ partial path metrics as the key values in the heap data structure. The satellite data in a heap are also the CC's input vectors, CC's states, intermediate LLR, and check-sum values. The heap data structure operations are of logarithmic time, and we store intermediate LLR and check-sum values in a vector of length $N-1$ and $N/2$, respectively.

3.5.1 Data Structure

In this part, we utilize a max-heap priority queue due to the simplicity of the design and the convenience of explanation (a similar implementation is possible using a min-max heap priority queue). A max-heap is a complete binary tree with the condition that, except for the leaf nodes, the parent node element is bigger than both child node elements. In our context, the partial path metric is the key of the heap priority queue. In this case, the root node of the max-heap is the decoding tree node with the greatest partial path metric value, which the sequential decoder should explore further. We just use the word "heap" when referring to the max-heap priority queue.

Maintaining the data structure is needed when decoding with a heap priority queue. We utilize the Heap-Insert operation to insert a new element, which needs $\mathcal{O}(\log_2 N)$ time. To have the best path from a priority queue, the Extract-Max algorithm is used. Additionally, in the case of finite-size heaps, when the heap is on the point of overflowing, the Delete-Min function will be called, which requires $\mathcal{O}(N)$ time. Utilizing double-ended priority queue (DEPQ)

data structures, such as a min-max heap or a pairing heap, enables efficient (in $\mathcal{O}(\log_2 N)$ time) extraction of both the maximum and minimum values, based on some ordering of the structure's keys (items).

In decoding PAC codes, we define H to represent the heap data structure, and each element of H is a five-attribute object. For the array H's ith entry, $H[i].m$ returns the partial path metric for the ith element; $H[i].v$ is the convolutional encoder input sequence corresponding to the ith element; $H[i].state$, $H[i].L$, and $H[i].us$ denote the CC's current state, intermediate LLR values, and intermediate check-sum values, respectively. Our presented decoding algorithm use $N-1$ and $N/2$ memory sizes for the $H[i].L$, and $H[i].us$, respectively.

Our simulations present the decoding computation in terms of the average number of cycles (ANC) per decoded bit. Each cycle includes one EXTRACT-MAX execution, one HEAP-INSERT operation if the bit is frozen, and two HEAP-INSERT operations if the bit is an information bit, each of which takes $\mathcal{O}(\log_2(H.\textit{heap-size}))$ time, where $H.\textit{heap-size}$ is the number of elements in the heap. Additionally, each cycle has one UPDATELLR operation. With aggregate analysis, the running time for a heap member in UPDATELLR is no more than $2N-2$, both for computing the intermediate LLR and check-sum values.

For a finite heap size, if the heap is full, one call of DELETE-MIN is made if the bit is frozen, and two calls are made if the bit is a data bit. The running time of DELETE-MIN is $\mathcal{O}(H.\textit{heap-size})$, and the heap size determines the frequency with which the operation is called.

3.5.2 Heap Sequential Decoding Algorithm

Algorithm 1 is the main function of the heap sequential decoding for the PAC codes. The MAIN-DECODER algorithm makes use of the LLR values for the channel output denoted by L_y, the code length N, the data index set $\mathcal{A}$, the connection polynomial c, and the vector of bit-channel bias values represented by *E0*. We use the the *max-cycles* and *max-heap-size* input parameters to bound the

Algorithm 1: MAIN-DECODER

Input: $L_y, N, \mathcal{A}, c$, *E0*, *max-cycles*, *max-heap-size*

Output: $\hat{u}$, *cycles*, *H.heap-size*

$H[1].m \leftarrow 0$
$H[1].v \leftarrow$ NIL
$H[1].state[1, \cdots, |g|-1] \leftarrow [0, \cdots, 0]$
$H[1].L[1, \cdots, N-1] \leftarrow [0, \cdots, 0]$
$H[1].us[1, \cdots, N/2] \leftarrow [0, \cdots, 0]$
$[H.heap\text{-}size, cycles, n] \leftarrow [1, 0, \log_2(N)]$
while *True* **do**
  if $cycles = max\text{-}cycles$ **then**
    break
  else
    $cycles \leftarrow cycles + 1$
  end
  $[max, H] \leftarrow$ EXTRACT-MAX(H)
  $cState \leftarrow max.state$
  $i \leftarrow |max.v| + 1$
  $[L, us] \leftarrow$ UPDATELLR$(L_y, max.L, i-1, max.us, n)$
  $L_i = L[N-1]$
  if $i \in \mathcal{A}^c$ **then**
    $[u_0, nState] \leftarrow$ CONVENC$(0, cState, c)$
    $M \leftarrow max.m +$ CALC-MET$(L_i, u_0, E0(i))$
    $v' \leftarrow [max.v] + [0]$
    $H \leftarrow$ HEAP-INSERT$(H, M, v', nextS, L, u, us)$
  else
    $[u_0, nState] \leftarrow$ CONVENC$(0, cState, c)$
    $M \leftarrow max.m +$ CALC-MET$(L_i, u_0, E0(i))$
    $v' \leftarrow [max.v] + [0]$
    $H \leftarrow$ HEAP-INSERT$(H, M, v', nextS, L, u, us)$
    $[u_1, nState] \leftarrow$ CONVENC$(1, cState, c)$
    $M \leftarrow max.m +$ CALC-MET$(L_i, u_1, E0(i))$
    $v' \leftarrow [max.v] + [1]$
    $H \leftarrow$ HEAP-INSERT$(H, M, v', nextS, L, u, us)$
  end
  if $|H[1].v| = N$ **then**
    break
  end
end
$\hat{u} = H[1].v$
return $[H.heap\text{-}size, cycles, \hat{u}]$

amount of cycles and heap size. The last element of L is the ith bit's updated LLR value, which is assigned to L_i on line 17. For the ith frozen bit, lines 19-22 compute the partial path metric using a one-bit convolutional encoder and metric calculator function and insert it into the heap.

The number of times this algorithm's main *while* loop is executed is a random variable dependent on the noise level. Each iteration has one call to EXTRACT-MAX, one call to UPDATELLR, one call to HEAP-INSERT for the frozen bits, and two calls of it for the information bits.

Algorithm 2: CONVENC

Input: $v, currState, c$
Output: $u, nextState$

```
1 u ← v · c[1]
2 for i ← 2 to |c| do
3 |   if c[i] = 1 then
4 |   |   u ← u ⊕ currState[i − 1]
5 |   end
6 end
7 nexState ← [v] + currState[1, ··· , |c| − 1]
8 return [u, nexState]
```

The Algorithm 2 is a one-bit convolutional encoder with a running time of $\mathcal{O}(|c|)$ that inputs the bit value, current state, and connection polynomial and outputs the next state of the CC and the encoded bit.

The Algorithm 3 corresponds to Figure 2.15's polar demapper block. This algorithm inputs the channel output LLR values; the best path's length, intermediate LLR, and check-sum values; and $n = \log_2(N)$. $sel = 1$ and $sel = 0$ in this algorithm correspond to the use of the f and g functions, respectively. Choosing f or g functions are by find-first-bit-set (ffs) operation introduced in [47]. The parameter s has a range of 0 to $n - 1$, with 0 indicating the leaf level of the tree. For the first bit of the codeword, the algorithm should begin at the root level and first call the f function. On line 5, the ffs function determines the level, and the g function is picked.

The check-sum vector has a length of $N/2$, and line 10 determines if the bit

Algorithm 3: UPDATE-LLR

Input: L_y, L, i, us, n
Output: L, us

$N \leftarrow 2^n$
if $i = 0$ **then**
  $[s, sel] \leftarrow [n-1, 1]$
else
  $[s, sel] \leftarrow [\text{ffs}(i), 0]$
end
if $s > 0$ **and** $sel = 0$ **then**
  $m \leftarrow 1$
  while $m \leq 2^{s-1}$ **do**
    if $i > N/2 + 1$ **then**
      $k \leftarrow i - 2^s + 1 + \lfloor \frac{2^s - 1}{2m} \rfloor 2m - N/2$
    else
      $k \leftarrow i - 2^s + 1 + \lfloor \frac{2^s - 1}{2m} \rfloor 2m$
    end
    $a \leftarrow us[k, \cdots k + m - 1]$
    $b \leftarrow us[k + m, \cdots, k + 2m - 1]$
    $us[k, \cdots, k + 2m - 1] \leftarrow [a \oplus b] + [b]$
    $m \leftarrow 2m$
  end
end
<Go to **Algorithm 4**>
return (L, us)

position is greater than $N/2 + 1$; if it is, it overwrites the existing check-sum values. This while loop has a maximum execution time of $\mathcal{O}(\log_2 N)$. If decoding completes for one member of the heap, the overall running time of the check-sum operation is $2N - 2$ when the aggregate analysis is used.

When the corresponding check-sum values for the g function are selected in line 26, lines 28-34 update the LLR values for the associated subtree. Line 37 contains the end and start indices of the associated subtree. Lines 38-44 choose the check-sum values that will be utilized in the g function on line 51. Line 54 initializes the next level's f function. Notably, if decoding completes for a single path, $N/2$ of the bits will have $s = 0$, and the corresponding loop's running duration will be 1. The loops run on length two for $N/4$ of the bits and so on.

The Algorithm 5 returns the heap element with the highest partial path metric value. Note that the extraction of the heap's maximum involves maintaining the heap structure and its execution time is in the order of $\mathcal{O}(\log_2 N)$.

The Algorithm 6 inserts a new element into the heap and updates its structure; its execution time is in the order of $\mathcal{O}(\log_2 N)$.

3.5.3 Simulation Results

The FER performance, ANC values, and average heap sizes for decoding the PAC$(128, 64)$ code are shown in Figure 3.6. As illustrated in this figure, the ANC varies just a little for different heap sizes, and for high SNR levels, the ANC per decoded bit is almost equal to one for all presented plots. Additionally, for high SNR levels, the average heap size approaches 64. Similar to the Figure 3.6, the Figure 3.7 also depicts the FER performance, ANC values, and average heap sizes of the PAC$(128, 99)$ code. This result can be compared with the stack decoding of polar codes, which needs a stack size of 2^{16} for a $(128, 99)$ polar code reported in [48].

Algorithm 4: Lines 23-55 in Algorithm 3

```
23 while s ≥ 0 do
24     if s = n − 1 then
25         if i > 0 and sel = 0 then
26             us′ ← us[i − 2^s + 1, ··· , i]
27         end
28         for p ← 1 to 2^s do
29             if sel = 1 then
30                 L[p] ← f(L_y[p], L_y[2^s + p])
31             else
32                 L[p] ← g(L_y[p], L_y[2^s + p], us′[p])
33             end
34         end
35         sel ← 1
36     else
37         [κ_1, κ_2] ← [2^n − 2^{s+1}, 2^n − 2^{s+2}]
38         if i > 0 and sel = 0 then
39             if i ≥ N/2 + 1 then
40                 us′ ← us[i − 2^s + 1 − N/2, ··· , i − N/2]
41             else
42                 us′ ← us[i − 2^s + 1, ··· , i]
43             end
44         end
45         for p ← 1 to 2^s do
46             L_a ← L[κ_2 + p]
47             L_b ← L[κ_2 + 2^s + p]
48             if sel == 1 then
49                 L[κ_1 + p] ← f(L_a, L_b)
50             else
51                 L[κ_1 + p] ← g(L_a, L_b, us′[p])
52             end
53         end
54         sel ← 1
55     end
56     s ← s − 1
57 end
```

Algorithm 5: EXTRACT-MAX

Input: the heap H
Output: the updated H, max

```
1  max ← H[1]
2  H[1] ← H[H.heap-size]
3  [i, j] ← [1, 2]
4  while j < H.heap-size − 1 do
5  |  if H[j + 1].m > H[j].m then
6  |  |  j ← j + 1
7  |  end
8  |  if H[i].m < H[j].m then
9  |  |  exchange H[i] with H[j]
10 |  |  [i, j] ← [j, 2j]
11 |  else
12 |  |  break
13 |  end
14 end
15 H.heap-size ← H.heap-size − 1
16 return (max, H)
```

Algorithm 6: HEAP-INSERT

Input: $H, M, v, state, L, u, us$
Output: the updated H

```
1  H.heap-size ← H.heap-size + 1
2  H[H.heap-size].m ← M
3  H[H.heap-size].v ← v
4  H[H.heap-size].state ← state + 1
5  H[H.heap-size].L ← L
6  H[H.heap-size].u ← u
7  H[H.heap-size].us ← us
8  temp ← H[H.heap-size]
9  i ← H.heap-size
10 while i > 1 and temp.m > H[⌊i/2⌋].m do
11 |  H[i] ← H[⌊i/2⌋]
12 |  i ← ⌊i/2⌋
13 end
14 H[i] ← temp
15 return H
```

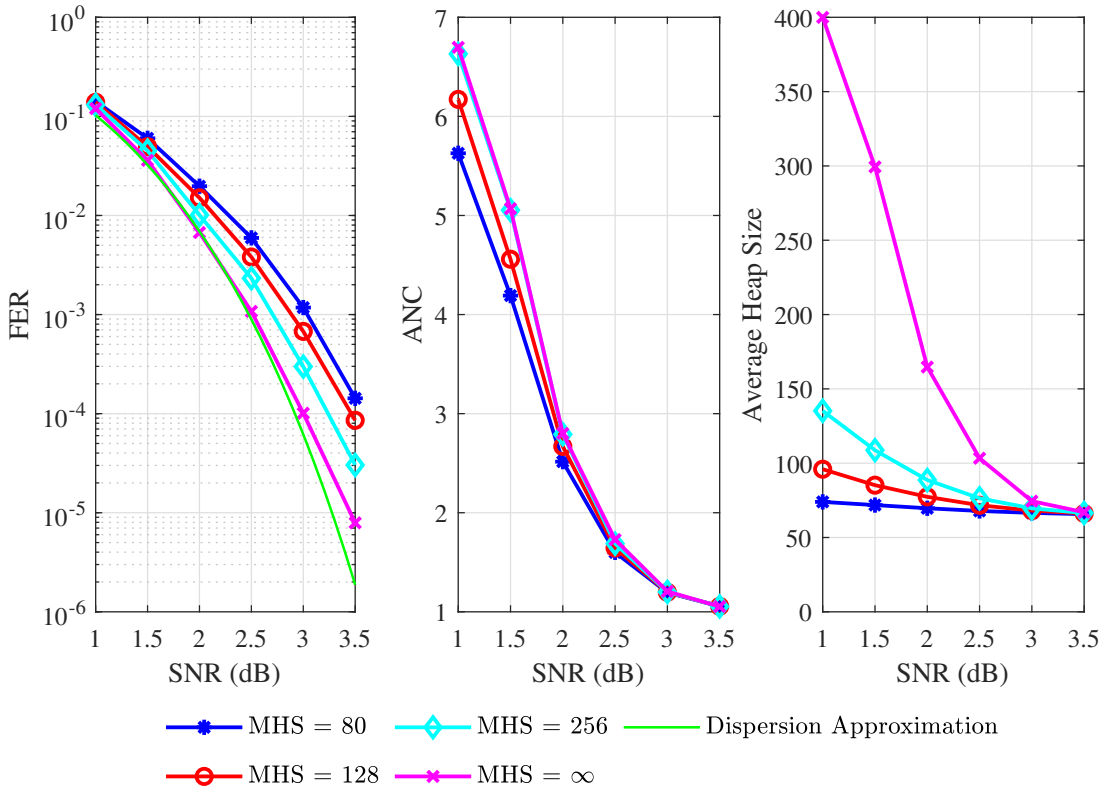

Figure 3.6: PAC$(128, 64)$ code performance.

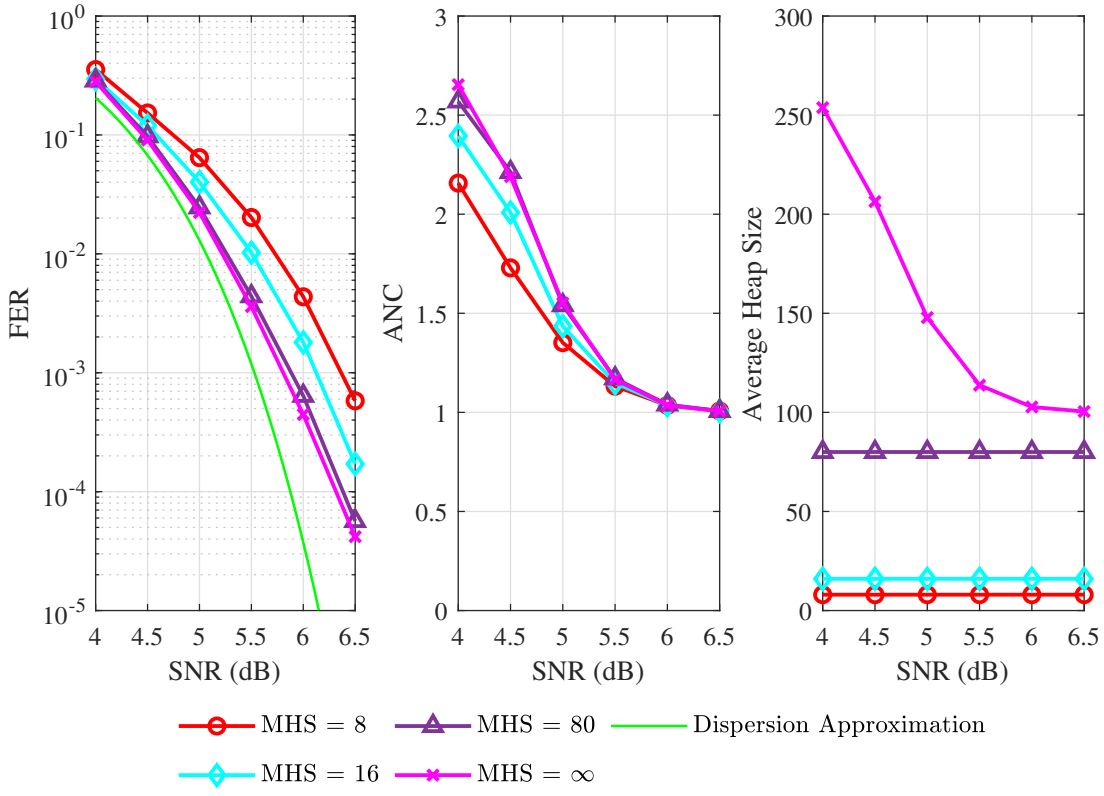

Figure 3.7: PAC$(128, 99)$ code performance.

Chapter 4

Computational Complexity of Sequential Decoding of PAC Codes

This chapter studies the upper and lower bounds on the computational complexity of the sequential decoding of PAC codes. We apply guessing to obtain a lower bound on the computational complexity of sequential decoding of PAC codes. Sequential decoding is an algorithm that performs the decoding by attempting to guess its path through an expanding tree of the most probable transmitted sequences. The computational complexity would be reduced in this manner. In the case of sequential decoding of CCs, this generally comes at the cost of communicating at rates strictly below capacity. The computational cutoff rate denotes the region between rates whose average decoding computational complexity is finite and those which is infinite. This chapter, benefiting from the polarized channels, proves that the computational cutoff rate in sequential decoding of PAC codes polarizes.

The FER performance of PAC codes is highly dependent on rate profiling. In this chapter, a Monte-Carlo based rate-profile construction method is proposed

that can improve the error-correction performance of PAC codes while guaranteeing a low mean sequential decoding complexity for signal-to-noise ratio (SNR) values beyond a target SNR value. This construction method is based on beginning from a bigger data index set $\mathcal{A}$ and freezing one by one the indices that result in the first bit error (FBE). This is similar to the offline version of the bit-flipping technique used in SC-flip decoding of polar codes [49]. The SC-flip decoder may identify wrongly decoded frames using a CRC check, and it can locate and freeze the FBE position using the log-likelihood ratio (LLR) values of the decoded bits. When the SC-flip decoding algorithm is employed, the simulation results demonstrate a slight improvement over the SC decoder.

The rate profiling of PAC codes also has been subjected to several different improvements. In [50], the PAC coding rate profile is obtained by using a discrete optimization approach based on simulated annealing. Their findings demonstrate that PAC codes with this rate profile have a high error-correction performance. In [51], a rate-profile construction method based on a reinforcement learning algorithm is proposed. This approach uses a set of reward and update techniques that aid in discovering the rate profile by the reinforcement learning agent. In [52], it is shown that a PAC$(256, 128)$ code can also achieve the theoretical bounds by constructing the code rate profile using a genetic algorithm. A rate profile based on the metaheuristic algorithms by targeting the FER performance can result in a sequential decoding algorithm with a very high computational complexity. In addition, in [53] a heuristic method based on maximum possible minimum distance is proposed for the construction of PAC code rate profiles.

4.1 Application of Guessing to Sequential Decoding of PAC codes

As discussed in the previous chapters, sequential decoding of PAC code has a variable computational complexity and, like sequential decoding of conventional CCs, suffers from the cutoff rate phenomenon. Furthermore, using polar codes in

the PAC codes makes the computational complexity of decoding different from when conventional CCs are decoded sequentially. The main objective of this section is to provide a lower bound on the computational complexity of sequential decoding of PAC codes utilizing the guessing function approach of [54]. Arıkan in [54] provides a tight lower bound on the average computation required for sequential decoding of conventional CCs by employing the relationship between the computational complexity of the sequential decoding and the guessing function. We address the computational complexity of sequential decoding of PAC codes using this lower bound and channel polarization approach.

In information theory, guessing traces its origins to Massey's work [55]. Massey proved that by guessing the value of a random variable X in decreasing order of the probabilities ($p_1 \geq p_2 \geq \cdots$), the number of guesses $G(X)$ would have the smallest average, where p_1 is the most likely symbol in the space of the random variable X, p_2 is the second most likely and so on. In this manner, for entropy $H(X) \geq 2$

$$\mathbb{E}[G(X)] = \sum_{i=1} i.p_i \geq \frac{1}{4} 2^{H(X)} + 1, \tag{4.1}$$

where $H(X)$ denotes the entropy function.

Arıkan [54] proved that for the random variable X with a finite alphabet $\mathcal{X}$ of size M, by guessing the values in a decreasing order of the probabilities the average number of successive guesses is upper and lower bounded as

$$\left[\sum_{x\in\mathcal{X}} \sqrt{P_X(x)}\right]^2 \geq \mathbb{E}[G(X)] \geq \frac{\left[\sum_{x\in\mathcal{X}} \sqrt{P_X(x)}\right]^2}{1+\ln M}, \tag{4.2}$$

where $P_X(.)$ is the probability distribution of X.

For a generalization that is useful in the channel coding problem [54], consider a pair of discrete random variables (X, Y) of the input and output of the channel where X has probability distribution P_X and takes one of the values in $\mathcal{X} = \{1, 2, \cdots, M\}$, and the channel output alphabet $\mathcal{Y}$ can be continuous. For a given Y, the number of successive guesses needed to guess the correct input X,

denoted by $G(X|Y)$, has a lower bound on its average as

$$\mathbb{E}[G(X|Y)] \geq \frac{\sum_{y\in\mathcal{Y}}\left[\sum_{x\in\mathcal{X}}\sqrt{P_{X,Y}(x,y)}\right]^2}{1+\ln M}. \tag{4.3}$$

where $P_{X,Y}(x,y)$ is the joint probability distribution of (X,Y). Since P_X has a uniform distribution and the size of $\mathcal{X}$ is equal to $M = e^{NR}$, the lower bound on the average of $G(X|Y)$ can be expressed as

$$\mathbb{E}[G(X|Y)] \geq \frac{e^{NR-R_0(W)}}{1+NR}, \tag{4.4}$$

where $R_0(W)$ is the cutoff rate function for (X,Y) and is defined as

$$R_0(W) = R_0(X,Y) = -\log\sum_{y\in\mathcal{Y}}\left[\sum_{x\in\mathcal{X}}P(x)\sqrt{P(y|x)}\right]^2. \tag{4.5}$$

Since, $\mathbb{E}[G(X|Y)]$ is upper bounded by $e^{NR-R_0(W)}$ [54], to denote the tight lower bound, we use notation

$$\mathbb{E}[G(X|Y)] \gtrapprox e^{NR-R_0(W)}. \tag{4.6}$$

Cutoff rate function is related to the Bhattacharyya parameter by

$$R_0(W) = \log_2\frac{2}{1+Z(W)}, \tag{4.7}$$

and consequently polarizing the Bhattacharyya parameter results in polarized cutoff rate.

To relate the number of guesses $G(X|Y)$ to sequential decoding of a PAC code, consider an arbitrary tree code of a PAC code and suppose that $\mathcal{X}$ is the set of all nodes at a fixed but arbitrary level N of the tree, and X is a random variable on $\mathcal{X}$ with a uniform distribution. We can think of X as the node in $\mathcal{X}$ which lies on the transmitted path or equivalently as the channel input sequence of length N. The number of paths from the origin of length N is equal to the number of nodes at level N, and there is a one-to-one correspondence between them. In this manner, the guessing function $G(.|.)$ is the sum of the number of nodes in $\mathcal{X}$ which are examined before, and the correct node $X = \mathbf{x}$ when $\mathbf{y}$ is received. No guess will be repeated in guessing the channel input, and whenever the correct

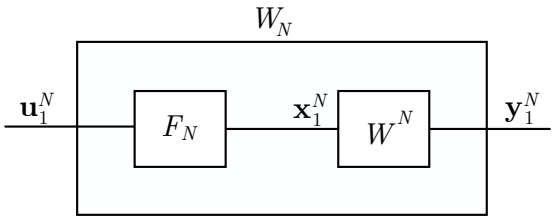

Figure 4.1: Polar code construction of length N.

channel input is guessed, the genie tells the decoder to stop. Thus, the number of guesses $G(X|Y)$ is a lower bound to the PAC decoder's computation in decoding the first N bits of the transmitted sequence. Then, the lower bound to the average of guessing number $G(X|Y)$ serves as a lower bound to average computation in sequential decoding.

As Figure 4.1 illustrates, the combined channel W_N that vector $\mathbf{u}^N$ sees is derived from a pre-processing on N parallel channels seen by the vector $\mathbf{x}^N$. Using N copies of channel W, channel W_N is obtained by the channel combining phase explained in [2]. Input-output pair of the channel W_N is $(U^N; Y^N)$ and with Gallager's parallel channel theorem the upper bound on the combined channel cutoff rate [34, p. 149-150] we have

$$R_0(W_N) = R_0(U^N; Y^N) \leq NR_0(W), \tag{4.8}$$

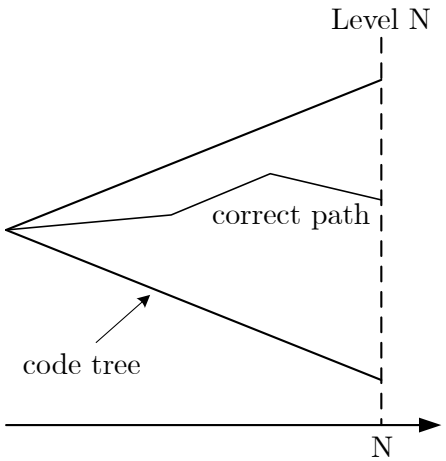

Figure 4.2: Decoding tree of PAC codes.

Consider a $(N, K, \mathcal{A}, T)$ PAC code with the tree code shown in Figure 4.2. By

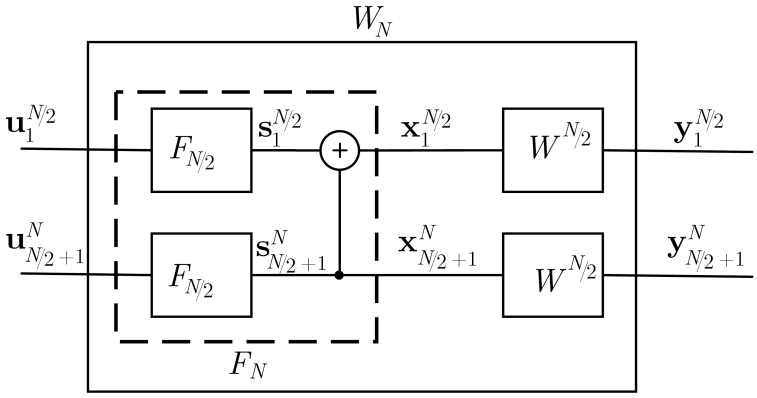

Figure 4.3: Recursive construction of polar code of length N.

using (4.6), the average number of guesses $\mathbb{E}[G_N]$ has a lower bound

$$\mathbb{E}[G_N] \gtrapprox e^{NR - R_0(U^N;Y^N)}, \tag{4.9}$$

and using (4.8), the lower bound on the average number of guesses becomes

$$\mathbb{E}[G_N] \gtrapprox e^{N(R - R_0(W))}. \tag{4.10}$$

The overall recursion of the polar mapper is illustrated in Figure 4.3. We have N parallel channels $W : X_i \to Y_i$, for $1 \leq i \leq N$. Suppose that K^- is the number of information bits in $\mathbf{v}^{N/2}$ and K^+ is the number of information bits in $\mathbf{v}_{N/2+1}^{N}$ s.t. $K = K^- + K^+$, and define

$$R^- \triangleq \frac{K^-}{N/2}, \quad R^+ \triangleq \frac{K^+}{N/2}. \tag{4.11}$$

Note that $R^- + R^+ = 2R$. Similarly, let us denote the first and second halves' cutoff rates after one step of polarization by $R_0(W^-)$ and $R_0(W^+)$, respectively. From the channel polarization theorem

$$R_0(W^-) + R_0(W^+) \geq 2R_0(W), \tag{4.12}$$

which shows that after one step of polarization cutoff is boosted [56]. The main idea of boosting the cutoff rate is to build correlated synthesized channels of independent channels such that the sum of the cutoff rates of synthesized channels becomes greater than the independent channels.

Suppose that the decoder in the tree code of Figure 4.4 wants to reach the level $N/2$. We show the required average number of guesses by $\mathbb{E}[G_{N/2}]$. We also show the average number of guesses needed to decode the second half of the code as $\mathbb{E}[G_{N/2,N}]$ assuming a genie gives us the $\mathbf{u}^{N/2}$.

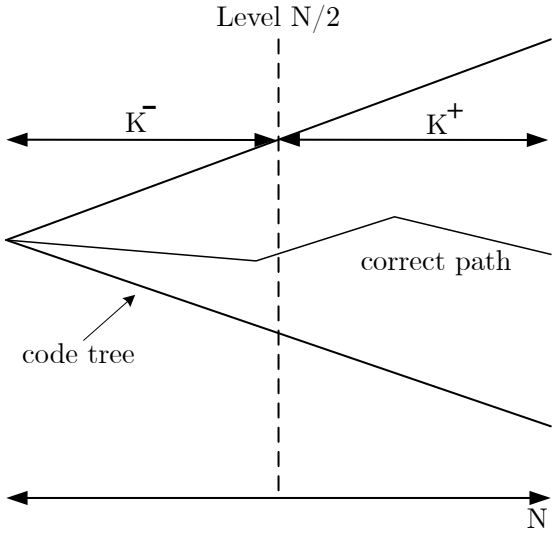

Figure 4.4: Decoding tree of PAC codes after one step polarization.

Theorem 1. *In sequential decoding of PAC codes, the computational cutoff rate polarizes, meaning that the lower bound on the average number of guesses for decoding the first and second halves of the codeword are exponential in $N/2$ as*

$$\begin{aligned} \mathbb{E}[G_{N/2}] &\gtrapprox e^{\frac{N}{2}(R^- - R_0(W^-))}, \\ \mathbb{E}[G_{N/2,N}] &\gtrapprox e^{\frac{N}{2}(R^+ - R_0(W^+))}. \end{aligned} \tag{4.13}$$

Proof. In one step polarization, we obtain $N/2$ parallel bad channels as

$$W^- : S_i \to (Y_i, Y_{N/2+i}), \tag{4.14}$$

for $1 \leq i \leq N/2$. Suppose that the decoder in the tree code of Figure 4.4 wants to reach the level $N/2$. The required average number of guesses has a lower bound as

$$\mathbb{E}[G_{N/2}] \gtrapprox e^{(\frac{N}{2}R^- - R_0(U^{N/2};Y^N))}. \tag{4.15}$$

Same as our first step, we have $N/2$ parallel copies of W^-, and a $F_{N/2}$ preprocessing is performed on the channel inputs $\mathbf{s}^{N/2}$ to obtain $\mathbf{u}^{N/2}$. As a result, by

using the parallel channel theorem for the first half of the bit channels, we have

$$R_0(U^{N/2}; Y^N) \leq \frac{N}{2} R_0(W^-). \tag{4.16}$$

Consequently, the average number of guesses required to decode the first half of the bits $\mathbb{E}[G_{N/2}]$ has the lower bound

$$\mathbb{E}[G_{N/2}] \gtrapprox e^{\frac{N}{2}(R^- - R_0(W^-))}. \tag{4.17}$$

Moreover, in one step polarization, we also obtain $N/2$ parallel good channels

$$W^+ : S_{N/2+i} \to (Y_i, Y_{N/2+i}, S_i), \tag{4.18}$$

for $1 \leq i \leq N/2$. In the same manner, if a genie provides the $\mathbf{u}^{N/2}$, the cutoff rate for the second half is obtained

$$R_0(U_{N/2+1}^N; Y^N, U^{N/2}) \leq \frac{N}{2} R_0(W^+). \tag{4.19}$$

With the genie aided decoding assumption for the first half, the average number of guesses required to decode the second half has a lower bound

$$\mathbb{E}[G_{N/2,N}] \gtrapprox e^{(\frac{N}{2}R^+ - R_0(U_{N/2+1}^N; Y^N, U^{N/2}))}. \tag{4.20}$$

Therefore, the average number of guesses required to decode the second half of the bits $\mathbb{E}[G_{N/2,N}]$ has the lower bound

$$\mathbb{E}[G_{N/2,N}] \gtrapprox e^{\frac{N}{2}(R^+ - R_0(W^+))}. \tag{4.21}$$

Lower bound on the number of guesses in (4.10) is exponential in blocklength N, and after one step polarization (4.17) and (4.21) are exponential in $N/2$ which is the gain in computational complexity of the PAC sequential decoder. This proves that the computational cutoff rate polarizes.

□

Figure 4.5 extends above operation recursively for the levels of size $N/4$. K^{--} denotes the number of information bits in the first $N/4$ bits and R^{--} is the corresponding rate. K^{-+}, K^{+-}, and K^{++} are defined likewise with their corresponding

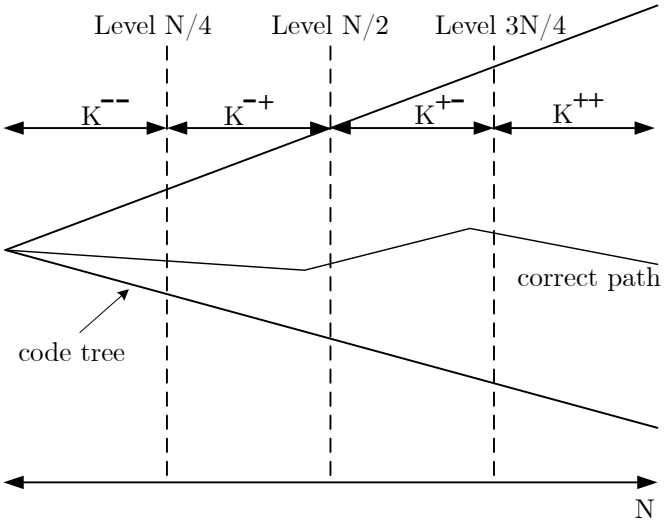

Figure 4.5: Decoding tree of PAC codes after two step polarization.

rates. From channel polarization we have

$$R_0(W^{--}) + R_0(W^{-+}) \geq 2R_0(W^{-}), \tag{4.22}$$

and

$$R_0(W^{+-}) + R_0(W^{++}) \geq 2R_0(W^{+}), \tag{4.23}$$

which results in

$$R_0(W^{--}) + R_0(W^{-+}) + R_0(W^{+-}) + R_0(W^{++}) \geq 4R_0(W). \tag{4.24}$$

Following this procedure implies that the PAC code rate profile should be lower than the channel cutoff rate profile in order to have a minimal lower bound.

It is worthwhile to impose an upper constraint B_N on the number of visits of sequential decoding during a decoding session in order to investigate the effect of the cutoff rate and channel polarization on decoding PAC codes. Take, for example, the assumption that decoding will be stopped if the total number of visits exceeds B_N. Similarly, presume that $B_{N/2}$ is the upper constraint on the number of visits to the first half of the decoding tree levels and that decoding will be ended if the number of visits exceeds this upper bound. Continuing this way, the upper bound to decode the first bit is B_1. By extreme limits if $2B_i = B_{i+1}$ and $B_1 = 1$, the decoding is like SC decoding. Hence, this proves that the PAC

code with infinite blocklength and a similar decoding complexity as SC decoding can achieve the channel capacity.

Table 4.1: Rate and cutoff rate after one and two steps of polarization for different cutoff rate values of PAC$(256, 128)$ code

	W	W^-	W^+	W^{--}	W^{-+}	W^{+-}	W^{++}
Cutoff rate, 1 dB	0.3838	0.1863	0.6394	0.0548	0.3455	0.4545	0.8881
Cutoff rate, 2 dB	0.4612	0.2613	0.7310	0.1058	0.4668	0.5680	0.9406
Polar rate profile	0.5	0.2422	0.7578	0.0625	0.4219	0.5781	0.9375
RM-polar	0.5	0.2656	0.7344	0.0781	0.4531	0.5781	0.8906

Table 4.1 compares the cutoff rate polarization at 1 and 2 dB SNR values with polar and RM-Polar rate profiles used in a PAC$(256, 128)$ code. As this table shows, at 1 and 2 dB SNR values, the cutoff rates of a BI-AWGN channel are respectively 0.3838 and 0.4612. From (4.10), the computational complexity of sequential decoding of a code with a rate of 0.5 should increase exponentially with the code length. After two steps of polarization, the rates $R^{--}, R^{-+}, R^{+-}, R^{++}$ for the polar and RM-Polar rate profiles are becoming close to the cutoff rates for 1 and 2 dB SNR values. Therefore, thanks to the polarization effect, the computational complexity of a PAC code is constant for the SNR values beyond the cutoff rate.

4.2 Upper Bound Estimation on the Distribution of Computation for Sequential Decoding

Code rates should be less than the channel cutoff rate to achieve finite mean computational complexity in sequential decoding of CCs. In the previous section, we proved that it is required to have R^- less than $R_0(W^-)$ to achieve a finite mean computational complexity in sequential decoding of the first half of the

codewords. For decoding the second half of the codewords, the $R^+ < R_0(W^+)$ condition can result in a finite mean computational complexity. Motivated by this, the requirement that CCs have a finite mean computational complexity may well be adapted to PAC codes for all $1 \leq l \leq N$ in the form of [57]

$$lR_l < \sum_{i=1}^{l} E_0(1, W_N^{(i)}), \tag{4.25}$$

where $R_l = \frac{\lambda_l}{l}$, and λ_l denotes the number of information bits in the first l bits of $\mathbf{v}$.

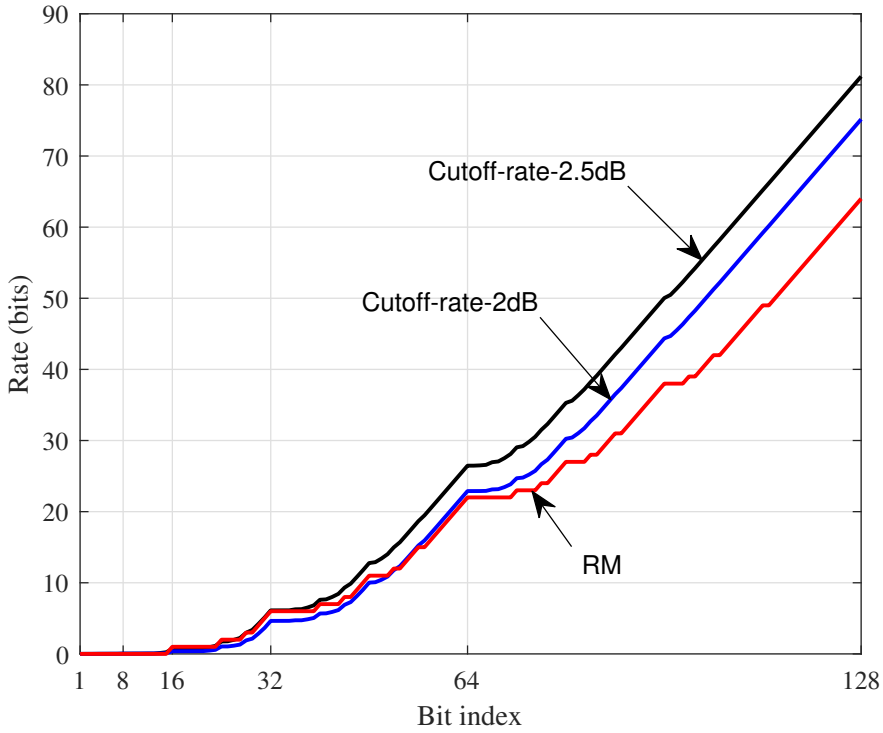

Figure 4.6: PAC$(128, 64)$ rate profiles.

As illustrated in Figure 4.6, a PAC$(128, 64)$ code utilizing the RM rate profile can have a constant complexity for SNR values greater than 2.5 dB, while from (4.25), by utilizing the polar rate profile a constant complexity may happen for all SNR ranges. Figure 4.7 supports this by illustrating the computational complexity of a PAC$(128, 64)$ code for an RM and polar rate profiles. This verifies our previous section's results about polarizing the computational cutoff rate.

Because CCs have memory, each bit of message may impact up to $b(m + 1)$

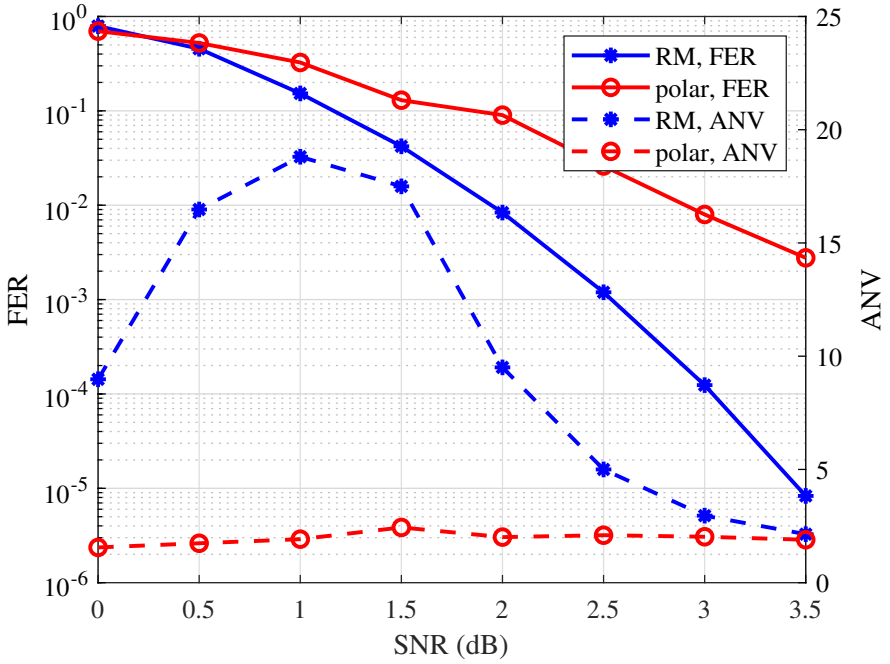

Figure 4.7: PAC$(128, 64)$ rate profiles.

bits of the output vector, and a bit error can cause a burst error of up to $b(m+1)$ length for a CC with a memory size m and a rate $1/b$ [16, p. 416]. Likewise, PAC codes may have burst errors up to length N owing to the presence of a polarized channel with a memory of N. In this connection, consider this scenario: If a first-bit error (FBE) occurs at index i, it is quite likely that the subsequent bits from $i+1$ to N would be decoded incorrectly as well.

In the case of a PAC$(256, 128)$ code, Figure 4.8 depicts the average proportion of the succeeding bits after the FBE that is decoded incorrectly. These statistics are based on 10^4 decoding failures of the PAC$(256, 128)$ code using polar and RM-Polar rate profiles. The polar rate profile is constructed by selecting the K highest reliable indices at a $E_b/N_0 = 2.5$ dB level. For the RM-Polar code construction, first, we select all the 93-row indices whose row has a weight of more than 32. After this, among the 70 rows of $F^{\otimes 8}$ with a weight of 16, the 35-row indices with the best reliabilities are chosen. Compared to the RM-polar construction described in Chapter 2, this construction results in a slightly better FER performance. Specifically, as seen in this figure, for all simulated SNR levels, almost over half of the succeeding bits of FBE are decoded incorrectly for both

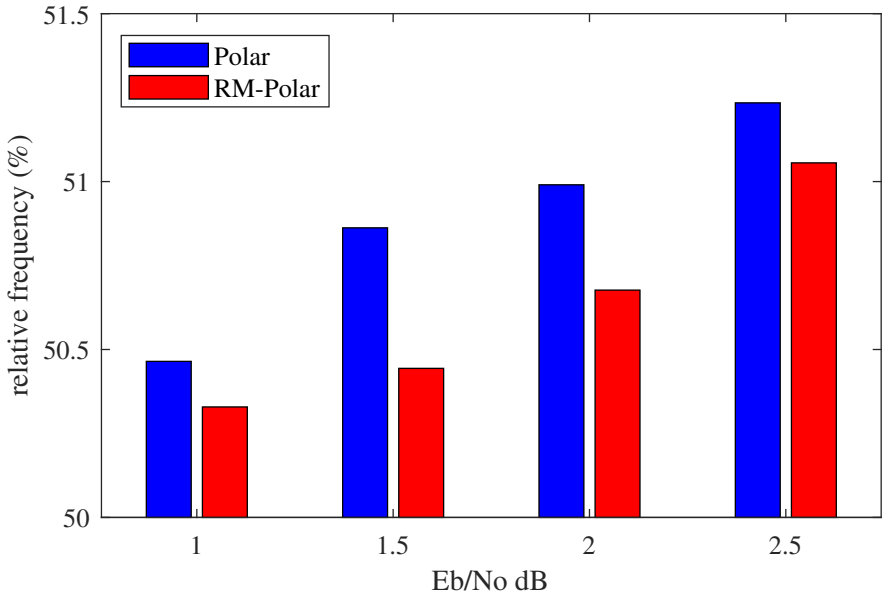

Figure 4.8: Error frequency of subsequent bits of FBE for PAC$(256, 128)$ code.

rate profiles, regardless of the SNR value.

Taking into account the constraint (4.25) and the findings of Figure 4.8, it is theoretically possible to create a rate profile $\mathcal{A}$ that results in reasonably excellent error-correction performance while also assuring low mean decoding complexity for SNR values that are higher than the target SNR value. In order to accomplish this, we suggest the following Monte-Carlo (MC) approach for a PAC code with a blocklength N, a code rate R, and a quantization level δ:

1. Select a desired SNR value that meets the condition
$$NR < \sum_{i=1}^{N} q(E_0(1, W_N^{(i)}), \delta).$$

2. Initiate a rate profile $\mathcal{A}$ such that index i for $i = 1, \cdots, N$ is included inside it if $q(E_0(1, W_N^{(i)}), \delta) = 1$.

3. Define an all-zero vector $\mathbf{h} = (h_1, \cdots, h_N)$ for the purpose of storing FBE index frequencies.

4. Conduct a Monte-Carlo simulation of a PAC$(N, |\mathcal{A}|, \mathcal{A}, T)$ code and, at the

completion of each iteration, pick j such that $j = \min(i) : v_i \neq \hat{v}_i$ and alter $\mathbf{h}$ by $h_j \leftarrow h_j + 1$.

5. Determine the index j of the largest element in $\mathbf{h}$ and delete it from $\mathcal{A}$.
6. If $|\mathcal{A}| > K$, continue to step 3.
7. Return $\mathcal{A}$.

We define a 1-bit quantization function $q(x, \delta)$ as

$$q(x, \delta) := \begin{cases} 1, & \text{if } x \geq \delta, \\ 0, & \text{otherwise,} \end{cases}$$

where $0 < \delta < 1$.

When the SNR value is larger than the desired SNR, step 1 (in accordance with (4.25)) guarantees a low mean computational complexity for the PAC sequential decoder. Aiming to improve error-correction performance, Step 4 is motivated by the findings of Figure 4.8 and tries to do so by avoiding assigning messages to those components of data carrier vector $\mathbf{v}$ that are most likely to be FBE (as a result, the frequency of burst error incidence is decreased).

Let us define $\mathcal{B}_i$ as the event of the occurrence of FBE at the ith bit:

$$\mathcal{B}_i = \{(\mathbf{u}, \mathbf{y}) : \hat{\mathbf{u}}^{i-1} = \mathbf{u}^{i-1}, \hat{u}_i \neq u_i\}, \tag{4.26}$$

and $\mathcal{C}_i$ as the event of the occurrence of an error at the ith bit:

$$\mathcal{C}_i = \{(\mathbf{u}, \mathbf{y}) : \hat{u}_i \neq u_i\}. \tag{4.27}$$

It is obvious that the $\mathcal{B}_i$ events for $i = 1, 2, \cdots, N$ are disjoint and the block error event $\mathcal{E}$ is the union of $\mathcal{B}_i$ events. Hence,

$$P(\mathcal{E}) = \sum_{i \in \mathcal{A}} P(\mathcal{B}_i). \tag{4.28}$$

Moreover, we can write [31]

$$P(\mathcal{E}) = \sum_{i \in \mathcal{A}} P(\mathcal{B}_i) \leq \sum_{i \in \mathcal{A}} P(\mathcal{C}_i) \leq \frac{1}{2} \sum_{i \in \mathcal{A}} Z(W_N^{(i)}), \tag{4.29}$$

where the first inequality is because $\mathcal{B}_i \subseteq \mathcal{C}_i$ for $i = 1, 2, \cdots, N$, and, as proved in [58], the last inequality is true for any arbitrary symmetric channel. A Gaussian approximation code construction [32] method looks for $Z(W_N^{(i)})$ values, whereas a density evolution code construction [31] method looks for $P(\mathcal{C}_i)$ values. As a consequence, the density evolution approach may provide slightly superior error correction performance. On the other hand, our suggested approach for constructing a code based on MC searches for $\mathcal{B}_i$ events whose probabilities yield the code's accurate error probability. This explains why our construction can outperform the density evolution code construction.

Simulation results

This subsection presents simulation results using the rate profiles obtained by our proposed technique for PAC$(256, 128)$ and PAC$(64, 32)$ codes over a BI-AWGN channel.

With $\delta = 0.5$ and $E_b/N_0 = 1.5, 2.5$ and 3 dB, we produced three alternative rate profiles for the PAC$(256, 128)$ code using our suggested construction approach. $|\mathcal{A}|$ is $144, 165$, and 176 at step 2 of the algorithm for the aforementioned SNR values; the method runs for $16, 37$, and 48 iterations in order to achieve the final rate profile for these three SNR values, and the resultant rate profiles are included in Table 4.2. The rate profiles are as binary vector forms of length N of the hexadecimal vector $\boldsymbol{\alpha}$, where in its binary format $\alpha_i = 1$ if $i \in \mathcal{A}$ and $\alpha_i = 0$, otherwise, for $1 \leq i \leq N$. This allows for a more compact representation of the rate profiles.

Table 4.2: Rate profiles.

(N, K)	E_b/N_0 (dB)	Rate profile $\boldsymbol{\alpha}$ (hexadecimal)
$(256, 128)$	1.5	00000000000001170001013F037F7FFF0001017F077F7FFF177F7FFF7FFFFFFF
	2.5	000000010001011F0001013F077FFFFF0001037F177F7FFF011F1FFF7FFFFFFF
	3	00000010001013F0001037F077FFFFF0001077F177F7FFF013F1FFF177F7FFF
$(64, 32)$	3	0001017F017F7FFF
	5	0007077F031F17FF

The acquired rate profiles (labeled with MC-) are compared to the rate profiles

of bit-channel cutoff rates at $E_b/N_0 = 1$ and 2 dB in Figure 4.9. As can be seen in this figure, owing to the first step of the algorithm, the produced rate profiles are always lower than the cutoff-rate profiles at the desired SNR values. However, the final rate profiles may be lower than the cutoff-rate profiles at an even lower SNR value, according to step 5, based on the values of N and K. For instance, when 48 indices are removed from a 3 dB cutoff-rate profile, the MC-3dB profile drops below the 2 dB cutoff-rate profile, indicating that the computational cutoff rate of the PAC decoder has been improved. Additionally, this figure depicts the RM-Polar rate profile built at $E_b/N_0 = 2.5$ dB, which coincides with the 1 dB cutoff rate profile.

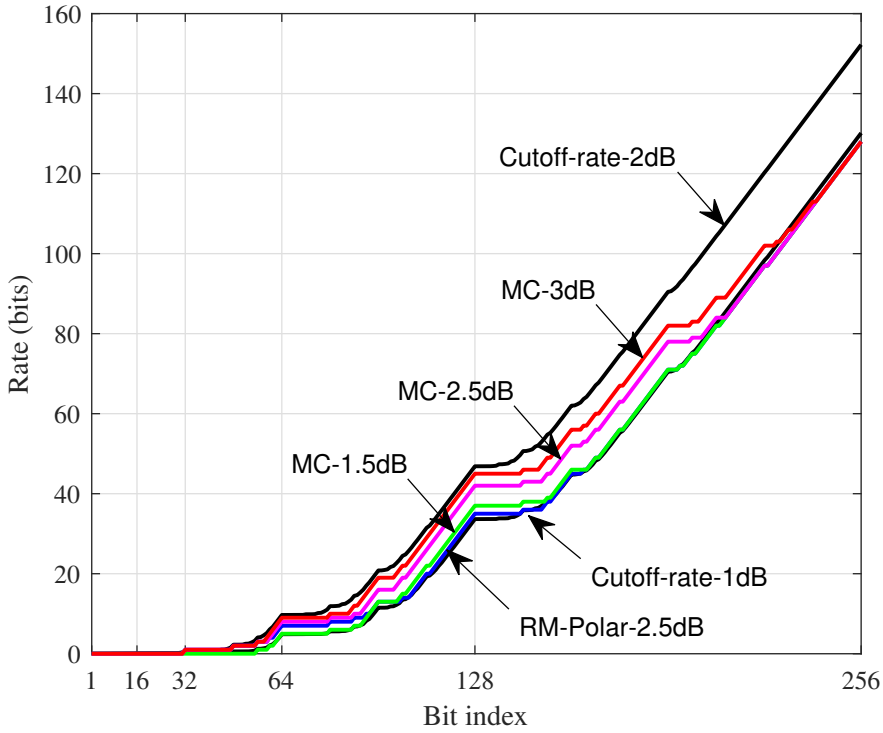

Figure 4.9: PAC$(256, 128)$ code rate profiles.

Figure 4.10 shows a comparison of the FER and ANV performances of PAC$(256, 128)$ codes generated at three different $E_b/N_0 = 1.5, 2.5$ and 3 dB values. The random-coding union (RCU) bound is also displayed on this graph [44]. It is demonstrated that the MC-1.5dB rate profile achieves a FER performance comparable to that of the RM-Polar rate profile. In addition, when the construction SNR advances, the number of potential indices in step 2 expands as well,

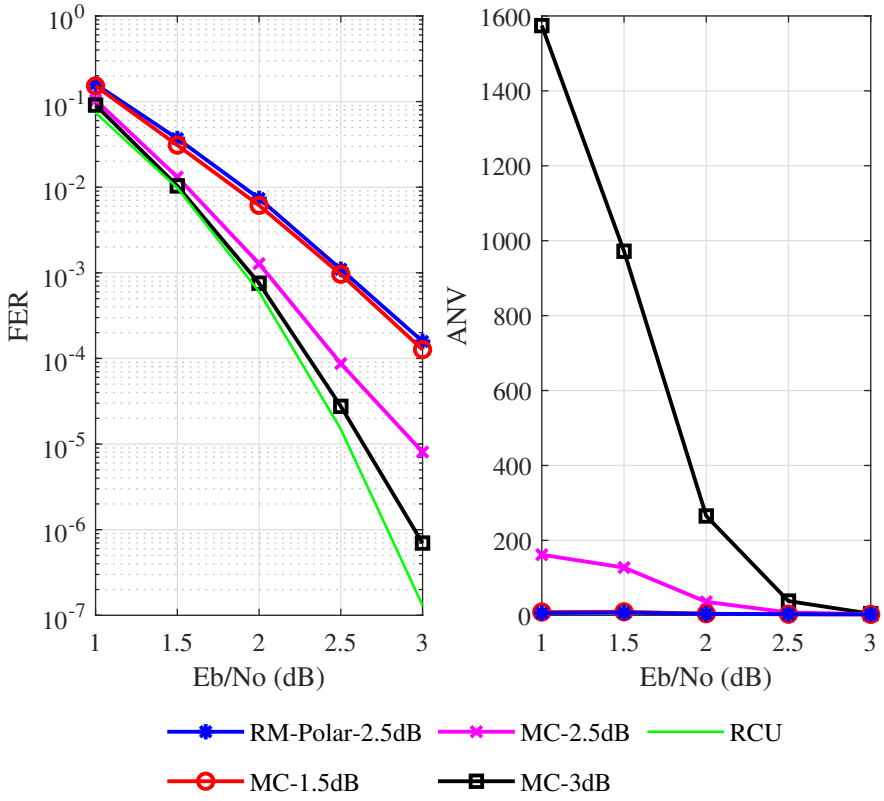

Figure 4.10: Error-correction performance and complexity of PAC$(256, 128)$ codes.

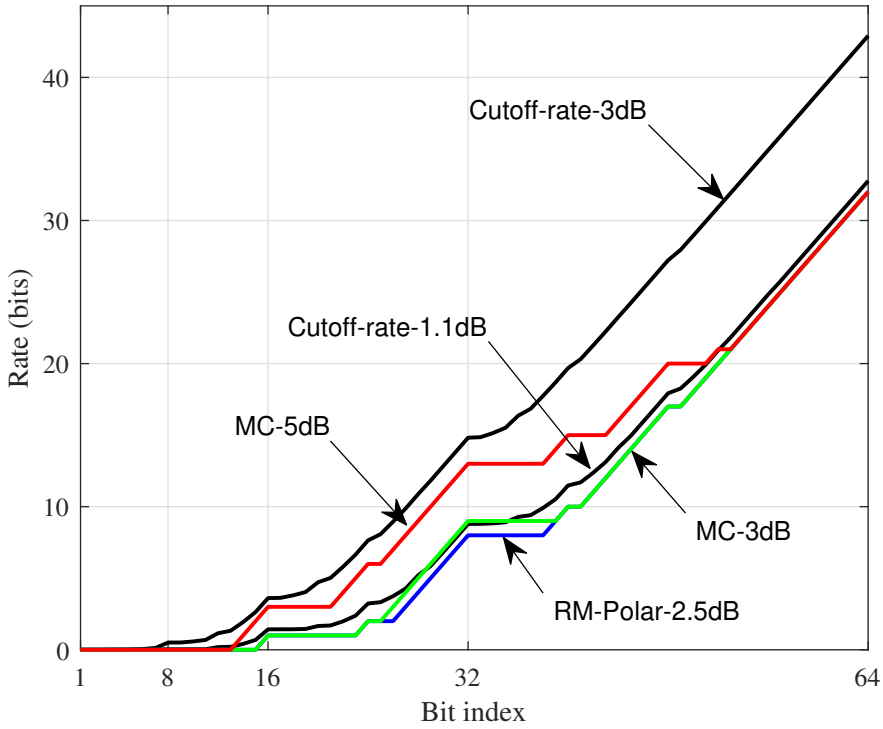

Figure 4.11: PAC$(64, 32)$ code rate profiles.

providing a wider choice of indices for step 4 to assess. As a result, raising the construction SNR of a PAC code helps to improve the code's error-correcting capability. This is why MC-2.5dB performs notably better than MC-1.5dB, and MC-3dB outperforms the other two. When compared to the RM-Polar rate profile, the MC-3dB rate profile has a gain of over 0.5 dB at FER $= 10^{-3}$.

Increasing the construction SNR, for understandable reasons, may cause the low mean computational decoding area to shift to higher SNR values. The ANV curves in Figure 4.10 provide an illustration of this impact. Due to the fact that RM-Polar and MC-1.5dB both sit on the cutoff-rate profile at 1 dB (Figure 4.9), they both lead to a low mean computational complexity over and above $E_b/N_0 = 1$ dB, which is consistent with the previous findings. MC-2.5dB and MC-3dB, on the other hand, are located below the cutoff-rate profile at 2 dB (Figure 4.9) and so have a low ANV for SNR values greater than $E_b/N_o = 2$ dB.

With $\delta = 0.5$ and $E_b/N_0 = 3$ and 5 dB, two alternative rate profiles for the PAC$(64, 32)$ code are generated by utilizing the MC construction approach. $|\mathcal{A}|$

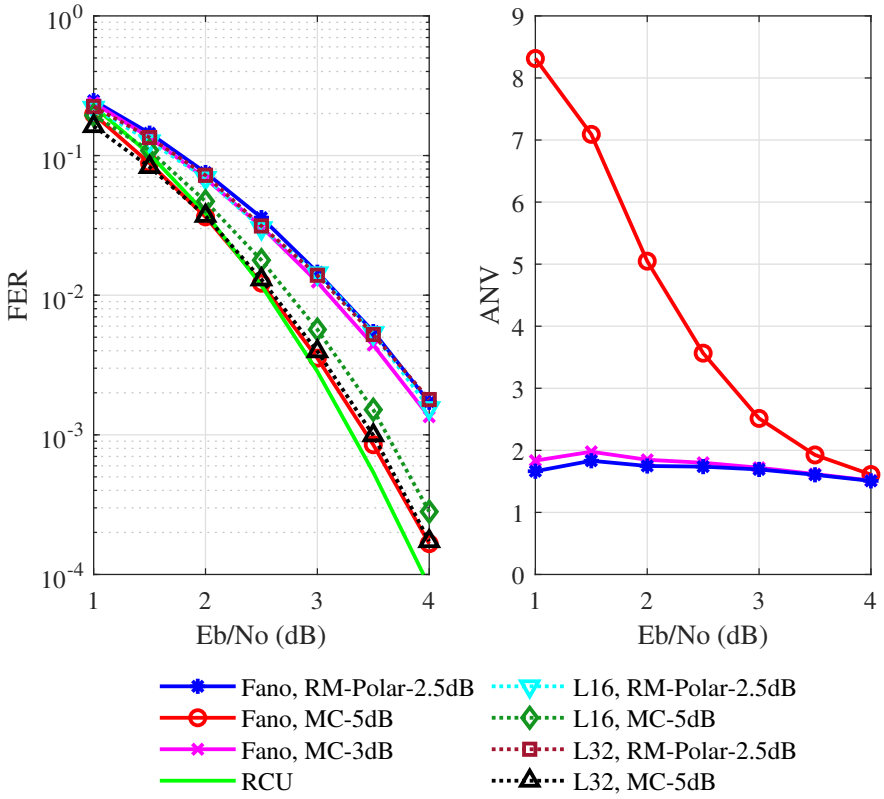

Figure 4.12: Error-correction performance and complexity of PAC$(64, 32)$ codes.

is 42 and 53 at step 2 of the algorithm, and to acquire the final rate profile, the process is repeated for 10 and 21 rounds, respectively. The resultant rate profiles are shown in Table 4.2. The produced rate profiles are shown and compared to the rate profiles of bit-channel cutoff rates at $E_b/N_0 = 1.1$ and 3 dB in Figure 4.11. Despite the fact that MC-5dB is created with $E_b/N_0 = 5$, the final rate profile is lower than the cutoff-rate profile at 3 dB due to the elimination of 21 indices from the original rate profile during the construction process. The RM-Polar and MC-3dB rate profiles placed just below the cutoff rate profile at $E_b/N_0 = 1.1$ dB.

A comparison of the FER and ANV performances of PAC$(64, 32)$ codes produced at $E_b/N_0 = 3$ and 5 dB is shown in Figure 4.12, and it is observed that the MC-3dB and the RM-Polar rate profiles have comparable FER and ANV performances. They both lead to a low mean computational complexity above $E_b/N_0 = 1.1$ dB as they both sit on the cutoff-rate profile at 1.1 dB (Figure 4.11). Compared with MC-3dB, the FER performance of MC-5dB is pretty close

to the RCU bound; however, this gain in FER performance comes at the expense of greater decoding complexity. In addition, the MC-5dB has a low ANV for SNR values greater than $E_b/N_o = 3$dB since its rate profile is located below the cutoff-rate profile at 3 dB.

It should be noted that the ANV values of the PAC$(64, 32)$ code are much lower when compared to PAC$(256, 128)$. Therefore, decoding PAC$(64, 32)$ code by using an SCL decoding with a small list size is achievable in this case. We utilize the algorithm proposed in [27] to implement the list decoder for PAC codes. According to this study, the FER performance of list decoding of PAC$(64, 32)$ code approaches that of Fano decoding of PAC$(64, 32)$ code for both the MC-5dB and RM-Polar rate profiles when the list size grows. When employing MC-5dB, the list decoding of PAC$(64, 32)$ code has roughly 0.5 dB gain at FER $= 10^{-3}$, which is similar to the coding gain of the Fano decoding algorithm.

Chapter 5

SCL and Stack Decoding for Polarized Channels

This chapter addresses an improvement to the SCL decoding for polarized channels, reducing the number of sorting operations involved without degrading the code's error-correction performance. We present an optimal metric function and prove that, on average, the correct path's partial path metrics should equal the bit-channel capacities partial summations. Likewise, we prove that the average bit metric values of the incorrect bits can be at most 0, implying that the partial path metric values of the wrong paths deviate from the bit-channel capacities partial summations. We take advantage of this by introducing an approach to reject the potential erroneous paths based on the departure of their bit metrics from the bit-channel mutual information. This approach avoids sorting on many branches while causing no frame error rate (FER) performance loss.

For the noiseless bit channels, we prove that the bit-channel metric value should be zero for the correct branch and $-\infty$ for the wrong branch. In this way, in the polarized channels, we would be able to identify incorrect branches and eliminate them from the list of metrics that should be sorted. One result indicates that when a list size of 4 is used, our suggested algorithm requires the decoder to do almost 92% fewer sorting operations than the conventional SCL algorithm while

decoding a $(1024, 512)$ polar code at a 3 dB SNR value.

Similar to the SCL decoding, we introduce an improvement to the stack algorithm that makes use of the bit-metric values and results in a stack size that is, on average, much smaller than the one used by the conventional stack algorithm. The results show that our proposed algorithm utilizes a much smaller stack size (over 90% is possible at a 3.5 dB SNR value) for a PAC$(128, 64)$ code while maintaining the same error-correction performance.

Finally, we prove that, for a threshold lower than the bit-channel cutoff rate, the probability of pruning the correct path from that bit of the decoding tree decreases exponentially by the given threshold, allowing us to suggest a technique for dynamically determining the threshold value. The dynamic threshold considerably reduces stack size (or, in the case of SCL decoding, the amount of sorting operations) for low SNR values.

5.1 Path Metric Function for SCL Decoder

In the SCL decoding, whenever the ith bit is an information bit, the decoding tree will consider both bifurcated paths and will choose L paths among all paths [9]. The decoder requires a partial path metric function, and it will determine the most likely path based on that function. For a given channel output $\mathbf{y}$, the optimal metric function in the terms of error probability is to choose $\hat{\mathbf{u}}^i$ as the sequence $\mathbf{u}^i$ for which the value of $P(\mathbf{u}^i|\mathbf{y})$ is maximized in the ith level of decoding (MAP rule). Note that the SC decoder has a likelihood value for frozen bits as well and that given a wrong path, the decoder may recommend a nonzero value for a frozen bit. This may assist the SCL decoder in determining that this path is one of the wrong paths, so the metric function should consider both the frozen and information bit likelihoods into account.

By using Bayes' rule, $P(\mathbf{u}^i|\mathbf{y})$ may be expressed as

$$P(\mathbf{u}^i|\mathbf{y}) = \frac{P(\mathbf{y}|\mathbf{u}^i)}{P(\mathbf{y})} P(\mathbf{u}^i). \tag{5.1}$$

Notice that the $P(\mathbf{u}^i)$ term is same for all the paths and can be avoided in calculating the path metrics. The fact that every monotonically growing function retains maximality means that an optimal metric also maximizes

$$\Phi(\mathbf{u}^i;\mathbf{y}) \triangleq \log_2 \frac{P(\mathbf{y}|\mathbf{u}^i)}{P(\mathbf{y})}, \tag{5.2}$$

where we call it partial path metric. Therefore, in the ith level of the decoding tree, the bit metric function can be obtained as

$$\begin{aligned}\phi(u_i;\mathbf{y},\mathbf{u}^{i-1}) &\triangleq \Phi(\mathbf{u}^i;\mathbf{y}) - \Phi(\mathbf{u}^{i-1};\mathbf{y}) \\ &= \log_2 \frac{P(\mathbf{y}|\mathbf{u}^i)}{P(\mathbf{y})} - \log_2 \frac{P(\mathbf{y}|\mathbf{u}^{i-1})}{P(\mathbf{y})} \\ &= \log_2 \frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}{P(\mathbf{y},\mathbf{u}^{i-1})}.\end{aligned} \tag{5.3}$$

For any given output $(\mathbf{y},\mathbf{u}^{i-1})$ of a bit-channel $W_N^{(i)}$, we have $\phi(u_i;\mathbf{y},\mathbf{u}^{i-1}) \geq \phi(\tilde{u}_i;\mathbf{y},\mathbf{u}^{i-1})$ if and only if $P(\mathbf{y},\mathbf{u}^{i-1}|u_i) \geq P(\mathbf{y},\mathbf{u}^{i-1}|\tilde{u}_i)$. That is, when dealing with a bit channel $W_N^{(i)}$, the bit metric ϕ is calculated according to the local ML rule as well.

The bit metric can be represented as

$$\begin{aligned}&\phi(u_i;\mathbf{y},\mathbf{u}^{i-1}) = \log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}{P(\mathbf{y},\mathbf{u}^{i-1})}\right) \\ &= \log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}{\frac{1}{2}\left[P(\mathbf{y},\mathbf{u}^{i-1}|u_i) + P(\mathbf{y},\mathbf{u}^{i-1}|u_i\oplus 1)\right]}\right) \\ &= 1 - \log_2\left(1 + \frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i\oplus 1)}{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}\right) \\ &= 1 - \log_2\left(1 + 2^{-\log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}{P(\mathbf{y},\mathbf{u}^{i-1}|u_i\oplus 1)}\right)}\right) \\ &= 1 - \log_2\left(1 + 2^{-L_i\cdot(-1)^{u_i}}\right)\end{aligned} \tag{5.4}$$

for a binary input channel with a uniform input distribution, where

$$L_i = \log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i=0)}{P(\mathbf{y},\mathbf{u}^{i-1}|u_i=1)}\right). \tag{5.5}$$

Therefore, for a 0 branch or a frozen bit, the bit metric would be $1 - \log_2\left(1 + 2^{-L_i}\right)$ and for a 1 branch the bit metric is $1 - \log_2\left(1 + 2^{L_i}\right)$.

Note that whenever $x \geq 0$, 1 in $\log_2(1 + 2^x)$ is negligible and it can be approximated by x, and whenever $x < 0$, 2^x term can be neglected and $\log_2(1 + 2^x) \approx \log_2(1) = 0$. Therefore, for a 0 branch or a frozen bit, the corresponding bit metric can be approximated as

$$1 - (-L_i) \cdot \mathbb{1}_{\{-L_i > 0\}}, \tag{5.6}$$

where $\mathbb{1}$ is the indicator function (an event's indicator function is a random variable that equals 1 when the event occurs and 0 when the event does not). For a 1 branch the corresponding metric can be approximated as

$$1 - (L_i) \cdot \mathbb{1}_{\{L_i > 0\}}. \tag{5.7}$$

Rather of maximizing the path metric

$$\Phi(\mathbf{u}^i; \mathbf{y}) = \sum_{j=1}^{i} \phi(u_j; \mathbf{y}, \mathbf{u}^{j-1}), \tag{5.8}$$

the path metric

$$\sum_{j=1}^{i} \log_2\left(1 + 2^{-L_j \cdot (-1)^{u_j}}\right) \tag{5.9}$$

is minimized in the literature [26].

In a decoding algorithm, the metric function determines which direction through the decoding tree should be chosen. As a matter of intuitive understanding, the bit-metric function value must always grow for every u_i bit on the correct path in order to detect the correct data vector $\mathbf{u}$. It is impossible to place a measure on this point of view for a certain code. Studying an ensemble of codes with the specified channel and blocklength may make the problem easier to understand.

The following discusses and examines the metric function of SCL decoding, similar to how the metric function of sequential decoding has been addressed

before. In the case of the ensemble of bit channels with output-input $(\mathbf{y}, \mathbf{u}^{i-1}; u_i)$ pair, the expectation of (5.3) is

$$\begin{aligned}
&\mathbb{E}_{U_i,(\mathbf{Y},\mathbf{U}^{i-1})}\left[\phi(U_i;\mathbf{Y},\mathbf{U}^{i-1})\right] \\
&= \sum_{u_i} q(u_i) \sum_{(\mathbf{y},\mathbf{u}^{i-1})} P(\mathbf{y},\mathbf{u}^{i-1}|u_i)\phi(u_i;\mathbf{y},\mathbf{u}^{i-1}) \\
&= \sum_{u_i} \sum_{(\mathbf{y},\mathbf{u}^{i-1})} q(u_i) P(\mathbf{y},\mathbf{u}^{i-1}|u_i) \left[\log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}{P(\mathbf{y},\mathbf{u}^{i-1})}\right)\right] \\
&= I(W_N^{(i)}),
\end{aligned} \tag{5.10}$$

where $I(W_N^{(i)})$ denotes the symmetric capacity of the bit-channel $W_N^{(i)}$. As a consequence, the heuristic result is that for the bit channels the average metric increases all the time for the correct branches.

Suppose now that $\tilde{u}_i$ is a wrong bit with a metric of $\phi(\tilde{u}_i;\mathbf{y},\mathbf{u}^{i-1})$ at level i. By averaging this bit-metric value over the correct and incorrect branches, we derive

$$\begin{aligned}
&\mathbb{E}_{U_i,\tilde{U}_i,(\mathbf{Y},\mathbf{U}^{i-1})}\left[\phi(\tilde{U}_i;\mathbf{Y},\mathbf{U}^{i-1})\right] \\
&= \sum_{\tilde{u}_i} q(\tilde{u}_i) \sum_{u_i} q(u_i) \sum_{(\mathbf{y},\mathbf{u}^{i-1})} P(\mathbf{y},\mathbf{u}^{i-1}|u_i)\phi(\tilde{u}_i;\mathbf{y},\mathbf{u}^{i-1}) \\
&= \sum_{\tilde{u}_i} q(\tilde{u}_i) \sum_{(\mathbf{y},\mathbf{u}^{i-1})} P(\mathbf{y},\mathbf{u}^{i-1})\phi(\tilde{u}_i;\mathbf{y},\mathbf{u}^{i-1}) \\
&= \sum_{\tilde{u}_i} \sum_{(\mathbf{y},\mathbf{u}^{i-1})} q(\tilde{u}_i) P(\mathbf{y},\mathbf{u}^{i-1}) \left[\log_2\left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|\tilde{u}_i)}{P(\mathbf{y},\mathbf{u}^{i-1})}\right)\right] \\
&\leq \sum_{\tilde{u}_i} \sum_{(\mathbf{y},\mathbf{u}^{i-1})} q(\tilde{u}_i) P(\mathbf{y},\mathbf{u}^{i-1}) \left[\frac{P(\mathbf{y},\mathbf{u}^{i-1}|\tilde{u}_i)}{P(\mathbf{y},\mathbf{u}^{i-1})} - 1\right] \\
&= 0.
\end{aligned} \tag{5.11}$$

It should be noted that we used the $\log_2(x) \leq x - 1$ inequality to obtain (5.11). The expectation stated above means that, on average, the bit metric is a negative or 0 value for each wrong branch. When the decoder travels in the wrong direction for numerous branches, as a result of not raising the path metric value, the decoder detects this and may possibly eliminate the wrong path from the list of possible paths.

The above discussion is about the behaviour of the bit-channel metric on average. Polarization attempts to obtain K noiseless bit channels out of all N bit channels. Lets assume that we achieve to this goal and the ith bit channel is noiseless. In this case, if we have transmitted u_i bit and $\tilde{u}_i = u_i \oplus 1$, we have

$$\begin{aligned}\phi(u_i; \mathbf{y}, \mathbf{u}^{i-1}) &= \log_2 \frac{P(\mathbf{y}, \mathbf{u}^{i-1}|u_i)}{P(\mathbf{y}, \mathbf{u}^{i-1})} \\ &= \log_2 \frac{P(\mathbf{y}, \mathbf{u}^{i-1}|u_i)}{\frac{1}{2}P(\mathbf{y}, \mathbf{u}^{i-1}|u_i) + \frac{1}{2}P(\mathbf{y}, \mathbf{u}^{i-1}|\tilde{u}_i)} \\ &= \log_2 \frac{1}{\frac{1}{2} + 0} = 1.\end{aligned} \tag{5.12}$$

This is in consist with (5.10) as for the noiseless bit channel, its bit-channel mutual information is 1. Also for the ith wrong branch, we have its bit-metric function as

$$\begin{aligned}\phi(\tilde{u}_i; \mathbf{y}, \mathbf{u}^{i-1}) &= \log_2 \frac{P(\mathbf{y}, \mathbf{u}^{i-1}|\tilde{u}_i)}{P(\mathbf{y}, \mathbf{u}^{i-1})} \\ &= \log_2 \frac{0}{0 + \frac{1}{2}} = -\infty.\end{aligned} \tag{5.13}$$

As a result of this finding, any pathways that include the wrong branch of a noiseless bit channel can never belong to a correct path. Also, from (5.11) we know that the bit metric of the wrong branch should be a negative number on average. Based on this discussion, if the branch metric is a big negative number, the branch is the wrong one, and in the SCL decoding, there is no need to consider this branch in the L survivor branches, and we can consider this bit channel as a noiseless bit channel. As explained, ideally (when the noise is small and the blocklength is large enough), the bit metric of the wrong branch should be $-\infty$ (the only way to get a noiseless bit channel is if the code block length is increased to infinity). It would be interesting to see whether we can take advantage of this in the case of short to moderate block length codes. We will put this to the test using both polar and PAC codes using SCL or stack decoding algorithms.

5.2 Improving SCL Decoding for Polarized Channels

The primary downside of SCL decoding is that, given a list size of L, it is necessary to discover the L biggest metric values out of $2L$ paths in each branching level of the decoding tree. There are many algorithms for determining the L biggest path metric values, and this stage of the SCL decoding process is often referred to as sorting [9]. This section proposes an improvement to the SCL decoding algorithm (PSCL) that omits the sorting step for many of the decoding tree's branching levels.

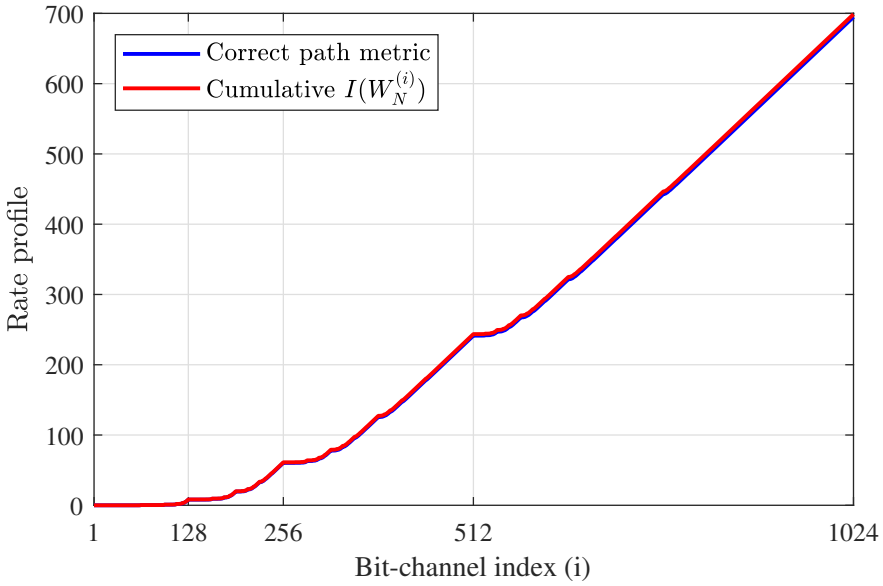

Figure 5.1: Comparison of the partial path metric corresponding to the correctly decoded codewords of list decoding of $(1024, 512)$ polar code with the capacity rate profile over BI-AWGN at 2.5 dB SNR.

As proved in the preceding section, the bit metric for the correct branch at the ith level of the decoding tree is $I(W_N^{(i)})$ on average, whereas the bit metric for an erroneous branch at this level on average would be less than 0. Figure 5.1 compares the average metric profiles for the correctly decoded codewords (10^4 codewords) to the capacity rate profile of a $(1024, 512)$ polar code over a BI-AWGN with an SNR of 2.5 dB. As this figure shows, for the almost noiseless bit channels (mostly the bit-channels that are at the end), the slope is 1

($I(W_N^{(i)}) \approx 1$). From this figure, it can be deduced that a wrong branch may be recognized from the correct branch when traversing through the decoding tree. To accomplish this, we propose modifying the SCL decoding algorithm as follows:

1. At a branching level i, out of $2L$ paths, discard paths having a bit-metric $\phi(u_i; \mathbf{y}, \mathbf{u}^{i-1})$ less than the threshold m_T, where $m_T < I(W_N^{(i)})$.
2. If the number of remaining paths exceeds the list size L, sort them and choose the L ones with the highest metric values; otherwise, no sorting is required.
3. Declare a decoding failure and end the decoding procedure if all $2L$ paths are rejected.
4. If $i = N$, select the path with the highest metric value.

Note that the sorting of a typical SCL decoding technique is a function of $2L$, whereas the number of elements that may be sorted in our proposed algorithm can be much less than $2L$.

In Figure 5.2, the FER performance of our proposed algorithm with the bit-metric threshold $m_T = -5$ for a $(1024, 512)$ polar code is compared to the performance of a conventional SCL decoding technique both with a list size of $L = 4$. As this figure demonstrates, there is no degradation in the FER performance.

Table 5.1: Average number of executed sorting for decoding $(1024, 512)$ polar code.

SNR [dB]	0.0	0.5	1.0	1.5	2.0	2.5	3.0
# of sorting ($L = 4$, $m_T = -5$)	132.23	116.11	89.23	58.49	37.18	37.22	40.63

The corresponding number of sorting operations performed in our proposed algorithm is listed in Table 5.1. Note that the conventional SCL decoding technique requires 510 sorting operations to decode a $(1024, 512)$ polar code with a $L = 4$ list size. When the SNR is 3 dB, our suggested decoding technique reduces the number of sorting operations to about 40 without affecting the FER performance.

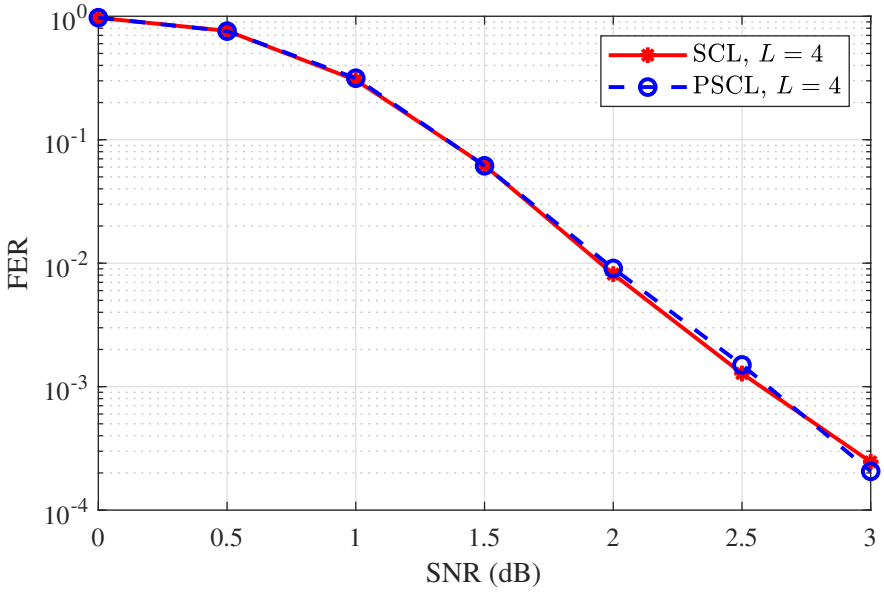

Figure 5.2: Comparison of the conventional SCL decoding with our proposed SCL decoding algorithm for a list size of $L = 4$ of a $(1024, 512)$ polar code with $m_T = -5$.

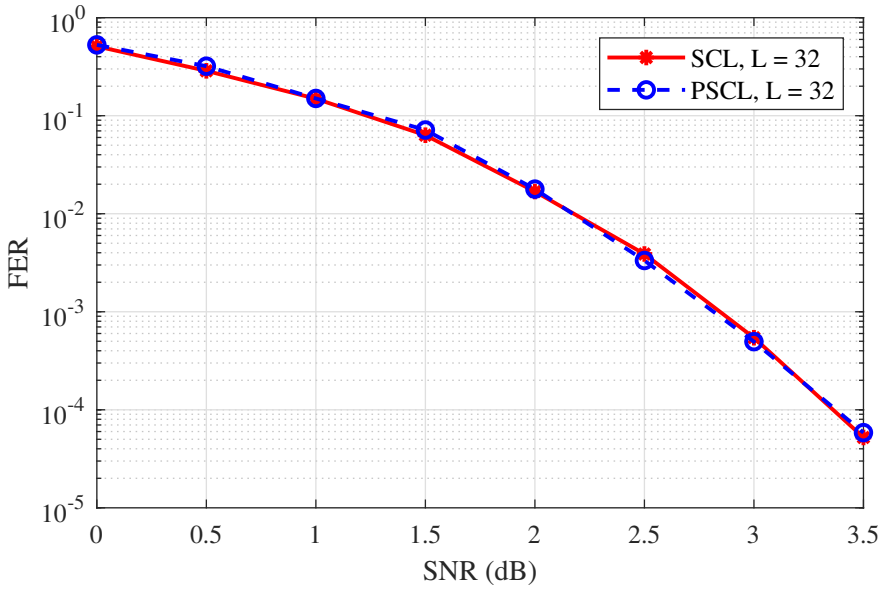

Figure 5.3: Comparison of the conventional list decoding with our proposed list decoding algorithm of a PAC$(128, 64)$ code when $m_T = -10$.

Also, for a list size of $L = 32$, the FER performance of our proposed method for a PAC$(128, 64)$ code is compared to the performance of a conventional SCL decoding technique in Figure 5.3. As this figure illustrates, no loss in FER performance occurs.

Table 5.2: Average number of executed sorting for decoding PAC$(128, 64)$ code.

SNR [dB]	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
# of sorting ($L = 32$, $m_T = -10$)	37.96	36.95	35.93	35.15	34.24	33.29	31.83	28.14

Table 5.2 lists the amount of sorting operations conducted by our proposed technique corresponding to the plot of Figure 5.3. The conventional SCL decoding technique requires 59 sorting operations to decode a PAC$(128, 64)$ code with a $L = 32$ list size. As an instance, when the SNR is 3.5 dB, our proposed decoding approach decreases the number of sorting operations to about 28 without compromising the FER performance.

In the following, we compare our proposed technique to two similar works. A pruning technique is suggested in [59] which is applied after the sorting step in the SCL algorithm and is based on the metric value of the best path. Finding the best path at each level i is costly, and our proposed algorithm avoids this by comparing the metric to the channel capacity profile, which is a priori information to the decoder. Similarly, in [60] the pruning step is also after the sorting step of SCL decoding. A stack containing the information of the L best paths (among those paths that are pruned, the paths in the stack have the highest metric values) that are already pruned is updated at every level of the decoding. At each level i, a path will survive if its metric is better than the metric of the stack top; otherwise, the path will be pruned. If the path is pruned and its metric is better than the metric of the worst path in the stack, the path will be inserted into the stack, and that worst path will be deleted out from the stack. If the decoder is at the ith level and a path is pruned at level j ($j < i$) with metric value $\tilde{\Gamma}_j$, in [60] it is proposed to add $\tilde{\Gamma}_j$ with an upper bound on the metric to be able to use it at the level i. We suggest updating the metric value of this pruned path as $\tilde{\Gamma}_j + \sum_{k=j+1}^{i} I(W_N^{(i)})$ which gives a better estimate of this path if it was not pruned and would be decoded correctly up to the ith level (stack contains L paths with such these updated metrics). The interesting thing about using this stack is that

in [60], they are trying to compare paths with different lengths. In a sequential decoding algorithm, the optimal way to compare the paths with different lengths is using a bias, and in [14] it is proved that the optimal bias should be the bit-channel capacities. Sorting $2L$ paths in the SCL decoding process and sorting and updating the stack are costly operations. The primary purpose of our approach is to do as few sorting operations as possible. They can eliminate the sorting step in these works by comparing the pathways to the bit-channel capacity profile (instead of the best path).

Probability of error

We make the assumption that the list size is infinite and that the failure to get correct decoding is solely due to the pruning strategy that we are proposing (this is the case for the stack decoding, which is explained in the next section). We fail to successfully decode a received vector if the bit metric of the correct path is less than the threshold m_T at the ith level for i from 1 to N. Responding to this issue is equal to determining an upper bound for

$$P\left\{\phi(u_i; \mathbf{y}, \mathbf{u}^{i-1}) \leq m_T\right\}, \tag{5.14}$$

where $m_T < I(W_N^{(i)})$ (in our simulations we use $m_T < 0$). To accomplish this, we use the Chernoff bound:

Chernoff bound. For a random variable X, if $m < \mathbb{E}[X]$, we have

$$P\{X \leq m\} \leq 2^{-sm} g(s); \qquad s < 0, \tag{5.15}$$

where $g(s)$ is the moment-generating function (MGF) of X. ■

Theorem 2. *For the bit-metric function $\phi(u_i; \boldsymbol{y}, \boldsymbol{u}^{i-1})$ and constant threshold m_T we have*

$$P\left\{\phi(u_i; \boldsymbol{y}, \boldsymbol{u}^{i-1}) \leq m_T\right\} \leq 2^{m_T - E_0(1, W_N^{(i)})}, \tag{5.16}$$

where $m_T < I(W_N^{(i)})$.

Proof. We found the mean of the random variable $\phi(U_i) \triangleq \phi(U_i; \mathbf{Y}, \mathbf{U}^{i-1})$ as $I(W_N^{(i)})$, and in the following, we derive an upper bound for its MGF when $-1 < s < 0$.

$$\begin{aligned} g(s) &\triangleq \mathbb{E}[2^{s\phi(U_i)}] = \mathbb{E}\left[2^{s\log_2\left(\frac{P(\mathbf{Y},\mathbf{U}^{i-1}|U_i)}{P(\mathbf{Y},\mathbf{U}^{i-1})}\right)}\right] \\ &= \mathbb{E}\left[\left(\frac{P(\mathbf{Y},\mathbf{U}^{i-1}|U_i)}{P(\mathbf{Y},\mathbf{U}^{i-1})}\right)^s\right] \\ &= \sum_{u_i} q(u_i) \sum_{(\mathbf{y},\mathbf{u}^{i-1})} P(\mathbf{y},\mathbf{u}^{i-1}|u_i) \left(\frac{P(\mathbf{y},\mathbf{u}^{i-1}|u_i)}{P(\mathbf{y},\mathbf{u}^{i-1})}\right)^s \\ &= \sum_{(\mathbf{y},\mathbf{u}^{i-1})} \underbrace{P(\mathbf{y},\mathbf{u}^{i-1})^{-s}}_{a} \underbrace{\sum_{u_i} q(u_i) P(\mathbf{y},\mathbf{u}^{i-1}|u_i)^{1+s}}_{b}. \end{aligned} \tag{5.17}$$

By defining $r = -s$ and considering $-1 < s < 0$ and using the Hölder's inequality (H) inequality:

$$\sum ab \le \left(\sum a^{\frac{1}{r}}\right)^r \left(\sum b^{\frac{1}{1-r}}\right)^{1-r}, \tag{5.18}$$

we have

$$\begin{aligned} g(s) &\overset{\text{H}}{\le} \left[\underbrace{\sum_{(\mathbf{y},\mathbf{u}^{i-1})} P(\mathbf{y},\mathbf{u}^{i-1})}_{=1}\right]^r \\ &\qquad \left[\sum_{(\mathbf{y},\mathbf{u}^{i-1})} \left[\sum_{u_i} q(u_i) P(\mathbf{y},\mathbf{u}^{i-1}|u_i)^{1-r}\right]^{\frac{1}{1-r}}\right]^{1-r} \\ &= 2^{(1-r)\log_2\left[\sum_{(\mathbf{y},\mathbf{u}^{i-1})}\left[\sum_{u_i} q(u_i) P(\mathbf{y},\mathbf{u}^{i-1}|u_i)^{1-r}\right]^{\frac{1}{1-r}}\right]} \\ &= 2^{-(1-r)E_0(\frac{r}{1-r}, W_N^{(i)})} = 2^{-(1+s)E_0(\frac{-s}{1+s}, W_N^{(i)})}. \end{aligned} \tag{5.19}$$

Therefore (to simplify), for $s = -1/2$ we have

$$\begin{aligned} P\{\phi(U_i) \le m_T\} &\le 2^{-sm_T} g(s) \\ &\le 2^{-sm_T} 2^{-(1+s)E_0(\frac{-s}{1+s}, W_N^{(i)})} \\ &= 2^{\frac{m_T - E_0(1, W_N^{(i)})}{2}}. \end{aligned} \tag{5.20}$$

This says that if the threshold is less than the bit-channel cutoff rate, the probability of pruning the correct path at the ith level of decoding goes exponentially to zero by m_T. □

Based on the obtained upper bound, in the next section, we propose a dynamic threshold for the pruning technique, and we will see that the technique extremely reduces (up to 90%) the required stack size for decoding on polarized channels.

5.3 Improving Stack (Heap) Decoding for Polarized Channels

According to [14], the ith bit-channel metric function (3.4) when using bit-channel cutoff rates as the bias values results in a sequential decoding with a low computational complexity, where u_i is the ith transmitted bit. Similar to what we described in the previous section for SCL decoding, also in the stack (heap) algorithm, the bit metric of an incorrect branch in a noiseless bit channel converges to $-\infty$ (channel polarization is used to locate noiseless bit channels). As a result of this finding, any pathways that include the wrong branch of a noiseless bit channel would never reach the top of the stack. Ideally, in the noiseless bit channel, there is no need to insert the path of the wrong branch into the stack at all, and the stack size may therefore be kept very small. The only way to get a noiseless bit channel is to increase the code block length to infinity. It would be interesting to see whether we can take advantage of this in the case of short block length codes. We will put this to the test using a PAC$(128, 64)$ stack decoding. Assume that the bit-metric value of less than $m_T = -20$ (instead of $-\infty$) corresponds to the bit-metric value of a wrong branch. For the bit channels with a bit-metric value smaller than -20, Table 5.3 displays the levels of the decoding tree, the associated bit-channel cutoff rates, and the bit-channel metric values of both corresponding branches in a single decoding trial at 2.5 dB SNR. This table has 54 rows, indicating that the stack size would not need to expand in this 54 times compared to the conventional stack algorithm.

Table 5.3: Levels, reliabilities, and bit-metric values of both branches (up and down branches) of noiseless bit channels at 2.5 dB SNR.

i	$E_0(1, W_N^{(i)})$	up	down
32	0.9978	0.00	-38.96
46	0.9367	0.06	-22.94
47	0.9612	-25.00	0.04
48	0.9997	-55.91	0.00
52	0.9531	0.05	-23.89
54	0.9786	0.02	-33.29
55	0.9876	0.01	-35.21
56	1.0000	0.00	-73.56
58	0.9921	-32.25	0.01
59	0.9954	-37.69	0.00
60	1.0000	0.00	-77.40
61	0.9975	-41.48	0.00
62	1.0000	0.00	-91.17
63	1.0000	0.00	-Inf
64	1.0000	-191.64	0.00
78	0.9904	0.01	-22.10
79	0.9946	0.01	-23.38
80	1.0000	0.00	-59.10
86	0.9977	-34.53	0.00
87	0.9988	0.00	-32.41
88	1.0000	-79.83	0.00
90	0.9992	-27.72	0.00
91	0.9995	0.00	-38.39
92	1.0000	0.00	-79.27
93	0.9998	0.00	-41.88
94	1.0000	-87.67	0.00
95	1.0000	-Inf	0.00
96	1.0000	0.00	-209.46
100	0.9987	-25.97	0.00
102	0.9995	-47.14	0.00
103	0.9997	-46.93	0.00
104	1.0000	-103.52	0.00
106	0.9998	-36.09	0.00
107	0.9999	0.00	-28.38
108	1.0000	0.00	-81.81
109	1.0000	-Inf	0.00
110	1.0000	-Inf	0.00
111	1.0000	0.00	-Inf
112	1.0000	0.00	-249.96
114	1.0000	-50.76	0.00
115	1.0000	-45.28	0.00
116	1.0000	-106.58	0.00
117	1.0000	0.00	-Inf
118	1.0000	-Inf	0.00
119	1.0000	0.00	-Inf
120	1.0000	0.00	-283.56
121	0.9999	0.00	-51.19
122	1.0000	0.00	-Inf
123	1.0000	0.00	-Inf
124	1.0000	-Inf	0.00
125	1.0000	0.00	-Inf
126	1.0000	-Inf	0.00
127	1.0000	-Inf	0.00
128	1.0000	0.00	-641.35

Figure 5.4 compares the FER performance of our proposed stack algorithm (Pstack) as we do not insert the paths containing the wrong branches (a branch with the bit-metric value less than $m_T = -20$) with a conventional stack (heap) algorithm for a PAC$(128, 64)$ code. As this figure shows, there is no loss in the error-correction performance. This figure also compares the corresponding average number of stack sizes. Our proposed stack algorithm needs a much smaller stack size compared to the conventional stack algorithm. The average stack size is reduced from 67.04 to 6.55 at 3.5 dB SNR.

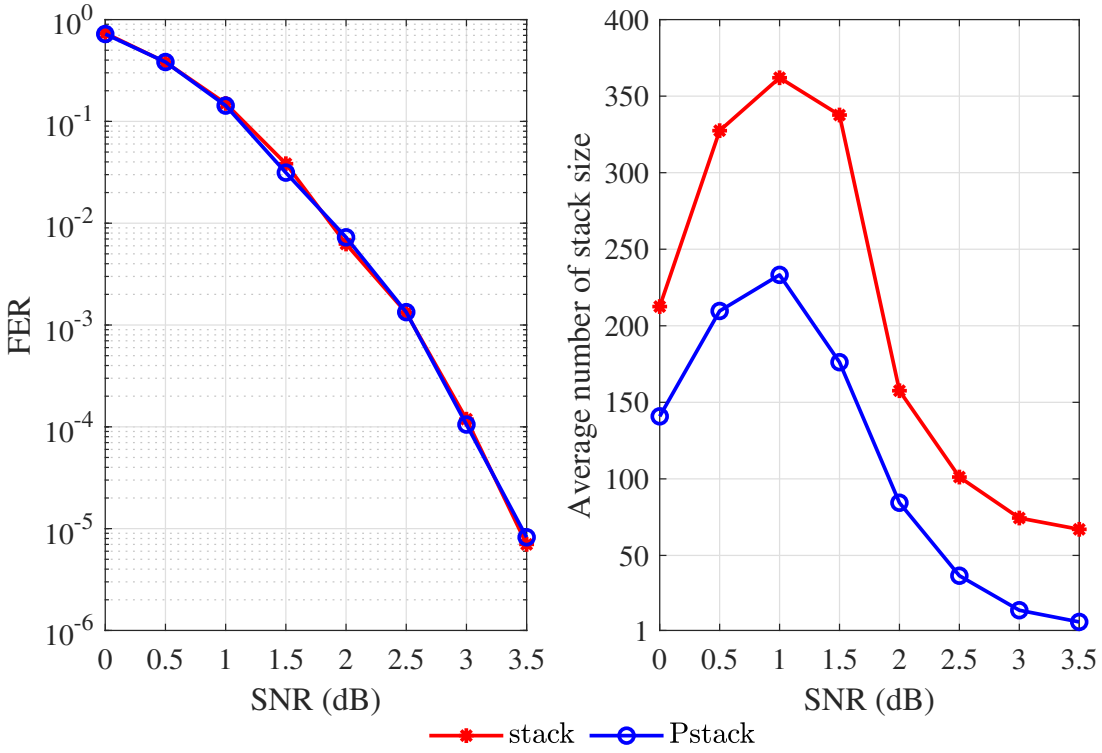

Figure 5.4: FER and average number of stack size performance comparison of PAC$(128, 64)$ codes with $m_T = -20$.

5.3.1 Dynamic Threshold

In the preceding section, we proved that the probability of error due to pruning decreases exponentially with m_T. If the FER performance at 3.5 dB is about 10^{-6} in the stack decoding with no pruning technique, it would be reasonable to use $m_T = -23$ ($2^{-23} \approx 10^{-7}$) at this SNR value (to make the performance loss due

to the pruning negligible). In this manner, we may choose an appropriate value for m_T for all other desired SNR values. The dispersion approximation yields a theoretical best attainable performance for every finite block-length code [44]. If the FER performance achieved by the dispersion approximation for a particular code is D at a specified SNR value, we use

$$m_T = \lfloor \log_2(D/10) \rfloor \tag{5.21}$$

as the dynamic bit-metric threshold for that SNR value. In this way, for a $(128, 64)$ code and SNR vector of $(0, 0.5, \cdots, 3.5)$, the corresponding threshold values are as

$$\mathbf{m}_T = (-5, -6, -7, -9, -11, -14, -18, -23).$$

Figure 5.5 compares the FER performance and average stack size of the conventional stack algorithm with our proposed algorithm when employing this dynamic threshold settings (PstackD). When compared to the Pstack approach, the PstackD technique results in a further reduction in the stack size for low SNR levels. For example, at a 1 dB SNR value, the average stack size of the conventional stack algorithm is 364, the average stack size of the Pstack approach is 233, and the average stack size of PstackD is around 134.

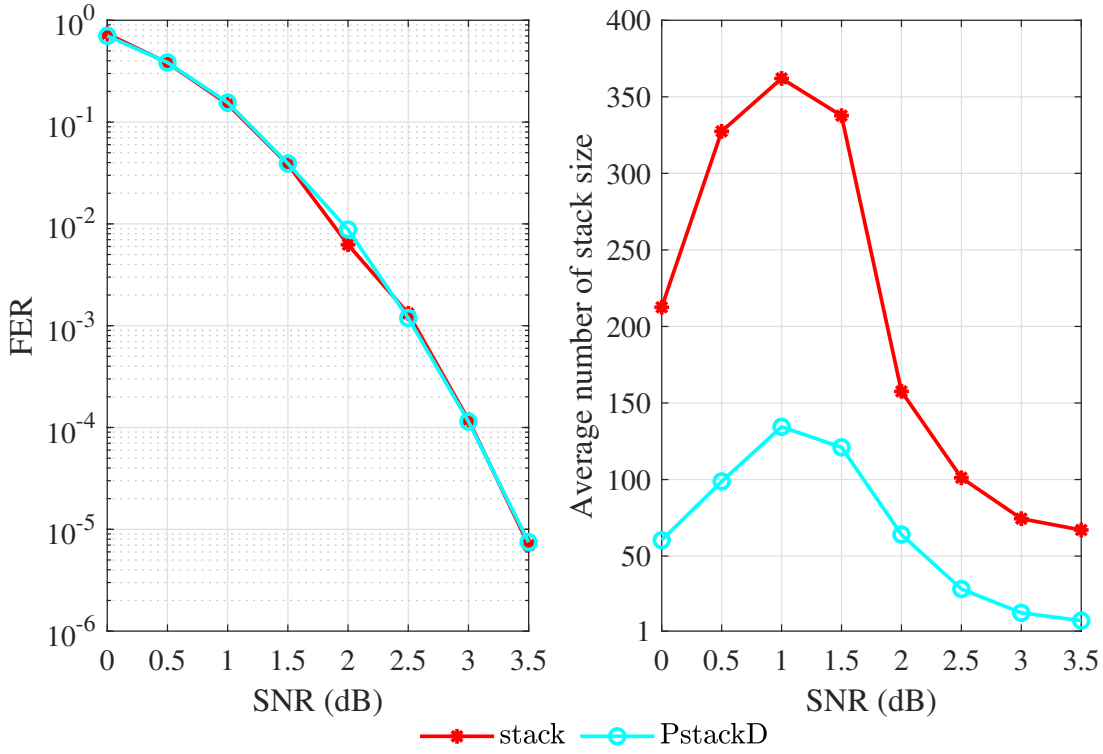

Figure 5.5: FER and average number of stack size performance comparison of PAC$(128, 64)$ codes with dynamic threshold.

Chapter 6

Weight Distribution of PAC Codes

This chapter investigates the weight distribution of PAC codes. By benefiting from cyclic shift matrices, we give a new proof to a result of [23] which says that $d_{\min}$ for PAC codes is greater than or equal to the $d_{\min}$ for polar codes. The usage of cyclic shifts in our work is motivated by the work of [61], which designs and analyzes a particular permutation set of polar codes relying on a $N/4$-cyclic shift for practical applications [24]. In our study, we generalize this algebraic result to the m-cyclic shift for $1 \leq m \leq N$, offer an explicit proof, and demonstrate how the findings may be applied to be used in the PAC codes. In [23], they also proved that the sum of $\mathbf{g}_i$ (ith row of $F^{\otimes n}$ for $1 \leq i < N$) with some rows below it (we represent this by $\underline{\mathbf{g}}_i$) has a weight greater than or equal to the weight of $\mathbf{g}_i$. We generalize this and prove that the summation of an odd number of clockwise cyclic shifts of $\underline{\mathbf{g}}_i$ has also a weight greater than or equal to the weight of $\mathbf{g}_i$. Also, we prove that summation of a row of the matrix $F^{\otimes n}$ with a row below it is equal to some clockwise cyclic shifts of that row, and we use this to prove that the $d_{\min}$ for PAC codes is greater than or equal to $d_{\min}$ for the polar codes. The weight distribution of linear codes dictates the performance of ML decoding, which can be well estimated by the union bound, particularly at high SNRs. This implies that PAC codes outperform polar codes in terms of error-correction performance.

6.1 PAC Codes v. Polar Codes

Assume $\mathbf{g}_i$ be the ith row of the matrix $G_n \triangleq F^{\otimes n}$ for $1 \leq i \leq N$. The weight of $\mathbf{g}_i$ is $w(\mathbf{g}_i) = 2^{\sum_{m=1}^{n} i_m}$, where $i-1 = i_n i_{n-1} \cdots i_1 = \sum_{m=1}^{n} i_m 2^{m-1}$ is the bit-index representation of index $i-1$ [2]. Let C_N^j be the $N \times N$ clockwise cyclic shift matrix in j places s.t. $\mathbf{g}_i C_N^j = (g_{i,1}, g_{i,2}, \cdots, g_{i,N}) C_N^j = (g_{i,N-j+1}, g_{i,N-j+2}, \cdots g_{i,N}, \cdots, g_{i,N-j})$.

Example 9. *Let $N = 8$, $j = 3$, and $i = 6$.*

$$C_8^3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For $\mathbf{g}_6 = (1, 1, 0, 0, 1, 1, 0, 0)$,

$$\mathbf{g}_6 C_8^3 = (1, 0, 0, 1, 1, 0, 0, 1).$$

For any nonzero binary vector $\mathbf{u} = (0, \cdots, 0, 1, u_{i+1}, u_{i+2}, u_N)$, where the first 1 is in its ith location, $\mathbf{u}G_n = \mathbf{g}_i \oplus u_{i+1}\mathbf{g}_{i+1} + \cdots \oplus u_N \mathbf{g}_N$. We use $\underline{\mathbf{g}}_i$ notation to show the summation of vector $\mathbf{g}_i$ with some other specified rows below it.

In the following proposition, we prove that adding an odd number of clockwise cyclic shifts of any row of the matrix G_n that is added with some rows below it (i.e., $\underline{\mathbf{g}}_i$ for the ith row of the matrix G_n) cannot decrease the weight of that row. In our proof we partition the generator matrix G_{n+1} into upper and lower parts as $G_{n+1} = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}$ s.t. the rows of H_1 and H_2 can be represented as $(\mathbf{h}_i, \mathbf{0})$ and $(\mathbf{h}_k, \mathbf{h}_k)$, respectively, where both $\mathbf{h}_i$ and $\mathbf{h}_k$ are rows of matrix G_n.

Proposition 1. *For any vector* $(u_{i+1}, u_{i+2}, \cdots, u_N) \in \{0,1\}^{N-i}$,

$$w\left(\underline{\mathbf{g}}_i \sum C_N^l\right) \geq w(\mathbf{g}_i), \tag{6.1}$$

where $1 \leq i \leq N$ *and* $\underline{\mathbf{g}}_i = \mathbf{g}_i \oplus u_{i+1}\mathbf{g}_{i+1} + \cdots \oplus u_N\mathbf{g}_N$ *for any* $l = 1, 3, 5, \cdots$ *and in mod* N *there are odd number of distinct number of cyclic shifts.*

Proof. The proof is by induction on n. For $n = 1$ the statement obviously holds. By induction suppose that the statement holds for n. We want to show that for any row $\mathbf{g}_i$ of G_{n+1},

$$w\left(\underline{\mathbf{g}}_i \sum C_{2N}^l\right) \geq w(\mathbf{g}_i), \tag{6.2}$$

where $1 \leq i \leq 2N$. We divide the proof into two cases based on whether the row index i is bigger than N or not.

Case 1: $i > N$.

In this case we can write the the vector $\underline{\mathbf{g}}_i$ as $(\underline{\mathbf{h}}_{i-N}, \underline{\mathbf{h}}_{i-N})$, where $\mathbf{h}_{i-N}$ is the $(i-N)$th row of matrix G_n. Hence,

$$\begin{aligned} w\left(\underline{\mathbf{g}}_i \sum C_{2N}^l\right) &= w\left((\underline{\mathbf{h}}_{i-N}, \underline{\mathbf{h}}_{i-N}) \sum C_{2N}^l\right) \\ &= w\left(\underline{\mathbf{h}}_{i-N} \sum C_N^l, \underline{\mathbf{h}}_{i-N} \sum C_N^l\right) \\ &= 2w\left(\underline{\mathbf{h}}_{i-N} \sum C_N^l\right) \geq 2w(\mathbf{h}_{i-N}) \\ &= w(\mathbf{h}_{i-N}, \mathbf{h}_{i-N}) = w(\mathbf{g}_i), \end{aligned} \tag{6.3}$$

where the inequality is by induction. Notice that in the third equality it is just possible for even number of shifts get equal (l and $l+N$) and again the inequality will hold.

Case 2: $i \leq N$.

In this case the ith row of the matrix G_{n+1} can be written as $\mathbf{g}_i = (\mathbf{h}, \mathbf{0})$ s.t. $\mathbf{h}$ is the ith row of matrix G_n. We have

$$\underline{\mathbf{g}}_i = (\underline{\mathbf{h}}, \mathbf{0}) \oplus (\underline{\mathbf{y}}, \underline{\mathbf{y}}), \tag{6.4}$$

where $(\mathbf{y}, \mathbf{y})$ is a zero vector or is the jth row of matrix G_{n+1} s.t. $j > N$. We have

$$\begin{aligned} \underline{\mathbf{g}}_i \sum C_{2N}^l &= (\underline{\mathbf{h}}, \mathbf{0}) \sum C_{2N}^l \oplus (\underline{\mathbf{y}}, \underline{\mathbf{y}}) \sum C_{2N}^l \\ &= (\underline{\mathbf{h}}, \mathbf{0}) \sum C_{2N}^l \oplus \left(\underline{\mathbf{y}} \sum C_N^l, \underline{\mathbf{y}} \sum C_N^l \right). \end{aligned} \tag{6.5}$$

Suppose $\gamma \triangleq \left(\underline{\mathbf{y}} \sum C_N^l\right)_k$. Also note that the kth bit

$$\left(\underline{\mathbf{h}} \sum C_N^l\right)_k = \underbrace{\left((\underline{\mathbf{h}}, \mathbf{0}) \sum C_{2N}^l\right)_k}_{\alpha} \oplus \underbrace{\left((\underline{\mathbf{h}}, \mathbf{0}) \sum C_{2N}^l\right)_{k+N}}_{\beta} \tag{6.6}$$

is equal to 1 if $(\alpha, \beta) = (1, 0)$ or $(\alpha, \beta) = (0, 1)$ and we can have one of the following four cases:

If $\gamma = 0$ and $(\alpha, \beta) = (0, 1)$, then $\left(\underline{\mathbf{g}}_i \sum C_{2N}^l\right)_{k+N} = 1$.

If $\gamma = 1$ and $(\alpha, \beta) = (0, 1)$, then $\left(\underline{\mathbf{g}}_i \sum C_{2N}^l\right)_k = 1$.

If $\gamma = 0$ and $(\alpha, \beta) = (1, 0)$, then $\left(\underline{\mathbf{g}}_i \sum C_{2N}^l\right)_k = 1$.

If $\gamma = 1$ and $(\alpha, \beta) = (1, 0)$, then $\left(\underline{\mathbf{g}}_i \sum C_{2N}^l\right)_{k+N} = 1$.

So

$$w(\underline{\mathbf{g}}_i \sum C_{2N}^l) \geq w(\underline{\mathbf{h}} \sum C_N^l) \geq w(\mathbf{h}) = w(\mathbf{g}_i), \tag{6.7}$$

where the last inequality is by induction. □

The following lemma is useful in the proof of the Theorem 3.

Lemma 1. *For $1 \leq m \leq N-1$ and $1 \leq k \leq N$, if $k + m > N$,*

$$\mathbf{g}_k = \mathbf{g}_k \sum_{\substack{1 \leq l \leq N-1 \ \ s.t. \\ g_{m+1,l+1}=1}} C_N^l, \tag{6.8}$$

where $\mathbf{g}_k$ and $\mathbf{g}_{m+1}$ are the kth and $(m+1)$th rows of the matrix G_n.

Example 10. *For $n = 3$, $k = 6$, and $m = 3$, we have that $m + k = 9 > N$. So for $\mathbf{g}_k = (1, 1, 0, 0, 1, 1, 0, 0)$ and $\mathbf{g}_{m+1} = (1, 1, 1, 1, 0, 0, 0, 0)$ we have $\mathbf{g}_k = \mathbf{g}_k(C^1 \oplus C^2 \oplus C^3)$.*

Proof. The proof is by induction. Suppose that Lemma 1 is true for n and we prove it for $n+1$ case. We divide the proof into two cases based on whether k is greater than N or not.

Case 1: $k > N$.

The kth row of matrix G_{n+1} can be written as $\mathbf{g}_k = (\mathbf{h}_{k-N}, \mathbf{h}_{k-N})$, where $\mathbf{h}_{k-N}$ is the $(k-N)$th row of matrix G_n. If $k+m > 2N$, then $k+m-N > N$ and by induction we have that

$$\mathbf{h}_{k-N} = \mathbf{h}_{k-N} \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{m+1,l+1}=1}} C_N^l. \tag{6.9}$$

From this we have

$$\begin{aligned} &\mathbf{g}_k \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{m+1,l+1}=1}} C_{2N}^l \\ &= (\mathbf{h}_{k-N}, \mathbf{h}_{k-N}) \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{m+1,l+1}=1}} C_{2N}^l \\ &= (\mathbf{h}_{k-N} \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{m+1,l+1}=1}} C_N^l, \mathbf{h}_{k-N} \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{m+1,l+1}=1}} C_N^l) \\ &= (\mathbf{h}_{k-N}, \mathbf{h}_{k-N}) = \mathbf{g}_k. \end{aligned} \tag{6.10}$$

Case 2: $k \le N$.

Suppose that $\mathbf{g}_k = (\mathbf{h}_k, \mathbf{0})$ is the kth row of matrix G_{n+1} and $k+m > 2N$, where $\mathbf{h}_k$ is the kth row of matrix G_n. Because $k \le N$, we can write m as $m = j+N$ s.t. $j < N$. From $j+N+k > 2N$, we have $j+k > N$ and by induction we have

$$\mathbf{h}_k = \mathbf{h}_k \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{j+1,l+1}=1}} C_N^l. \tag{6.11}$$

Also note that for any vector $\mathbf{x}$ of length N and $1 \le l \le N-1$ we have

$$(\mathbf{x}, 0)(C_{2N}^l \oplus C_{2N}^{l+N}) = (\mathbf{x}C_N^l, \mathbf{x}C_N^l). \tag{6.12}$$

So we have

$$\begin{aligned}
\mathbf{g}_k \sum_{\substack{1\le l\le 2N-1 \text{ s.t.}\\ g_{m+1,l+1}=1}} C_{2N}^l &= (\mathbf{h}_k, \mathbf{0}) \sum_{\substack{1\le l\le N-1 \text{ s.t.}\\ (h_{j+1,l+1},h_{j+1,l+1})=(1,1)}} C_{2N}^l \\
&= (\mathbf{h}_k \sum_{\substack{1\le l\le N-1 \text{ s.t.}\\ h_{j+1,l+1}=1}} C_N^l, \mathbf{h}_k \sum_{\substack{1\le l\le N-1 \text{ s.t.}\\ h_{j+1,l+1}=1}} C_N^l) \qquad (6.13)\\
&= (\mathbf{h}_k, \mathbf{h}_k) = \mathbf{g}_k,
\end{aligned}$$

where the second equality is by (6.12) and the third equality is by induction. □

Let us define an $N \times N$ upper-triangular bidiagonal matrix D_N as

$$D_N \triangleq \begin{bmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 1 & & \vdots \\ \vdots & 0 & \ddots & \ddots & \\ & \vdots & & 1 & 1 \\ 0 & 0 & \dots & & 1 \end{bmatrix}, \qquad (6.14)$$

which is an upper-triangular Toeplitz matrix that has 1 in its main diagonal and upper diagonal elements (its first row has 1 in the first and second positions), with all other entries being zero.

We also define matrix $D_N^m \triangleq (D_N)^m$ which is an upper-triangular Toeplitz matrix that its first row has 1 in the first and $(m+1)$th positions, where $1 \le m \le N-1$.

The following theorem relates the matrix D_N^m to clock-wise cyclic shift matrices C_N^l.

Theorem 3. *For any $m \le N-1$,*

$$D_N^m G_n = G_n \sum_{\substack{1\le l\le N-1 \ s.t.\\ g_{m+1,l+1}=1}} C_N^l. \qquad (6.15)$$

This is equivalent to saying that if $i = k+m \le N$, then

$$\mathbf{g}_k \oplus \mathbf{g}_i = \mathbf{g}_k \sum_{\substack{1\le l\le N-1 \ s.t.\\ g_{i-k+1,l+1}=1}} C_N^l, \qquad (6.16)$$

and if $k+m>N$*, then*

$$\mathbf{g}_k = \mathbf{g}_k \sum_{\substack{1\le l\le N-1 \ \ s.t. \\ g_{m+1,l+1}=1}} C_N^l \tag{6.17}$$

Example 11. *For* $n=2$*,*

$$G_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, D^2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, C^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

$$D^2G_n = G_nC^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{6.18}$$

Example 12. *For* $n=2$*,*

$$G_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, D^3 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$C^1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, C^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, C^3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$D^3G_n = G_n(C^1 \oplus C^2 \oplus C^3) = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{6.19}$$

Proof. The Lemma 1 is used to show the equation (6.17), and we provide a proof for the case $i=k+m\le N$. Proof is by induction and we assume that (6.16) is true for G_n and we prove it for G_{n+1}. In this respect, we have $i=k+m\le 2N$. We divide the proof into 6 cases based on the values of i and k. First case is

for $k > N$, the second case is when $i, k \leq N$, and the other four cases are when $k \leq N$ and $i > N$.

Case 1: $k > N$.

We have

$$\mathbf{g}_k \oplus \mathbf{g}_i = (\mathbf{h}_{k-N} \oplus \mathbf{h}_{i-N}, \mathbf{h}_{k-N} \oplus \mathbf{h}_{i-N}), \tag{6.20}$$

where $\mathbf{h}_{k-N}$ and $\mathbf{h}_{i-N}$ are $(k-N)$th and $(i-N)$th rows of matrix G_N, respectively. By induction we have

$$\begin{aligned}
\mathbf{g}_k \oplus \mathbf{g}_i &= (\mathbf{h}_{k-N} \oplus \mathbf{h}_{i-N}, \mathbf{h}_{k-N} \oplus \mathbf{h}_{i-N}) \\
&= (\mathbf{h}_{k-N} \sum_{\substack{1 \leq l \leq N-1 \text{ s.t.} \\ h_{i-k+1,l+1}=1}} C_N^l, \mathbf{h}_{k-N} \sum_{\substack{1 \leq l \leq N-1 \text{ s.t.} \\ h_{i-k+1,l+1}=1}} C_N^l) \\
&= (\mathbf{h}_{k-N}, \mathbf{h}_{k-N}) \sum_{\substack{1 \leq l \leq 2N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_{2N}^l \\
&= \mathbf{g}_k \sum_{\substack{1 \leq l \leq 2N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_{2N}^l.
\end{aligned} \tag{6.21}$$

As an example for $N = 8$, $k = 12$, and $i = 15$, the 12th and 15th rows of matrix G_4 are

$$\begin{aligned}
\mathbf{g}_{12} &= (\mathbf{h}_4, \mathbf{h}_4) = (1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0), \\
\mathbf{g}_{15} &= (\mathbf{h}_7, \mathbf{h}_7) = (1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0).
\end{aligned}$$

By knowing that $i - k + 1 = 4$ we have

$$\mathbf{g}_{12} \oplus \mathbf{g}_{15} = \mathbf{g}_{12}(C_{16}^1 \oplus C_{16}^2 \oplus C_{16}^3).$$

Case 2: $k \leq N$ and $i \leq N$.

The kth and ith rows of the matrix G_{n+1} can be written as $\mathbf{g}_k = (\mathbf{h}_k, \mathbf{0})$ and $\mathbf{g}_i = (\mathbf{h}_i, \mathbf{0})$, respectively, where $\mathbf{h}_k$ and $\mathbf{h}_i$ are the corresponding rows of the matrix G_n. By induction we have

$$\mathbf{h}_k \oplus \mathbf{h}_i = \mathbf{h}_k \sum_{\substack{1 \leq l \leq N-1 \text{ s.t.} \\ h_{i-k+1,l+1}=1}} C_N^l. \tag{6.22}$$

Note that $\mathbf{h}_k$ has its last 1 at the kth position and shifting based on the vector $\mathbf{h}_{i-k+1}$ can shift that last 1 up to $i - k$ position. As a result, the last 1 of the

vector $\mathbf{h}_k$ can be shifted up to the ith position which is less that or equal to N. Considering this we have

$$\begin{aligned}
&\mathbf{g}_k \oplus \mathbf{g}_i = (\mathbf{h}_k, \mathbf{0}) \oplus (\mathbf{h}_i, \mathbf{0}) = (\mathbf{h}_k \oplus \mathbf{h}_i, \mathbf{0}) \\
&= (\mathbf{h}_k \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{i-k+1,l+1}=1}} C_N^l, \mathbf{0}) \\
&= (\mathbf{h}_k, \mathbf{0}) \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_{2N}^l = \mathbf{g}_k \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_{2N}^l,
\end{aligned} \tag{6.23}$$

where the third equality is by induction and the fourth equality is by the discussion above.

As an example for $N = 8$, $k = 2$, and $i = 8$, the 2nd and 8th rows of matrix G_4 are

$$\begin{aligned}
\mathbf{g}_2 &= (\mathbf{h}_2, \mathbf{0}) = (1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0), \\
\mathbf{g}_8 &= (\mathbf{h}_8, \mathbf{0}) = (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0).
\end{aligned}$$

By knowing that $i - k + 1 = 7$ we have

$$\mathbf{g}_2 \oplus \mathbf{g}_8 = \mathbf{g}_2(C_{16}^2 \oplus C_{16}^4 \oplus C_{16}^6).$$

Case 3: $k \le N$, $i = j + N$ and $j \ge k$.
The kth, jth, and the ith rows of the matrix G_{n+1} are as $\mathbf{g}_k = (\mathbf{h}_k, \mathbf{0})$, $\mathbf{g}_j = (\mathbf{h}_j, \mathbf{0})$, and $\mathbf{g}_i = (\mathbf{h}_j, \mathbf{h}_j)$, respectively, where $\mathbf{h}_k$ and $\mathbf{h}_j$ are the corresponding rows of the matrix G_n. By induction, in shifting based on the $r = j-k+1$ row of matrix G_n, we have

$$\mathbf{h}_k \oplus \mathbf{h}_j = \mathbf{h}_k \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{j-k+1,l+1}=1}} C_N^l. \tag{6.24}$$

Note that the shifts are by the positions greater than or equal to 2 of the vector $\mathbf{h}_r$ which we show as $(\times, h_{r,2}, h_{r,3}, \cdots, h_{r,N})$. In general, we say that shifting vector $\mathbf{h}_k$ based on vector $(\times, h_{r,2}, h_{r,3}, \cdots, h_{r,N})$ results in $\mathbf{h}_k \oplus \mathbf{h}_j$ as an alternative way to (6.24).

Note that from case 2, we know that with shifting $\mathbf{h}_k$ by row $j-k+1$, the last 1 element of $\mathbf{h}_k$ will be shifted at most to the Nth position. From this we can say

that shifting $(\mathbf{h}_k, \mathbf{0})$ based on $(\times, h_{r,2}, h_{r,3}, \cdots, h_{r,N}, \mathbf{0})$ is $(\mathbf{h}_k \oplus \mathbf{h}_j, \mathbf{0})$. Similarly, shifting the kth row $(\mathbf{h}_k, \mathbf{0})$ based on $(\mathbf{0}, \times, h_{r,2}, h_{r,3}, \cdots, h_{r,N})$ is $(\mathbf{0}, \mathbf{h}_k \oplus \mathbf{h}_j)$. Also, note that shifing a vector $(x_1, x_2, \cdots x_N, \mathbf{0})$ based on a vector that only has 1 at its $(N+1)$th position is $(\mathbf{0}, x_1, x_2, \cdots x_N)$, i.e.

$$(x_1, x_2, \cdots x_N, \mathbf{0}) C_{2N}^N = (\mathbf{0}, x_1, x_2, \cdots x_N). \tag{6.25}$$

As a results, shifting the kth row $(\mathbf{h}_k, \mathbf{0})$ based on vector

$$(\times, h_{r,2}, h_{r,3}, \cdots, h_{r,N}, 1, h_{r,2}, h_{r,3}, \cdots, h_{r,N})$$

results in $(\mathbf{h}_k \oplus \mathbf{h}_j, \mathbf{h}_k \oplus (\mathbf{h}_k \oplus \mathbf{h}_j)) = (\mathbf{h}_k \oplus \mathbf{h}_j, \mathbf{h}_j)$. This means that

$$\begin{aligned}
&\mathbf{g}_k \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1, l+1}=1}} C_{2N}^l \\
&= (\mathbf{h}_k, \mathbf{0}) \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1, l+1}=1}} C_{2N}^l \\
&= (\mathbf{h}_k \oplus \mathbf{h}_j, \mathbf{0}) \oplus (\mathbf{0}, \mathbf{h}_k) \oplus (\mathbf{0}, \mathbf{h}_k \oplus \mathbf{h}_j) \\
&= (\mathbf{h}_k \oplus \mathbf{h}_j, \mathbf{h}_j) = \mathbf{g}_k \oplus \mathbf{g}_i.
\end{aligned} \tag{6.26}$$

As an example, for $N = 8$, $k = 7$, and $i = 16$, the 7th and the 16th rows of matrix G_4 are

$$\begin{aligned}
\mathbf{g}_7 &= (\mathbf{h}_7, \mathbf{0}) = (1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0), \\
\mathbf{g}_{16} &= (\mathbf{h}_8, \mathbf{h}_8) = (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1).
\end{aligned}$$

As $j = 8$ and $j-k+1 = 2$, the shift is based on $(\times, h_{2,2}, \cdots, h_{2,8}, 1, h_{2,2}, \cdots, h_{2,8}) = (\times, 1,0,0,0,0,0,0,1,1,0,0,0,0,0,0)$. So

$$\mathbf{g}_7 \oplus \mathbf{g}_{16} = \mathbf{g}_7 (C_{16}^1 \oplus C_{16}^8 \oplus C_{16}^9).$$

Case 4: $k \le N$, $i = j + N$, $j \le k$, and $j > N/2$.
We have $N/2 < k \le N$ and $N + N/2 < i$. The kth and ith rows of the matrix G_{n+1} can be written as $\mathbf{g}_k = (\mathbf{h}_k, \mathbf{0}) = (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0})$ and $\mathbf{g}_i = (\mathbf{h}_j, \mathbf{h}_j) = (\mathbf{y}, \mathbf{y}, \mathbf{y}, \mathbf{y})$, respectively, where $\mathbf{h}_k$ and $\mathbf{h}_j$ are the corresponding rows of the matrix G_N, and $\mathbf{x}$ and $\mathbf{y}$ are corresponding rows of the matrix $G_{N/2}$.

We have

$$\mathbf{h}_{k-N/2} \oplus \mathbf{h}_j = (\mathbf{x}, \mathbf{0}) \oplus (\mathbf{y}, \mathbf{y}) = (\mathbf{x} \oplus \mathbf{y}, \mathbf{y}), \tag{6.27}$$

and by induction

$$(\mathbf{x} \oplus \mathbf{y}, \mathbf{y}) = (\mathbf{x}, \mathbf{0}) \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{j-k+N/2+1, l+1}=1}} C_N^l. \tag{6.28}$$

Notice that $j - k + N/2 + 1 \le N/2$. So $\mathbf{g}_{j-k+N/2+1} = (\mathbf{r}, \mathbf{0}, \mathbf{0}, \mathbf{0})$, where $\mathbf{r}$ is a row of matrix $G_{N/2}$. In this way, $i - k + 1 = (j - k + N/2 + 1) + N/2$ and $\mathbf{g}_{i-k+1} = (\mathbf{r}, \mathbf{r}, \mathbf{0}, \mathbf{0})$.

From (6.28) we can see that shifting $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0}, \mathbf{0})$.

Moreover, shifting $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ by $(\mathbf{0}, 1, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{0}, \mathbf{x} \oplus (\mathbf{x} \oplus \mathbf{y}), \mathbf{y}, \mathbf{0}) = (\mathbf{0}, \mathbf{y}, \mathbf{y}, \mathbf{0})$.

Likewise, shifting $(\mathbf{0}, \mathbf{x}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ is also $(\mathbf{0}, \mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0})$.

Finally, shifting $(\mathbf{0}, \mathbf{x}, \mathbf{0}, \mathbf{0})$ by $(\mathbf{0}, 1, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{0}, \mathbf{0}, \mathbf{x} \oplus (\mathbf{x} \oplus \mathbf{y}), \mathbf{y}) = (\mathbf{0}, \mathbf{0}, \mathbf{y}, \mathbf{y})$.

By considering these four shifting together, the shifting of $(\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_{N/2}, 1, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0}, \mathbf{0}) \oplus (\mathbf{0}, \mathbf{y}, \mathbf{y}, \mathbf{0}) \oplus (\mathbf{0}, \mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0}) \oplus (\mathbf{0}, \mathbf{0}, \mathbf{y}, \mathbf{y}) = (\mathbf{x} \oplus \mathbf{y}, \mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{y})$. So, we conclude that

$$\begin{aligned} \mathbf{g}_k \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1, l+1}=1}} C_{2N}^l &= (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0}) \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1, l+1}=1}} C_{2N}^l \\ &= (\mathbf{x} \oplus \mathbf{y}, \mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{y}) \\ &= (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0}) \oplus (\mathbf{y}, \mathbf{y}, \mathbf{y}, \mathbf{y}) = \mathbf{g}_k \oplus \mathbf{g}_i. \end{aligned} \tag{6.29}$$

As an example, for $N = 8$, $k = 8$, and $i = 15$, the 8th and 15th rows of matrix G_4 are

$$\begin{aligned} \mathbf{g}_8 &= (\mathbf{h}_8, \mathbf{0}) = (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0}) = (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0), \\ \mathbf{g}_{15} &= (\mathbf{h}_7, \mathbf{h}_7) = (\mathbf{y}, \mathbf{y}, \mathbf{y}, \mathbf{y}) = (1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0). \end{aligned}$$

As $i-k+1 = 8$, Shifting is based on the vector $\mathbf{g}_{i-k+1} = (\times, r_2, \cdots, r_{N/2}, 1, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0}) = (\times, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)$. So $\mathbf{g}_8 \oplus \mathbf{g}_{15} = \mathbf{g}_8(C^1 \oplus C^2 \oplus \cdots \oplus C^7)$.

Case 5: $k \leq N$, $i = j + N$, $j \leq k$, and $1 \leq j \leq N/2$ and $k > N/2$.
We have that $N < i \leq N + N/2$ and $1 \leq i - k + 1 \leq N$. The kth and the ith rows of the matrix G_{n+1} can be written as $\mathbf{g}_k = (\mathbf{h}_k, \mathbf{0}) = (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0})$, and $\mathbf{g}_i = (\mathbf{h}_j, \mathbf{h}_j) = (\mathbf{y}, \mathbf{0}, \mathbf{y}, \mathbf{0})$, respectively, where $\mathbf{h}_k$ and $\mathbf{h}_j$ are the corresponding rows of the matrix G_N, and $\mathbf{x}$ and $\mathbf{y}$ are corresponding rows of the matrix $G_{N/2}$. Also $\mathbf{h}_{k-N/2} = (\mathbf{x}, \mathbf{0})$ and $\mathbf{h}_{i-N/2} = (\mathbf{y}, \mathbf{y})$. By induction we have that

$$\begin{aligned}(\mathbf{x}, \mathbf{0}) \sum_{\substack{1 \leq l \leq N-1 \text{ s.t.} \\ h_{i-k+1,l+1}=1}} C_N^l &= \mathbf{h}_{k-N/2} \sum_{\substack{1 \leq l \leq N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_N^l \\ &= \mathbf{h}_{k-N/2} \oplus \mathbf{h}_{i-N/2} = (\mathbf{x} \oplus \mathbf{y}, \mathbf{y}).\end{aligned} \tag{6.30}$$

As $i - k + 1 \leq N$, $\mathbf{g}_{i-k+1} = (\mathbf{r}, \mathbf{0})$.

From (6.30) we have that shifting $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_N, \mathbf{0})$ is $(\mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0}, \mathbf{0})$.

Also shifting $(\mathbf{0}, \mathbf{x}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_N, \mathbf{0})$ is $(\mathbf{0}, \mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0})$.

By considering these two shifting together, the shifting of $(\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_N, \mathbf{0})$ is $(\mathbf{x} \oplus \mathbf{y}, \mathbf{y} \oplus (\mathbf{x} \oplus \mathbf{y}), \mathbf{y}, \mathbf{0}) = (\mathbf{x} \oplus \mathbf{y}, \mathbf{x}, \mathbf{y}, \mathbf{0})$. So we conclude that

$$\begin{aligned}\mathbf{g}_k \sum_{\substack{1 \leq l \leq 2N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_{2N}^l &= (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0}) \sum_{\substack{1 \leq l \leq N-1 \text{ s.t.} \\ g_{i-k+1,l+1}=1}} C_{2N}^l \\ &= (\mathbf{x} \oplus \mathbf{y}, \mathbf{x}, \mathbf{y}, \mathbf{0}) \\ &= (\mathbf{x}, \mathbf{x}, \mathbf{0}, \mathbf{0}) \oplus (\mathbf{y}, \mathbf{0}, \mathbf{y}, \mathbf{0}) = \mathbf{g}_k \oplus \mathbf{g}_i.\end{aligned} \tag{6.31}$$

Case 6: $k \leq N$, $i = j + N$, $j \leq k$, and $1 \leq j \leq N/2$ and $k \leq N/2$.
We have that $N < i \leq N + N/2$ and $N/2 < i - k + 1 \leq N$. The kth and the ith rows of the matrix G_{n+1} can be written as $\mathbf{g}_k = (\mathbf{h}_k, \mathbf{0}) = (\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$, and $\mathbf{g}_i = (\mathbf{h}_j, \mathbf{h}_j) = (\mathbf{y}, \mathbf{0}, \mathbf{y}, \mathbf{0})$, respectively, where $\mathbf{h}_k$ and $\mathbf{h}_j$ are the corresponding

rows of the matrix G_N, and $\mathbf{x}$ and $\mathbf{y}$ are corresponding rows of the matrix $G_{N/2}$. Also $\mathbf{h}_k = (\mathbf{x}, \mathbf{0})$ and $\mathbf{h}_{i-N/2} = (\mathbf{y}, \mathbf{y})$. By induction we have that

$$\begin{aligned}(\mathbf{x}, \mathbf{0}) \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{i-N/2-k+1, l+1}=1}} C_N^l &= \mathbf{h}_k \sum_{\substack{1 \le l \le N-1 \text{ s.t.} \\ h_{i-N/2-k+1, l+1}=1}} C_N^l \\ = \mathbf{h}_k \oplus \mathbf{h}_{i-N/2} &= (\mathbf{x} \oplus \mathbf{y}, \mathbf{y}).\end{aligned} \tag{6.32}$$

Note that $1 < i - N/2 - k + 1 \le N/2$ and $N/2 < i - k + 1 \le N$. So $\mathbf{g}_{i-k+1} = (\mathbf{r}, \mathbf{r}, \mathbf{0}, \mathbf{0})$.

From (6.32), shifting $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ by $\mathbf{g}_{i-N/2-k+1} = (\times, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{x} \oplus \mathbf{y}, \mathbf{y}, \mathbf{0}, \mathbf{0})$.

Likewise, shifting $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ by $(\mathbf{0}, 1, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{0}, \mathbf{x} \oplus (\mathbf{x} \oplus \mathbf{y}), \mathbf{y}, \mathbf{0})$.

By considering these two shifting together, the shifting of $(\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ by $(\times, r_2, \cdots, r_{N/2}, 1, r_2, \cdots, r_{N/2}, \mathbf{0}, \mathbf{0})$ is $(\mathbf{x} \oplus \mathbf{y}, \mathbf{0}, \mathbf{y}, \mathbf{0})$.

So we conclude that

$$\begin{aligned}\mathbf{g}_k \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1, l+1}=1}} C_{2N}^l &= (\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0}) \sum_{\substack{1 \le l \le 2N-1 \text{ s.t.} \\ g_{i-k+1, l+1}=1}} C_{2N}^l \\ = (\mathbf{x} \oplus \mathbf{y}, \mathbf{0}, \mathbf{y}, \mathbf{0}) & \\ = (\mathbf{x}, \mathbf{0}, \mathbf{0}, \mathbf{0}) \oplus (\mathbf{y}, \mathbf{0}, \mathbf{y}, \mathbf{0}) &= \mathbf{g}_k \oplus \mathbf{g}_i.\end{aligned} \tag{6.33}$$

□

Assume that the matrix T_N is an $N \times N$ upper-triangular Toeplitz matrix and that the elements on each diagonal of the matrix are same. We can see that $T_N = \sum_m D_N^m$ for some values of m and by Theorem 3, $D_N^m G_n = G_n \sum_l C_N^l$ for some values of l. Therefore, for a data word $\mathbf{v}$,

$$\mathbf{v} T_N G_n = \mathbf{v} \sum_m D_N^m G_n = \mathbf{v} G_n \sum_s C_N^s, \tag{6.34}$$

where the second equality is by Theorem 3.

Assume that the data vector $\mathbf{v}$ has its first 1 at its ith location, i.e. $\mathbf{v}G_n = \underline{\mathbf{g}}_i$, where $\mathbf{g}_i$ is the ith row of the matrix G_n. From (6.34) we have

$$w(\mathbf{v}T_N G_n) = w(\mathbf{v}G_n \sum_s C_N^s) = w(\underline{\mathbf{g}}_i \sum_s C_N^s) \geq w(\mathbf{g}_i), \tag{6.35}$$

where the inequality is by Proposition 1. By noticing that the $d_{\min}$ of polar code is the minimum row weight of $G_{N,\mathcal{A}}$, this proves that $d_{\min}$ for PAC codes is greater than or equal to $d_{\min}$ for the polar code.

6.2 On the Weight Distribution of PAC Codes

For an (N, K) code with a weight distribution $\{A_0 = 1, A_1, A_2, \cdots, A_N = 1\}$, the union upper bound of the probability of error is

$$P_E \leq \sum_{k=2}^{2^K} Z(W)^{w_k} = \sum_{w_d \geq d_{min}} A_d Z(W)^{w_d}, \tag{6.36}$$

where $Z(W)$ is the Bhattacharyya parameter of the channel W, w_k is the weight of the codeword k, and d_{min} is the minimum non-zero codeword weight [34]. When the error is low, we can approximate the probability of error with the first term of the upper bound as

$$P_E \approx A_{d_{min}} Z(W)^{w_{d_{min}}}. \tag{6.37}$$

There is an algebraic formula to calculate the weight distribution of second-order binary (N, K) RM codes [62], and in the previous section we proved that a pre-multiplication of an upper-triangular Toeplitz matrix to the polar codes generator matrix, the minimum distance will not decrease.

By approximation of the error probability for low error rates in (6.37), decreasing $A_{d_{min}}$ is equivalent to improving the low error rate performance. In the PAC codes, the pre-multiplication of the Toeplitz matrix tries to improve the weight distribution of the code by summing the row i of the polar transformation matrix with the rows below it. Our simulation results on the first 1200 nontrivial

connection polynomials show that for an RM$(2,5)$ code, pre-multiplication of a Toeplitz matrix will result in 4 different weight distributions listed in Table 6.1, where D_i is the set of all connection polynomials with the weight distribution of type i. Connection polynomial 3, 3165, and identity matrix are examples of D_4. The sets $\{7, 3253\}$, $\{13, 133, 3207\}$, and $\{1131, 3211\}$ are connection polynomial examples of D_3, D_2, and D_1, respectively. From Table 6.1, it can be seen that a good connection polynomial can decrease $A_{d_{min}}$ from 620 (for the RM$(2,5)$ codes) to 236 (for the PAC$(32,16)$ codes).

We use the polynomials of D_1 as the connection polynomials for the PAC$(128,29)$ codes (corresponding to an RM$(2,7)$). For RM$(2,7)$ codes, $A_{d_{min}} = 10668$, while by using the connection polynomial 3211 in the PAC$(128,29)$ codes, $A_{d_{min}}$ decreases to 324. A good choice of connection polynomial with improving the weight distribution will result in better error-correction performance. The effect of different connection polynomials on FER performance of PAC$(128,29)$ codes is shown in Figure 6.1. All the connection polynomials used in this figure result in almost the same ANV.

Table 6.1: Weight Distributions of the PAC$(32,16)$ codes with different polynomial connections.

D_1		D_2		D_3		D_4	
w	$A(w)$	w	$A(w)$	w	$A(w)$	w	$A(w)$
0	1	0	1	0	1	0	1
8	236	8	364	8	492	8	620
10	3072	10	2048	10	1024	12	13888
12	3136	12	6720	12	10304	16	36518
14	21504	14	14336	14	7168	20	13888
16	9638	16	18598	16	27558	24	620
18	21504	18	14336	18	7168	32	1
20	3136	20	6720	20	10304		
22	3072	22	2048	22	1024		
24	236	24	364	24	492		
32	1	32	1	32	1		

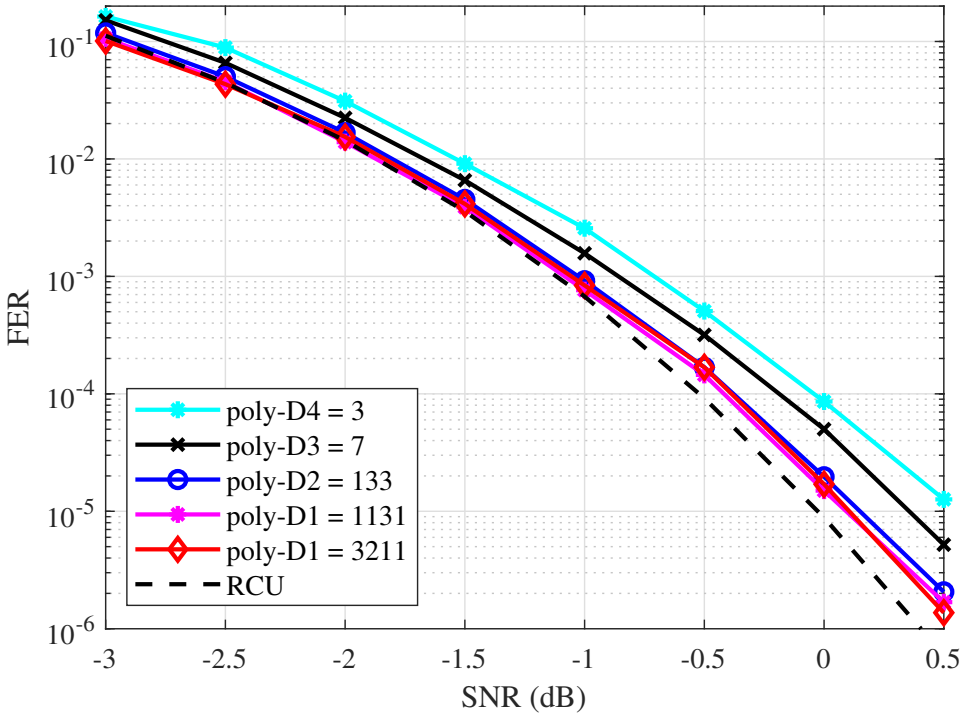

Figure 6.1: PAC codes with different polynomials.

Chapter 7

Summary and Conclusions

From a practical standpoint, short block-length codes with a low complexity and good error-correction probability are desirable. Over BI-AWGN channels, PAC codes with sequential decoding [13] are shown to have an error-correction performance close to the dispersion approximation. In this dissertation, we have studied the error-correction and computational complexity performances of PAC codes both from theoretical and experiential aspects. We studied stack, Fano, and SCL decoding of PAC codes and provided an in-depth review of each.

Then, we studied the metric function of sequential decoding. We provided a metric function for sequential decoding of PAC codes that employs bit-channel mutual information and cutoff rate values as the bias. This benefits in a favorable trade-off between computational complexity and error-correction performance. Additionally, we demonstrated that sequential decoding of PAC codes has a Pareto distribution upper bound on its computational complexity when bias values are smaller than the bit-channel cutoff rates. As a result, the probability of encountering a high level of computational complexity is negligible. Also shown was the possibility for superior error-correction performance of the PAC codes under sequential decoding compared to the 5G polar codes under CRC-aided SCL decoding with similar computational complexity. The main drawback of sequential decoding is its variable complexity. We addressed this problem by

introducing a bound on the complexity of sequential decoding.

The amount of computation of sequential decoding of PAC codes is a random variable. We addressed the lower bound on the amount of computation by the guessing technique and channel polarization approach, and we proved that for the PAC codes, the computational cutoff rate polarizes. Additionally, we presented a technique for constructing rate profiles for PAC codes that allows for a trade-off between decoding complexity and error-correction performance. This approach tries to enhance the FER performance of PAC codes while ensuring a low mean sequential decoding complexity above a predetermined SNR. In comparison to RM-Polar rate profiles, numerical results indicated that our proposed rate profiles resulted in a 0.5 dB coding gain at FER $= 10^{-3}$ for PAC$(64, 32)$ and PAC$(256, 128)$ codes. Our suggested approach for constructing rate profiles can be applied to any pre-transformed polar code by substituting the desired matrix for the CC generator (Toeplitz) matrix.

In another line of investigation, we analyzed the optimal metric function of the SCL decoding algorithm for polar and PAC codes. On average, the path metric of the correct path should be equal to the sum of the bit-channel mutual information, and the path metric of the incorrect branch can be no more than 0. We took advantage of this by introducing an approach to reject the erroneous paths based on the departure of their path metric from the bit-channel mutual information. This approach avoids sorting on many branches while causing no loss in the FER performance. Moreover, we proposed a similar approach to the stack algorithm based on the bit-metrics of the polarized channels. For the noiseless bit channels, we proved that the bit-channel metric value should be zero for the correct branch and $-\infty$ for the wrong branch. This way, the decoding algorithm can identify the incorrect branch and avoid adding it to the stack. This reduces the needed stack size up to 90% in our simulation results. Additionally, we proved that the probability of pruning the correct path exponentially approaches zero for a threshold value smaller than the bit-channel cutoff rate, allowing us to suggest a technique for determining the threshold value.

Finally, we investigated the weight distribution of PAC codes. We proved that

the summation of an odd number of clockwise cyclic shifts of any row of the matrix $F^{\otimes n}$ could not decrease the weight of that row. Also, we prove that summation of a row of the matrix $F^{\otimes n}$ with a row below it is equal to some clockwise cyclic shifts of that row, and we used this to prove that the $d_{\min}$ for PAC codes is greater than or equal to $d_{\min}$ for the polar codes. The weight distribution of linear codes dictates the performance of ML decoding, which can be well estimated by the union bound, particularly at high SNR values. This implies that PAC codes outperform polar codes in terms of error-correction performance. We also studied the impact of different connection polynomial $c(x)$ on the performance of PAC codes through its effect on the multiplicity $A_{d\min}$ of minimum-distance codewords. The simulation findings indicate that a good connection polynomial may result in significantly better FER performances.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[2] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[3] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[4] D. J. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *IMA International Conference on Cryptography and Coding*, pp. 100–111, Springer, 1995.

[5] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, no. 18, p. 1645, 1996.

[6] M. Sipser and D. A. Spielman, "Expander codes," *IEEE transactions on Information Theory*, vol. 42, no. 6, pp. 1710–1722, 1996.

[7] D. A. Spielman, "Linear-time encodable and decodable error-correcting codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1723–1731, 1996.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2, pp. 1064–1070, IEEE, 1993.

[9] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[10] I. Dumer and K. Shabunov, "Recursive list decoding for Reed-Muller codes," *arXiv preprint arXiv:1703.05304*, 2017.

[11] I. Dumer, "On decoding algorithms for polar codes," *arXiv preprint arXiv:1703.05307*, 2017.

[12] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, 2012.

[13] E. Arıkan, "From sequential decoding to channel polarization and back again," *arXiv preprint arXiv:1908.09594*, 2019.

[14] M. Moradi, "On sequential decoding metric function of polarization-adjusted convolutional (PAC) codes," *IEEE Transactions on Communications*, vol. 69, no. 12, pp. 7913–7922, 2021.

[15] M. Moradi and A. Mozammel, "A Monte-Carlo based construction of polarization-adjusted convolutional (PAC) codes," *arXiv preprint arXiv:2106.08118*, 2021.

[16] J. M. Wozencraft, "Sequential decoding for reliable communication," Tech. Rep. 325, Research Laboratory of Electronics, MIT, Cambridge, 1957.

[17] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Transactions on Information Theory*, vol. 9, no. 2, pp. 64–74, 1963.

[18] K. Zigangirov, "Some sequential decoding procedures," *Problemy Peredachi Informatsii*, vol. 2, no. 4, pp. 13–25, 1966.

[19] F. Jelinek, "Fast sequential decoding algorithm using a stack," *IBM journal of research and development*, vol. 13, no. 6, pp. 675–685, 1969.

[20] A. Mozammel, "Hardware implementation of Fano decoder for polarization-adjusted convolutional (PAC) codes," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1632–1636, 2022.

[21] J. M. Geist, "Algorithmic aspects of sequential decoding," tech. rep., 1970.

[22] O. Shalvi, N. Sommer, and M. Feder, "Signal codes: Convolutional lattice codes," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5203–5226, 2011.

[23] B. Li, H. Zhang, and J. Gu, "On pre-transformed polar codes," *arXiv preprint arXiv:1912.06359*, 2019.

[24] H. Luo, G. Zhang, J. Wang, R. Li, Y. HuangFu, H. Zhang, Y. Chen, and J. Wang, "Polar code transmission method and apparatus," in *US Patent App*, pp. 16/673,581, IEEE, 2020.

[25] M. Moradi and A. Mozammel, "Concatenated Reed-Solomon and polarization-adjusted convolutional (PAC) codes," *arXiv preprint arXiv:2106.08822*, 2021.

[26] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[27] H. Yao, A. Fazeli, and A. Vardy, "List decoding of Arıkan's PAC codes," *Preprints: 2021050235*, 2021.

[28] M. Moradi, A. Mozammel, K. Qin, and E. Arikan, "Performance and complexity of sequential decoding of PAC codes," *arXiv preprint arXiv:2012.04990*, 2020.

[29] S. Lin and D. J. Costello, *Error control coding*, vol. 2. New York: Prentice hall, 2001.

[30] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Communications Letters*, vol. 12, no. 6, pp. 447–449, 2008.

[31] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," in *2009 IEEE International Symposium on Information Theory*, pp. 1496–1500, 2009.

[32] D. Wu, Y. Li, and Y. Sun, "Construction and block error rate analysis of polar codes over AWGN channel based on Gaussian approximation," *IEEE Communications Letters*, vol. 18, no. 7, pp. 1099–1102, 2014.

[33] B. Li, H. Shen, and D. Tse, "A RM-polar codes," *arXiv preprint arXiv:1407.5483*, 2014.

[34] R. G. Gallager, *Information theory and reliable communication*, vol. 2. New York: Wiley, 1968.

[35] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[36] G. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[37] A. J. Viterbi and J. K. Omura, *Principles of digital communication and coding*. New York: McGraw-Hill, 1979.

[38] I. M. Jacobs and J. Wozencraft, *Principles of communication engineering*. New York: John Wiley and Sons, 1965.

[39] M. Rowshan, A. Burg, and E. Viterbo, "Polarization-adjusted convolutional (PAC) codes: Sequential decoding vs list decoding," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1434–1447, 2021.

[40] M.-O. Jeong and S.-N. Hong, "SC-Fano decoding of polar codes," *IEEE Access*, vol. 7, pp. 81682–81690, 2019.

[41] K. Niu and K. Chen, "Stack decoding of polar codes," *Electronics letters*, vol. 48, no. 12, pp. 695–697, 2012.

[42] P. Trifonov, "A score function for sequential decoding of polar codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1470–1474, IEEE, 2018.

[43] J. Massey, "Variable-length codes and the Fano metric," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 196–198, 1972.

[44] Y. Polyanskiy, H. V. Poor, and S. Verdu, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[46] N. Sommer, M. Feder, and O. Shalvi, "Closest point search in lattices using sequential decoding," in *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pp. 1053–1057, IEEE, 2005.

[47] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *Journal of Signal Processing Systems*, vol. 69, no. 3, pp. 305–315, 2012.

[48] P. Trifonov, "Performance and complexity of the sequential successive cancellation decoding algorithm," *arXiv preprint arXiv:2012.08139*, 2020.

[49] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 2116–2120, IEEE, 2014.

[50] S. Seyedmasoumian and T. M. Duman, "Approximate weight distribution of polarization-adjusted convolutional (PAC) codes," *arXiv preprint arXiv:2202.12885*, 2022.

[51] S. K. Mishra, D. Katyal, and S. A. Ganapathi, "A modified Q-learning algorithm for rate-profiling of polarization adjusted convolutional (PAC) codes," *arXiv preprint arXiv:2110.01563*, 2021.

[52] T. Tonnellier and W. J. Gross, "On systematic polarization-adjusted convolutional (PAC) codes," *IEEE Communications Letters*, vol. 25, no. 7, pp. 2128–2132, 2021.

[53] S. K. Mishra, D. Katyal, and S. A. Ganapathi, "A heuristic algorithm for rate-profiling of polarization adjusted convolutional (PAC) codes," 2021.

[54] E. Arıkan, "An inequality on guessing and its application to sequential decoding," *IEEE Transactions on Information Theory*, vol. 42, no. 1, pp. 99–105, 1996.

[55] J. L. Massey, "Guessing and entropy," in *Proceedings of 1994 IEEE International Symposium on Information Theory*, p. 204, IEEE, 1994.

[56] E. Arıkan, "On the origin of polar coding," *IEEE journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 209–223, 2015.

[57] M. Moradi, "On the metric and computation of PAC codes," *arXiv preprint arXiv:2012.05511*, 2020.

[58] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[59] K. Chen, K. Niu, and J. Lin, "A reduced-complexity successive cancellation list decoding of polar codes," in *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, 2013.

[60] K. Chen, B. Li, H. Shen, J. Jin, and D. Tse, "Reduce the complexity of list decoding of polar codes by tree-pruning," *IEEE Communications Letters*, vol. 20, no. 2, pp. 204–207, 2016.

[61] H. Luo, G. Zhang, A. Maevskiy, V. Gritsenko, Y. Zhou, Y. Chen, R. Li, Y. Ge, J. Wang, and J. Wang, "Analysis and application of permuted polar codes," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–5, IEEE, 2018.

[62] N. Sloane and E. Berlekamp, "Weight enumerator for second-order Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 16, no. 6, pp. 745–751, 1970.