STUDY AND PERFORMANCE ANALYSIS OF CONSTRUCTION AND DECODING ALGORITHMS FOR POLAR CODES IN ADDITIVE WHITE GAUSSIAN NOISE (AWGN) COMMUNICATION CHANNELS

RODRIGO ANDRES RODRIGUES FISCHER

MONOGRAFIA DE TRABALHO DE CONCLUSÃO DE CURSO EM
ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

STUDY AND PERFORMANCE ANALYSIS OF CONSTRUCTION AND DECODING ALGORITHMS FOR POLAR CODES IN ADDITIVE WHITE GAUSSIAN NOISE (AWGN) COMMUNICATION CHANNELS

ESTUDO E ANÁLISE DE DESEMPENHO DE ALGORITMOS DE CONSTRUÇÃO E DECODIFICAÇÃO DE CÓDIGOS POLARES EM CANAIS DE COMUNICAÇÃO COM RUÍDO DO TIPO BRANCO GAUSSIANO (AWGN)

RODRIGO ANDRES RODRIGUES FISCHER

ORIENTADOR: PROF. DR. JOÃO PAULO LEITE

MONOGRAFIA DE TRABALHO DE CONCLUSÃO
DE CURSO EM ENGENHARIA ELÉTRICA

BRASÍLIA/DF: DEZEMBRO - 2020

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

STUDY AND PERFORMANCE ANALYSIS OF CONSTRUCTION AND DECODING ALGORITHMS FOR POLAR CODES IN ADDITIVE WHITE GAUSSIAN NOISE (AWGN) COMMUNICATION CHANNELS

RODRIGO ANDRES RODRIGUES FISCHER

MONOGRAFIA DE TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA.

BRASÍLIA, 11 DE DEZEMBRO DE 2020.

FICHA CATALOGRÁFICA

RODRIGUES FISCHER, RODRIGO ANDRES
Study and Performance Analysis of Construction and Decoding Algorithms for Polar Codes in Additive White Gaussian Noise (AWGN) Communication Channels [Distrito Federal] 2020.
xix, 117p., 210 x 297 mm (ENE/FT/UnB, Engenheiro Eletricista, Engenharia Elétrica, 2020).
Monografia de Trabalho de Conclusão de Curso – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Códigos Polares	2. AWGN
3. Algoritmos de Construção	4. Algoritmos de Decodificação
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

FISCHER, R. (2020). Study and Performance Analysis of Construction and Decoding Algorithms for Polar Codes in Additive White Gaussian Noise (AWGN) Communication Channels . Monografia de Trabalho de Conclusão de Curso em Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 117p.

Rodrigo Andres Rodrigues Fischer

Departamento de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedico este trabalho a meu pai Christian e a minha mãe Héden

AGRADECIMENTOS

Eu gostaria primeiramente de agradecer aos meus pais Christian Fischer e Héden Cardoso por todo o carinho, amor e cuidado que sempre recebi, e agradecer também por ter recebido as condições que tornaram possível que eu desenvolvesse meus estudos. Em seguida, agradeço ao meu irmão Davi Fischer por sempre estar ao meu lado e ter sido um grande companheiro de inúmeras aventuras. Agradeço também às incríveis pessoas que pude chamar de avós: Marta Maria, Guaracy, Diana e Rodolfo. Honro esse trabalho ao meu avô Guaracy, que tinha o sonho de ver-me tornar engenheiro, e ao meu avô Rodolfo, que me inspirou a seguir sua profissão.
Gostaria de agradecer também à minha namorada e melhor amiga Hiandra Tomasi por ter compartilhado comigo 5 incríveis, porém trabalhosos, anos de graduação em Engenharia Elétrica. Agradeço também por ter sempre estado ao meu lado nos altos e baixos da vida, sempre me apoiando e me ajudando a passar pelos momentos mais difíceis.
Gostaria de agradecer ao Prof. João Paulo Leite do Dep. de Engenharia Elétrica da UnB por ter me introduzido ao mundo acadêmico e por ter me ajudado a dominar suas ferramentas e particularidades. Agradeço imensamente a sua disposição, paciência e ao suporte que recebi em nossos trabalhos conjuntos. Agradeço também aos Profs. João Luiz, Eduardo Peixoto, Paulo Portela, Geovany Borges, Daniel Café, Francis Arody, Kleber Melo, Lúcio Martins e Leonardo RAX pelo incrível trabalho feito na Universidade de Brasília e pela oportunidade de ter sido seu aluno. Agradeço também aos funcionários do SG11 pela cordialidade e pelo excelente trabalho realizado em manter o prédio no qual passei incontáveis horas durante os últimos anos.
Agradeço imensamente ao Prof. Celius Magalhães do Dep. de Matemática da UnB por todo o apoio, incentivo e oportunidades que recebi, frutos da sua carinhosa e voluntária tutoria. Você me deu valiosas ferramentas que guardo no meu cinto de utilidades, e você me inspira a um dia ensinar com a maestria que você possui.

Agradeço o meu incrível companheiro de inúmeras matérias, trabalhos e monitorias, Diego Neves, por ter me inspirado com sua curiosidade inata e por ter me convidado a participar de inúmeras experiências engrandecedoras durante a graduação.
Agradeço também ao meu amigo de longa data André Seiki por sempre compartilhar comigo conquistas, idéias e curiosidades. Com poucas palavras você diversas vezes abriu minha mente para conceitos e pensamentos poderosos.
Gostaria de agradeçer aos meus colegas do Laboratório de Estruturas de Microondas e Ondas Milimétricas (LEMOM) Gustavo Viana, Josh Peña, Lucas Baião, Bruno Chaves e Natália Viana. Agradeço a vocês pelos inúmeros almoços, jantas no RU e cafezinhos. Nossas teorias discutidas no almoço ainda vão render um Nobel.
Gostaria de agradeçer aos meus amigos Thiago Dantas, Pedro Ferreira, Letícia Almeida, Arthur Dias, Pedro Caio, Marcelo Guedes, Gabriel Lins, Allan Souza, Rodrigo Borba e Felipe Gomes por tornarem minha graduação divertida e por terem me acolhido em seu grupo de amigos. Sem vocês a graduação teria sido monótona.

RESUMO

Título: Estudo e Análise de Desempenho de Algoritmos de Construção e Decodificação de Códigos Polares em Canais de Comunicação com Ruído do Tipo Branco Gaussiano (AWGN)
Autor: Rodrigo Andres Rodrigues Fischer
Orientador: Prof. Dr. João Paulo Leite
Curso de Graduação em Engenharia Elétrica
Brasília, 11 de dezembro de 2020

Propostos em 2009 por Arikan, os códigos polares compõem a primeira família de códigos corretores de erros a provadamente atingir a capacidade de canais de comunicação. Além disso, os códigos polares apresentam esquemas de codificação e decodificação de baixa complexidade. Contudo, a capacidade é atingida apenas quando o tamanho do bloco tende ao infinito, e, além disso, esses códigos apresentam grande latência na decodificação. Na última década, foram feitos esforços para tornar os códigos polares competitivos com a tecnologia estado-da-arte, os códigos LDPC e códigos Turbo, tanto em termos de desempenho quanto em termos de complexidade de implementação.

O presente trabalho apresenta os aspectos básicos da teoria de códigos polares, bem como algumas das técnicas consideradas estado-da-arte de construção de códigos e decodificação que os tornam competitivos com as tecnologias já consolidadas. No campo teórico, é mostrado que a decodificação em ordem natural dos códigos polares são consequência da estrutura de recursão considerada, não da matriz de codificação em si. O trabalho aborda os decodificadores Successive Cancelation (SC) e Successive Cancelation List (SCL). Vertentes de menor latência de ambos os decodificadores são tratadas, como o Simplified SC e Fast-SC, Simplified SCL e SSCL-SPC (Single Parity Check). Além disso, o desempenho desses decodificadores é avaliado com diferentes técnicas de construção, entre elas a construção Bhattacharyya, BEE (Bit Error Evolution), DEGA (Density Evolution with Gaussian Approximation) e, por fim, Modified DEGA. Diversos parâmetros de projeto, como o tamanho da lista, razão sinal ruído de otimização e taxa de código serão também explorados nas simulações de desempenho dos códigos.

Palavras-chave: Códigos Polares, AWGN, Algoritmos de Construção, Algoritmos de Decodificação.

ABSTRACT

Title: Study and Performance Analysis of Construction and Decoding Algorithms for Polar Codes in Additive White Gaussian Noise (AWGN) Communication Channels
Author: Rodrigo Andres Rodrigues Fischer
Supervisor: Prof. Dr. João Paulo Leite
Undergraduate Program in Electrical Engineering
Brasília, December 11th, 2020

Proposed by Arikan in 2009, polar codes were the first family of error correcting codes to provably achieve the communications channel capacity. Also, polar codes present low complexity encoding and decoding schemes. However, channel capacity is only achieved as the block size tends to infinity, and these codes are shown to have large decoding latency. In the last decade, efforts were made in order to make polar codes competitive with state-of-the-art codes, namely LDPC and Turbo codes, both in performance and complexity.

The present work approaches the basic aspects of polar coding theory, as well as some of the state-of-the-art code construction and decoding techniques, which make them competitive to already consolidated technologies. We show that the natural order decoding of polar codes are a consequence of the recursive structure considered to represent the codes, not the encoding matrix itself. This work addresses the Successive Cancellation (SC) and Successive Cancellation List (SCL) decoders. Lower latency versions of both decoding techniques are presented, namely Simplified Successive Cancellation (SSC) and Fast-SC, Siplified SCL and SSCL-SPC. Also, the error performance of these decoders is assessed using different construction methods such as the Bhattacharyya construction, DEGA (Density Evolution with Gaussian Approximation), Modified DEGA and BEE (Bit Error Evolution). Several design settings, such as the list size, design signal-to-noise ratio and code rate will also be explored in the simulations presented.

Keywords: Polar Codes, AWGN, Construction Algorithms, Decoding Algorithms.

SUMMARY

LIST OF FIGURES

LIST OF SYMBOLS

Mathematical symbols

$I(W)$	The symmetric capacity of the B-DMC W
$Z(W)$	The Bhattacharyya parameter of the B-DMC W
$\log(x)$	The natural logarithm of x
$\log_a(x)$	The base a logarithm of x
$\mathbf{A} \otimes \mathbf{B}$	The Kroneker product of matrices **A** and **B**
$\mathbf{A}^{\otimes n}$	The n-th Kroneker power of matrix **A**
$\mathbf{R}_N$	The reverse shuffling matrix
$\mathbf{S}_N$	The shuffling matrix
$\lvert A \rvert$	The cardinality of set A
A^n	The cartesian product $\{(a_0, ..., a_{n-1}) \mid a_i \in A,\, 0 \leq i \leq n-1\}$
W^n	The channel $W^n : \mathcal{X}^n \to \mathcal{Y}^n$ obtained by n parallel uses of W
$W_n^{(i)}$	The i-th split channel $W_n^{(i)} : M \to \mathcal{Y}^n \times \mathcal{X}^{i-1}$
W_n	The W combined channel $W_n : M^k \to \mathcal{Y}^n$

LIST OF ACRONYMS AND ABBREVIATIONS

AWGN	Additive White Gaussian Noise. xi, xiii, 2, 9–11, 14, 21, 43, 45, 46, 48, 50, 63, 64
B-DMC	Binary-input Discrete Memoryless Channel. 8, 15, 16, 30, 31, 33, 43
BCH	Bose–Chaudhuri–Hocquenghem. 18
BEC	Binary Erasure Channel. 8, 9, 14, 15, 23, 43, 44
BEE	Bit Error Evolution. 2, 48
BER	Bit Error Rate. 50, 63, 96
BPSK	Binary Phase-Shift Keying. xi, xiii, 11, 13, 14, 21, 45, 46, 48, 50, 63, 64
BSC	Binary Symmetric Channel. xi, 8, 14, 16, 20, 21, 23–25
CD	Compact Disc. 5, 18
CRC	Cyclic Redundancy Check. 80, 81, 95
DEGA	Density Evolution - Gaussian Approximation. xiii, 2, 45–48, 73–76
DMC	Discrete Memoryless Channel. 7, 10, 24, 25
DVB-RCS2	Digital Video Broadcasting - Return Channel via Satellite - Second Generation. 18, 60, 95
DVB-S2	Digital Video Broadcasting - Satellite - Second Generation. 19
DVD	Digital Video Disc. 18
EB	Exabytes. 1
Fast-SSC	Fast Simplified Successive Cancellation. xiii, 74–76
FEC	Forward Error Correction. 1
FER	Frame Error Rate. 50, 63, 96
i.i.d.	independent and identically distributed. 10
LDPC	Low-Density Parity-Check. 1, 18, 19, 24
LLR	Log-Likelihood Ratio. 13–15, 17, 66, 69
LR	Likelihood Ratio. 13, 34

MAP	Maximum *a Posteriori*. 12, 20–22, 27
ML	Maximum Likelihood. 12, 16, 27, 33, 34, 76, 79, 88, 94
PC	Personal Computer. 1
PSD	Power Spectral Density. 11
QPSK	Quadrature Phase-Shift Keying. xi, 11
QR	Quick Response. 18
REP	REPetition. xiii, 74, 75, 82, 85
RM	Reed-Muller. 18, 19, 33
RS	Reed-Solomon. 18
RV	Random Variable. 34
SC	Successive Cancellation. i, xiii, 2, 24, 27, 33–35, 39, 43, 48, 58, 64, 66, 73, 76, 79, 81, 88, 92, 94–96
SCL	Successive Cancellation List. i, 3, 60, 66, 78, 96
SNR	Signal-to-Noise Ratio. 11, 18, 19, 96
SPC	Single Parity Check. xiii, 75, 82–84
SSC	Simplified Successive Cancellation. i, xiii, 3, 72, 74, 95
SSCL	Simplified Successive Cancellation List. 3, 82, 85, 96
TV	Television. 1
VR	Virtual Reality. 1
WiMAX	Worldwide Interoperability for Microwave Access. 18

NOTATION

iff We represent the sentence "if and only if" by "iff".. 16

vectors Vectors are represented by bold lowercase letters, indicating $\mathbf{x} = (x_0, ..., x_{n-1})$. Another equivalent notation is $x_1^n = (x_1, ..., x_n)$, where the indexing starts at 1. Also, $x_{1,o}^n$ is the vector with all odd entries of x_1^n and $x_{1,e}^n$ is the vector with all even entries of x_1^n.. 27

1 INTRODUCTION

The last 20 years have experienced the rapid evolution of the Internet. Uncountable services such as social media and content streaming, initially available mainly for Personal Computers (PCs), are now available for mobile phones. Also, new streaming modalities such as gaming, Virtual Reality (VR) and high resolution video are being introduced. An immediate consequence of this is the increase of global mobile data traffic: just in the last five years, the global mobile data traffic went from under 10 Exabytes (EB) in 2015 to 50 EB in 2020, with projected traffic of over 200 EB for 2026 [1]. This projection accounts for approximately 100 EB of data traffic in 2G/3G/4G mobile technologies and approximately 100 EB in 5G [1].

Even though we sometimes experience connection loss while using our phones, we certainly don't see any noise while watching videos online, as we used to see in old television. This is made possible through two techniques. The first one is to use digital communications, which allows the ideal regeneration of the information bits transmitted [2]. However, the noise we saw in old Television (TV) didn't disappear. It is still there, and that's why we need the second technique: Forward Error Correction (FEC) that detects and corrects the errors introduced.

Such error correcting techniques are bounded by a fundamental limit imposed by the Noisy-Channel Coding Theorem proved by Shannon in 1948 [3]. However, no technique to provably achieve this limit was known until 2009, when Arikan [4] proposed *polar codes*. These codes take advantage of a phenomenon called *channel polarization*, in which some bits are transmitted with great reliability and others don't. Despite being proven to achieve channel capacity, they do so only by letting the block length go to infinity, and, using the initial definitions proposed by Arikan, these codes show poor error performance when compared to state-of-the-art Turbo and Low-Density Parity-Check (LDPC) codes [5].

The 4G LTE [6] system used Turbo [7] codes as the data channel FEC, while the 5G [8] system uses LDPC [7]. Both these techniques are the state-of-the-art in channel coding and are empirically shown to have capacity achieving properties [7]. Since their discovery, polar codes were improved and refined, in a manner that they are able to achieve comparable performance to these other state-of-the-art techniques [5]. In fact, polar codes are proposed as the FEC scheme in 5G control channels [8]. Polar codes are still an active area of research as it makes way into new systems.

1.1 OBJECTIVES

The first objective of this work is to present a comprehensive review of polar coding theory and state-of-the-art techniques. Three main elements are presented: the basic theory of polar codes, approximate construction methods and efficient decoding algorithms that allow reduced latency. We also provide the reader with the required concepts in communications and coding theory. We expect that the reader is able to gain intuition on the techniques approached as detailed descriptions of the algorithms are presented.

The second objective of this work is to develop our own software implementations of the algorithms reviewed, in order to generate comprehensive simulation results. We used the Python programming language, with parts of the code optimized using the Pythran [9] static compiler. Without this optimization, the simulation campaigns would not have been completed in practical time. We won't get into the details of the code development, however, behind the scenes, there are countless hours spent on code debugging and optimization.

Thirdly, we present the reader with the aforementioned simulation results. We compare the construction methods presented for multiple decoding algorithms and show an analysis on the similarities between the construction methods, which highlights potential future areas of research. We also explore how changing the modulation used causes optimality loss in polar codes and highlight possible workarounds.

As another contribution, we also show that the natural order decoding of polar codes isn't a consequence of the encoding matrix, but of the way we define the recursive structure of polar codes instead. We derive new recursive relations, which are similar to the ones obtained by Arikan [4], for the new recursive structure proposed.

1.2 WORK OVERVIEW

We begin in Chapter 2 by introducing the communication and coding theory required to the development of polar codes. We approach themes such as *discrete memoryless channels*, *Additive White Gaussian Noise (AWGN) channel* and *block codes*. This chapter is organized in a different than usual format: it contains definitions and examples that help illustrate the concepts presented. Also, some results used in later chapters will be derived there.

Chapter 3 contains the fundamental blocks of polar coding theory. Among other topics, we present the Channel Polarization and Polar Coding theorems and introduce the Successive Cancellation (SC) decoding [4]. We then present four approximated methods of code construction, namely the Density Evolution - Gaussian Approximation (DEGA) [10], Modified DEGA [11], Bit Error Evolution (BEE) [11] and Bhattacharyya [11]. We follow by presenting comprehensive simulation results that allow us to compare the different construction methods. We also approach polar systematic encoding and other practical aspects of polar encoding.

Chapter 4 begins by introducing the binary tree decoding algorithm, which sets ground for us to review the Simplified Successive Cancellation (SSC) and Fast-SSC algorithms [12, 13]. Then, we introduce the Successive Cancellation List (SCL) decoding and CRC-concatenated polar codes [5, 14]. Next, we present the Simplified Successive Cancellation List (SSCL) and SSCL-SPC decoding algorithms [15]. We show simulation results that assess the effect of multiple code settings on the error performance and follow by analysing the error performance behaviour of the four construction methods.

Finally, Chapter 5 presents the conclusions and some insights on future works.

2 COMMUNICATION THEORY

2.1 INTRODUCTION

The main goal of this chapter is to introduce fundamental elements that will be used throughout the development of polar coding theory. Here, we explore them in a general context of communications and coding theory; we expect that this approach will help the reader to understand the essence of such concepts and later, when polar codes are introduced in the next chapters, it becomes clear which parts of the theory belong to a more general context and which belong to polar coding theory. Also, this chapter adopts a different than usual format: it is built around definitions and examples, used to illustrate concepts and to derive results that will be used in later chapters.

Section 2.2 introduces the general abstraction of a communications link. Next, Section 2.3 approaches the mathematical modelling of the communications channel: we review concepts such as *discrete memoryless channels*, *Binary Erasure Channels (BEC)*, *AWGN channels*, *decision rules*, *likelihood ratios*, and introduce the *symmetric capacity* and *Bhattacharyya parameter* definitions. Section 2.4 presents a brief history of error correcting codes and goes through some of their fundamental definitions, such as *block codes*, *linear codes* and *coset codes*. Lastly, Section 2.5 presents the conclusions.

2.2 COMMUNICATIONS LINK

We begin by defining *communication* as the process of transferring information between two points in space or time. It is easy to think of examples were information is transferred between two points in space: when someone sends a text message to someone else far away or when the radio station broadcasts FM radio to thousands of receivers at different places simultaneously. However, sometimes we fail to see that our modern everyday life depends heavily on communication systems, and that we are constantly transferring information in time, space or both at once. When we store information we are essentially transferring information from the past to the present. When one searches the web for something using their smartphones they are retrieving information stored some time ago, on a server or computer far away, to the present time and location.

We define a *communications link* as the collection of procedures performed in order to effectively transfer information between two points in space or time. The following steps show to be sufficient to describe a communications link [2]:

a) some information is generated at the *transmitter*;

b) the information is inserted into a signal that must be compatible with transmission on a particular physical medium; we call this process *modulation*;

c) the signal is propagated throughout the physical medium, called the *channel*; remember, this medium can be both space and/or time;

d) the signal is received at the *receiver* and the information is recovered from the received signal; this process is called *demodulation*;

e) the information is consumed at the receiver.

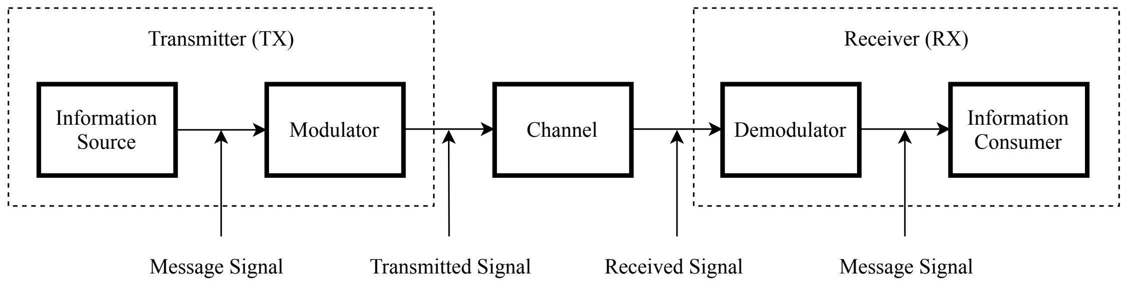


Figure 2.1 – An illustration of a generic communications link.

We can say that a communications link begins at the generation of information at the transmitter and ends when this information is consumed at the receiver. The steps above are illustrated on Figure 2.1.

As expected, the transmitted signal is susceptible to impairments that the channel may introduce in a way that the recovery of the information from the received signal doesn't always occur perfectly. Examples of such impairments are the interference from other signals propagating in the same medium, thermal noise generated at the receiver or even a scratch on a Compact Disc (CD). As an attempt to overcome such impairments, we need to introduce redundancy on the information, in a process called *channel coding*. A brief introduction to this topic and the hereby adopted mathematical modeling of the channel will be addressed in Sections 2.4 and 2.3 respectively. For further information on these topics and on communications systems, refer to [2, 7, 16].

2.3 MATHEMATICAL MODELLING OF THE CHANNEL

We can approach the communications problem using many abstraction levels, one of which is shown in Figure 2.1. Note that this level of abstraction does not dictate how the transmitted signal should be represented. The complete description of the signal in all points of space and time within its propagation medium is often unnecessary. Instead, we model the communications link with statistical quantities that approximate or simplify the representation of the real physical quantities [2, 16].

Imagine that a communications link is used to transmit bits from the source to the consumer. In order to test this system, you program the source to transmit a known sequence of bits, and you observe what is being received. Now, regardless of how the actual signal transmission occurs, you are only observing bits; you notice that some of them are received properly, but some others don't. You then start to compute statistics of how often bits are received incorrectly. However, you realize that simply computing how often errors occur may not be sufficient in order to describe your link: the probability of error can change over time or past bits or past errors can affect the result of future bits.

We then generalize the idea of this previous thought experiment. Suppose we only have access to the message signal and that it is made of *symbols* from a certain alphabet $\mathcal{X}$. At the output of the channel we have symbols from some other alphabet $\mathcal{Y}$. Why the alphabet $\mathcal{Y}$ may be different from $\mathcal{X}$ will be clarified using examples. This simplified setup is shown in Figure 2.2.

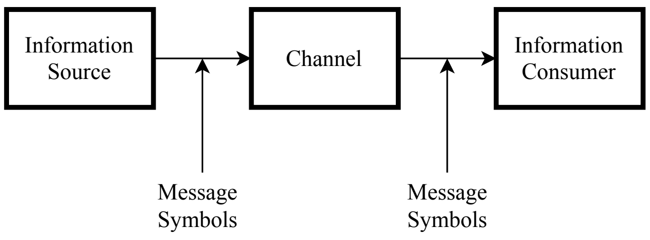


Figure 2.2 – Simplified communications link assuming the symbols generated are transmitted directly through the channel.

2.3.1 Discrete Memoryless Channels

As is expected, the mathematical modelling of the channel is based on statistics and the definition of a *discrete channel* itself is made only by defining a collection of probability measure functions.

Definition 2.1 (Discrete Channel)**.** Given the finite sets $\mathcal{X}$ and $\mathcal{Y}$, respectively called *input alphabet* and *output alphabet*, and an arbitrary set S called *set of states*, a *discrete channel* is a system of probability measure functions

$$p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n; s),$$

with

$$
\begin{gathered}
\alpha_1, ..., \alpha_n \in \mathcal{X}, \\
\beta_1, ..., \beta_n \in \mathcal{Y}, \\
s \in S, \\
n = 1, 2, ..., \qquad (2.1)
\end{gathered}
$$

that is, a function system that satisfies

1. $p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n; s) \geq 0$ for all $n,\ \alpha_1, ..., \alpha_n,\ \beta_1, ...,\ \beta_n,\ s$;
2. $\sum_{\beta_1,...,\beta_n} p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n; s) = 1$ for all $n,\ \alpha_1, ..., \alpha_n,\ s$.

We interpret $p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n; s)$ as the probability of receiving the symbol sequence $\beta_1, ..., \beta_n$ at the channel output after applying the sequence $\alpha_1, ..., \alpha_n$ on its input, given the channel was at state s just before α_1 was transmitted.

On this work we are particularly interested on channels that are said to be *memoryless*. This is because a core assumption used on the theory of polar codes is that the channels used are memoryless.

Definition 2.2. A discrete channel is said to be memoryless if it satisfies

1. the probability $p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n; s)$ doesn't depend on the state s and can be written as $p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n)$;
2. $p_n(\beta_1, ..., \beta_n \mid \alpha_1, ..., \alpha_n) = \prod_{i=1}^{n} p_1(\beta_i \mid \alpha_i)$.

We can see that on memoryless channels the transition probabilities depends only on its input and output symbols, therefore there is no initial condition or internal state on the channel that changes over time. Also, the transmission of multiple symbols sequentially can be treated symbol by symbol individually: each symbol transmission is independent from the others.

Then, for a Discrete Memoryless Channel (DMC), knowing the probabilities $p_1(\beta \mid \alpha)$ is sufficient to characterize the transmission of any sequence. We adopt therefore the following notation.

Definition 2.3. $W : \mathcal{X} \to \mathcal{Y}$ represents a DMC W with input alphabet $\mathcal{X}$ and output alphabet $\mathcal{Y}$. We denote $p_1(y \mid x)$ by $W(y \mid x)$, $y \in \mathcal{Y}$, $x \in \mathcal{X}$.

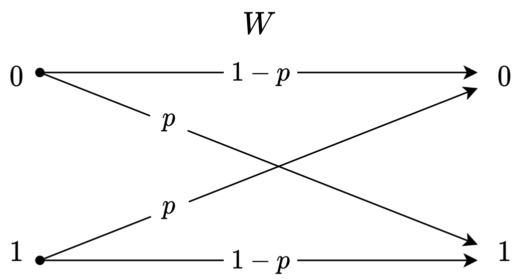


Figure 2.3 – Binary symmetric channel W with crossover probability p.

Definition 2.4 (Binary-Input Channel)**.** We say a channel $W : \mathcal{X} \to \mathcal{Y}$ has binary-input if $|\mathcal{X}| = 2$. Without loss of generality, we will represent the symbols of $\mathcal{X}$ as $\{0, 1\}$.

Definition 2.5 (Binary Symmetric Channel)**.** A Binary-input Discrete Memoryless Channel (B-DMC) W is said to be *symmetric* if there exists a permutation π such that

1. $\pi = \pi^{-1}$;
2. $\forall y \in \mathcal{Y}, W(y \mid 1) = W(\pi(y) \mid 0)$.

In a symmetrical channel, for each transition $W(y \mid 1)$ there exists a symbol $\pi(y)$ such that $W(\pi(y) \mid 0) = W(y \mid 1)$. Also, it is easy to see that $W(\pi(y) \mid 1) = W(\pi(\pi(y)) \mid 0) = W(y \mid 0)$.

Example 2.1 (Binary Symmetric Channel)
If

1. $\mathcal{X} = \mathcal{Y} = \{0, 1\}$,
2. $W(0|1) = W(1|0) = p$,
3. $W(0|0) = W(1|1) = 1 - p$,

we say that W is a Binary Symmetric Channel (BSC) with crossover probability p. A visual representation of this channel is shown in Figure 2.3.

Definition 2.6 (Binary Erasure Channel)**.** A B-DMC W is said to be a Binary Erasure Channel (BEC) if for each $y \in \mathcal{Y}$ either

$$W(y \mid 0)W(y \mid 1) = 0$$

or

$$W(y \mid 0) = W(y \mid 1).$$

In the latter case, y is said to be an *erasure symbol*.

We can see that erasure symbols have equally likely transitions from 0 or 1. For non-erasure symbols, it must be true that the transition from 1 or 0 has probability zero.

Example 2.2 (Binary Erasure Channel)
If

1. $\mathcal{X} = \{0, 1\}, \mathcal{Y} = \{0, 1, e\}$,
2. $W(1|0) = W(0|1) = 0$,
3. $W(0|0) = W(1|1) = 1 - \epsilon$,
4. $W(e|0) = W(e|1) = \epsilon$,

then W is a BEC with erasure probability ϵ. A visual representation of this channel is shown in Figure 2.4.

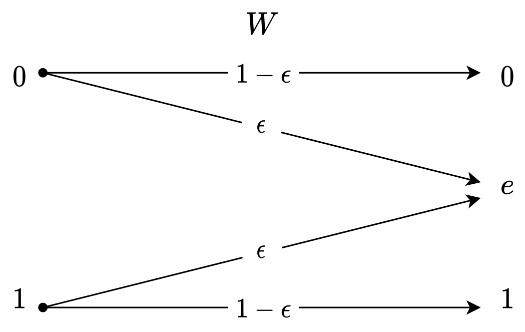


Figure 2.4 – Binary erasure channel W with erasure probability ϵ.

We can think of the BEC on Example 2.2 as a channel where there is no crossover probability. In this channel, there is no chance a 0 at the input will generate a 1 at the output, and vice versa.

2.3.2 AWGN Channel

The channel definition at Section 2.3.1 is a mathematical abstraction that doesn't reveal any physical aspect of what may be happening at a real communications link. When actual physical channel impairments are concerned, a basic model used widely in communications theory is the Additive White Gaussian Noise (AWGN) channel. We can define this kind of noise by being:

a) *additive*, since it adds to the signal and other channel impairments;

b) *white*, meaning its power is evenly distributed in all of the signal frequency range;

c) *Gaussian*, meaning it follows a normal distribution. In addition, we consider that this distribution has zero mean.

A common technique to transmit signals over space is using linear modulation over passband channels [2]. Using this technique, one can transmit different information on two time-orthogonal signals called in-phase (I) and quadrature (Q) components, which can be recovered independently at the receiver and can be represented as the real and imaginary parts of complex numbers. There are modulation schemes that have a dimension higher than two, by using signals orthogonal in frequency, as an example. However, we will focus on two-dimensional modulation.

Definition 2.7. The AWGN channel will be described using the input alphabet

$$\mathcal{X} = \{s_0, s_1, ..., s_{M-1}\} \subset \mathbb{C}$$

and output alphabet

$$\mathcal{Y} = \mathbb{C}.$$

Given $x \in \mathcal{X}$ at the channel input, the output is $y = x + N \in \mathcal{Y}$, where $N = n_I + j \cdot n_Q$ and n_I, n_Q are independent and identically distributed (i.i.d.) samples from a zero mean normal distribution with variance σ, that is, $n_I, n_Q \sim N(0, \sigma)$.

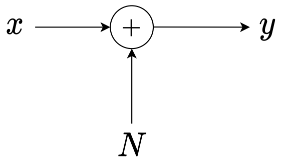


Figure 2.5 – A representation of the AWGN channel.

A representation of the AWGN channel can be seen on Figure 2.5. Definition 2.7 creates an algebraic relation between the output and the input of the channel, in contrast with the pure statistical Definition 2.1 of DMCs. Nevertheless, one can obtain probability density functions $p_1(y \mid x), x \in \mathcal{X}, y \in \mathcal{Y}$ defined by [2]

$$p_1(y \mid x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\|y - x\|^2}{2\sigma^2}\right). \tag{2.2}$$

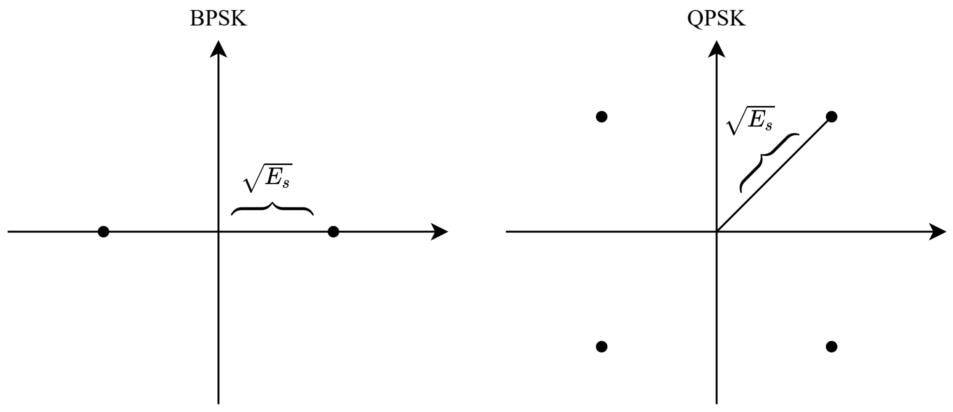


Figure 2.6 – BPSK and QPSK constellations.

If we limit the symbols in $\mathcal{X}$ to the real numbers and $\mathcal{Y}$ also to $\mathbb{R}$, we have that [2]

$$p_1(y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|y-x\|^2}{2\sigma^2}\right). \tag{2.3}$$

By our definition, the AWGN channel is memoryless.

An important measure that is related to the transmitted signal power is the *symbol energy*.

Definition 2.8. The symbol energy E_s is defined as

$$E_s = \frac{1}{M}\sum_{i=0}^{M-1} \|s_i\|^2.$$

One can model the AWGN in frequency as a random process with constant Power Spectral Density (PSD) of $N_0/2$ [2]. It can be shown that after each component of the signal is retrieved the variance per dimension is $\sigma = N_0/2$. There are many ways to measure the Signal-to-Noise Ratio (SNR), but a common way that of doing so is using the symbol energy and the noise PSD N_0, defined as

$$\frac{E_s}{N_0}. \tag{2.4}$$

The input alphabet $\mathcal{X}$, also called *constellation*, is made of symbols s_i that are points on the complex plane. Two common constellations are shown in Figure 2.6, namely Binary Phase-Shift Keying (BPSK) and Quadrature Phase-Shift Keying (QPSK). At the lowest level, however, these symbols usually also represent bits, and so $M = 2^n$ and each sequence of n bits is mapped to a symbol s_i, as exemplified in Figure 2.7.

$$\begin{tikzcd}\underbrace{01...1}_{n\text{ bits}} \arrow[rrr, "\text{symbol mapping}"] & & & s_i\end{tikzcd}$$

Figure 2.7 – An example of a symbol mapping operation.

2.3.3 Decision Rules

Upon receiving a symbol y at the output of the channel, one needs to decide which symbol $s_i \in \mathcal{X} = \{s_0, ..., s_{M-1}\}$ was transmitted, or how likely was s_i transmitted. In order to accomplish this, we define the Maximum *a Posteriori* (MAP) and Maximum Likelihood (ML) decision rules.

Definition 2.9. A decision rule is a function $\delta : \mathcal{Y} \to \mathcal{X}$ which maps the received symbols $y \in \mathcal{Y}$ to an input symbol $x \in \mathcal{X}$.

Definition 2.10 (ML Decision Rule)**.** The ML decision rule is defined as

$$\delta_{\text{ML}}(y) = \underset{i \in \{0,...,M-1\}}{\arg\max}\; p(y \mid x = s_i), \tag{2.5}$$

where $p(y \mid x = s_i)$ is the conditional probability of receiving y given s_i was transmitted in the case the output alphabet $\mathcal{Y}$ is discrete or, if $\mathcal{Y}$ is continuous, it is the conditional probability density.

Definition 2.11 (MAP Decision Rule)**.** The MAP decision rule is defined as

$$\begin{aligned}\delta_{\text{MAP}}(y) &= \underset{i \in \{0,...,M-1\}}{\arg\max}\; p(x = s_i \mid y) \\ &= \underset{i \in \{0,...,M-1\}}{\arg\max}\; \tau_i p(y \mid x = s_i),\end{aligned} \tag{2.6}$$

where $\tau_i = p(x = s_i)$ is known as the *prior* probability that $x = s_i$ and $p(x = s_i \mid y)$ is the conditional probability that $x = s_i$ given the *a posteriori* observation y. The inversion of the conditional was made using the Bayes rule.

The MAP rule accounts for the case where not all symbols are equally likely to appear on the input. However, if $\tau_i \equiv 1/M$ the MAP and ML rules are equivalent. It can be shown [2] that the MAP rule produces the minimum average error probability.

Definition 2.12 (Likelihood Ratio)**.** When bits are mapped to the symbols $s_i \in \mathcal{X}$, we can define the Likelihood Ratio (LR) of the i-th bit b_i on the bit sequence $(b_0, ..., b_{n-1})$ as

$$l_{b_i}(y) = \frac{p(b_i = 0 \mid y)}{p(b_i = 1 \mid y)},$$

where $p(b_i = j \mid y)$ is the conditional probability of $b_i = j$ given y was observed at the channel output.

Using the Bayes rule, we get

$$\begin{aligned} l_{b_i}(y) &= \frac{p(b_i = 0 \mid y)}{p(b_i = 1 \mid y)} \\ &= \frac{p(y \mid b_i = 0)\tau_0}{p(y \mid b_i = 1)\tau_1}. \end{aligned}$$

Note that, after swapping the conditionals, p may represent a probability density or a probability function.

Definition 2.13 (Log-Likelihood Ratio)**.** For algebraic convenience, we define the Log-Likelihood Ratio (LLR) as

$$\begin{aligned} L_{b_i}(y) &= \log\left(l_{b_i}(y)\right) \\ &= \log\left(\frac{p(y \mid b_i = 0)}{p(y \mid b_i = 1)}\right) + \log\left(\frac{\tau_0}{\tau_1}\right) \\ &= L_{b_i}^{\text{posterior}}(y) + L^{\text{prior}}. \end{aligned} \tag{2.7}$$

From Eq. (2.7) one can see that for equal priors the sign of $L_{b_i}(y)$ shows which conditional probability is bigger, and that its magnitude shows how big one probability is when compared to the other. This is a measure of reliability of an observation, and this extra information besides the signal of the LLR will be used later when bits are transmitted using redundancy.

We can compute $p(y \mid b_i = j)$ by summing probabilities on all symbols on which $b_i = j$

$$p(y \mid b_i = j) = \frac{1}{\tau_j} \sum_{s_k \mid b_i = j} p(y \mid s_k) p(s_k). \tag{2.8}$$

Example 2.3 (BPSK Transmission)

Consider the BPSK transmission under an AWGN channel with noise variance σ. Let $\mathcal{X} = \{-1, 1\}$ with a mapping $x = (-1)^b$ when the bit b is transmitted, $\mathcal{Y} = \mathbb{C}$ and

$\tau_0 = \tau_1 = 1/2$. If $y = r + j \cdot q$ we have that

$$\begin{aligned} l(y) &= \frac{\exp\left(-\frac{(r-1)^2+q^2}{2\sigma^2}\right)}{\exp\left(-\frac{(r+1)^2+q^2}{2\sigma^2}\right)} \\ &= \exp\left(\frac{2r}{\sigma^2}\right) \\ L(y) &= \frac{2r}{\sigma^2}. \end{aligned} \tag{2.9}$$

From this example, we can see that the output likelihood for the BPSK AWGN depends on a channel characteristic, namely the noise variance σ^2. If we observe $r = 50$ with $\sigma = 1$, then $L(y) = 100$, which would show that with a great reliability $x = 0$. However, note that it is extremely unlikely to receive $r = 50$ both when $x = 0$ or when $x = 1$, but the ratio of the conditional densities is very large.

Example 2.4 (BSC)
For a BSC with $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ and $\tau_0 = \tau_1 = 1/2$,

$$\begin{aligned} l(0) &= \frac{1-p}{p}, \\ l(1) &= \frac{p}{1-p}. \end{aligned} \tag{2.10}$$

It can also be seen on Example 2.4 that the LLRs depend on a channel characteristic, namely p.

Example 2.5 (BEC)
For a BEC W with $\mathcal{X} = \{0, 1\}$, $\mathcal{Y} = \{0, 1, e\}$ and $\tau_0 = \tau_1 = 1/2$,

$$\begin{aligned} l(0) &= \frac{1-\epsilon}{0} && = \text{inf}, \\ l(1) &= \frac{0}{1-\epsilon} && = 0, \\ l(e) &= \frac{\epsilon}{\epsilon} && = 1. \end{aligned} \tag{2.11}$$

Using LLR,

$$\begin{aligned} L(0) &= \text{inf}, \\ L(1) &= -\text{inf}, \\ L(e) &= 0. \end{aligned} \tag{2.12}$$

Example 2.5 shows that for a BEC the LLR values polarize towards $-\inf$ and $\inf$ for non-erasure symbols, since $W(y \mid 0)W(y \mid 1) = 0$, and that for erasure symbols one can extract no information of whether $x = 0$ or $x = 1$, since $W(y \mid 0) = W(y \mid 1)$.

2.3.4 Channel Capacity and the Bhattacharyya Parameter

An important quantity related to a communications channel is its *capacity*. Shannon showed in his famous 1948 paper [3] that the transmission of information bits can be achieved with arbitrarily small probability of error, given that the transmission rate is below this quantity.

Definition 2.14 (Mutual Information)**.** Given two discrete random variables X, Y with sample spaces $\mathcal{X}, \mathcal{Y}$, their mutual information is calculated as

$$I(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p(x,y) \log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right). \tag{2.13}$$

For continuous variables, the corresponding sums in Definition 2.14 should be replaced by integrals. Note that if X, Y are independent, $I(X;Y) = 0$, meaning that Y doesn't convey any information on X.

Definition 2.15 (Channel Capacity)**.** Consider a memoryless channel modeled by $p(y \mid x)$, $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. The channel capacity is defined as

$$C = \max_{p(x)} I(X;Y), \tag{2.14}$$

where $p(x)$ is the marginal distribution of X.

Using the properties that $p(x,y) = p(x)p(y \mid x)$ and $p(y) = \sum_{x \in \mathcal{X}} p(x,y)$, we can rewrite the mutual information as

$$I(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p(y \mid x)p(x) \log_2 \left(\frac{p(y \mid x)}{\sum_{x \in \mathcal{X}} p(x)p(y \mid x)} \right). \tag{2.15}$$

By assuming a binary-input channel with equal priors, we get the following definition.

Definition 2.16 (Symmetric Capacity)**.** Given a B-DMC $W : \mathcal{X} \to \mathcal{Y}$, we define the symmetric capacity as

$$I(W) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} \frac{1}{2} W(y \mid x) \log_2 \left(\frac{W(y \mid x)}{\frac{1}{2} W(y \mid 0) + \frac{1}{2} W(y \mid 1)} \right). \tag{2.16}$$

The symmetric capacity is equal to the Shannon capacity in Definition 2.15 when W is symmetric.

Definition 2.17 (Bhattacharyya Parameter)**.** Given a B-DMC $W : \mathcal{X} \to \mathcal{Y}$, we define the Bhattacharyya parameter as

$$Z(W) = \sum_{y \in \mathcal{Y}} \sqrt{W(y \mid 0) W(y \mid 1)}. \tag{2.17}$$

The Bhattacharyya parameter is an upper bound on the probability of ML decision error when W is used once to transmit a 0 or a 1. The following definition shows the relationship between $I(W)$ and $Z(W)$.

Proposition 2.1. For any B-DMC W, we have

$$I(W) \geq \log_2 \frac{2}{1 + Z(W)} \tag{2.18}$$

$$I(W) \leq \sqrt{1 - Z(W)^2}. \tag{2.19}$$

This proposition makes clear the intuition that $I(W) \approx 1$ if and only if $Z(W) \approx 0$, and $I(W) \approx 0$ if and only if $Z(W) \approx 1$. From this point, we define "iff" as "if and only if".

2.4 INTRODUCTION TO ERROR CORRECTING CODES

The modern communications and information theory is considered to be founded by Claude Shannon in his 1948 paper *A Mathematical Theory of Communication* [3]. There, he showed that there is a fundamental limit on how many information symbols, or bits, one can transmit through a noisy channel per unit of time with an arbitrarily small probability of error. However, practical applications only approached this limit in the early 1990s.

As it was seen on the previous sections, the transmission of information through a channel is prone to errors. In order to avoid such errors, one can transmit information with redundancy. A simple example is, instead of transmitting the bit 0 or 1 through a BSC, transmit the triple 000 or 111. By doing this, one could assume, upon receiving 001, that the sequence

000 was sent.

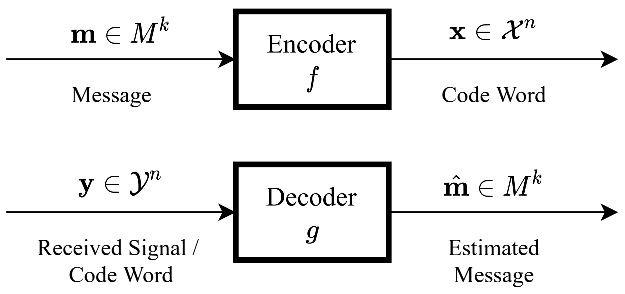

$\mathbf{m} \in M^k$
Message
Encoder f
$\mathbf{x} \in \mathcal{X}^n$
Code Word
$\mathbf{y} \in \mathcal{Y}^n$
Received Signal / Code Word
Decoder g
$\hat{\mathbf{m}} \in M^k$
Estimated Message

Figure 2.8 – Diagram representing the block encoding/decoding operations.

The process of generating the redundant sequence is called *encoding*, and the estimation of the message from the received sequence is called *decoding*. In this introduction, we are interested in *block codes*, codes in which blocks of message symbols are encoded into blocks of *code word* symbols. Block codes are memoryless: the encoding/decoding operations are independent between blocks. This process can be represented in general as seen in Figure 2.8.

The encoding operation can be represented by an injective function $f : M^k \to \mathcal{X}^n$ that maps the message sequences composed of symbols from M into sequences of encoded symbols from $\mathcal{X}$. We can call the image of the encoding function f as the code word set $C \subset A^n$. Also, the decoding operation can be seen as a function $g : \mathcal{Y}^n \to M^k$ that maps the received signal into the message set. Note that in Figure 2.8 the decoder input can be either a sequence from $\mathcal{X}^n$ or a signal directly from the channel output, such as LLR values in $\mathbb{R}^n$. As seen before, the LLR values convey more information about the transmitted bit, such as the reliability of the estimate, and this information can be used by the decoder to make a better estimate of the message.

Definition 2.18. A block code with k information bits, $n - k$ redundancy bits and block size n is represented by (n, k). The code rate is defined as

$$R = \frac{k}{n}. \tag{2.20}$$

The code rate is a measure of how much redundancy is inserted into the encoded block when compared to the message length.

Example 2.6

Let $M = \{a, b\}$ and $C = \{000, 111\}$, with the mapping $a \to 000, b \to 111$. We can see that $\mathcal{X} = \{0, 1\}$ and $\mathcal{X}^n$ has eight elements, but only two are valid code words, which enables the decoder to detect errors. We are treating a, b and $0, 1$ as symbols only, and there is no explicit mathematical relationship between these symbols. Also, without the knowledge of the channel, one cannot determine how a received message like 010 should be decoded. Figure 2.9 shows all the possible received sequences in a cube lattice.

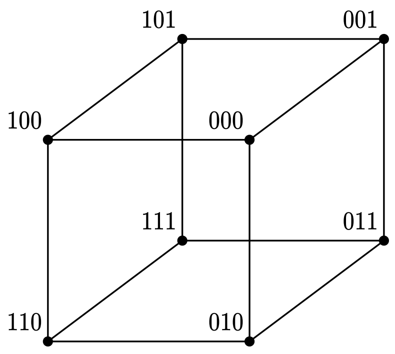


Figure 2.9 – 3-bit binary cube.

Since the code word length is bigger than the message length, more symbols will be transmitted in order to transmit one information bit. Then, by measuring the SNR using only the symbol energy one would not account that more energy is being used. We then define the bit energy per N_0 ratio as

$$\frac{E_b}{N_0} = \frac{1}{R}\frac{E_s}{N_0}. \tag{2.21}$$

2.4.1 A Brief History

The story of error correcting codes dates back to 1950 when Richard Hamming proposed the Hamming codes as an attempt to automatically correct errors introduced in punched cards used in computers at that time. However, this technique couldn't approach the limit proposed by Shannon and the search for better performing codes continued. Later in 1954 the Reed-Muller (RM) codes were proposed and further used in deep-space communication. Around 1959-60, the Bose–Chaudhuri–Hocquenghem (BCH) codes were invented, which are still used until the present time in satellite communications, CDs, Digital Video Discs (DVDs) and bar codes. Around the same time, the Reed-Solomon (RS) codes were introduced and were subsequently used in CDs, DVDs, Blu-ray discs and Quick Response (QR) codes. It wasn't until 1993 when the turbo codes were introduced that practical codes could approach the Shannon capacity. Turbo codes are used in 3G and 4G mobile telephony, DVB-RCS2 satellite communications and WiMAX wireless communications standards. Later on, in 1996, a family of codes developed by Robert Gallager in 1963 in his doctoral dissertation were rediscovered: the Low-Density Parity-Check (LDPC) codes. At the time of their discovery, LDPC codes were impractical to implement. However, this technique found usages in DVB-S2 satellite communications standard and in the data channel of the recent mobile 5G standard.

Even though LDPC and turbo codes have shown capacity-achieving capabilities in practical applications [7, 16], they are not mathematically proven to do so and suffer from error floors at high SNR [7, 16]. Over seventy years after Shannon predicted the existence of a capacity achieving codes, Arikan [4] proposed polar codes as provably capacity-achieving codes, which are closely related to RM codes. The last decade experienced a large amount of research regarding this technique, which allowed polar codes to present comparable performance to LDPC codes [5].

2.4.2 Linear Codes

It is not imperative that $\mathcal{X}^n$ or $\mathcal{X}$ have an algebraic structure that allows elements of these sets to be operated with others using addition and/or multiplication operations. However, by introducing such structure to these sets one can create more efficient mappings and encoding/decoding algorithms.

Definition 2.19 (Linear Codes)**.** Let K be a finite field and K^k, K^n be vector spaces with element-wise addition and multiplication. A linear code maps messages from the message space K^k to the code word space $C \in K^n$, $k \leq n$, by multiplying the message **u** $\in K^k$ by a generator matrix **G**, obtaining **x** $=$ **uG**.

Example 2.7
A code does not have to be binary. Consider $K = \{0, 1, 2\}$ with modulo 3 addition and multiplication. If

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.22}$$

then

$$K^k = \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)\} \tag{2.23}$$

and

$$\begin{aligned} C = \{&(0,0,0),(1,2,1),(2,1,2),(1,1,0),(2,0,1), \\ &(0,2,2),(2,2,0),(0,1,1),(1,0,2)\}. \end{aligned} \tag{2.24}$$

Definition 2.20 (Coset Codes)**.** Given a linear code C over a finite field K generated by $\mathbf{G}$ and $\mathbf{x} \in K^n$, we define $\mathbf{x} + C \triangleq \{\mathbf{x} + \mathbf{y} | \mathbf{y} \in C\}$ as a $\mathbf{G}$-coset code. It is easy to see that if $\mathbf{x} \in C$ then $\mathbf{x} + C = C$, since C is subspace.

Example 2.8

Continuing Example 2.7, let $\mathbf{x} = (1, 1, 1)$. Then

$$\begin{aligned}\mathbf{x} + C = \{&(1,1,1),(2,0,2),(0,2,0),(2,2,1),(0,1,2),\\ &(1,0,0),(0,0,1),(1,2,2),(2,1,0)\}.\end{aligned} \tag{2.25}$$

Since $\mathbf{x} + C$ only contains $(0, 0, 0)$ if $\mathbf{x} \in C$, we can see that $\mathbf{x} + C$ won't always be a subspace of K^n.

2.4.3 Decoding and Distances

When we introduced redundancy in Example 2.6, we represented each message word with an amount of symbols greater than the necessary to list all possible messages. This means that we introduced extra possible received words that don't correspond to any valid input message. Figure 2.9 shows every sequence placed in vertices of a cube in a manner that sequences connected by an edge have only one bit of difference, sequences separated by square diagonals differ by two bits and sequences separated by cube diagonals differ by three bits.

A possible decoding strategy would be to choose the code word closest to the received sequence. It will be shown later that for BSC with equal priors this corresponds to MAP decoding. In fact, the notion of distance is at the core of the decoding problem.

Definition 2.21 (Hamming Distance)**.** The Hamming distance is the number of indexes on which two sequences of symbols $\mathbf{x}, \mathbf{y}$ are different. More formally, we define the Hamming distance of two sequences $\mathbf{x} = (x_0, ..., x_{n-1})$ and $\mathbf{y} = (y_0, ..., y_{n-1})$ of K^n as

$$d_H(\mathbf{x}, \mathbf{y}) = |\{i | x_i \neq y_i\}|, \tag{2.26}$$

where $|A|$ is the number of elements in set A. Also, the Hamming weight of a sequence is defined by

$$d_H(\mathbf{x}, 0), \tag{2.27}$$

when $x \in \mathcal{X} \subset \mathbb{N}$.

Example 2.9 (Block Codes in BSC Channel)
Consider $\mathcal{X} = \{0, 1\}$ and $f : \mathcal{X}^k \to \mathcal{X}^n$ an encoding function with code word set C. Then, a code word $\mathbf{x} \in C$ is transmitted though a BSC channel with crossover probability p and $\mathbf{y} \in \mathcal{X}^n$ is received at the output of the channel. Assuming equal priors, each code word $\mathbf{x} \in C$ is equally likely to happen, and the MAP rule is

$$\delta_{\text{MAP}}(\mathbf{y}) = \arg\max_{\mathbf{x} \in C} p(\mathbf{y} \mid \mathbf{x}). \tag{2.28}$$

Given $\mathbf{x} \in C$, we have that $d_H(\mathbf{y}, \mathbf{x})$ is equal to the number of different entries, and, assuming $\mathbf{x}$ was transmitted, it is also the amount of errors. Then,

$$p(\mathbf{y} \mid \mathbf{x}) = p^{d_H(\mathbf{x},\mathbf{y})}(1-p)^{n-d_H(\mathbf{x},\mathbf{y})}. \tag{2.29}$$

Assuming $p < 1/2$, we have that $p^i(1-p)^{n-i} \geq p^j(1-p)^{n-j}$ if $i \leq j$, and the MAP rule reduces to

$$\delta_{\text{MAP}}(\mathbf{y}) = \arg\min_{\mathbf{x} \in C} d_H(\mathbf{y}, \mathbf{x}). \tag{2.30}$$

Following the steps in Example 2.9, we can show that if somehow $p > 1/2$, then the MAP rule would reduce to finding the code word with the greater distance. Also, if the crossover probability was not the same for 0 and 1 transmission, this minimum distance rule wouldn't coincide with the MAP rule.

Example 2.10 (Block Codes in AWGN Channel)
Consider $\mathcal{X} = \{0, 1\}$ and $f : \mathcal{X}^k \to \mathcal{X}^n$ an encoding function with code word set C. Then, a code word $\mathbf{x} \in C$ is transmitted though an AWGN channel with variance $\sigma = N_0/2$ using BPSK mapping and $\mathbf{y} \in \mathbb{R}^n$ is received at the output of the channel. Assuming equal priors, each code word $\mathbf{x} \in C$ is equally likely to happen, and the MAP rule is

$$\delta_{\text{MAP}}(\mathbf{y}) = \arg\max_{\mathbf{x} \in C} p(\mathbf{y} \mid \mathbf{x}). \tag{2.31}$$

Given $\mathbf{x} \in C$, let $\mathbf{z} \in \{-1, 1\}^n$ be the mapped version of $\mathbf{x}$, using $0 \to 1$ and $1 \to -1$

and $C_{\mathrm{B}} \subset \{-1, 1\}^n$ be the set of the mapped code words. We have that

$$\begin{aligned} p(\mathbf{y} \mid \mathbf{x}) = p(\mathbf{y} \mid \mathbf{z}) &= \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - z_i)^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\sum_{i=0}^{n-1}(y_i - z_i)^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\|\mathbf{y} - \mathbf{z}\|^2}{2\sigma^2}\right). \end{aligned}$$

Then, the MAP rule reduces to

$$\delta_{\mathrm{MAP}}(\mathbf{y}) = \underset{\mathbf{z} \in C_{\mathrm{B}}}{\arg\min} \|\mathbf{y} - \mathbf{z}\|^2. \tag{2.32}$$

Once again, the MAP rule could be expressed in terms of distances. However, when the message length k is big enough, testing all the distances of the 2^k possible code words is impractical. The main challenge of coding theory is to find a mathematical encoding rule f that generates a set C with known properties and structure that allows simpler decoding other than exhaustion of all possibilities. Besides allowing such efficient decoding algorithms, the structure of C must allow good correcting performance. On Example 2.6, we deliberately chose 111 and 000 as code words in order to maximize the distance between them. If 000 and 010 were chosen instead, with only one error one could falsely detect the other code word.

Given a generator matrix $\mathbf{G}$, there is a procedure for finding the smallest Hamming distance between code words [7]. However, for large enough message length k, it is impractical to perform such procedure for an arbitrary matrix $\mathbf{G}$.

Shannon's fundamental theorem [3] shows that an arbitrarily small probability of error is achievable through a large enough code length n. The following example shows an intuitive reasoning on why larger codes are able to show better error performance.

Example 2.11

Let $\mathcal{X} = \{0, 1\}$ be the message and code word alphabet of a code $f : \mathcal{X}^k \to \mathcal{X}^n$ with code word set C. We have 2^k possible messages, 2^k possible code words and 2^n possible received sequences. Lets have a look at the ratio

$$\frac{|C|}{|\mathcal{X}^n|} = \frac{|\mathcal{X}^k|}{|\mathcal{X}^n|} = \frac{2^k}{2^n} = 2^{k-n}. \tag{2.33}$$

If the code rate

$$R = \frac{k}{n} \tag{2.34}$$

is fixed, then

$$\frac{C}{|\mathcal{X}^n|} = 2^{n(R-1)} \tag{2.35}$$

and, as $n \to \infty$, assuming $R < 1$,

$$\frac{C}{|\mathcal{X}^n|} \to 0. \tag{2.36}$$

This means that the set C gets smaller and smaller when compared to $\mathcal{X}^n$ and the code words in C can be spread out throughout $\mathcal{X}^n$ more sparsely, allowing a greater distance between them, making lower the probability that a received sequence **y** is too far apart from the transmitted code word **c** and closer to another code word **c**′.

2.5 CONCLUSION

We began this chapter by introducing fundamental concepts in communications theory and by presenting a mathematical modeling of the communications channel. We used the BEC and BSC channels to exemplify discrete channels and later defined the AWGN channel. Important concepts, such as *symbol energy*, *signal-to-noise ratio*, along with *decision rules* and *likelihood ratios* were defined. Important metrics of rate and reliability, namely the *symmetric capacity* and the *Bhattacharyya parameter* were also defined.

Next, an introduction and a brief history of error correcting codes was made, where it became clear that, at least for BPSK under AWGN channel, optimal decoding of block codes is equivalent to finding the code word with minimum distance. For large codes, finding this code word may be computationally unfeasible, and one of the main goals of error correcting theory is to find encoding schemes that allow less complex decoding with good error performance. Finally, we illustrated why large blocks allow better error performance. In the next chapter we proceed by applying the concepts introduced in this chapter to polar codes.

3 THEORETICAL ASPECTS AND CONSTRUCTION OF POLAR CODES

3.1 INTRODUCTION

Discovered in 2009 by Arikan [4], polar codes are the first family of error correcting codes to provably achieve the channel capacity. More precisely, Arikan showed that for any code rate R smaller than the channel capacity, arbitrary small block error probability is achieved as the block size N goes to infinity. Another key feature of polar codes is that they are shown to have encoding and decoding complexity $O(N \log N)$ [4], which immediately made them attractive to further research. The possibility that polar codes can show similar or better performance than state-of-the-art codes such as Turbo and Low-Density Parity-Check (LDPC) codes has made this an active area of research [17].

The objective of this chapter is to introduce fundamental concepts of polar codes and to illustrate them with simulation results. Such concepts include *virtual channels*, approached in Section 3.2, and *channel polarization*, presented in Section 3.3. We then study the basic principles of the *Successive Cancellation (SC)* decoding proposed by Arikan [4] in Section 3.4. In Section 3.5 we show that the natural order decoding of polar codes is a property of the recursive structure proposed, not the encoding matrix. We show how this codes are constructed in Section 3.6 and present the simulation results and analysis in Section 3.7. Practical aspects of polar encoding are discussed in more detail in Section 3.8. Finally, the conclusions are presented in Section 3.9. The more detailed description of the decoder and decoding optimizations are left to Chapter 4.

3.2 VIRTUAL CHANNELS

It was seen in Example 2.9 that the decoding of a block code through a Binary Symmetric Channel (BSC) $W : \{0,1\} \to \{0,1\}$ can be performed considering the whole block as a single output of the channel. To illustrate this, Figure 3.1 shows the example where repetition encoding is used to transmit through a BSC: each 3-symbol block at the output can be considered as a symbol of a new output alphabet of a new channel $W' : \{0,1\} \to \{0,1\}^3$. The new transition probabilities can be obtained using the Hamming distances, as shown in Example 2.9.

Given a Discrete Memoryless Channel (DMC) $W : \mathcal{X} \to \mathcal{Y}$, there are three types of synthetic channels that are of interest:

1. parallel use channels;
2. combined channels;
3. split channels.

Parallel use channels concatenate n copies of a DMC W to represent symbols transmitted in series as being transmitted all at once in parallel. Combined channels, just like W' in the example above, incorporate the encoder structure into the channel. Lastly, split channels consider the idea that bits are decoded serially, and that estimates of the previous bits may already be available. The next subsections approach these channels.

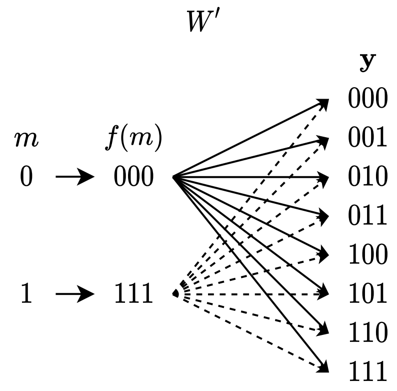


Figure 3.1 – Combined BSC channel with repetition encoding.

3.2.1 Parallel Use Channels

Suppose we transmit the sequence $\mathbf{x} \in \mathcal{X}^n$ symbol by symbol in a sequential manner through W and obtain $\mathbf{y} \in \mathcal{Y}^n$ at the output. Since the channel is memoryless, the transmission through the channel can be thought as happening in parallel, as depicted in Figure 3.2, and we can consider the equivalent channel $W^n : \mathcal{X}^n \to \mathcal{Y}^n$ with transition probabilities

$$W^n(\mathbf{y} \mid \mathbf{x}) = \prod_{i=0}^{n-1} W(y_i \mid x_i). \tag{3.1}$$

3.2.2 Combined Channels

A combined channel is obtained by inserting the encoding structure into the channel, as illustrated in Figure 3.3. Note that the channel is represented as a parallel use channel

W^n instead of using serial transmission. Since the encoding function f is injective, and considering the notation $\mathbf{x} = f(\mathbf{m})$, we have that the transition probabilities of the combined channel $W_n : M^k \to \mathcal{Y}^n$ are given by

$$W_n(\mathbf{y} \mid \mathbf{m}) = W^n(\mathbf{y} \mid f(\mathbf{m})) = \prod_{i=0}^{n-1} W(y_i \mid x_i). \tag{3.2}$$

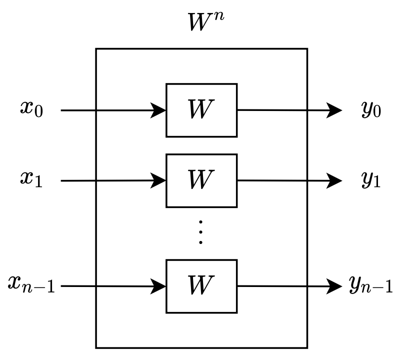
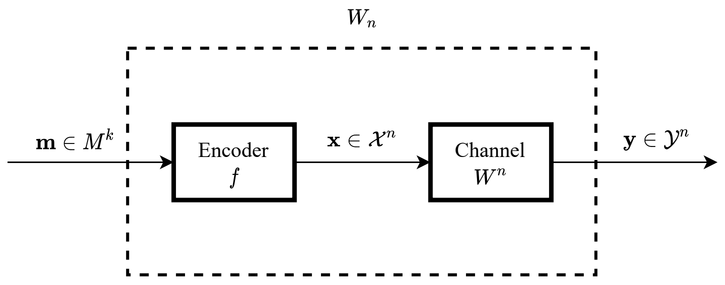


Figure 3.2 – Representation of a parallel use channel with W as the base channel.



W_n
$\mathbf{m} \in M^k$
Encoder f
$\mathbf{x} \in \mathcal{X}^n$
Channel W^n
$\mathbf{y} \in \mathcal{Y}^n$

Figure 3.3 – Generic representation of a combined channel.

Example 3.1 (Polar Encoding Kernel)
Polar encoding is done recursively, and the building block used will be introduced in this example. Figure 3.4 shows a combined channel with linear block coding of two bits, with

$$\mathbf{x} = \mathbf{uF}\,, \tag{3.3}$$

where

$$\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{3.4}$$

The $\oplus$ operation represents mod-2 summation or, equivalently, a sum in the finite field GF(2). The transition probabilities are

$$W_2(y_1, y_2 \mid u_1, u_2) = W(y_1 \mid u_1 \oplus u_2)W(y_2 \mid u_2). \tag{3.5}$$

Note that instead of transmitting u_1 and u_2 directly, we are transmitting u_2 and its differential to u_1, $u_1 \oplus u_2$.

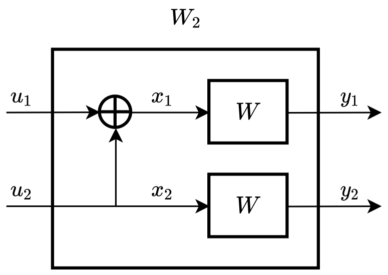


Figure 3.4 – Polar encoding kernel and combined channel W_2.

3.2.3 Split Channels

Before going into the split channel definition, we introduce a vector notation that will be handy. The notation a_1^N represents the vector $(a_1, ..., a_N)$. We write a_i^j, $1 \leq i \leq j \leq N$, to represent the subvector $(a_i, ..., a_j)$. In the case when $i > j$, we have that a_i^j represents an empty set. Given $\mathcal{A} \in \{1, ..., N\}$ a index subset, the subvector $(a_k : k \in \mathcal{A})$ is indicated by $a_{\mathcal{A}}$.

Polar codes are provably shown to achieve channel capacity under SC decoding [4]. This type of decoding considers that the message bits are decoded in a serial manner: m_1 is decoded first, and then its estimate $\hat{m}_1$ is used jointly with the channel output y_1^n to decode the next bit m_2. Next, both estimates $\hat{m}_1$ and $\hat{m}_2$ are used along with y_1^n to decode m_3. Inspired by this notion, we start our path on defining the split channels. After decoding m_1, the most natural thought is to look at the probability $p(y_1^n \mid m_2 = i)\,, i \in \{0, 1\}$. This probability considers all bits but m_2 to be unknown. The Maximum *a Posteriori* (MAP) rule would require to compute for both $i = 0$ and $i = 1$ and decide which one is larger. However, after doing this first for m_1, we have a hint of its value. What if we computed the probability $p(y_1^n, m_1 = j \mid m_2 = i)$ instead? As it turns out, computing probabilities this way, using the previously estimated bits, allows very reliable bit decisions at some positions, but also very unreliable bit decisions at other positions when using polar coding [4]. Then, if information bits are only transmitted on the reliable positions, channel capacity can be achieved [4].

The probability $p(y_1^n, m_1 = j \mid m_2 = i)$ can be interpreted as the transition probability of a channel that has y_1^n and m_1 at its output and m_2 at its input. In principle, m_1 can assume any estimate: it can be the actual m_1 value that entered the channel, or it can be the Maximum Likelihood (ML) estimate of m_1, or it can even be a random value. We wish then to compute

$p(y_1^n, m_1^{i-1} \mid m_i)$, and begin by noting that

$$\begin{aligned}
p(y_1^n, m_1^{i-1} \mid m_i) &= \frac{p(y_1^n, m_1^{i-1}, m_i)}{p(m_i)} \\
&= \frac{1}{p(m_i)} \sum_{m_{i+1}^k \in \mathcal{X}^{k-i}} p(y_1^n, m_1^k) \\
&= \frac{1}{p(m_i)} \sum_{m_{i+1}^k \in \mathcal{X}^{k-i}} p(y_1^n \mid m_1^k) \cdot p(m_1^k) \\
&= \frac{p(m_1^k)}{p(m_i)} \sum_{m_{i+1}^k \in \mathcal{X}^{k-i}} W_n(y_1^n \mid m_1^k) \\
&= \frac{1}{2^{k-1}} \sum_{m_{i+1}^k \in \mathcal{X}^{k-i}} W_n(y_1^n \mid m_1^k), \qquad (3.6)
\end{aligned}$$

where it was assumed that all messages m_1^k are equally likely and the split channel W_n was used.

We define the i-th split channel $W_n^{(i)} : M \to \mathcal{Y}^n \times M^{i-1}$ as having transition probabilities

$$W_n^{(i)}(y_1^n, m_1^{i-1} \mid m_i) = \frac{1}{2^{k-1}} \sum_{m_{i+1}^k \in \mathcal{X}^{k-i}} W_n(y_1^n \mid m_1^k). \qquad (3.7)$$

Figure 3.5 shows a split channel representation. This representation may wrongly lead one to believe that the actual inputs m_1^{i-1} are available to the output of the channel. Instead, this representation means that m_1^{i-1} is considered to be given at the input of the combined channel W_n: the terms $W_n(y_1^n \mid m_1^k)$ in Eq. (3.7) have m_1^{i-1} fixed.

Example 3.2 (Polar Coding Core Split Channels)
Following the Example 3.1, we have that

$$\begin{aligned}
W_2^{(1)}(y_1^2 \mid u_1) &= \sum_{u_2} \frac{1}{2} W_2(y_1^2 \mid u_1^2) \\
&= \sum_{u_2} \frac{1}{2} W^2(y_1^2 \mid u_1 \oplus u_2, u_2) \\
&= \sum_{u_2} \frac{1}{2} W(y_1 \mid u_1 \oplus u_2) W(y_2 \mid u_2), \qquad (3.8)
\end{aligned}$$

and

$$\begin{aligned}W_2^{(2)}(y_1^2, u_1 \mid u_2) &= \frac{1}{2} W_2(y_1^2 \mid u_1^2)\\ &= \frac{1}{2} W^2(y_1^2 \mid u_1 \oplus u_2, u_2)\\ &= \frac{1}{2} W(y_1 \mid u_1 \oplus u_2) W(y_2 \mid u_2). \end{aligned} \quad (3.9)$$

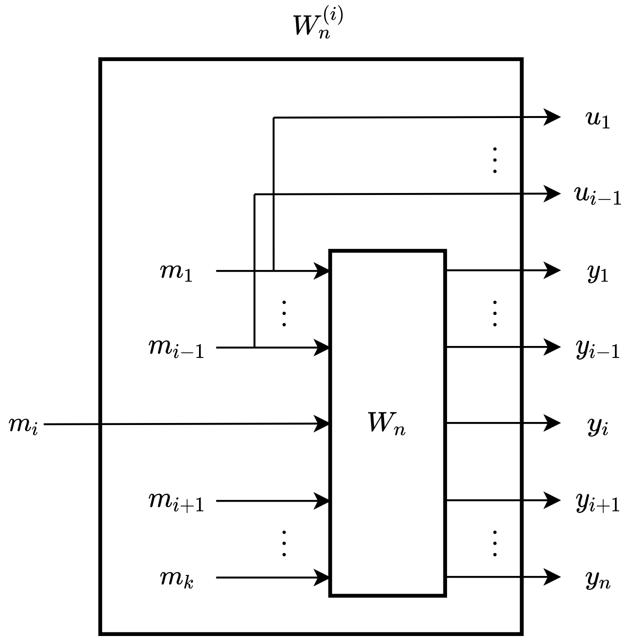


Figure 3.5 – Representation of a split channel.

3.3 THEORETICAL PRELIMINARIES ON POLAR CODING

We begin by defining the Kronecker product of two matrices **A** m-by-n and **B** r-by-s as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}\mathbf{B} & \dots & A_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ A_{m1}\mathbf{B} & \dots & A_{mn}\mathbf{B} \end{bmatrix}, \quad (3.10)$$

which is an mr-by-ns matrix. The Kroneker power $\mathbf{A}^{\otimes n}$ is defined as $\mathbf{A} \otimes \mathbf{A}^{\otimes(n-1)}$, for all $n \geq 1$, and $\mathbf{A}^{\otimes 0} \triangleq [1]$.

Polar encoding is really straightforward. Given the encoder kernel

$$\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad (3.11)$$

polar codes are a family of linear block codes obtained by the encoding matrix $\mathbf{F}_N = \mathbf{F}^{\otimes n}$, where $N = 2^n$. The encoder maps the bits u_1^N to the code word, which will be the input of the channel, by the operation

$$x_1^N = u_1^N \mathbf{F}_N. \tag{3.12}$$

Using this definition, we have that

$$\mathbf{F}_4 = \left[\begin{array}{cc:cc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ \hdashline 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right], \tag{3.13}$$

$$\mathbf{F}_8 = \left[\begin{array}{cc:cc:cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right]. \tag{3.14}$$

Here, the dashed lines are only to indicate how the previous matrices are used to construct the larger ones. Note how these matrices are obtained by tiling copies of the previous matrix next to each other, leaving zeros on the upper right corner. A very important property of $\mathbf{F}_N$ is that it is its own inverse, namely $\mathbf{F}_N^{-1} = \mathbf{F}_N$.

From this point on, let $W : \mathcal{X} \to \mathcal{Y}$ be a Binary-input Discrete Memoryless Channel (B-DMC) used to transmit the encoded bits. Also, when used on the context of polar coding, the combined and split channels have the notation $W_N : \mathcal{X}^N \to \mathcal{Y}^N$ and $W_N^{(i)} : \mathcal{X} \to \mathcal{Y}^N \times \mathcal{X}^{i-1}$, respectively, and the mapping between the input u_1^N and the bits $u_1^N \mathbf{F}_N$ available at the parallel use channel W^N is left implicit.

The recursive definition of the polar encoding matrix $\mathbf{F}_N$ allows us to define a recursive construction of the polar encoder and the polar split and combined channels. Figure 3.4 shows how a combined channel W_2 can be constructed using 2 copies of W and some encoding operations and, taking this iteration to the next level, Figure 3.6 shows how two combined channels W_2 can be used to construct the combined channel W_4. More generally, Figure 3.7 shows the recursive construction of the channels W_N. Note that the upper channel $W_{N/2}$ receives the bit by bit sum of the first and second half bits, and that the second half is passed to the next channel $W_{N/2}$ untouched. On the next channel, each half is then also divided and the process is repeated. Each copy of $W_{N/2}$ is called a constituent code.

The following theorem proved by Arikan [4] is a key result of split channels and gives the idea on why polar codes are named in such way.

Theorem 3.1 (Channel Polarization) For any B-DMC W, the channels $\{W_N^{(i)}\}$ *polarize* in the sense that, for any fixed $\delta \in (0, 1)$, as N goes to infinity through powers of two, the fraction of indices $i \in \{1, ..., N\}$ for which $I(W_N^{(i)}) \in (1 - \delta, 1]$ goes to $I(W)$ and the fraction for which $I(W_N^{(i)}) \in [0, \delta)$ goes to $1 - I(W)$.

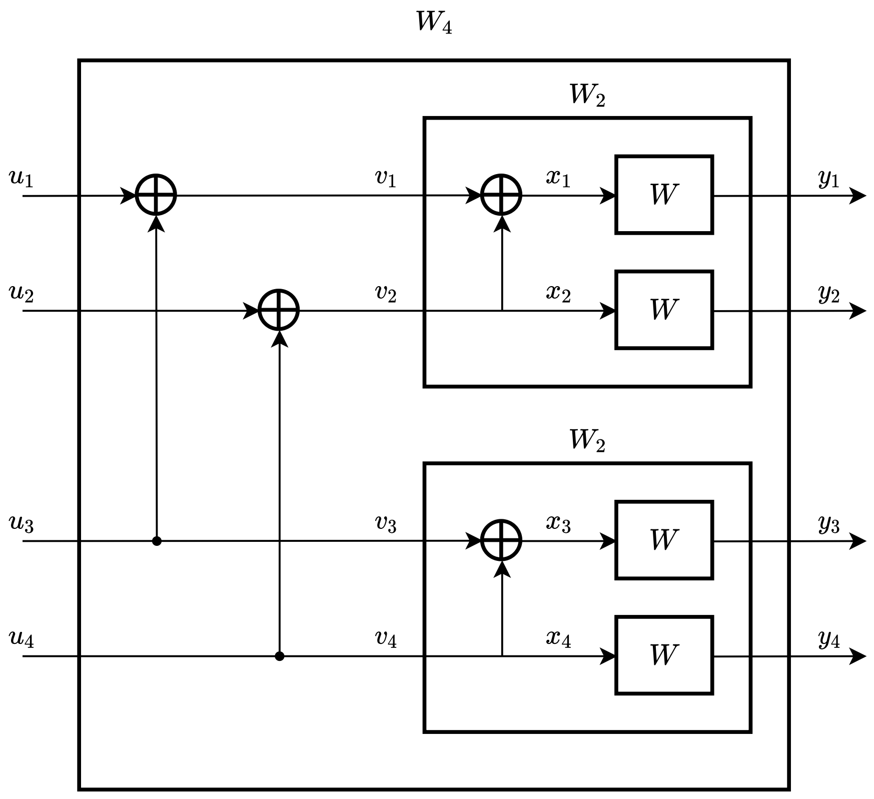


Figure 3.6 – The channel W_4 using polar encoding.

This theorem basically states that out of the N bit channels available, approximately $N \cdot I(W)$ of them are extremely reliable, with capacities near 1, and approximately $N \cdot (1 - I(W))$ are extremely unreliable, with capacities near 0. Note that capacities near 1 do not necessarily mean reliability, since the capacity is a measure of maximum rate. However, in Chapter 2 we saw that $I(W) \approx 1$ iff $Z(W) \approx 0$, and $Z(W)$ is a measure of reliability. Considering the $N \cdot I(W)$ channels with capacity near 1, one should expect that the total capacity is near $N \cdot I(W) \cdot 1$: in fact, we have that [4]

$$\sum_{i=1}^{N} I(W_N^{(i)}) = N \cdot I(W). \tag{3.15}$$

This is the same total capacity of N parallel copies of W transmitting uncoded bits. It then

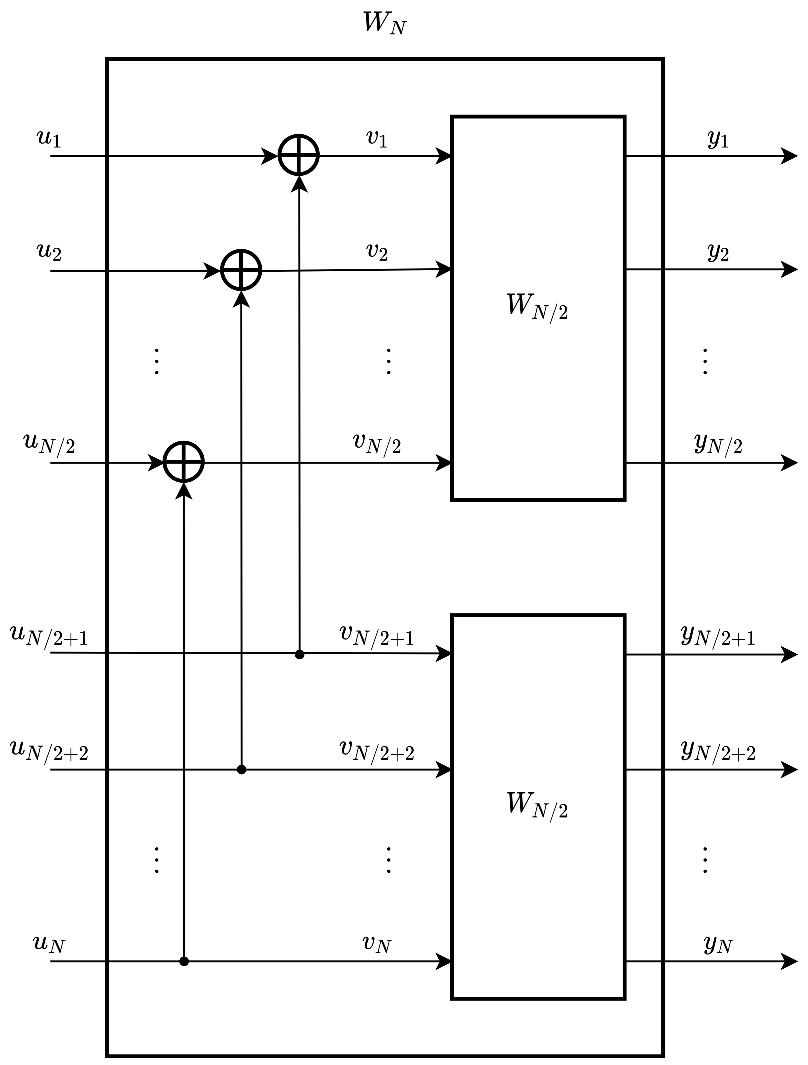


Figure 3.7 – The recursive construction of the combined channel W_N using two $W_{N/2}$ channels.

becomes clear that the total capacity is conserved, but instead redistributed and concentrated in a group of bit channels.

However, this result alone is not enough to ensure that polar coding is capacity achievable. Instead, it shows that we should only send data bits on the channels that are most reliable, leaving the remaining positions on a fixed known value, and it motivates the following definition.

Definition 3.1 ($\mathbf{F}_N$-Coset Codes)**.** Let $\mathcal{A}$ be an arbitrary subset of $\{1, ..., N\}$. Then, we can rewrite Eq. (3.12) as

$$x_1^N = u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}) \oplus u_{\mathcal{A}^C}\mathbf{F}_N(\mathcal{A}^C), \tag{3.16}$$

where $\mathbf{F}_N(\mathcal{A})$ denotes the submatrix of $\mathbf{F}_N$ composed of the rows of indexes in $\mathcal{A}$. The idea is to fix some bit values $u_{\mathcal{A}^C}$ at fixed positions $\mathcal{A}^C$, while varying $u_{\mathcal{A}}$. Using Definition 2.20, these are coset codes of the linear code with generator matrix $\mathbf{F}_N(\mathcal{A})$. These codes will be identified by $(N, K, \mathcal{A}, u_{\mathcal{A}^C})$, where K is the size of $\mathcal{A}$, which is also called the *information set*. Also, $u_{\mathcal{A}^C}$ is referred as *frozen* bits or vector.

The family of Reed-Muller (RM) codes are closely related to polar codes. In fact, RM codes are $\mathbf{F}_N$-coset codes, with $\mathcal{A}$ chosen so that no deleted row has a Hamming weight larger than the remaining rows and $u_{\mathcal{A}^C}$ is set to zero. Arikan [4] showed that RM codes are asymptotically unreliable under SC decoding.

Now, the question is: how do we select the most reliable bit indexes $\mathcal{A}$? The definition of polar codes makes this explicit.

Definition 3.2 (Polar Codes)**.** Given a B-DMC W, a $\mathbf{F}_N$-coset code with parameters $(N, K, \mathcal{A}, u_{\mathcal{A}^C})$ will be called a *polar code* for W if the information set $\mathcal{A}$ is chosen as a K-element subset of $\{1, ..., N\}$ such that $Z(W_N^{(i)}) \leq Z(W_N^{(j)})$ for all $i \in \mathcal{A}$, $j \in \mathcal{A}^C$.

Remember that $Z(W_N^{(i)})$ is a measure of reliability of the split channel $Z(W_N^{(i)})$: it represents an upper bound for the decision error probability under ML decoding. Choosing the most reliable bit positions to transmit information bits results in the following theorem [4].

Theorem 3.2 (Polar Coding) For any symmetric B-DMC W and any fixed $R < I(W)$, consider any sequence of $\mathbf{F}_N$-coset codes $(N, K, \mathcal{A}, u_{\mathcal{A}^C})$ with N increasing to infinity, $K = \lfloor NR \rfloor$, $\mathcal{A}$ chosen according to Definition 3.2, and $u_{\mathcal{A}^C}$ fixed arbitrarily.

The block error probability under SC decoding satisfies

$$P_e(N, K, \mathcal{A}, u_{\mathcal{A}^C}) = O(N^{-1/4}). \tag{3.17}$$

Theorem 3.2 shows that polar codes are capacity achieving, since choosing any rate smaller than the channel capacity leads to a probability of block error that decreases to 0 as $N \to \infty$. Also, the choice of $u_{\mathcal{A}^C}$ can be arbitrary, since the error bound doesn't depend on this value. Hence, we opt for the choice where $u_{\mathcal{A}^C}$ is the all zero vector. Therefore, a polar code will be identified by the parameters $(N, K, \mathcal{A})$.

3.4 SUCCESSIVE CANCELLATION DECODING

The idea behind SC decoding is to decode the bits in a serial manner and to use the bit estimates already obtained to decode the next bit. The decoding is carried out using the following expression to obtain the i-th bit estimate $\hat{u}_i$

$$\hat{u}_i = \begin{cases} 0, & \text{if } i \in \mathcal{A}^C \\ h_i(y_1^N, \hat{u}_1^{i-1}), & \text{if } i \in \mathcal{A}, \end{cases} \tag{3.18}$$

i starting at 1 and going to N, and $h_i : \mathcal{Y}^N \times \mathcal{X}^{i-1} \to \mathcal{X}\,, i \in \mathcal{A}$, are decision functions defined as

$$h_i(y_1^N, \hat{u}_1^{i-1}) = \begin{cases} 0, & \text{if } \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|1)} \geq 1 \\ 1, & \text{otherwise.} \end{cases} \tag{3.19}$$

This decision rule compares the transition probabilities for the split channels and decides for 0 or 1 based on which probability is larger, resembling the ML rule. Indeed, considering that only the values of $y_1^N, \hat{u}_1^{i-1}$ are available, this rule makes the best choice. However, future frozen bits are treated as Random Variables (RVs): as seen in Eq. (3.7), the sum is carried over all possibilities of future bits. In this situation, optimality is traded by the possibility of efficient computation of h_i using recursive formulas. For sake of simplicity, we adopt the following notation for the split channel Likelihood Ratios (LRs)

$$l_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) = \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} \mid 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} \mid 1)} \tag{3.20}$$

$$L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) = \log(l_N^{(i)}). \tag{3.21}$$

Example 3.3 (SC Decoding with $N = 2$)
Following example 3.2, we have that

$$\begin{aligned}
l_2^{(1)}(y_1^2) &= \frac{\sum_{u_2} \frac{1}{2} W(y_1 \mid 0 \oplus u_2) W(y_2 \mid u_2)}{\sum_{u_2} \frac{1}{2} W(y_1 \mid 1 \oplus u_2) W(y_2 \mid u_2)} \\
&= \frac{W(y_1 \mid 0)W(y_2 \mid 0) + W(y_1 \mid 1)W(y_2 \mid 1)}{W(y_1 \mid 1)W(y_2 \mid 0) + W(y_1 \mid 0)W(y_2 \mid 1)} \\
&= \frac{\dfrac{W(y_1 \mid 0)W(y_2 \mid 0)}{W(y_1 \mid 1)W(y_2 \mid 1)} + 1}{\dfrac{W(y_1 \mid 1)W(y_2 \mid 0)}{W(y_1 \mid 1)W(y_2 \mid 1)} + \dfrac{W(y_1 \mid 0)W(y_2 \mid 1)}{W(y_1 \mid 1)W(y_2 \mid 1)}} \\
&= \frac{l_1^{(1)}(y_1) l_1^{(1)}(y_2) + 1}{l_1^{(1)}(y_2) + l_1^{(1)}(y_1)},
\end{aligned}$$

and

$$\begin{aligned}
l_2^{(2)}(y_1^2, \hat{u}_1) &= \frac{\frac{1}{2} W(y_1 \mid \hat{u}_1 \oplus 0) W(y_2 \mid 0)}{\frac{1}{2} W(y_1 \mid \hat{u}_1 \oplus 1) W(y_2 \mid 1)} \\
&= \begin{cases} \dfrac{W(y_1 \mid 0)W(y_2 \mid 0)}{W(y_1 \mid 1)W(y_2 \mid 1)}, & \text{if } \hat{u}_1 = 0 \\ \dfrac{W(y_1 \mid 1)W(y_2 \mid 0)}{W(y_1 \mid 0)W(y_2 \mid 1)}, & \text{if } \hat{u}_1 = 1 \end{cases} \\
&= \begin{cases} l_1^{(1)}(y_1) \cdot l_1^{(1)}(y_2), & \text{if } \hat{u}_1 = 0 \\ \dfrac{l_1^{(1)}(y_2)}{l_1^{(1)}(y_1)}, & \text{if } \hat{u}_1 = 1. \end{cases}
\end{aligned}$$

The expressions obtained in Example 3.3 are important and will be shown in Section 3.5 to be valid not only for $N = 2$. We then define

$$f(l_a, l_b) = \frac{l_a \cdot l_b + 1}{l_a + l_b}, \tag{3.22}$$

$$g(l_a, l_b, u) = l_b \cdot (l_a)^{(1-2u)}, \tag{3.23}$$

where we note that the value of $u \in \{0, 1\}$ determines whether we multiply or divide the likelihoods. Figure 3.8 shows how the encoding structure converts into decoding structures. From now on we take the liberty of representing $l_1^{(1)}(y_j)$ by l_j.

Let's walk through the decoding performed considering the structure on Figure 3.9. First, y_1^4 is used to obtain the channel likelihood ratios l_1^4. At the upper copy of W_2, l_1 and l_2 are combined though an f node to obtain $l_2^{(1)}(y_1^2)$, and at the lower copy of W_2, l_3 and l_4 are

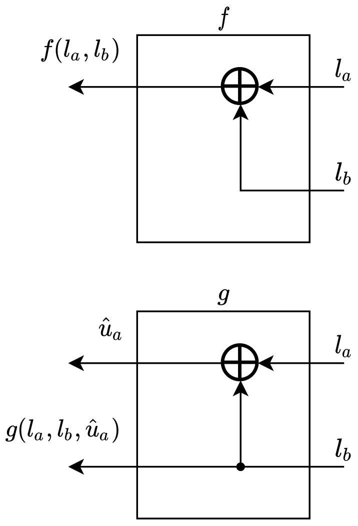


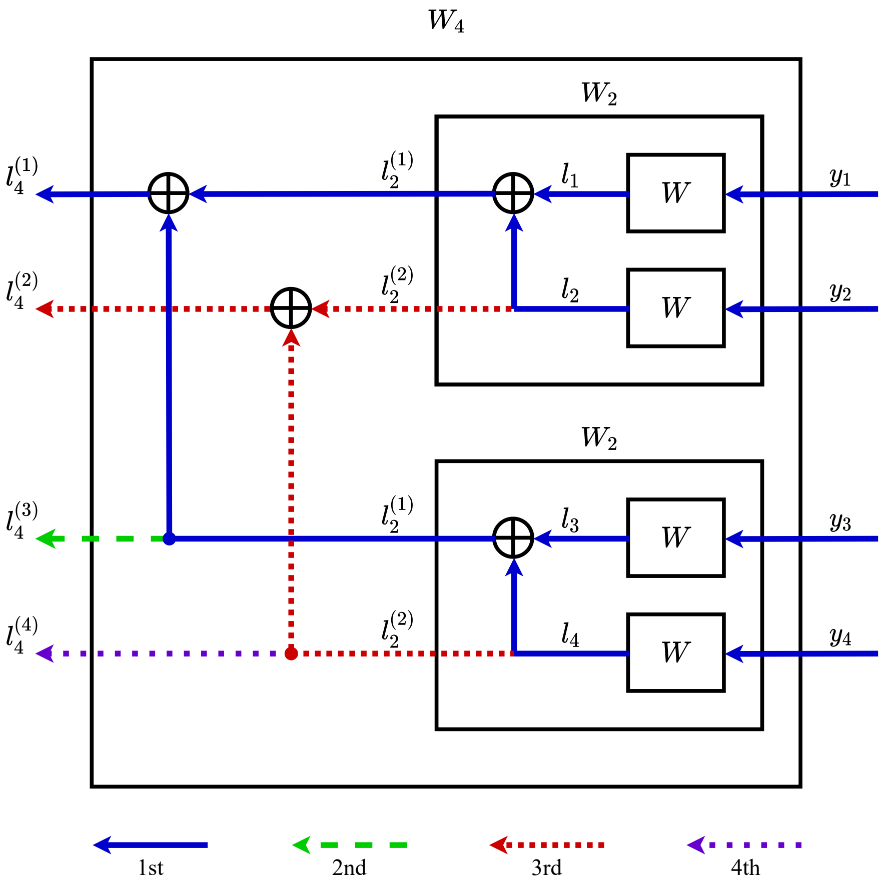
Figure 3.8 – Polar decoding basic nodes.

also combined through an f node to obtain $l_2^{(1)}(y_3^4)$. Note that for each constituent code the decoding is carried out identically, given the different inputs y_1^2 and y_3^4. If we were to decode the smaller components W_2 separately, we would then obtain the estimates $\hat{v}_1, \hat{v}_3$ using the decision function on Eq. (3.19). However, these decoders are part of a greater decoding structure and the decoding carries on. The next step is to use both $l_2^{(1)}$ likelihoods to compute $l_4^{(1)}(y_1^4) = f(l_2^{(1)}(y_1^2), l_2^{(1)}(y_3^4))$ using an f node and u_1 is estimated using $l_4^{(1)}(y_1^4)$. With this value in hand we proceed by computing $l_4^{(3)}(y_1^4, \hat{u}_1)$, since u_3 is the bit paired with the g node in this structure. After $\hat{u}_3$ is obtained, we can compute $\hat{v}_1 = \hat{u}_1 \oplus \hat{u}_3$ and $\hat{v}_3 = \hat{u}_3$. Now, we have the first bit estimate of the two component decoders W_2 and can proceed to compute $l_2^{(2)}(y_1^2, \hat{v}_1)$ and $l_2^{(2)}(y_3^4, \hat{v}_3)$. The same procedure for obtaining $\hat{u}_1, \hat{u}_3$ is used to obtain $\hat{u}_2$ and $\hat{u}_4$, in this sequence. Figure 3.9 makes clear that $\hat{x}_1^4$ are only obtained after $\hat{u}_4$ is decoded.

But what if we demand the bits to be decoded on the natural order? Also, one may notice that first one starts decoding the smaller constituent codes, W, then follow to $W_2, W_4, \ldots$ and so on. The decoding is made inside out. It would be interesting if we could decode outside in: after obtaining the channel likelihood ratios we would proceed to a decoder of order $N/2$, and so on, recursively.

We can solve both problems using the equivalent combined channel diagram shown in Figure 3.10. Since $\mathbf{F}_N = \mathbf{F}_N^{-1}$ for all N, we can flip the entire encoding structure left to right, swapping inputs and outputs. However, we can only do this with the encoding part of the combined channel; that's why the structure of $W_{N/2}$ was dissolved into the channels W and the encoding structures $\mathbf{F}_{N/2}$. Note how on Figure 3.10 the $\oplus$ operations that were previously carried at the input of W_N are now carried just before the channels W.

If we proceed the decoding using the structure on Figure 3.10, we decode the bits u_i in a)



b)

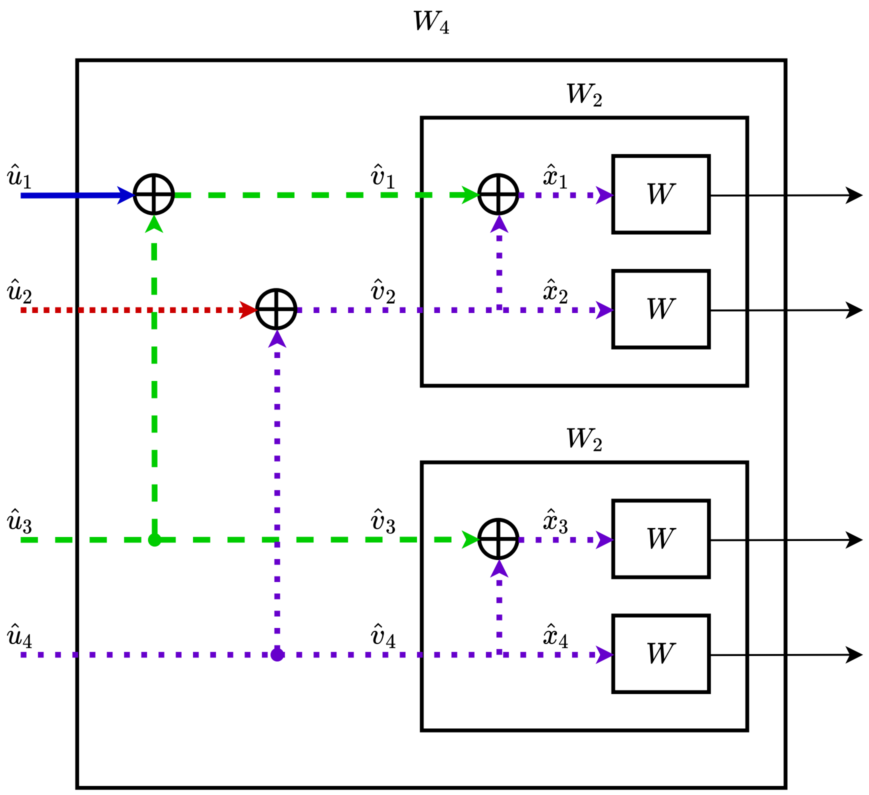


Figure 3.9 – Successive cancellation decoding steps for $N = 4$. a) Likelihood flow, from right to left. b) Estimated bits flow, from left to right.

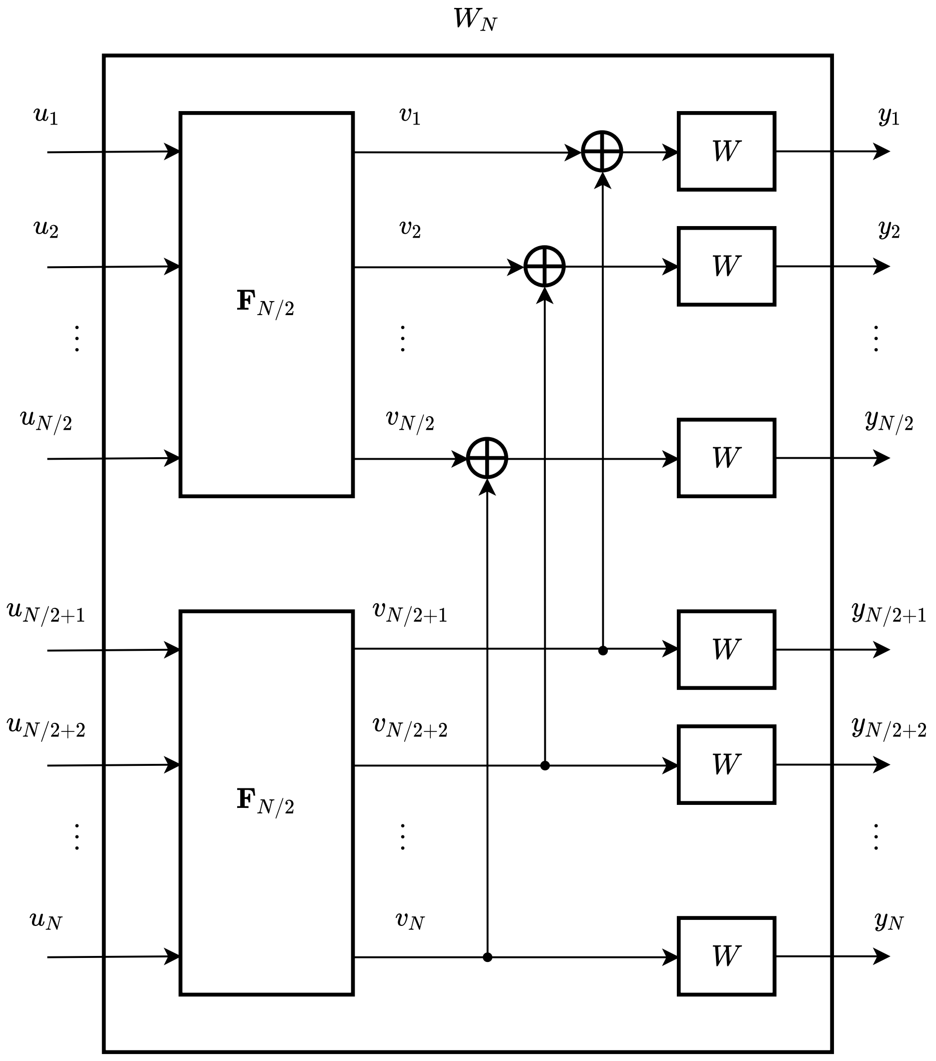


Figure 3.10 – An equivalent representation of the combined channel using two $\mathbf{F}_{N/2}$ encoders.

the natural order. In fact, we prove this by induction. For $\mathbf{F}_1$, it follows immediately. Now, given $x_1^N = u_1^N \mathbf{F}_N$, we have that every bit on x_1^N depends on the last bit of the input u_N, since the last row of $\mathbf{F}_N$ is an all one vector. Therefore, we do not know the value of any bit of $\hat{v}_1^{N/2}$ until the estimate $\hat{u}_{N/2}$ is made, and we have no option but to compute $f(l_i, l_{i+N/2})$ for $1 \leq i \leq N/2$ and feed this values to $\mathbf{F}_{N/2}$. Now, using the induction hypothesis, inside the structure of $\mathbf{F}_{N/2}$ the decoding will yield $\hat{u}_1^{N/2}$ in the natural order. At this point, $\hat{v}_1^{N/2}$ becomes available all at once after $\hat{u}_{N/2}$ is known and we compute $g(l_i, l_{i+N/2}, \hat{v}_i)$ for $1 \leq i \leq N/2$. These g values are fed to the lower $\mathbf{F}_{N/2}$ structure and, using the induction hypothesis again, $\hat{u}_{N/2+1}^N$ becomes available in the natural order.

On Figure 3.11 we illustrate for $N = 4$ that just by inverting the encoder we can decode the input bits on the natural order. Just inverting the encoder is a solution to the decoding order problem, but is not the only one. Arikan [4] proposed that the inputs are permuted by the *reverse shuffle* matrix $\mathbf{R}_N$ that separates all the even from the odd indexes by mapping

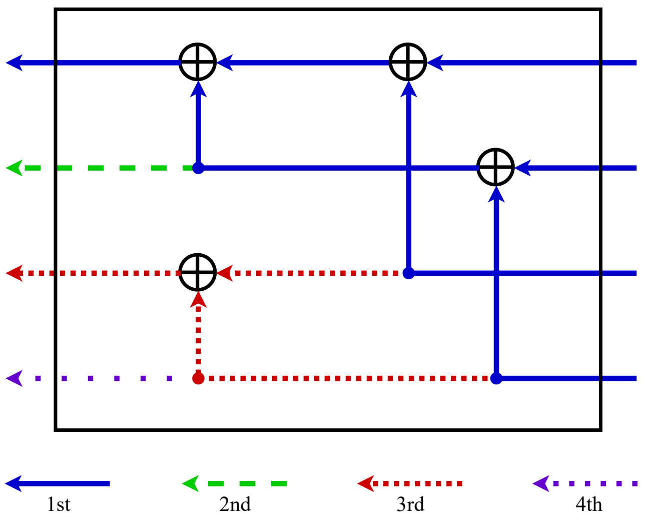


Figure 3.11 – Equivalent successive cancellation decoding steps for $N = 4$.

$(u_1, ...u_N)$ into $(u_1, u_3, ..., u_N - 1, u_2, u_4, ..., u_N)$, and you can also show by induction that this change results in decoding in the natural order. In fact, Arikan uses a different encoding matrix, namely

$$\mathbf{G}_N = \mathbf{B}_N \mathbf{F}_N, \tag{3.24}$$

where $\mathbf{B}_N$ is constructed recursively using

$$\mathbf{B}_N = \mathbf{R}_N (\mathbf{I}_2 \otimes \mathbf{B}_{N/2}). \tag{3.25}$$

We have that $\mathbf{B}_N$ is a permutation matrix [4], in a way that $\mathbf{F}_N$ and $\mathbf{G}_N$ are equivalent encoding matrices.

The advantage of SC decoding is its low complexity of $O(N \log N)$ [4]. Due to its serial nature, SC decoding shows large latency. Arikan [4] showed that SC decoding can be performed with latency $2N - 1$ for a code with length N. It would be desired, however, for the decoding latency to be sublinear in N. On Chapter 4 we show how this can be achieved without introducing any performance loss.

3.5 RECURSIVE RELATIONS

One of the greatest advantages of polar coding are the recursive relations for the split channel probabilities. These recursive relations are important, since the computation using Eq. (3.7) requires to sum on all the possibilities of u_{i+1}^N and is computationally unfeasible. However, neither of the generic structures shown in Figures 3.7 or 3.10 are suited for the derivation of a recursive formula between the split channels. The structure on Figure 3.7 has the problem that the decoding is not made in the natural order, and the structure on

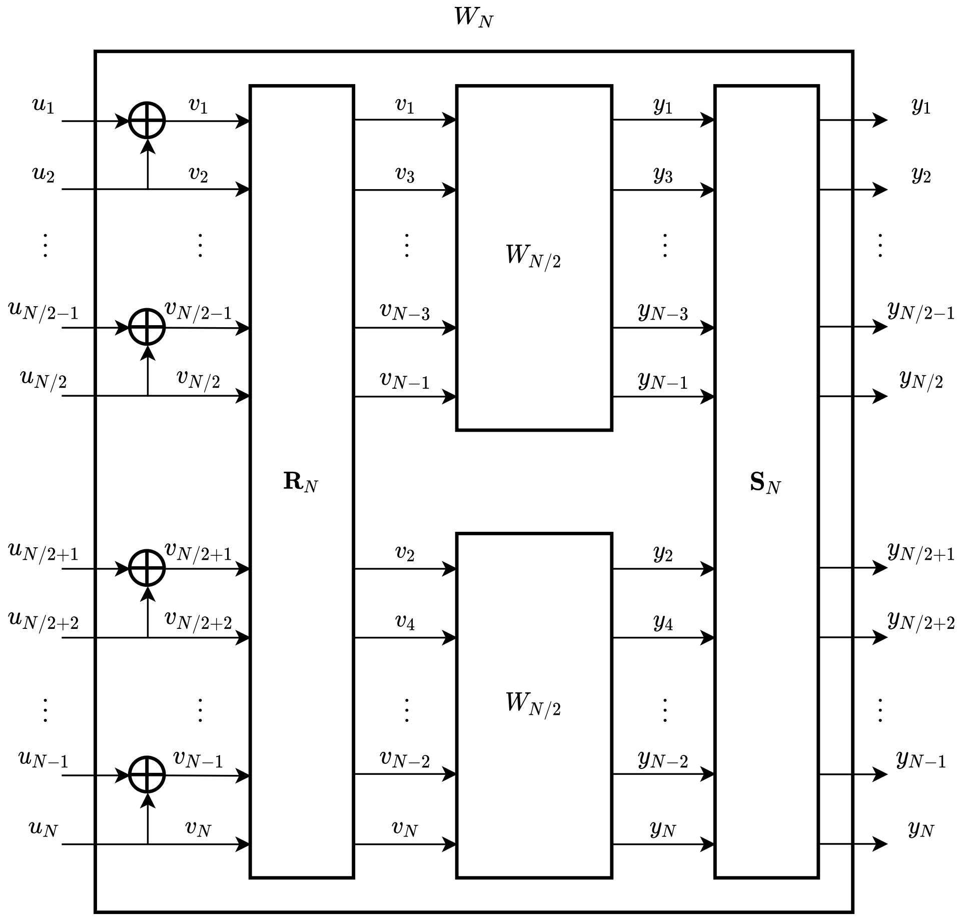


Figure 3.12 – Another equivalent representation of the combined channel using recursive construction.

Figure 3.10 doesn't show recursive construction in terms of $W_{N/2}$. Arikan [4] solved this problem and obtained the recursive relations by introducing the permutation matrix $\mathbf{R}_N$ to the encoding matrix. We wish to show similar relations, but maintaining the encoding matrix as $\mathbf{F}_N$ and natural order decoding.

The equivalent structure proposed in Figure 3.12 shows how the combined channel can be represented in order to facilitate the derivation of the recursive formulas. Note that even though the $\mathbf{R}_N$ matrix is in our schematic, this representation is equivalent to the one in Figure 3.10. Here, $\mathbf{R}_N$ represents the reverse shuffling matrix and $\mathbf{S}_N$ represents the shuffling matrix, which maps $(s_1, s_2, ..., s_{N/2}, s_{N/2+1}, ..., s_N)$ into $(s_1, s_{N/2+1}, s_2, ..., s_{N/2}, s_N)$, shuffling the lower and upper halves of the vector. An important notation used here is that $a_{1,e}^k$ designates all the even entries of a_1^k, and $a_{1,o}^k$ designates all the odd entries of a_1^k. As an example, $a_{1,o}^6 = (a_1, a_3, a_5)$.

We have that the recursive relations for our proposed scheme are

$$W_{2N}^{(2i-1)} = \sum_{u_{2i}} \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}) \tag{3.26}$$

and

$$W_{2N}^{(2i)} = \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}), \tag{3.27}$$

derived with the steps in Appendix A. Note how these formulas resemble the ones obtained in Example 3.2. Remember that the expressions obtained for f and g nodes result from this shape of recursion: $W_{2N}^{(2i-1)}$ is the sum of two product terms and $W_{2N}^{(2i)}$ is a single product term. If we plug these expressions on the likelihood definition at Eq. (3.20) we get

$$l_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-2}) = \frac{l_{N/2}^{(i)}(y_{1,o}^N, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) l_{N/2}^{(i)}(y_{1,e}^N, \hat{u}_{1,e}^{2i-2}) + 1}{l_{N/2}^{(i)}(y_{1,o}^N, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) + l_{N/2}^{(i)}(y_{1,e}^N, \hat{u}_{1,e}^{2i-2})} \tag{3.28}$$

and

$$l_N^{(2i)}(y_1^N, \hat{u}_1^{2i-2}) = \left[l_{N/2}^{(i)}(y_{1,o}^N, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) \right]^{1-2\hat{u}_{2i-1}} \cdot l_{N/2}^{(i)}(y_{1,e}^N, \hat{u}_{1,e}^{2i-2}). \tag{3.29}$$

These recursive expressions seem to be only a headache. However, they are necessary to show that the intuitive idea of evolving the likelihood ratios using the f and g nodes used in Figure 3.8 is actually equivalent to computing the split channel likelihood ratios. This intuitive idea presented earlier will lead to a binary tree decoding algorithm in Chapter 4. Figure 3.12 shows that the input of a W_N channel is made of several f and g nodes, and that this construction is recursive. This means that whenever we see the f and g node pattern and compute likelihood ratios using Eqs. (3.22, 3.23) we are actually computing split channel likelihoods at the input of some channel W_N.

Consider that we want to decode the bit $1 \leq k \leq N$ from a code with $N = 2^n$. If we do some re-indexing to start at 0, we now are decoding the bit $k' = k - 1$. In this new scenario, if $k' = 2i$ we use Eq. (3.28) and if $k' = 2i + 1$ we use Eq. (3.29). Let k' have a binary representation of $b_n ... b_1$. If $b_1 = 0$, then k' is even and the first recursion step will be an f node that will compute Eq. (3.28). If $b_1 = 1$, then k' is odd and the first recursion step will be a g node that will compute Eq. (3.29). In either case, we have that i has the binary representation $b_n ... b_2$, which means that this reasoning can be repeated with b_2 to obtain the recursive relations for $l_{N/2}^{(i)}$, in a manner that the digits $b_1, ..., b_n$ show all the recursive steps needed at once. Since we can't perform any calculation until we reach $l_1^{(0)}$, knowing the binary representation $b_n ... b_1$ of k' allows us to start the recursion at the channel level and evolve until we reach the split channel desired, by applying f and g according to the bits b_j, starting at b_n with $l_1^{(0)}$ and ending at b_1 with $l_N^{(k')}$. Figure 3.13 shows the split channel evolution on the example where $N = 8$.

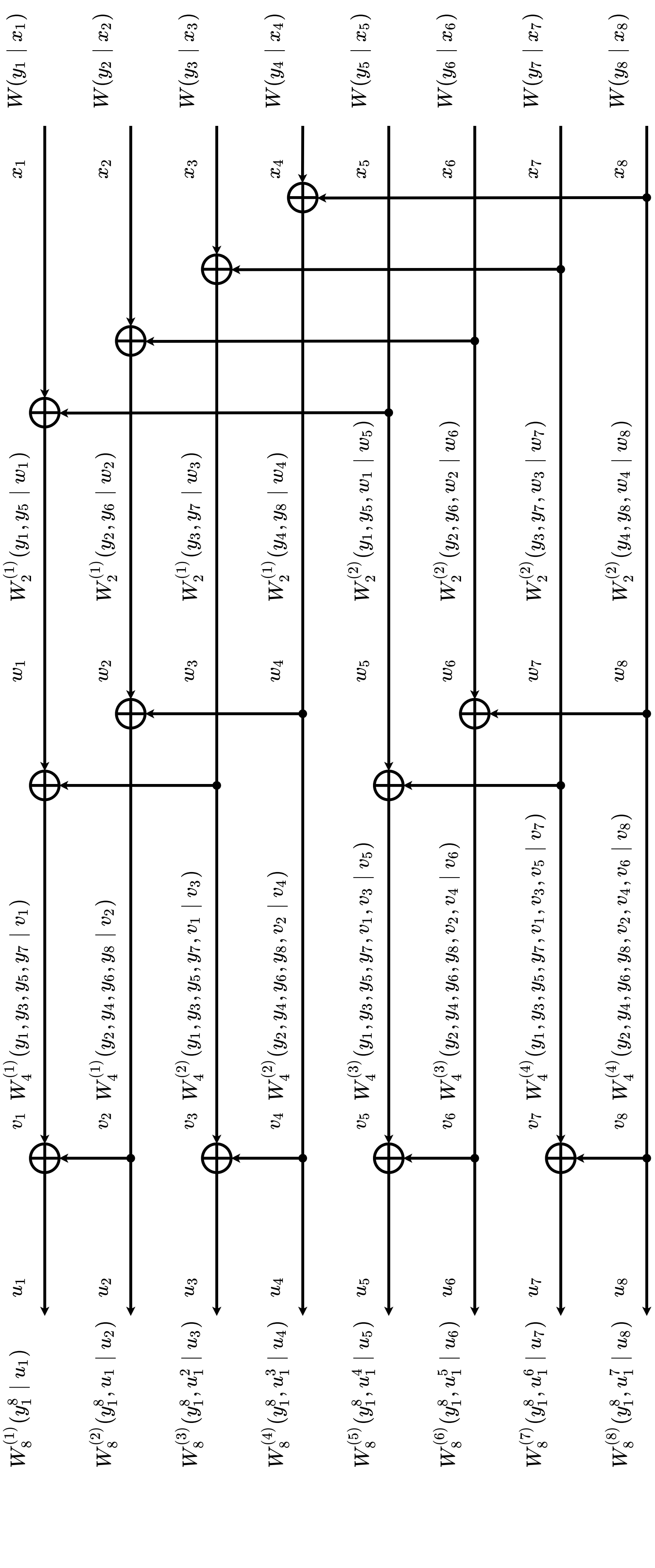

$W_8^{(1)}(y_1^8 \mid u_1)$ u_1 v_1 $W_4^{(1)}(y_1,y_3,y_5,y_7 \mid v_1)$ w_1 $W_2^{(1)}(y_1,y_5 \mid w_1)$ x_1 $W(y_1 \mid x_1)$
$W_8^{(2)}(y_1^8,u_1 \mid u_2)$ u_2 v_2 $W_4^{(1)}(y_2,y_4,y_6,y_8 \mid v_2)$ w_2 $W_2^{(1)}(y_2,y_6 \mid w_2)$ x_2 $W(y_2 \mid x_2)$
$W_8^{(3)}(y_1^8,u_1^2 \mid u_3)$ u_3 v_3 $W_4^{(2)}(y_1,y_3,y_5,y_7,v_1 \mid v_3)$ w_3 $W_2^{(1)}(y_3,y_7 \mid w_3)$ x_3 $W(y_3 \mid x_3)$
$W_8^{(4)}(y_1^8,u_1^3 \mid u_4)$ u_4 v_4 $W_4^{(2)}(y_2,y_4,y_6,y_8,v_2 \mid v_4)$ w_4 $W_2^{(1)}(y_4,y_8 \mid w_4)$ x_4 $W(y_4 \mid x_4)$
$W_8^{(5)}(y_1^8,u_1^4 \mid u_5)$ u_5 v_5 $W_4^{(3)}(y_1,y_3,y_5,y_7,v_1,v_3 \mid v_5)$ w_5 $W_2^{(2)}(y_1,y_5,w_1 \mid w_5)$ x_5 $W(y_5 \mid x_5)$
$W_8^{(6)}(y_1^8,u_1^5 \mid u_6)$ u_6 v_6 $W_4^{(3)}(y_2,y_4,y_6,y_8,v_2,v_4 \mid v_6)$ w_6 $W_2^{(2)}(y_2,y_6,w_2 \mid w_6)$ x_6 $W(y_6 \mid x_6)$
$W_8^{(7)}(y_1^8,u_1^6 \mid u_7)$ u_7 v_7 $W_4^{(4)}(y_1,y_3,y_5,y_7,v_1,v_3,v_5 \mid v_7)$ w_7 $W_2^{(2)}(y_3,y_7,w_3 \mid w_7)$ x_7 $W(y_7 \mid x_7)$
$W_8^{(8)}(y_1^8,u_1^7 \mid u_8)$ u_8 v_8 $W_4^{(4)}(y_2,y_4,y_6,y_8,v_2,v_4,v_6 \mid v_8)$ w_8 $W_2^{(2)}(y_4,y_8,w_4 \mid w_8)$ x_8 $W(y_8 \mid x_8)$

Figure 3.13 – An example of the split channel recursive evolution for $N = 8$.

3.6 CODE CONSTRUCTION

We showed earlier in this chapter that polar codes must be constructed choosing the bit indexes i on which the Bhattacharyya parameter of the split channel $Z(W_N^{(i)})$ are the smallest. Polar codes are non-universal, that is, its construction depends on the channel W. However, [18] showed that the non-universality of polar codes are an aspect of the SC decoding proposed by Arikan, not the encoding matrix itself.

Code construction for an arbitrary B-DMC W is possible, though impractical. This is because the Bhattacharyya parameter of the split channels would be obtained through

$$Z(W_N(i)) = \sum_{y_1^N \in \mathcal{Y}^N} \sum_{u_1^{i-1} \in \mathcal{X}^{i-1}} \sqrt{W_N^{(i)}(y_1^N, u_1^{i-1} \mid 0) W_N^{(i)}(y_1^N, u_1^{i-1} \mid 1)}, \tag{3.30}$$

which is a sum over all possible outputs and is, for larger N, impractical. We have, however, some recursive inequalities that hold for every W, namely [4]

$$Z\left(W_{2N}^{(2i-1)}\right) \leq 2Z\left(W_N^{(i)}\right) - Z\left(W_N^{(i)}\right)^2, \tag{3.31}$$

$$Z\left(W_{2N}^{(2i)}\right) = Z\left(W_N^{(i)}\right)^2, \tag{3.32}$$

where equality is only observed iff W is a Binary Erasure Channel (BEC).

Arikan [4] proposed a Monte Carlo method for determining $Z(W_N^{(i)})$. However, since extensive computer simulations have to be carried out, a search for other methods and approximations began. Next, we explore four approximate construction methods available at the literature [11]. On the rest of this section, we consider the indexing of the bit channels starting at 0.

3.6.1 Bhattacharyya Method

If we consider that the inequalities at Eqs. (3.31, 3.32) hold approximately, we can determine the Bhattacharyya parameters by recursion. The proposed approximation for the Additive White Gaussian Noise (AWGN) channel requires us to start at

$$Z(W) = \exp\left(-\frac{E_s}{N_0}\right) \tag{3.33}$$

and then follow by applying Eqs. (3.31, 3.32). The derivation of Eq. (3.33) is left to the Appendix B.

For a BEC with erasure probability ϵ, it is easy to verify that $Z(W) = \epsilon$. Then, since Eqs. (3.31, 3.32) hold for any BEC, this construction corresponds to the construction for a

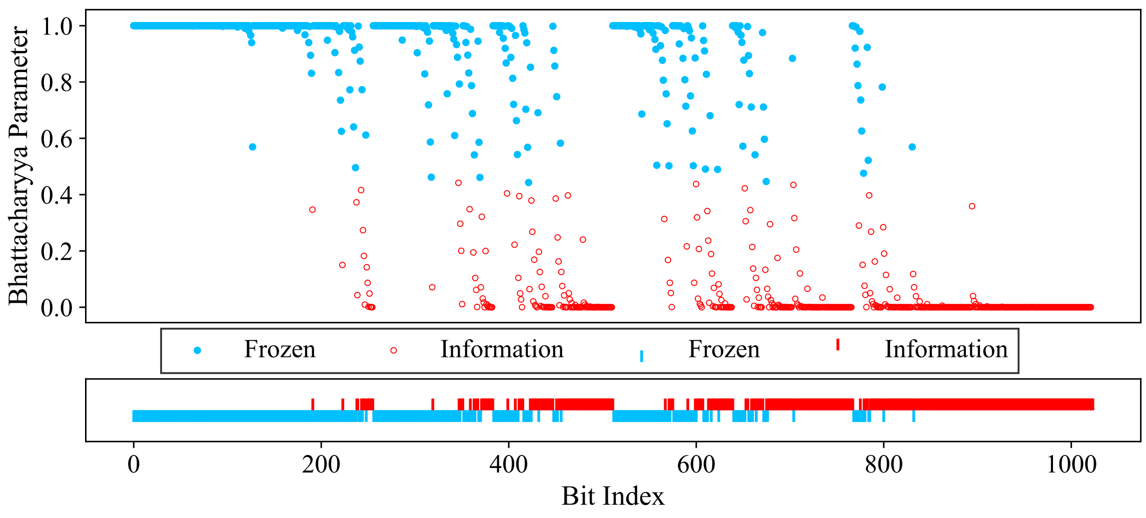


Figure 3.14 – Plot of the Bhattacharyya parameters $Z(W_N^{(i)})$ obtained with the Bhattacharyya construction method for $N = 1024$ and $E_s/N_0 = -1.5$ dB. The information and frozen indexes are chosen for $K = 512$.

BEC with erasure probability $\exp(-E_s/N_0)$.

This process is summarized by Algorithm 1. At the end of the process, the vector Z is sorted from lower to higher values, and the information set $\mathcal{A}$ is chosen to contain the indexes with the smaller Bhattacharyya parameters.

Figure 3.14 shows the Bhattacharyya parameters obtained using Algorithm 1 for $N = 1024$ and $E_s/N_0 = -1.5$ dB. Choosing the $K = 512$ smallest values results on the division shown. It can be seen that the information bits concentrate on the higher indexes, while frozen bits concentrate in lower indexes. However, we see clusters of frozen bits among the information bits, and vice-versa.

Algorithm 1: Bhattacharyya Method

```
Input: Design $E_s/N_0$ in dB ($E_s/N_0|_{\text{dB}}$), code order $n$ (length $N = 2^n$)
Output: Vector of Bhattacharyya parameters ($Z$)
1 $Z'[0] = \exp(-10^{E_s/N_0|_{\text{dB}}/10})$;
2 for $i = 1$ to $n$ do
3     for $j = 0$ to $2^{i-1} - 1$ do
4         $z = Z'[j]$;
5         $Z[2j] = 2z - z^2$;
6         $Z[2j+1] = z^2$;
7     $Z' = Z$
8 return $Z$;
```

3.6.2 Density Evolution - Gaussian Approximation (DEGA)

Recall that on Example 2.3 we showed that the log-likelihood ratios for a Binary Phase-Shift Keying (BPSK) system under AWGN transmission was given by

$$L(y) = \frac{2r}{\sigma^2}, \tag{3.34}$$

where y is the received complex sample $y = r + j \cdot q$. Under the assumption of all zero transmission, we have that $L(y)$ is random variable drawn from a Gaussian distribution with mean

$$\begin{aligned} E\left[\frac{2r}{\sigma^2}\right] &= \frac{2}{\sigma^2}E[r] \\ &= \frac{2}{\sigma^2} \end{aligned} \tag{3.35}$$

and variance

$$\begin{aligned} \text{var}\left[\frac{2r}{\sigma^2}\right] &= \frac{4}{\sigma^4}\text{var}[r] \\ &= \frac{4}{\sigma^2}. \end{aligned} \tag{3.36}$$

Here, we assumed $E_s = 1$.

An alternative to finding the Bhattacharyya parameters is to find the distribution of the values $L_N^{(i)} = \log(l_N^{(i)})$. Remember that if $L_N^{(i)}$ is close to zero, then the decision of the bit i is unreliable. It is known that the distribution of this values is not Gaussian, since transforming $L(y)$ into $L_N^{(i)}$ requires non-linear transformations on the f and g nodes. However, the Density Evolution - Gaussian Approximation (DEGA) method assumes two key things:

1. that the density evolves approximately as Gaussian as we do these transformations;
2. that the relationship between the mean and variance $\text{var}[L_N^{(i)}] = 2 \cdot E[L_N^{(i)}]$ holds as the densities evolve.

Then, one just needs to keep track of the distributions means. At the end of the process, the lower the absolute value of the mean is, the closer to zero the values of $L_N^{(i)}$ usually lie and less reliable the bit channel i is.

Using the terminology $m_N^i = E[L_N^{(i)}]$, the DEGA method performs the following updates [10]

$$m_{2N}^{2i} = \phi^{-1}\left(1 - \left(1 - \phi\left(m_N^i\right)\right)^2\right), \tag{3.37}$$

$$m_{2N}^{2i+1} = 2m_N^i, \tag{3.38}$$

where ϕ is an approximation function [19] of the one at [10]

$$\phi(x) = \begin{cases} \exp(-0.4527x^{0.86} + 0.0218), & 0 < x < 10, \\ \sqrt{\frac{\pi}{x}} \exp\left(\frac{-x}{4}\right)\left(1 - \frac{10}{7x}\right), & x \geq 10. \end{cases} \tag{3.39}$$

The inverse function ϕ^{-1} was approximated numerically by a piecewise function.

Algorithm 2 shows the complete procedure. After obtaining the mean vector, the values are sorted from higher to lower values, and the indexes with the largest means are selected to compose $\mathcal{A}$.

Figure 3.15 shows the means m_N^i obtained with the DEGA construction method for $N = 1024$ and $E_s/N_0 = -1.5$ dB. It can be seen that the behaviour of the means is quite erratic, even for low and high indexes. However, for this E_s/N_0 setting the distribution of frozen and information bits is quite similar to the one in Figure 3.14.

Algorithm 2: DEGA Method

```
Input: Design $E_s/N_0$ in dB ($E_s/N_0|_{\text{dB}}$), code order $n$ (length $N = 2^n$)
Output: Vector of approximated means ($M$)
1 $M'[0] = 4 \times 10^{E_s/N_0|_{\text{dB}}/10}$;
2 for $i = 1$ to $n$ do
3     for $j = 0$ to $2^{i-1} - 1$ do
4         $m = M'[j]$;
5         $M[2j] = \phi^{-1}(1 - (1 - \phi(m))^2)$;
6         $M[2j+1] = 2m$;
7     $M' = M$
8 return $M$;
```

3.6.3 Modified DEGA (M-DEGA)

Instead of using the ϕ function to update the means, another approximation is derived at [11], inspired by the BPSK error probability under AWGN channel

$$P_e = Q\left(\sqrt{2\frac{E_s}{N_0}}\right), \tag{3.40}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\frac{t^2}{2}} dt\,. \tag{3.41}$$

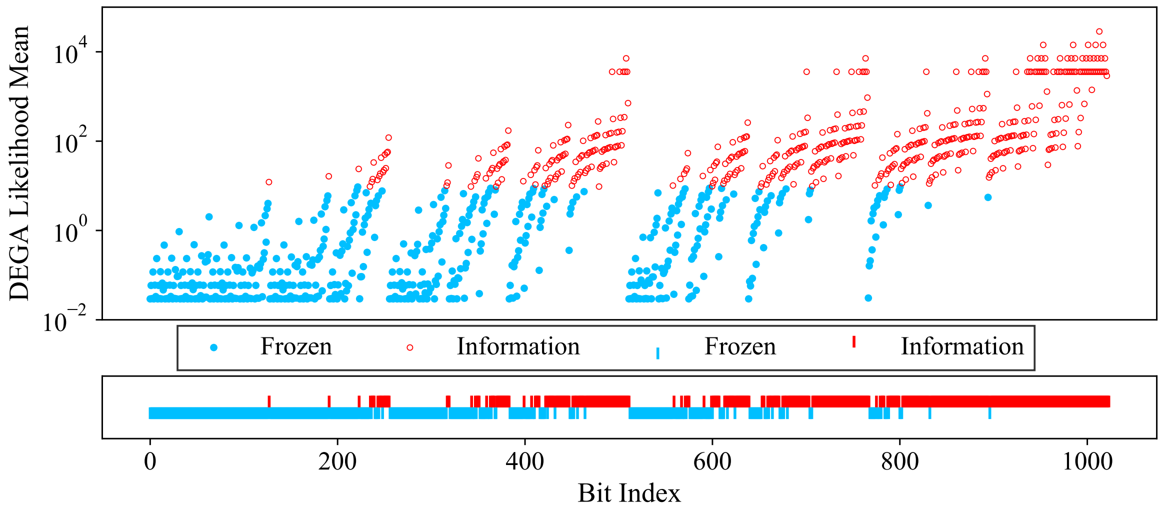


Figure 3.15 – Plot of the likelihood means m_N^i obtained with the DEGA construction method for $N = 1024$ and $E_s/N_0 = -1.5$ dB. The information and frozen indexes are chosen for $K = 512$.

The means are updated by

$$m_{2N}^{2i} = 2Q^{-1}\left(2Q\left(\sqrt{\frac{m_N^i}{2}}\right)\left(1 - Q\left(\sqrt{\frac{m_N^i}{2}}\right)\right)\right)^2, \tag{3.42}$$

$$m_{2N}^{2i+1} = 2m_N^i, \tag{3.43}$$

and the complete algorithm is shown in Algorithm 3. At the end of the algorithm, just like in DEGA, the indexes with the largest means are chosen to compose $\mathcal{A}$, since these values are associated to more certainty of $L_N^{(i)}$.

Algorithm 3: M-DEGA Method

```
   Input: Design $E_s/N_0$ in dB ($E_s/N_0|_{\text{dB}}$), code order $n$ (length $N = 2^n$)
   Output: Vector of approximated means ($M$)
1  $M'[0] = 4 \times 10^{E_s/N_0|_{\text{dB}}/10}$;
2  for $i = 1$ to $n$ do
3      for $j = 0$ to $2^{i-1} - 1$ do
4          $m = M'[j]$;
5          $M[2j] = 2Q^{-1}\left(2Q\left(\sqrt{\frac{m}{2}}\right)\left(1 - Q\left(\sqrt{\frac{m}{2}}\right)\right)\right)^2$;
6          $M[2j+1] = 2m$;
7      $M' = M$
8  return $M$;
```

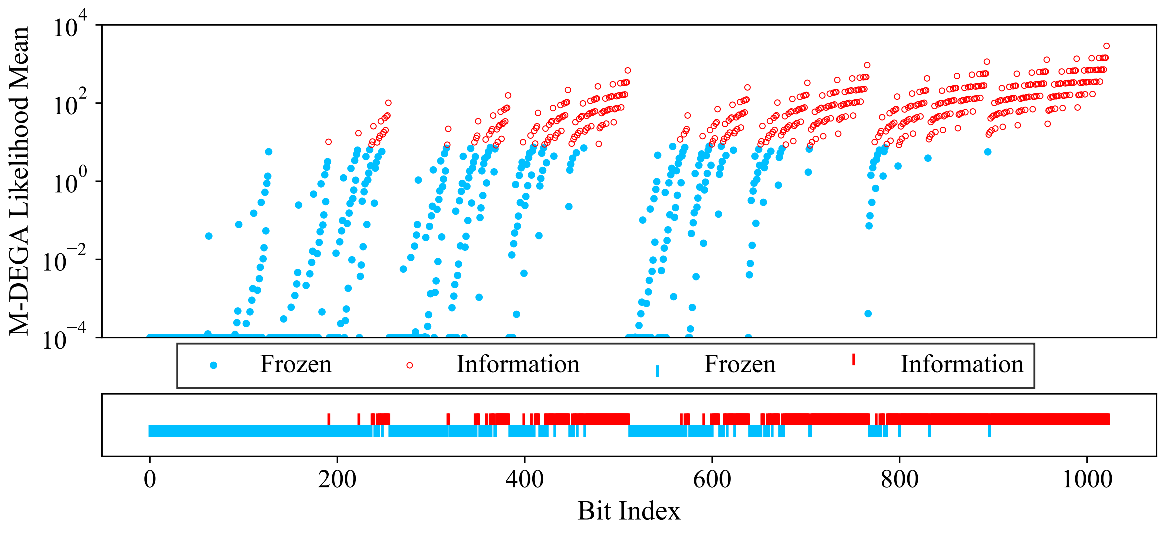


Figure 3.16 – Plot of the likelihood means m_N^i obtained with the M-DEGA construction method for $N = 1024$ and $E_s/N_0 = -1.5$ dB. Values below the plot range were clipped to 10^{-4}. The information and frozen indexes are chosen for $K = 512$.

Figure 3.16 shows the means m_N^i obtained with the M-DEGA construction method for $N = 1024$ and $E_s/N_0 = -1.5$ dB. Similar to DEGA, it can be seen that the behaviour of the means is quite erratic. We also observe that for this E_s/N_0 setting the distribution of frozen and information bits is quite similar to the one in Figure 3.14.

3.6.4 Bit Error Evolution (BEE)

Instead of obtaining the Bhattacharyya parameters or evolving the means, [11] proposes the evolution of the bit error rate of the channels $W_N^{(i)}$ when all bits other than i are frozen. Using the notation $p_N^{(i)}$ to denote the bit error probability of $W_N^{(i)}$, the approximated evolution of these probabilities is

$$p_{2N}^{2i} = 2p_N^i(1 - p_N^i), \tag{3.44}$$

$$p_{2N}^{2i+1} = Q\left(\sqrt{2}Q^{-1}(p_N^i)\right). \tag{3.45}$$

The complete algorithm is shown in Algorithm 4. At the end of the algorithm, the probability vector is sorted and the indexes with the smaller probabilities of error are chosen to compose the information set $\mathcal{A}$.

We simulated the $N = 16$ polar code with all bits frozen, except for bit i, using SC decoding and AWGN channel under BPSK transmission. Figure 3.17 shows the simulation results along with the bit channel error rates obtained with the Bit Error Evolution (BEE) approximation. We immediately see that the approximation matches the simulation results and that bit error rates below 10^{-5} are achieved.

Algorithm 4: BEE Method

```
    Input: Design $E_s/N_0$ in dB ($E_s/N_0|_{\text{dB}}$), code order $n$ (length $N = 2^n$)
    Output: Vector of approximated bit error probabilities ($P$)
1   $P'[0] = Q(\sqrt{2 \cdot 10^{E_s/N_0|_{\text{dB}}/10}})$;
2   for $i = 1$ to $n$ do
3       for $j = 0$ to $2^{i-1} - 1$ do
4           $p = P'[j]$;
5           $P[2j] = 2p(1-p)$;
6           $P[2j+1] = Q(\sqrt{2}Q^{-1}(p))$;
7       $P' = P$
8   return $P$;
```

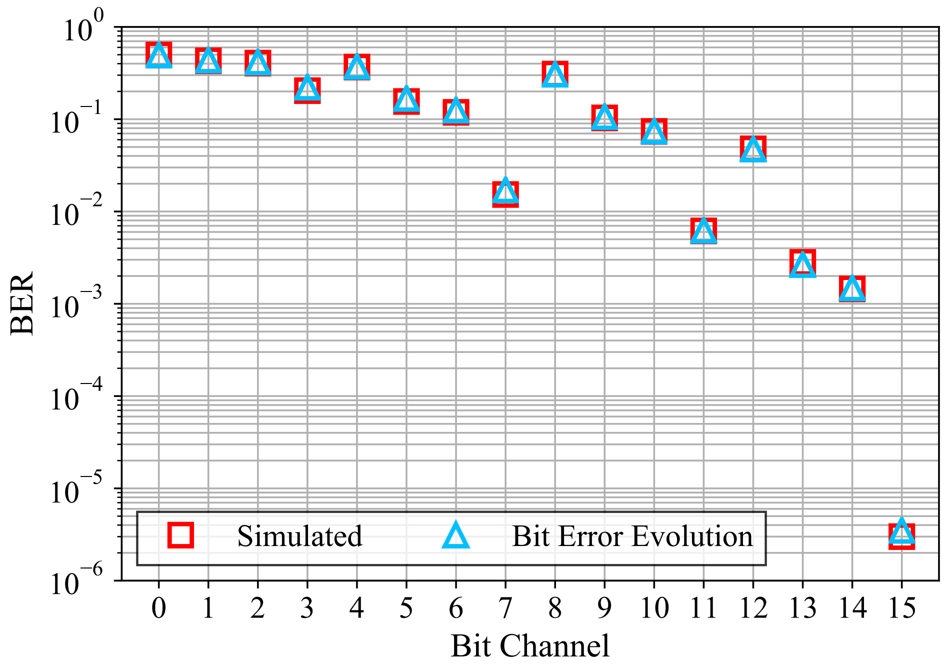


Figure 3.17 – A comparison between the simulated and BEE bit channel error rate for $E_s/N_0 = -2$ dB and $N = 16$.

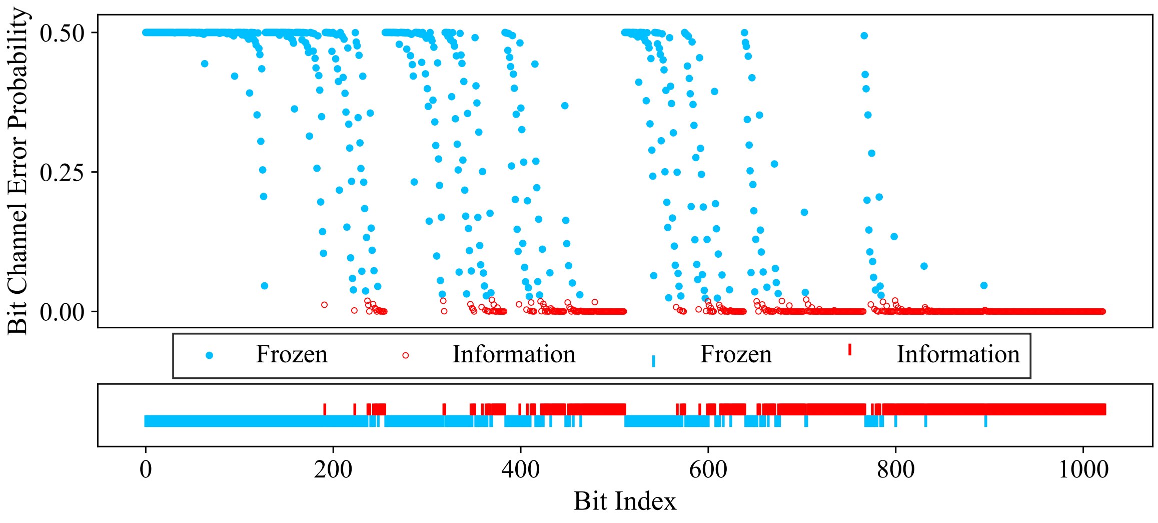


Figure 3.18 – Plot of the bit channel error probability $p_N^{(i)}$ obtained with the BEE construction method for $N = 1024$ and $E_s/N_0 = -1.5$ dB. The information and frozen indexes are chosen for $K = 512$.

Also, Figure 3.18 shows the bit channel error probabilities obtained using Algorithm 4 for $N = 1024$ and $E_s/N_0 = -1.5$ dB. At this point it becomes clear that, at least visually, all of the four methods presented have a similar frozen/information bit distribution.

3.7 SIMULATION RESULTS

At this point we are ready to present some simulation results. Monte Carlo [20] simulations were carried out in order to obtain the Bit Error Rate (BER) and Frame Error Rate (FER) of multiple of polar codes with multiple settings.

3.7.1 Code Construction

Figures 3.19-3.21 show the polar code BER and FER performance comparison using the different construction methods presented for code rates 1/3, 1/2 and 3/4 under AWGN channel and BPSK transmission, with $N = 4096$. The code rate 1/3 cannot be obtained for blocks with size $N = 4096$ and was approximated. When FER is of concern, all three code rates exhibit the same behaviour: DEGA, M-DEGA and BEE have similar performance, DEGA performing slightly better, while the Bhattacharyya method performs the worst. When BER is of concert, at low E_b/N_0 the DEGA method performs worse and at high E_b/N_0 the Bhattacharyya method performs worse. From these results, it is clear that the Bhattacharyya approximation shows poor performance when compared to DEGA, M-DEGA and BEE, while it is unclear which of these last three approximations performs best.

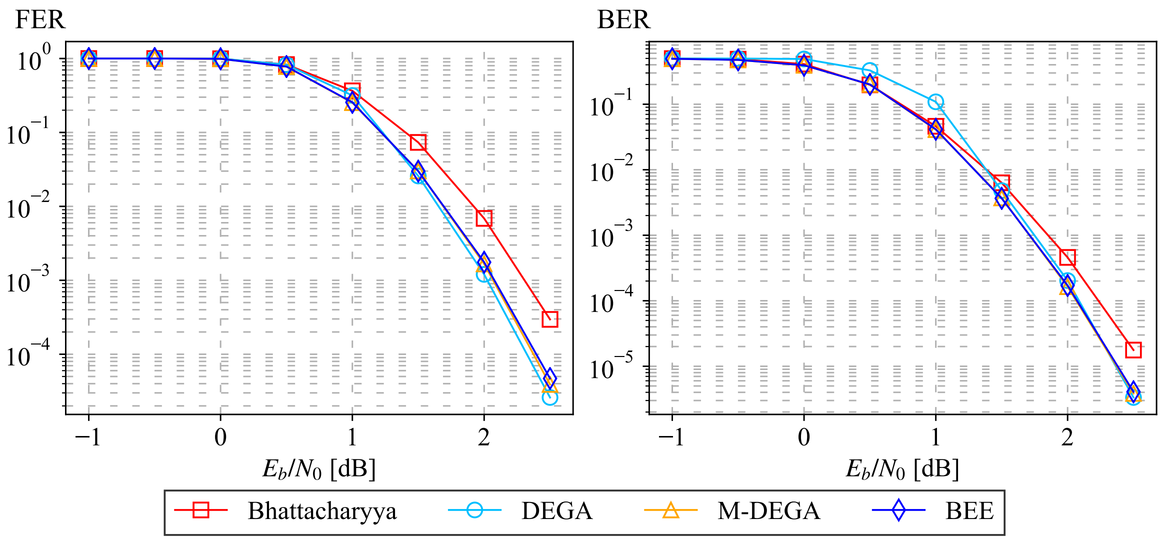


Figure 3.19 – Simulation results for $N = 4096$ and $K = 1365$ ($R \approx 1/3$).

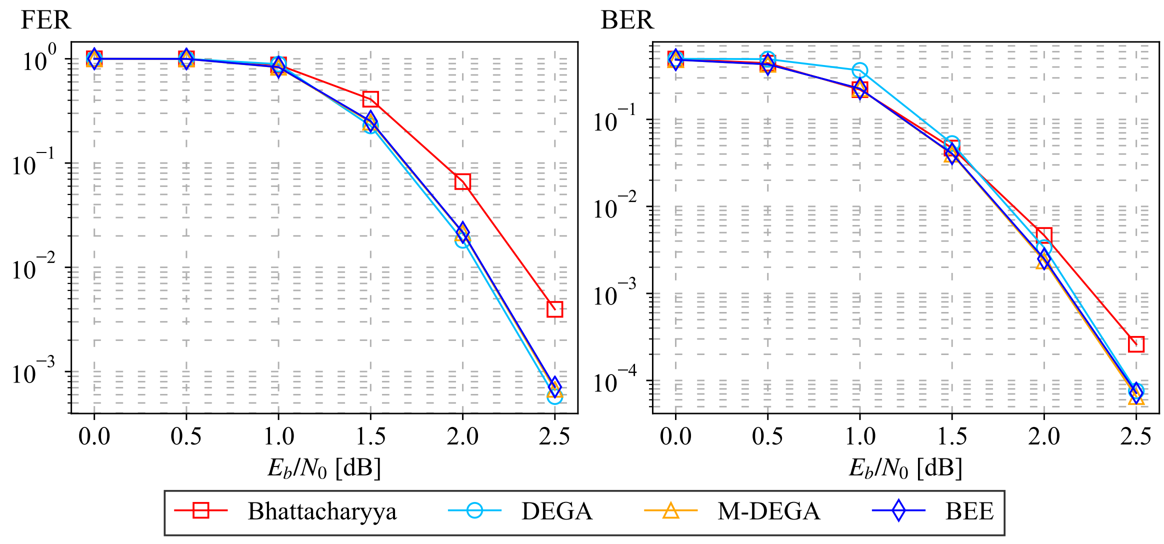


Figure 3.20 – Simulation results for $N = 4096$ and $K = 2048$ ($R = 1/2$).

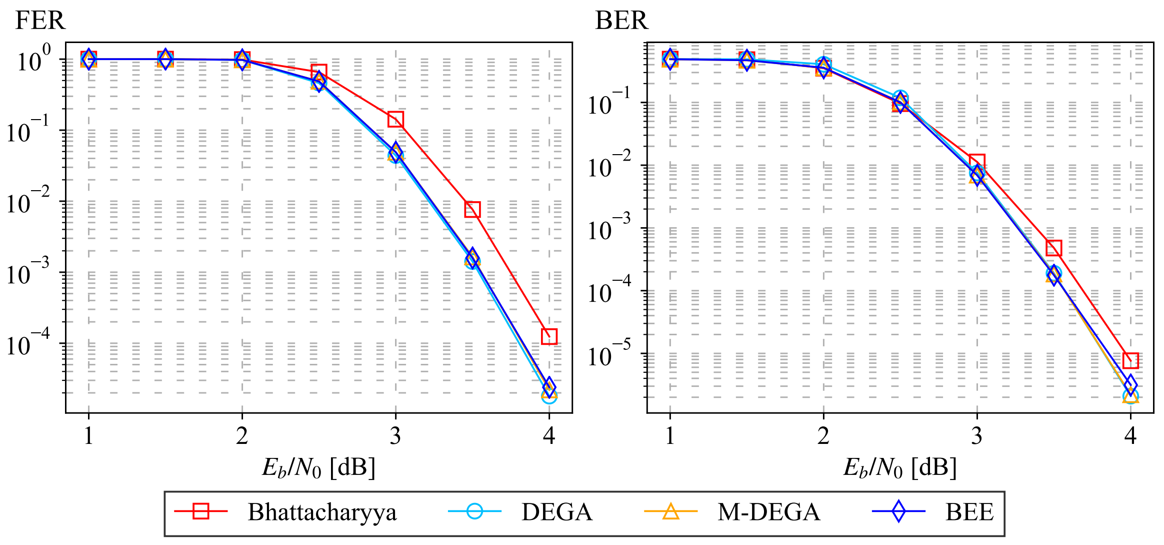


Figure 3.21 – Simulation results for $N = 4096$ and $K = 3072$ ($R = 3/4$).

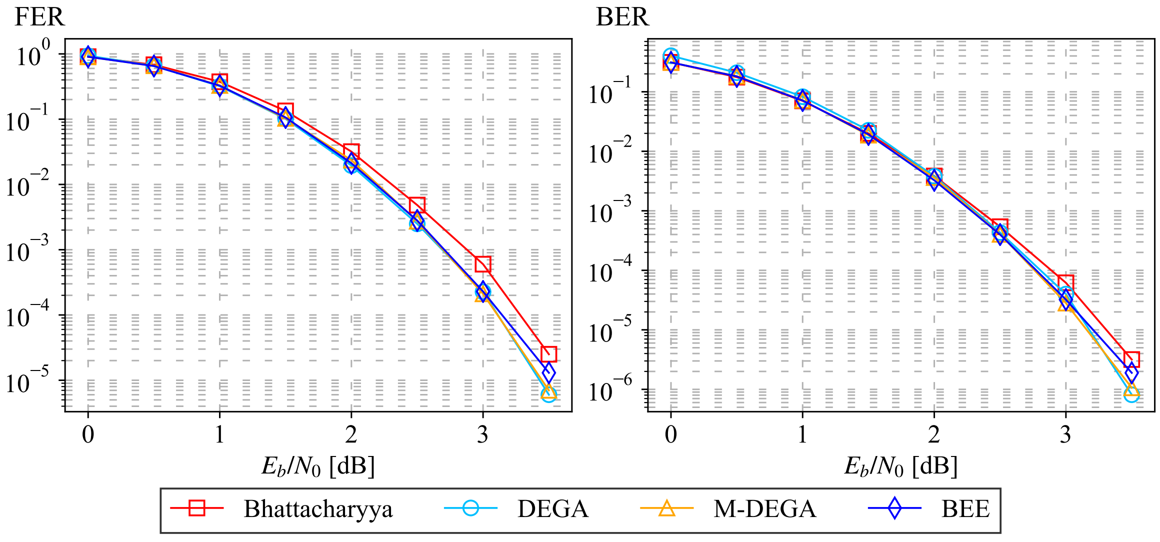


Figure 3.22 – Simulation results for $N = 1024$ and $K = 341$ ($R \approx 1/3$).

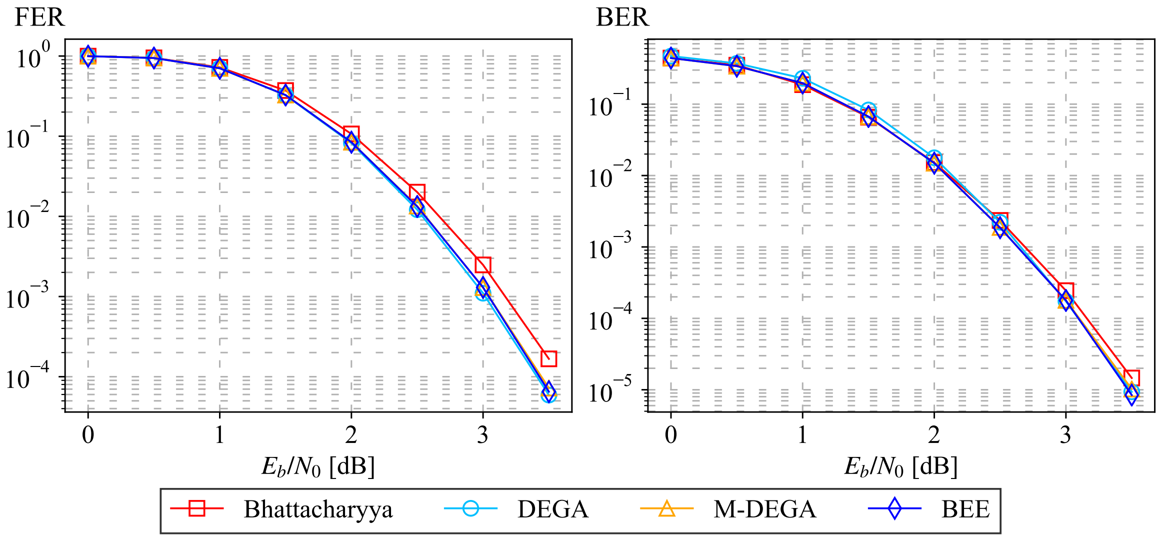


Figure 3.23 – Simulation results for $N = 1024$ and $K = 512$ ($R = 1/2$).

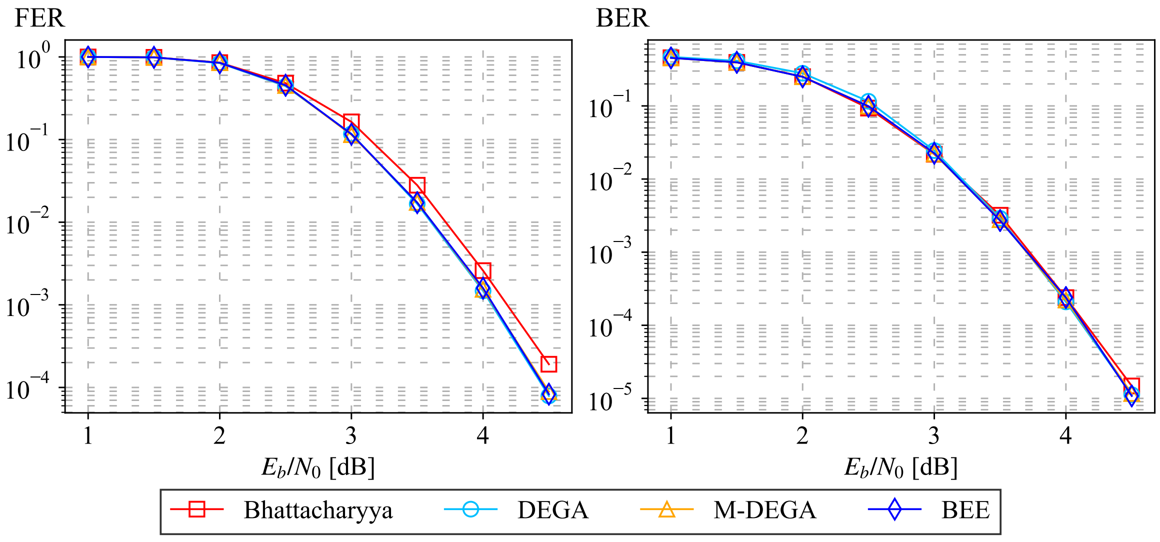


Figure 3.24 – Simulation results for $N = 1024$ and $K = 768$ ($R = 3/4$).

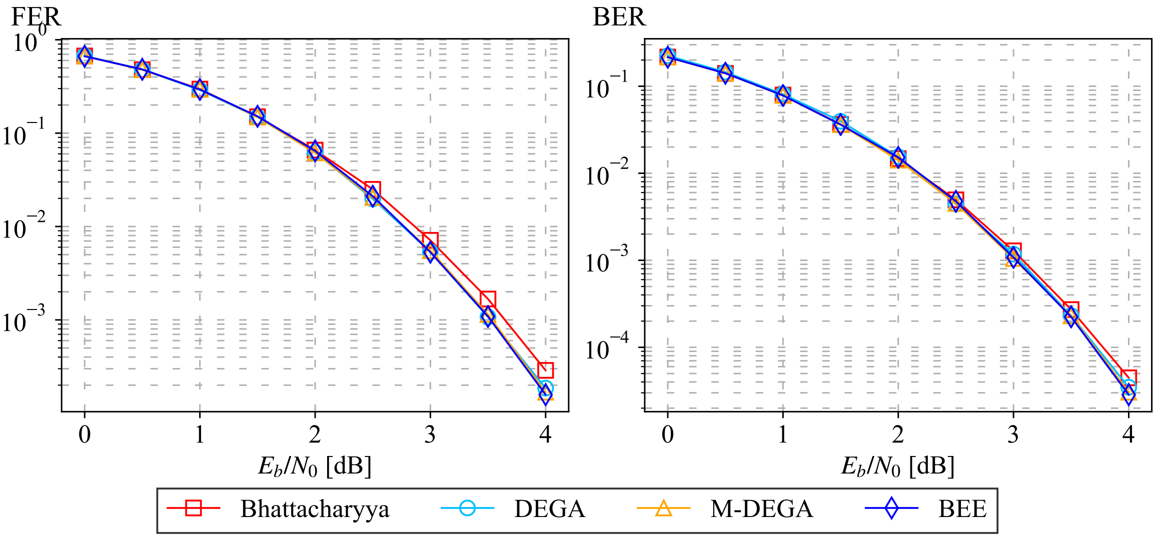


Figure 3.25 – Simulation results for $N = 256$ and $K = 85$ ($R \approx 1/3$).

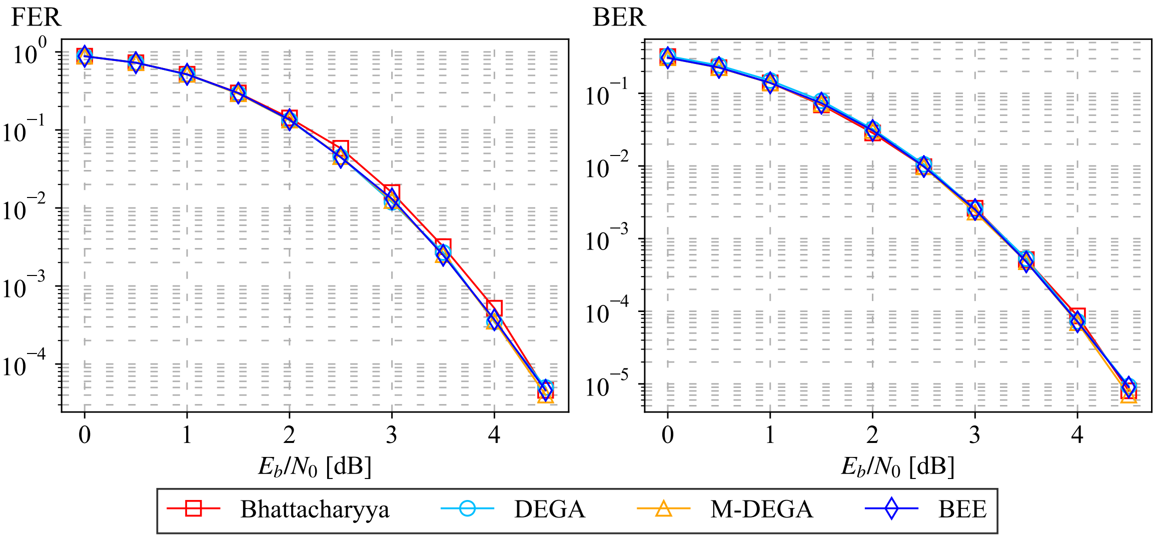


Figure 3.26 – Simulation results for $N = 256$ and $K = 128$ ($R = 1/2$).

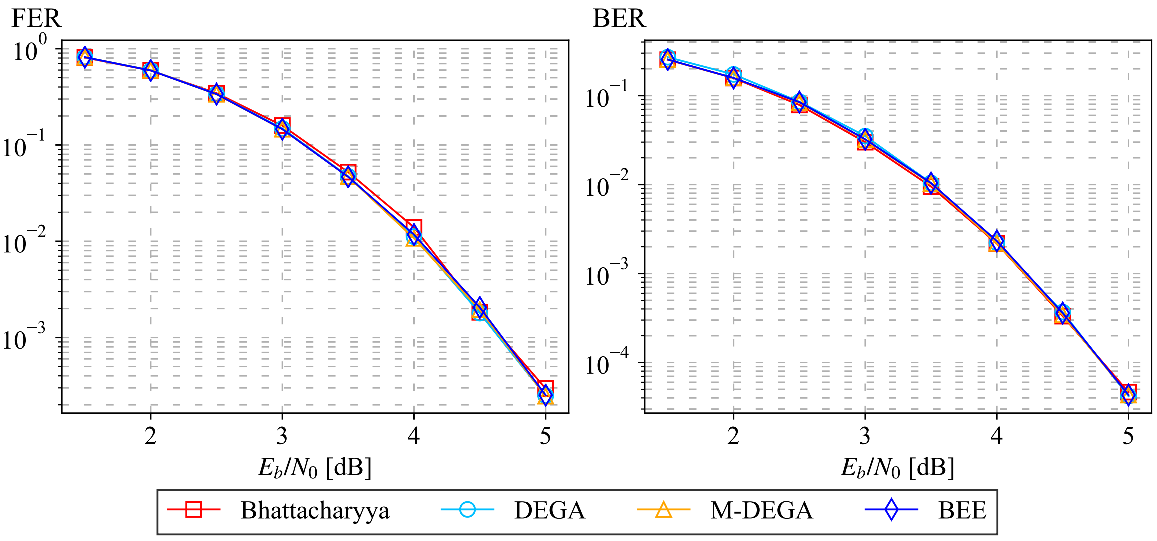


Figure 3.27 – Simulation results for $N = 256$ and $K = 192$ ($R = 3/4$).

Figures 3.22-3.24 show the results for $N = 1024$ and Figures 3.25-3.27 show the results for $N = 256$. We can see that the Bhattacharyya approximation still performs worse, but the overall performance difference for these block lengths is small. Specially, at $N = 256$ and $R = 3/4$, all methods show the same performance.

These curves show that for small N the construction methods are equivalent. Perhaps, the approximation error builds up when constructing larger N, which explains why the curves differ the most for $N = 4096$. To investigate further, we analyze how similar the final distribution of information and frozen bits is for the different methods. We first begin by analysing a case in which a small difference between the methods was observed: $N = 256$, $K = 192$ and $E_b/N_0 = 5$ dB, in Figure 3.27. Figure 3.28 shows the bit indexes on which the DEGA, M-DEGA and BEE approximations have a different result than the Bhattacharyya approximation in this scenario. The Bhattacharyya method was chosen as a reference since all the other methods have similar performance. The up-ticks show positions that were classified as information bits in Bhattacharyya but as frozen bits on the methods shown, and down-ticks show positions that were classified as frozen bits in Bhattacharyya but as information bits on the methods shown. We see that in this case DEGA, M-DEGA and BEE have the same construction, differing only in two positions from the Bhattacharyya method. Since the number of information bits is K for every construction method, the difference observed can be thought as information bit positions that moved to other positions, leaving behind gaps with frozen bits. This also means that the up- and down-ticks come in pairs.

Since the curves for $N = 4096$ and $R = 1/2$ in Figure 3.20 showed a considerable performance difference between the Bhattacharyya method and the other three methods, we investigated the similarity between the frozen and information bit distribution in this case. Figure 3.29 shows the bit indexes on which the DEGA, M-DEGA and BEE approximations have a different result than the Bhattacharyya approximation for $N = 4096$, $K = 2048$ and $E_b/N_0 = 2.5$ dB. We can see that DEGA, M-DEGA and BEE have similar distributions of information and frozen bits. The up-ticks that are seen mostly on higher bit positions represent information positions in Bhattacharyya that were relocated to where the down-ticks are.

In order to analyze how similar the construction methods are for different E_b/N_0, Figures 3.30-3.31 show the similarity between the shown methods to the Bhattacharyya method. The similarity is computed as the number of bit positions that have a different classification than the Bhattacharyya classification. For $N = 256$ and $K = 192$, the largest observed difference was 97.65% for the DEGA method, which corresponds to 6 different bit classifications. For $N = 4096$ and $K = 2048$, the largest observed difference was 97.46% for the DEGA method, which corresponds to 104 different bit classifications. These results show that a small percentage of different bit positions are responsible for the error performance differences observed.

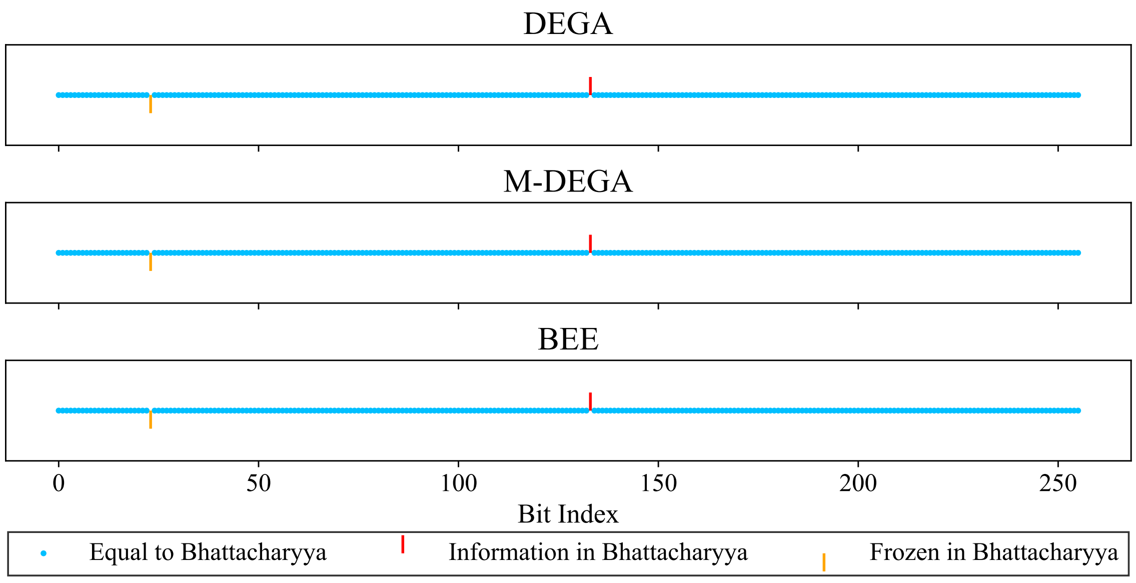


Figure 3.28 – Information and frozen bit differences comparing the shown methods to the Bhattacharyya approximation, with $N = 256$, $K = 192$ and $E_b/N_0 = 5$ dB.

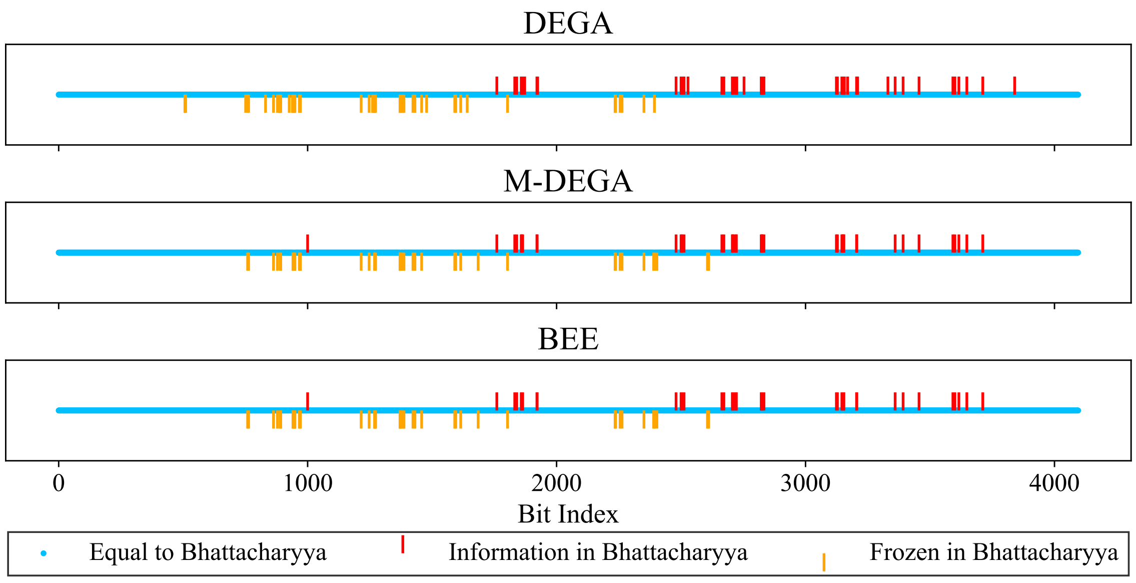


Figure 3.29 – Information and frozen bit differences comparing the shown methods to the Bhattacharyya approximation, with $N = 4096$, $K = 2048$ and $E_b/N_0 = 2.5$ dB.

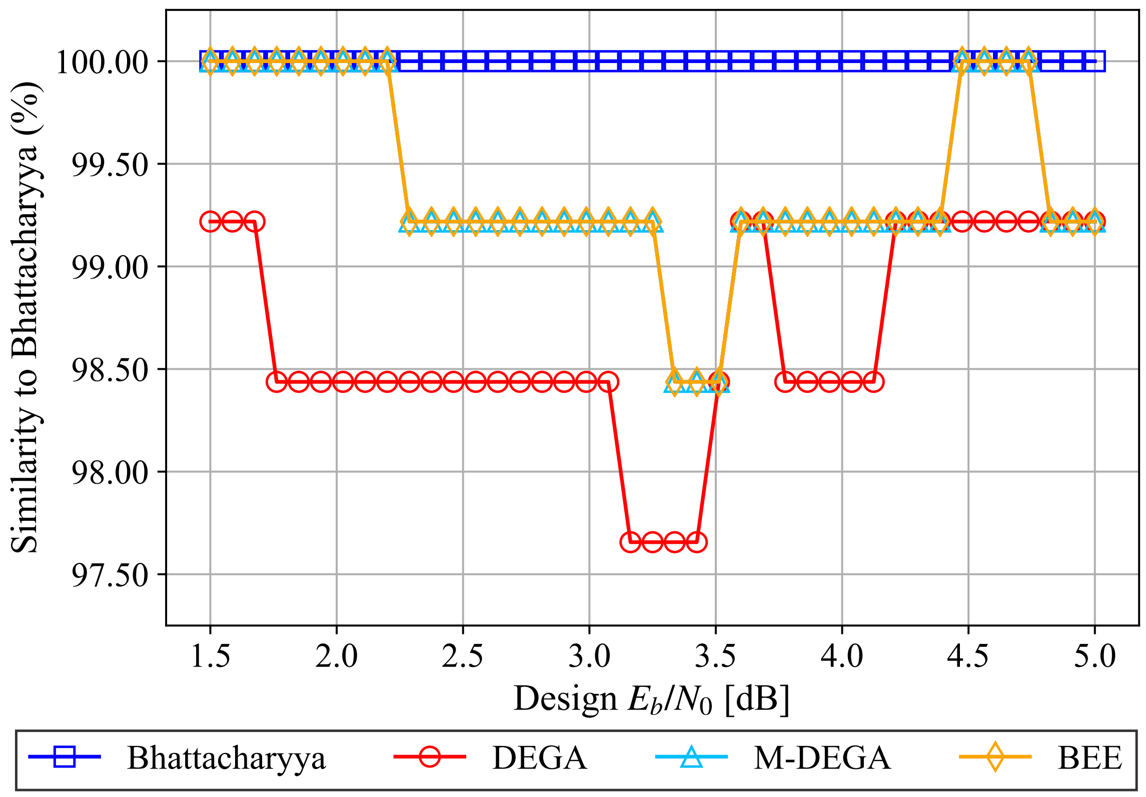


Figure 3.30 – Similarity of the information and frozen bit distributions using the Bhattacharyya approximation as reference, with $N = 256$ and $K = 192$.

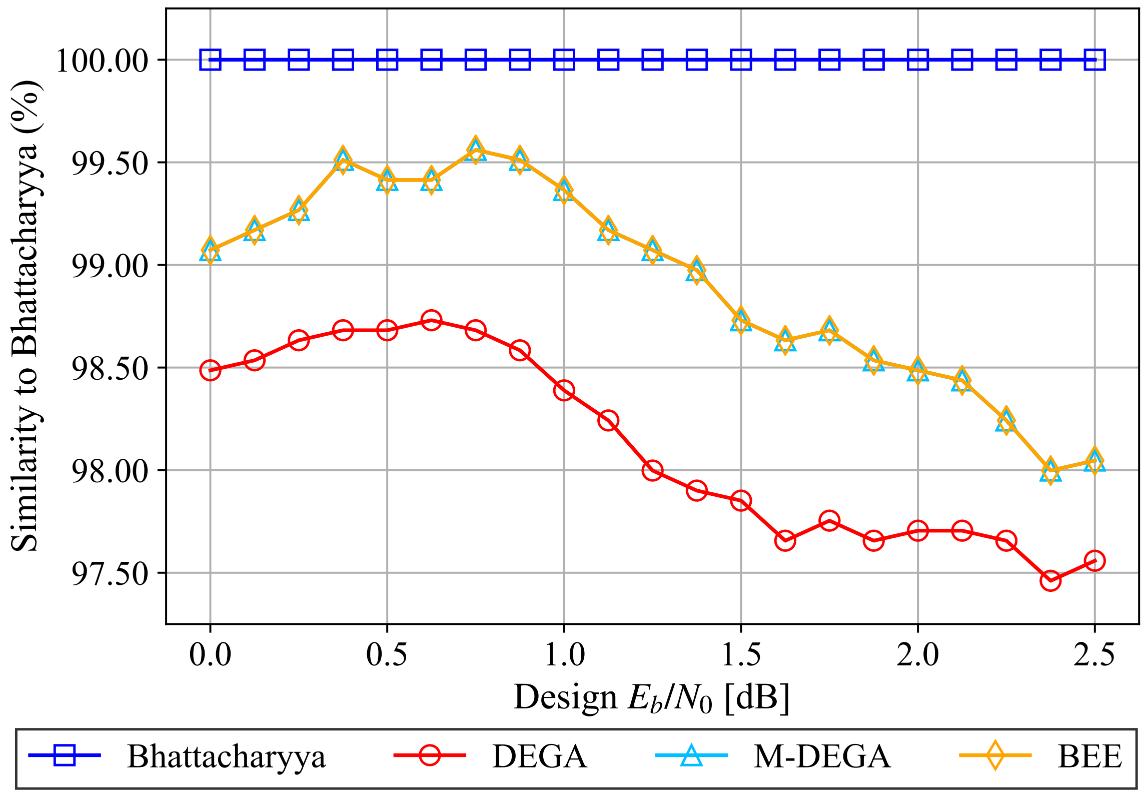


Figure 3.31 – Similarity of the information and frozen bit distributions using the Bhattacharyya approximation as reference, with $N = 4096$ and $K = 2048$.

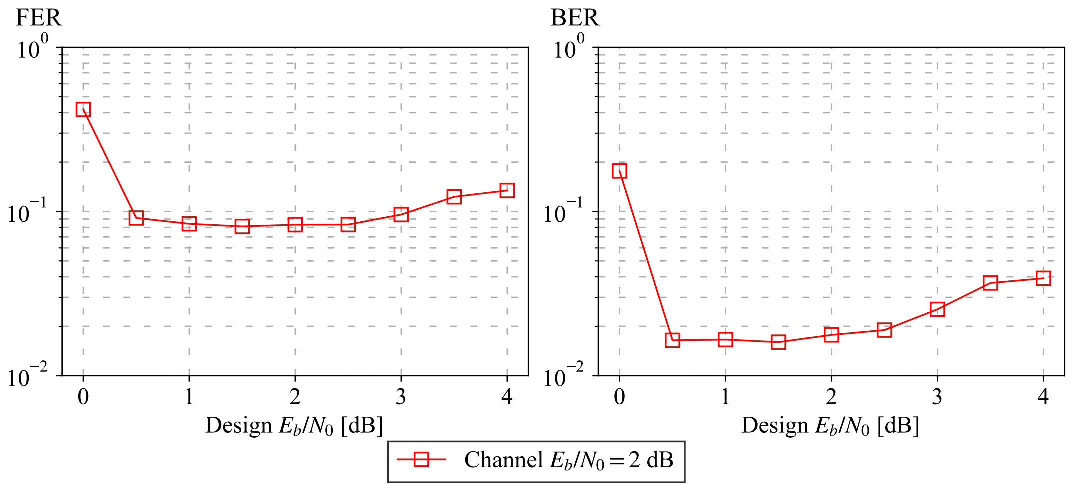


Figure 3.32 – Polar code performance for fixed channel $E_b/N_0 = 2$ dB while varying the design E_b/N_0, for $N = 1024$, $K = 512$ and using the DEGA construction method.

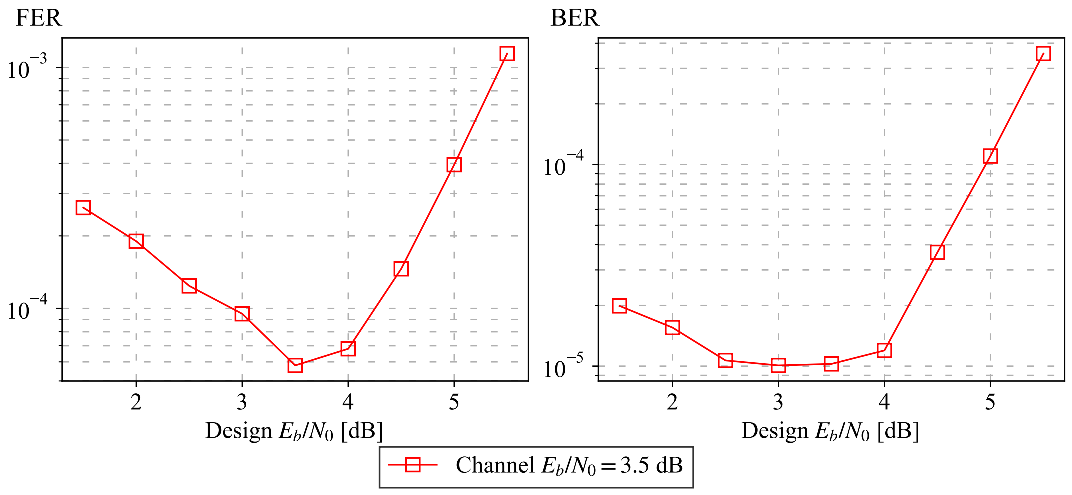


Figure 3.33 – Polar code performance for fixed channel $E_b/N_0 = 3.5$ dB while varying the design E_b/N_0, for $N = 1024$, $K = 512$ and using the DEGA construction method.

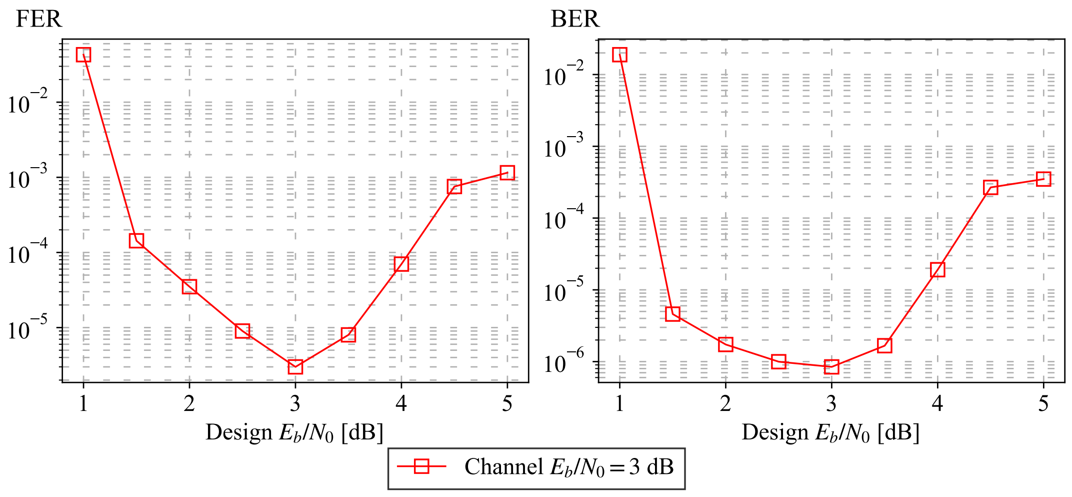


Figure 3.34 – Polar code performance for fixed channel $E_b/N_0 = 3$ dB while varying the design E_b/N_0, for $N = 4096$, $K = 2048$ and using the DEGA construction method.

Another important thing to consider is that in practical applications matching the design E_s/N_0 to the channel E_s/N_0 may not be possible. This is because the channel E_s/N_0 is unknow; it has to be estimated. Another problem is how often one can estimate the channel E_s/N_0, or how often can the encoder and decoder change their frozen bit configuration. A possible strategy is to quantize the design E_s/N_0 domain to some fixed values and transmit with the compromise of performance in favor of the convenience of using the same encoder and decoder design for a broader operation range.

Figures 3.32-3.34 show what happens for a fixed channel E_b/N_0 while varying the design E_b/N_0 using the DEGA construction method. On the $N = 1024$, $K = 512$ and channel $E_b/N_0 = 2$ dB scenario, we can see a small variation on BER and FER performance. Changing the channel E_b/N_0 to 3.5 dB makes the design E_b/N_0 choice more critical, since a local minimum in FER occurs at the design $E_b/N_0 = 3.5$ dB. On the $N = 4096$, $K = 2048$ and channel $E_b/N_0 = 3$ dB scenario we see dramatic performance variation while varying the design E_b/N_0. Figure 3.34 shows a clear local minimum on FER when the design E_b/N_0 matches the channel E_b/N_0, and the FER performance ranges from above 10^{-2} to below 10^{-5}.

Figures 3.32-3.34 also show how polar codes under SC decoding are non-universal codes, meaning that the code construction depends on the channel W. Another way of showing this is by Figure 3.35. There, the bit channel error rate is shown for three different bit channels, and it can be seen that which channel is the most reliable varies as the channel E_s/N_0 changes.

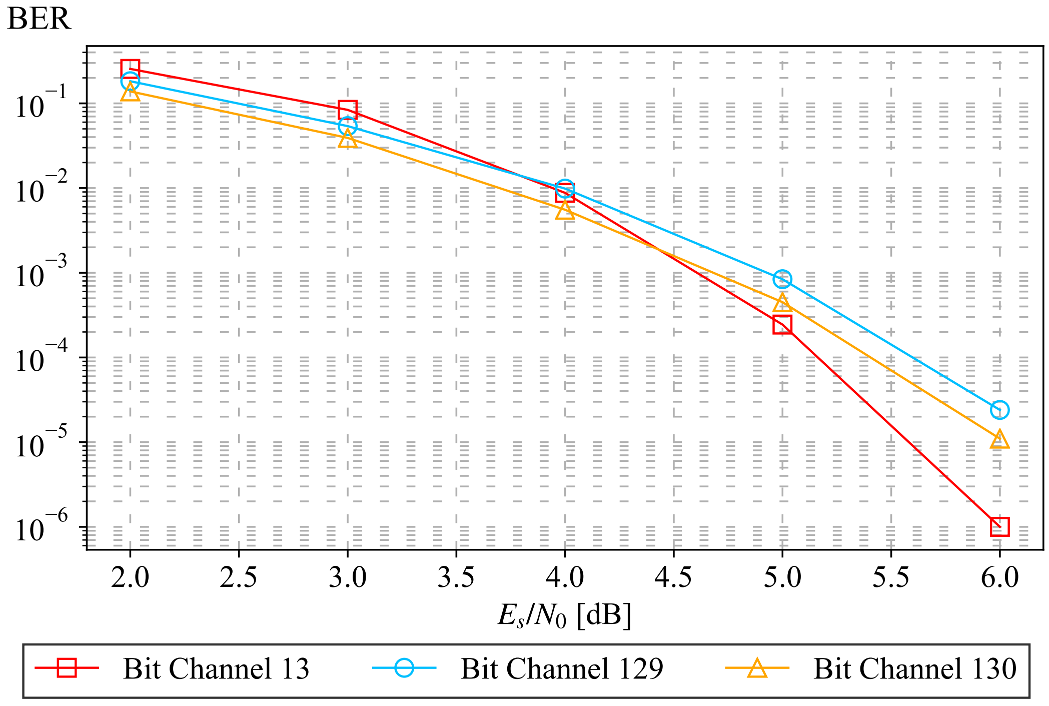


Figure 3.35 – Simulated bit channel error rate for $N = 256$ for the channels 13, 129 and 130.

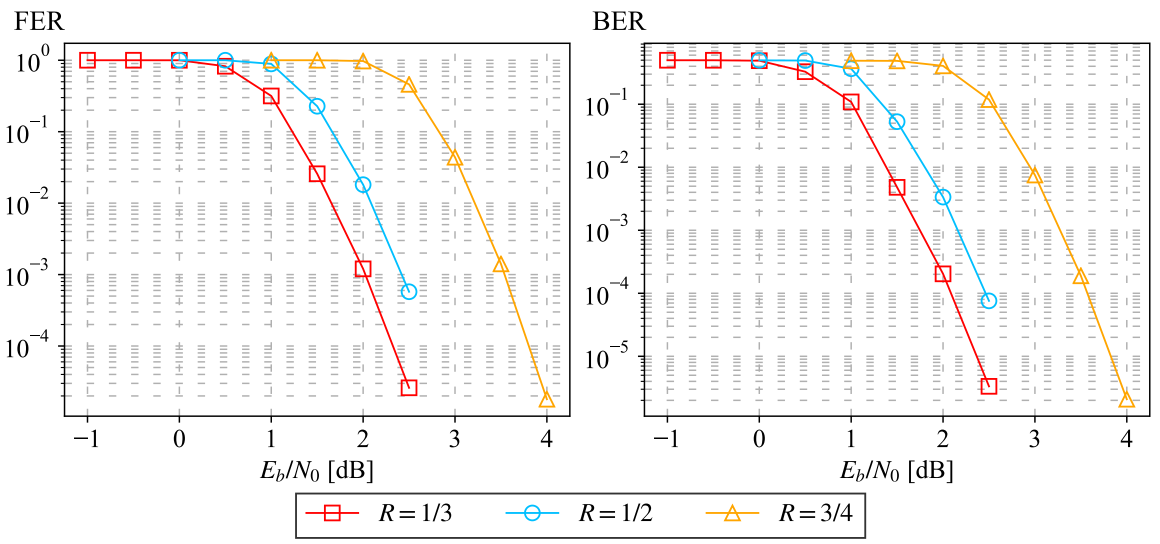


Figure 3.36 – Simulation results for $N = 4096$ using the DEGA construction method.

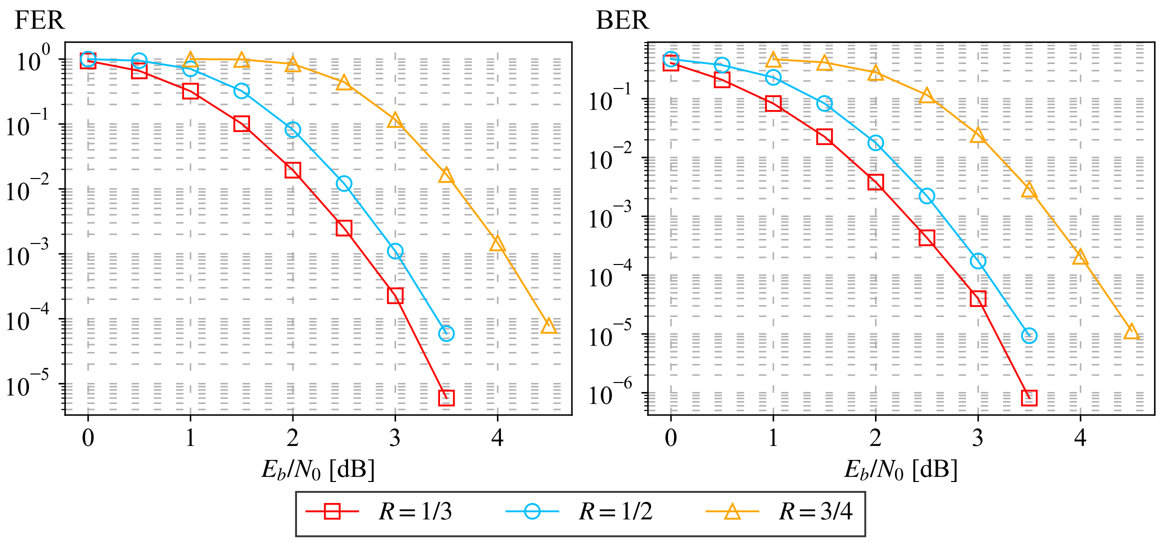


Figure 3.37 – Simulation results for $N = 1024$ using the DEGA construction method.

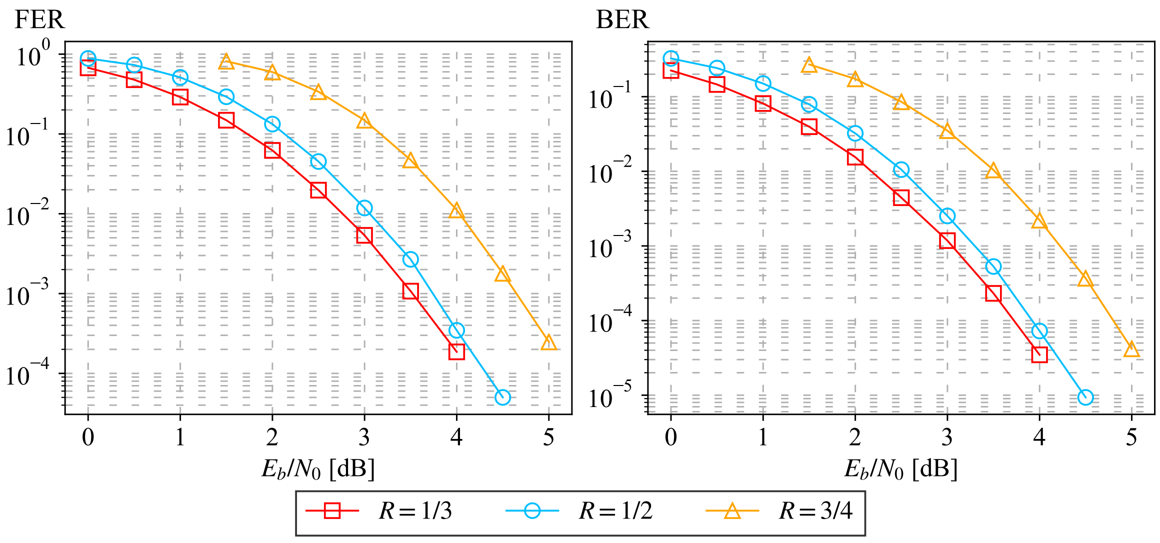


Figure 3.38 – Simulation results for $N = 256$ using the DEGA construction method.

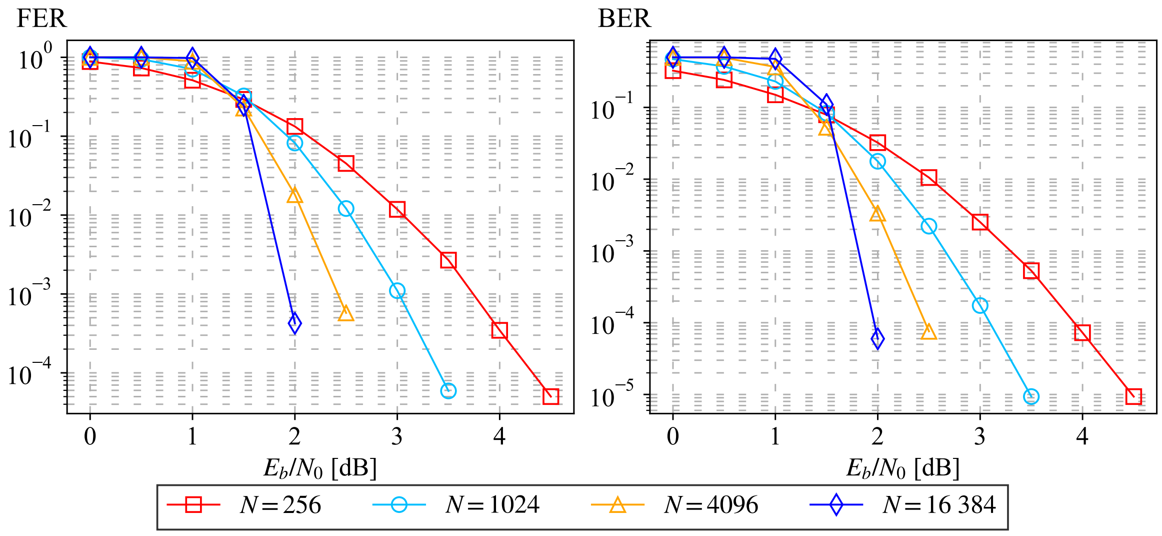


Figure 3.39 – Simulation results for $R = 1/2$ using the DEGA construction method.

3.7.2 Code Rate and Block Size

Figures 3.36-3.38 show how the code rate affects the polar coding performance, using the DEGA construction method for $N = 4096$, $N = 1024$ and $N = 256$. As expected, the more redundancy the block has, the best the code performs. Lastly, Figure 3.39 shows the effect of the block size N on the code performance. Clearly, using larger blocks provide better FER and BER performance. However, as was stated earlier in Section 3.4, using large blocks have a penalty on the decoding latency.

3.7.3 Comparison with Turbo Codes

Figure 3.40 shows a comparison between the state-of-the-art Turbo codes used in the Digital Video Broadcasting - Return Channel via Satellite - Second Generation (DVB-RCS2) system and the DEGA constructed polar code, both with block length $N = 1024$ and code rate $R = 1/2$. It is clear that polar codes perform worse than Turbo codes for the same block length. In order to address this performance issue, a different decoding scheme will be introduced in Chapter 4, namely the Successive Cancellation List (SCL) decoding scheme.

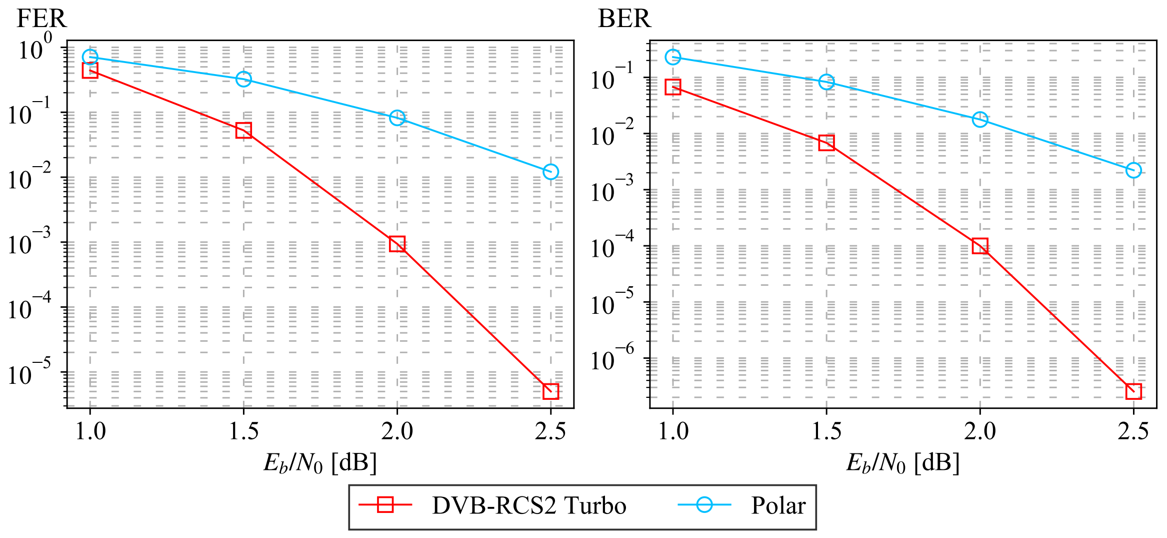


Figure 3.40 – Simulation results for $R = 1/2$ comparing the DVB-RCS2 Turbo codes with the DEGA constructed polar codes for $N = 1024$.

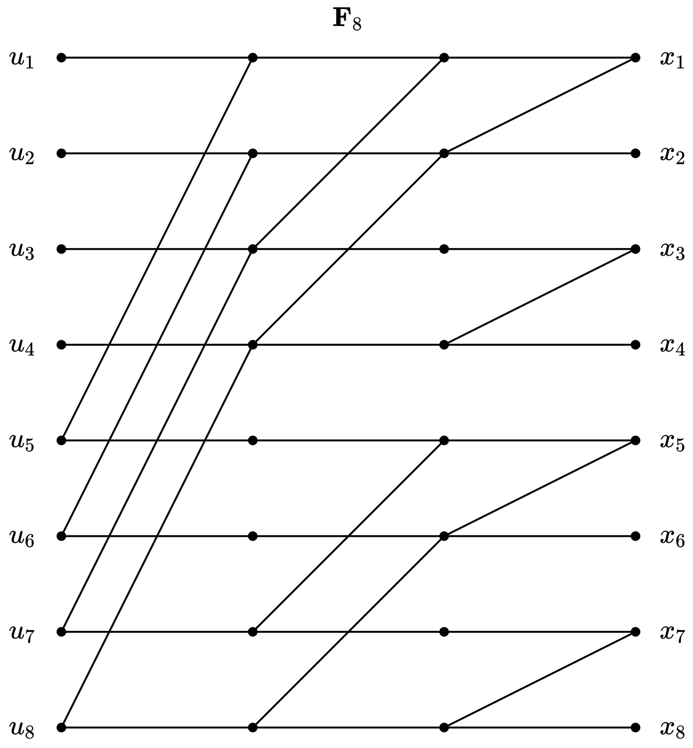
3.8 PRACTICAL ASPECTS OF THE ENCODER IMPLEMENTATION

The first step of the encoder is to fill the K most reliable positions of the vector u_1^N with K message bits, leaving all the other entries *frozen* with zeroes. Then, the second step is to encode u_1^N using the encoding matrix $\mathbf{F}_N$. Although just multiplying the vector u_1^N by $\mathbf{F}_N$ is an option, it isn't the most efficient one. Firstly, if you are implementing polar codes in software, these matrix multiplication will use unnecessary integer multiplications, resulting in large integers that will need to be taken modulo-2 afterwards. Secondly, if implemented in hardware without any parallelization, one would need N time steps, one for each bit, to complete the encoding.

In fact, one can take advantages of the recursive structure of the construction of $\mathbf{F}_N$ and perform the encoding in $n = \log_2 N$ time steps [4]. Figure 3.41 shows such implementation for $N = 8$. Note that if all sums are implemented in parallel only 3 time steps are required to compute $x_1^8 = u_1^8\mathbf{F}_8$, each time step corresponding to a layer in this diagram. This can be generalized for every $N = 2^n$. As discussed earlier, there are two equivalent encoder implementations, one being the mirrored version of the other.

3.8.1 Systematic Encoding

Some linear codes are systematic: given the encoding matrix $\mathbf{G}_{k\times n}$ and the message bits m_1^k, we perform the operation $x_1^n = m_1^k\mathbf{G}$ and get the vector $x_1^n = (m_1^k, p_1^{n-k})$, or a permutation of this vector, meaning that the message bits always appear unaltered on the exact same positions, and the rest of the bits p_1^{n-k} are parity bits. When we perform the polar encoding $x_1^N = u_1^N\mathbf{F}_N$, we in fact obtain a non-systematic code word x_1^N, where the bits $u_\mathcal{A}$



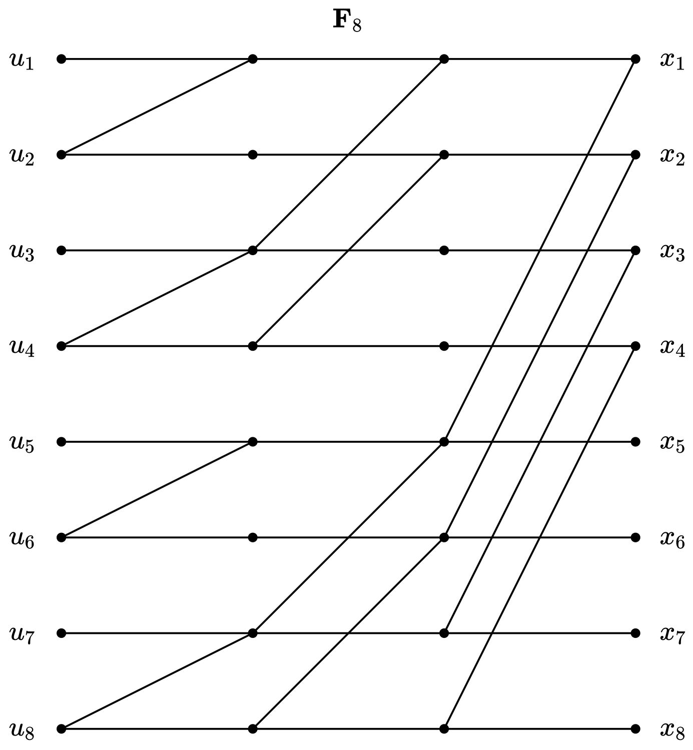


Figure 3.41 – a) An encoder implementation for $\mathbf{F}_8$. Each edge carries a bit value, 0 or 1, and each node adds modulo-2 the values carried by the edges at the left and transmits the results to all edges on the right. b) Equivalent implementation that uses the fact that $\mathbf{F}_N^{-1} = \mathbf{F}_N$.

do not appear in x_1^N explicitly. As we will see later, systematic encoding will be useful when we use decoding algorithms based on binary trees.

Recall that polar codes are $\mathbf{F}_N$-coset codes, meaning we have

$$\begin{aligned} x_1^N &= u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}) \oplus u_{\mathcal{A}^C}\mathbf{F}_N(\mathcal{A}^C) \\ &= u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}), \end{aligned} \tag{3.46}$$

considering that the frozen bits $u_{\mathcal{A}^C}$ are all zero. By separating the columns of $\mathbf{F}_N(\mathcal{A})$ with indexes in $\mathcal{A}$, we obtain the sub-vectors $x_{\mathcal{A}}$ and $x_{\mathcal{A}^C}$, namely

$$x_{\mathcal{A}} = u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}, \mathcal{A}) \tag{3.47}$$

$$x_{\mathcal{A}^C} = u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}, \mathcal{A}^C), \tag{3.48}$$

where $\mathbf{F}_N(\mathcal{A}, \mathcal{B})$ denotes the sub-matrix $[\mathbf{F}_{N i,j}]$ with $i \in \mathcal{A}$ and $j \in \mathcal{B}$. It is easy to verify that $\mathbf{F}_N(\mathcal{A}, \mathcal{A})$ is lower-triangular, meaning it is invertible. Then, if it is desired that $x_{\mathcal{A}}$ contains the information bits, one must feed the encoder with

$$u_{\mathcal{A}} = x_{\mathcal{A}}[\mathbf{F}_N(\mathcal{A}, \mathcal{A})]^{-1}. \tag{3.49}$$

It was shown that this encoding has time complexity $O(N \log_2 N)$[21].

For some, but not all, choices of $\mathcal{A}$, $\mathbf{F}_N(\mathcal{A}, \mathcal{A})$ is its own inverse, meaning that if we feed $x_{\mathcal{A}} = u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}, \mathcal{A})$ again to the encoder and force $x_{\mathcal{A}^C}$ to zero, then we get

$$\begin{aligned} {x_{\mathcal{A}}}' &= x_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}, \mathcal{A}) \\ &= u_{\mathcal{A}}\mathbf{F}_N(\mathcal{A}, \mathcal{A})\mathbf{F}_N(\mathcal{A}, \mathcal{A}) \\ &= u_{\mathcal{A}}, \end{aligned} \tag{3.50}$$

in a way that systematic encoding is possible by passing the information bits through the encoder twice, ensuring that the frozen bits are 0 on each turn. Since the encoding by $\mathbf{F}_N$ is done in time $\log_2 N$ [4], this reduces the amount of time steps to $O(\log_2 N)$.

Arikan [21] showed empirically that systematic polar encoding performs better on the BER, while having the exact same FER. This may be an advantage depending on the application considered: when the whole block is discarded if any error is detected, then both types of encoding are equivalent. Figure 3.42 shows the BER and FER performance of a $N = 4096$, $K = 2048$ polar code constructed using DEGA with BPSK transmission over AWGN obtained by our implementation. We see a decoding behaviour compatible with Arikan's findings [21].

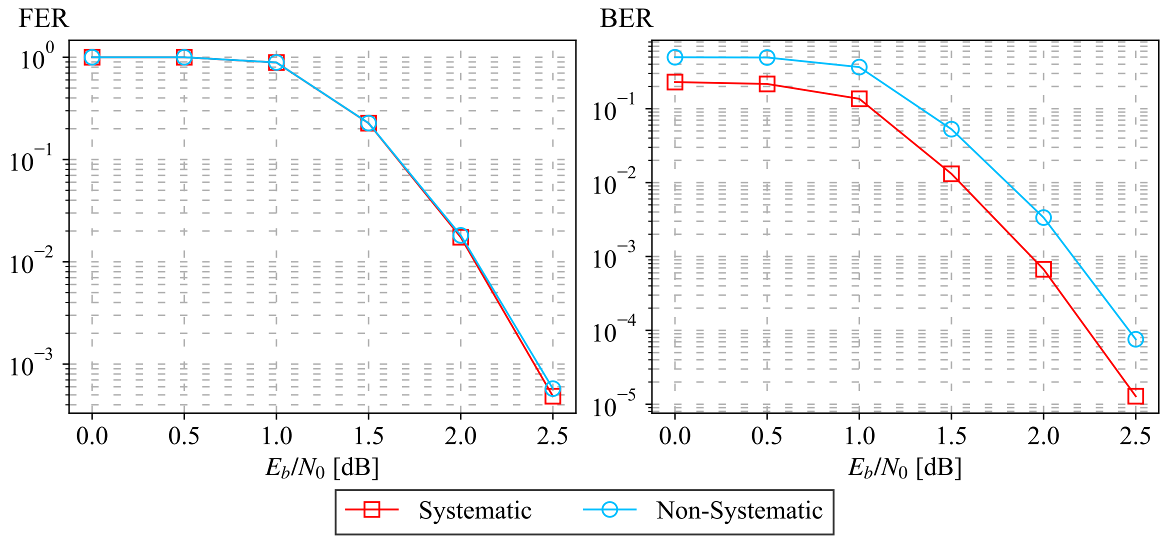


Figure 3.42 – Polar code performance for systematic and non-systematic encoding under SC decoding, with $N = 4096$, $K = 2048$ and constructed using DEGA, transmitted using BPSK over AWGN channel.

3.9 CONCLUSION

We began this chapter by introducing *virtual channels*, a concept on the theoretical foundation of polar codes. While exposing the theoretical preliminaries of polar codes we reviewed the Channel Polarization and the Polar Coding theorems. This last theorem is the one that proves that polar codes are capacity achieving. We also reviewed SC decoding, which makes polar codes attractive due to their low complexity $O(N \log N)$ [4]. However, its latency of $2N - 1$ [4] is a weak point of this technique. We showed that the natural order decoding is not a property of the encoding matrix, but of the recursive structure considered instead. We showed new recursive formulas compatible with the new recursive structure proposed.

Our simulation results showed that polar codes have poor performance for small N, which makes their counterpart Turbo codes more attractive. Also, polar codes exhibit non-universal code construction, which requires code optimization for every channel type and SNR. The optimal construction method is unfeasible and approximations are required. We reviewed the DEGA, M-DEGA, BEE and Bhattacharyya construction methods and presented figures that allowed the visualization of how the parameters used to construct the code evolve for each algorithm. It was shown that for small N all methods have similar performance. For larger N, the DEGA, M-DEGA and BEE methods performed better than the Bhattacharyya method. By comparing the similarity between the construction methods it was shown that a small percentage of bit positions is responsible for the difference observed.

On the next chapter we introduce simplifications to the SC decoding that tackle the latency issue of polar codes. We also introduce the list decoding scheme, which enables error performance gains, and present simulation results that confirm this. We then present an analysis on how list decoders are affected by code construction settings.

4 DECODING ALGORITHMS FOR POLAR CODES

4.1 INTRODUCTION

As it was seen before, polar codes have the disadvantage of having high latency and low performance for small N. After the original Successive Cancellation (SC) decoder was proposed by Arikan [4], some improvements became to appear in the literature that tackled both this problems.

After introducing hardware friendly decoding expressions and an algorithm based on binary trees in Sections 4.2 and 4.3, we show how the latency of polar decoding can be improved by removing redundant steps on the decoding algorithm, without introducing performance loss, in Sections 4.4 and 4.5. Introducing another type of decoding in Section 4.6, namely Successive Cancellation List (SCL) decoding, the performance of polar codes was improved. By introducing CRC concatenated polar codes in Section 4.7, comparable performance to state-of-the-art codes was observed. In Section 4.8 optimizations in the list decoding, similar to those for SC decoding, are presented. The simulation results are presented in Section 4.9 and an analysis on the impact of code construction settings on the error performance was made. Finally, Section 4.10 presents the conclusions.

4.2 HARDWARE FRIENDLY DECODING

On Chapter 3 we showed how we operate the bit likelihood ratios using the f and g functions defined as

$$f(l_a, l_b) = \frac{l_a \cdot l_b + 1}{l_a + l_b} \tag{4.1}$$

$$g(l_a, l_b, u) = l_b \cdot (l_a)^{(1-2u)} \tag{4.2}$$

in order to perform polar decoding. Since this values involve likelihoods, they can get very close to zero or very large. It is then advantageous to use Log-Likelihood Ratio (LLR) instead. Adapting these expressions to work with log-likelihoods we get

$$f(L_a, L_b) = \log\left(\frac{e^{L_a+L_b}+1}{e^{L_a}+e^{L_b}}\right), \tag{4.3}$$

$$g(L_a, L_b, u) = L_b + (-1)^u L_a. \tag{4.4}$$

However, we are not completely free of large values appearing at the exponential, or small values appearing on the quotient in f. In order to overcome this, and also to perform these computations in a more hardware friendly way, we use the following approximation for f [22]

$$f(L_a, L_b) \approx \operatorname{sgn}(L_a)\operatorname{sgn}(L_b)\min(|L_a|, |L_b|). \tag{4.5}$$

It then becomes clear that g nodes can increase the certainty of the bit decision by summing values with equal sign. On the other hand, f nodes never increase the certainty since the result absolute value is always the minimum from its inputs. Using this approximation causes of course performance loss.

4.3 DECODING TREE

From now on, lets use the convention that the bit indexing will start at zero, for convenience. On Chapter 3 we showed that, in order to decode the i-th bit of a polar code with $N = 2^n$, one uses the binary representation $b_{n-1}...b_0$ of i, where we start by combining some channel likelihoods using f if $b_{n-1} = 0$ or using g if $b_{n-1} = 1$. We can see that the decoding structure is recursive, because if we consider $k = 0b_{n-2}...b_0$ and $k + 2^{n-1} = 1b_{n-2}...b_0$, we see that the remaining decoding steps will be the same. Figure 4.1 makes this more precise.

In order to decode $u_0^{N/2-1}$ we start by applying $f(l_i, l_{i+N/2})$ for $0 \leq i \leq N/2 - 1$. The rest of the decoding can be thought as decoding a polar code with length $N/2$ and inputs $f(l_i, l_{i+N/2})$. After obtaining the estimates $\hat{u}_0^{N/2-1}$, we encode these values using $\mathbf{F}_{N/2}$ to obtain $\hat{v}_0^{N/2-1}$, which we use to compute $g(l_i, l_{i+N/2}, \hat{v}_i)$ for $0 \leq i \leq N/2 - 1$. Again, the rest of the decoding to obtain $\hat{u}_{N/2}^{N-1}$ can be thought as another polar decoder with length $N/2$ fed by $g(l_i, l_{i+N/2}, \hat{v}_i)$. At this point, the bit estimates $\hat{u}_0^{N-1}$ are all available, and if non-systematic encoding is used, one can stop the decoding here. However, there are two scenarios on which one must perform another step. Firstly, for systematic codes, the information bits are in $\hat{x}_0^{N-1}$. Also, if the decoding of u_0^{N-1} is part of a larger polar code, then one also may need to compute $\hat{x}_0^{N-1}$ and feed these values to larger code. On both these scenarios, we continue the algorithm by encoding $\hat{u}_{N/2}^{N-1}$ using $\mathbf{F}_{N/2}$ to obtain $\hat{v}_{N/2}^{N-1}$, and then get $\hat{x}_i = \hat{v}_i \oplus \hat{v}_{i+N/2}$ for $0 \leq i \leq N/2 - 1$ and $\hat{x}_i = \hat{v}_i$ for $N/2 \leq i \leq N - 1$.

This process can be thought of traversing a binary tree, constructed from the structure in Figure 4.2. Each node represents a polar decoder with size 2^s, where s is the level of the node. At a level $s = k$ the following sequence is followed:

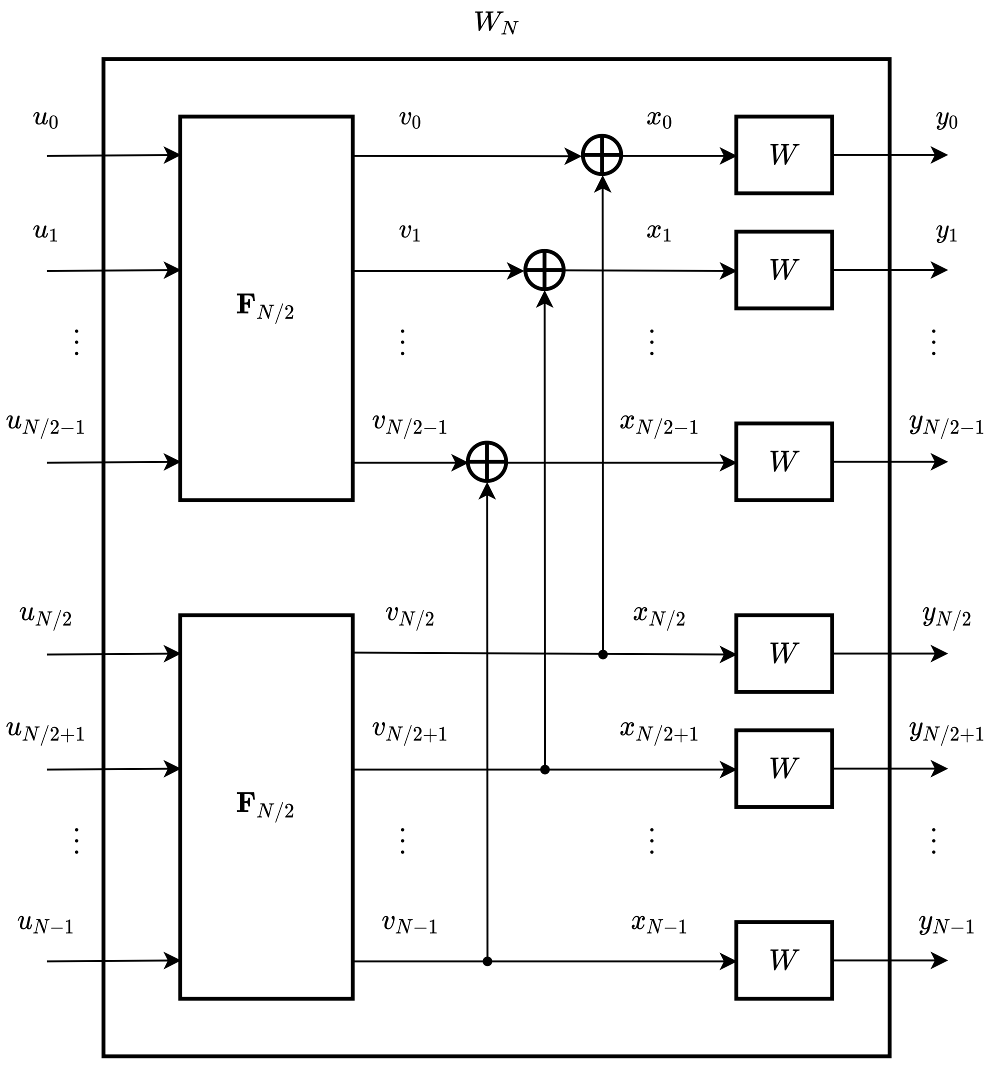


Figure 4.1 – An equivalent representation of the combined channel using two $\mathbf{F}_{N/2}$ encoders.

1. the node ν at $s = k$ receives the LLR vector $\boldsymbol{\alpha} = \{\alpha_0, ..., \alpha_{2^k-1}\}$ from its parent node;
2. using $\boldsymbol{\alpha}$, the node ν computes $\boldsymbol{\alpha}^l = \{\alpha_0^l, ..., \alpha_{2^{k-1}-1}^l\}$ and passes these values to the left child decoder;
3. the left child decoder goes through all the decoding operations and provides ν with the node bit vector $\boldsymbol{\beta}^l = \{\beta_0^l, ..., \beta_{2^{k-1}-1}^l\}$;
4. the node ν uses $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}^l$ to compute $\boldsymbol{\alpha}^r = \{\alpha_0^r, ..., \alpha_{2^{k-1}-1}^r\}$ and passes these values to the right child decoder;
5. the right child decoder goes through all the decoding operations and provides ν with the node bit vector $\boldsymbol{\beta}^r = \{\beta_0^r, ..., \beta_{2^{k-1}-1}^r\}$;
6. the node ν finishes its decoding operations by computing $\boldsymbol{\beta}$ and providing these values to its parent node.

Given the LLR vector $\boldsymbol{\alpha} = \{\alpha_0, ..., \alpha_{2^k-1}\}$ at level $s = k$, the left likelihoods are computed using

$$\alpha_i^l = f(\alpha_i, \alpha_{i+2^{k-1}}), \tag{4.6}$$

where f can be the exact version at Eq. (4.3) or the hardware friendly version at Eq. (4.5). Next, after $\boldsymbol{\beta}^l = \{\beta_0^l, ..., \beta_{2^{k-1}-1}^l\}$ is available, the right likelihoods are computed using

$$\alpha_i^r = g(\alpha_i, \alpha_{i+2^{k-1}}, \beta_i^l), \tag{4.7}$$

where g is the function at Eq. (4.4). After $\boldsymbol{\beta}^r = \{\beta_0^r, ..., \beta_{2^{k-1}-1}^r\}$ is available, $\boldsymbol{\beta} = \{\beta_0, ..., \beta_{2^k-1}\}$ is computed using

$$\beta_i = \begin{cases} \beta_i^l \oplus \beta_i^r, & \text{if } 0 \leq i \leq 2^{k-1} - 1 \\ \beta_i^l & \text{otherwise.} \end{cases} \tag{4.8}$$

The recursion stops at leaf nodes where $s = 0$. Leaves receive just α_0 from the parent

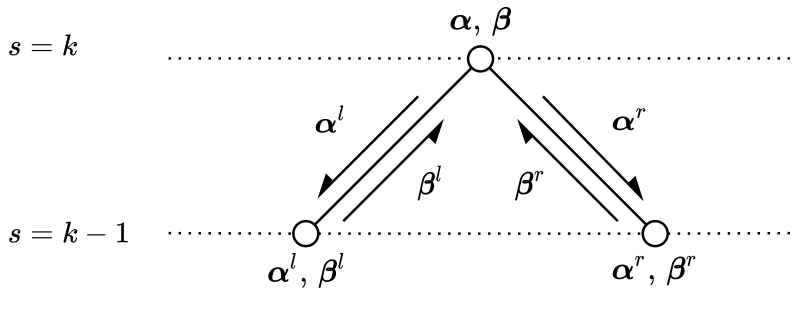

$\boldsymbol{\alpha}$, $\boldsymbol{\beta}$
$s = k$
$\boldsymbol{\alpha}^l$
$\boldsymbol{\alpha}^r$
$\boldsymbol{\beta}^l$
$\boldsymbol{\beta}^r$
$s = k - 1$
$\boldsymbol{\alpha}^l$, $\boldsymbol{\beta}^l$
$\boldsymbol{\alpha}^r$, $\boldsymbol{\beta}^r$

Figure 4.2 – Representation of the recursive polar decoding process.

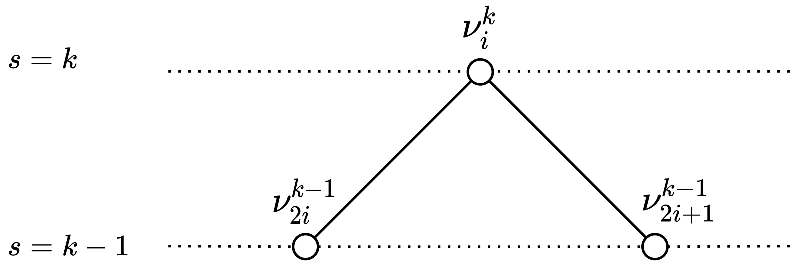


Figure 4.3 – Polar decoding binary tree node notation.

node, and no left and right calculation is needed. The leaf β_0 is obtained by

$$\beta_0 = \begin{cases} 0, & \text{if the leaf index is frozen or } \alpha_0 > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4.9}$$

Figure 4.3 shows a convenient node notation that uniquely identifies each node on the tree using two numbers, the level $s = k$ and the level index i. Left node indexes are obtained by updating the index i of the parent node to $2i$ and right node indexes are obtained by updating i to $2i + 1$. Figure 4.4 shows an example of such notation on a tree with depth 3.

According to this proposed notation, the likelihoods at node ν_i^k can be identified by $\boldsymbol{\alpha}_i^k = \{\alpha_{i,0}^k, ..., \alpha_{i,2^k-1}^k\}$, and the node bits can be identified by $\boldsymbol{\beta}_i^k = \{\beta_{i,0}^k, ..., \beta_{i,2^k-1}^k\}$. The leaves indexes of the node ν_i^k are identified by the set

$$\begin{aligned} V_i^k &= \{i \in \{0, ..., 2^k - 1\} \mid i \text{ is leaf of } \nu_i^k\} && (4.10) \\ &= \{i \cdot 2^k, i \cdot 2^k + 1, ..., (i + 1) \cdot 2^k - 1\}. && (4.11) \end{aligned}$$

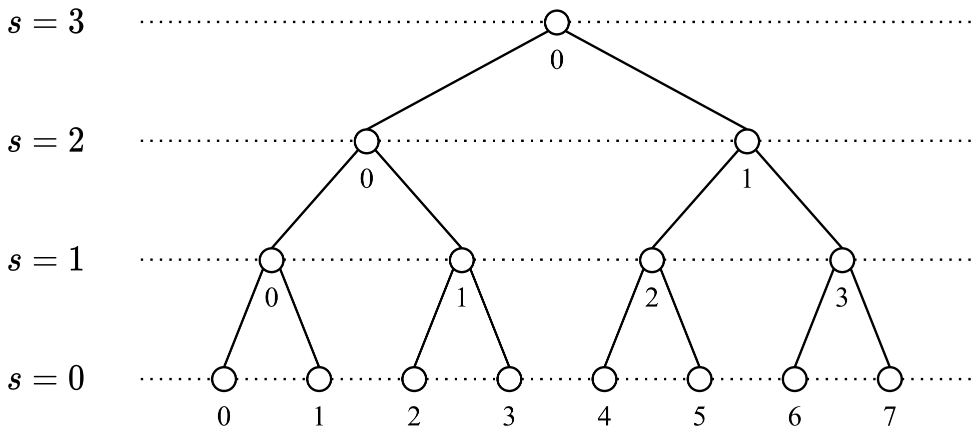


Figure 4.4 – Example of the proposed node notation for a tree with depth 3.

The vector of the leaf bits is identified as

$$\boldsymbol{\beta}(V_i^k) = \{\beta_{i,0}^0 \mid i \in V_i^k\}. \tag{4.12}$$

Given this, we have that $\hat{u}_i = \beta_i^0$ and that the relationship between the node bits and the leaf bits is $\boldsymbol{\beta}_i^k = \boldsymbol{\beta}(V_i^k)\mathbf{F}_{2^k}$, meaning that the leaf bits are encoded to obtain the node bits, and vice-versa. For a polar code with $N = 2^n$, we have that $\hat{u}_0^{N-1} = \boldsymbol{\beta}(V_0^n)$ and $\hat{x}_0^{N-1} = \boldsymbol{\beta}_0^n$. Also, we have that the bit likelihoods $L_N^{(i)} = \alpha_i^0$.

4.3.1 Linear Node Indexing

The node indexing presented so far uses two numbers to identify a node. This notation is useful for the theoretical approach so far. However, since the level $s = k$ has more nodes than the level $s = k - 1$, this type of indexing may lead to non-uniform data containers. Also, two numbers have to be passed to the children nodes for their complete identification. We introduce next an indexing scheme that was particularly useful when implementing the binary tree decoding algorithm in software. For hardware applications, the convenience of using this scheme must be assessed.

For a binary tree with depth $s = k$, there is a 1 to 1 mapping between the numbers $\{0, ..., 2^{k+1} - 2\}$ and the nodes on the tree, which allows a linear 1-Dimensional indexing of the tree. Figure 4.5 shows how the left and right children node indexes are obtained, and Figure 4.6 shows the example of a binary tree with depth 3 and 1-D indexing. One disadvantage of this node indexing is that it is absolute: the tree starts at the node with index $i = 0$ and can't be expanded upwards. However, this indexing scheme was proven to be useful on our software implementation of the tree decoding algorithm.

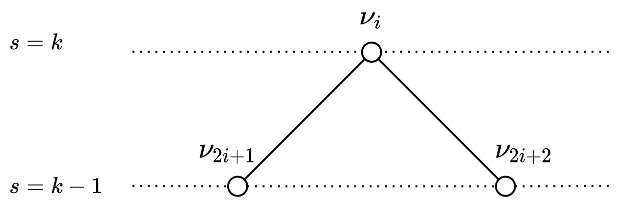


Figure 4.5 – Linearly indexed binary tree index construction.

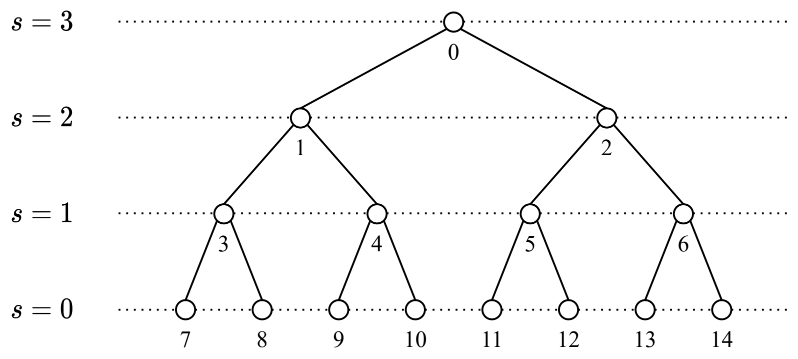


Figure 4.6 – Example of linear node indexing for a binary tree with depth 3.

4.4 SIMPLIFIED SUCCESSIVE CANCELLATION DECODING

As the tree algorithm makes explicit, the recursive structure of polar codes allows us to break the decoding process into the decoding of smaller codes. One can take advantage of this and shorten the decoding time by decoding specific nodes with reduced complexity. We then introduced the Simplified Successive Cancellation (SSC) [12] decoding.

Figure 4.7 shows a binary tree with $N = 8$, where the frozen leaves are depicted as black circles and the information leaves are depicted as white circles. We then begin by defining two special types of nodes, namely Rate-0 nodes and Rate-1 nodes.

4.4.1 Rate-0 Nodes

Rate-0 nodes have only frozen leaves. On the example in Figure 4.7, at node ν_0^2 there is no need to compute $\boldsymbol{\alpha}^l$, since its left child has only frozen leaves and we know that $\boldsymbol{\beta}^l$ will be all zero. We then proceed directly to computing $\boldsymbol{\alpha}^r$. By doing this, we are actually pruning out the tree.

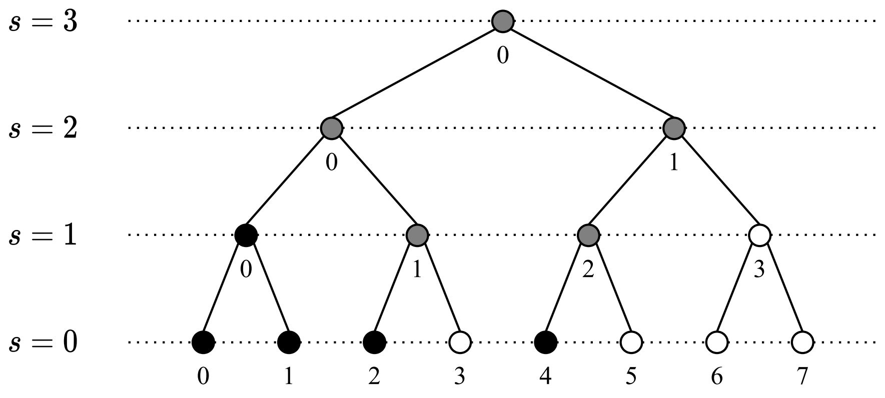


Figure 4.7 – Binary decoding tree for $N = 8$, where Rate-0 nodes are represented with black circles, Rate-1 nodes are represented with white circles and mixed nodes are represented with grey circles.

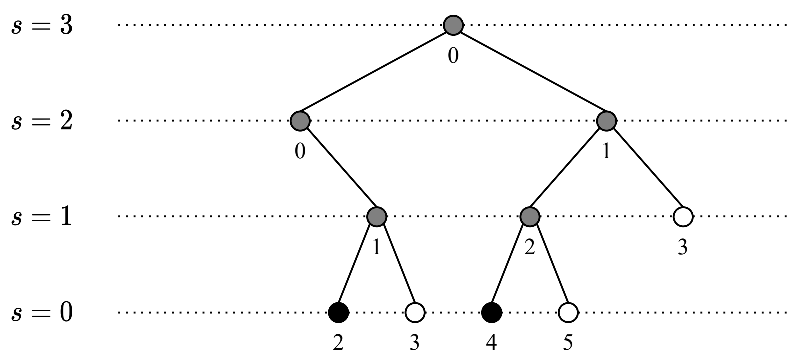


Figure 4.8 – Pruned decoding tree, considering that Rate-0 nodes do not have to be visited and that Rate-1 nodes are decoded directly.

4.4.2 Rate-1 Nodes

Rate-1 nodes have only information leaves, and it turns out that there is no need to traverse this node to obtain $\boldsymbol{\beta}_i^k$. We compute these values directly from the node likelihoods $\boldsymbol{\alpha}_i^k$ [12]

$$\beta_{i,j}^k = \begin{cases} 0, & \text{if } \alpha_{i,j}^k > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4.13}$$

Note that one does not obtain the estimates of the leaf bits $\hat{u}_j$, $j \in V_i^k$, directly. Instead, one obtains the node bits $\boldsymbol{\beta}_i^k$. Figure 4.8 shows how the decoding tree can be pruned considering these approximations.

After decoding a Rate-1 node ν_i^k, one has two options: encode the node bits $\boldsymbol{\beta}_i^k$ using $\mathbf{F}_{2^k}$ to obtain the leaf bits or continue the decoding until $\boldsymbol{\beta}_0^n$ is reached at the top of the tree. In the case systematic encoding is used, it is advantageous to obtain the top node bits $\boldsymbol{\beta}_0^n$, since they contain the information bits. In addition, in this case, it is unnecessary to obtain the leaf bits. If non-systematic encoding is used, one can obtain $\boldsymbol{\beta}_0^n$ and encode these bits to obtain $\hat{u}_0^{N-1}$ all at once, or compute the leaf bits for every Rate-1 node, but halt the decoding without reaching the top node at the end. We do not have any claim of which method is best for non-systematic encoding.

Figure 4.9 shows a tree example with $N = 1024$ and $K = 512$, using the DEGA construction method. We can see that we have Rate-0 and Rate-1 nodes as large as 64 bits, at $s = 6$. It was shown at [23] that this type of decoding has sublinear latency $O(N^{1-1/\mu})$, where μ is a parameter called the scaling exponent of the transmission channel. Note that this form of decoding has the exact same performance as the traditional SC decoding [12].

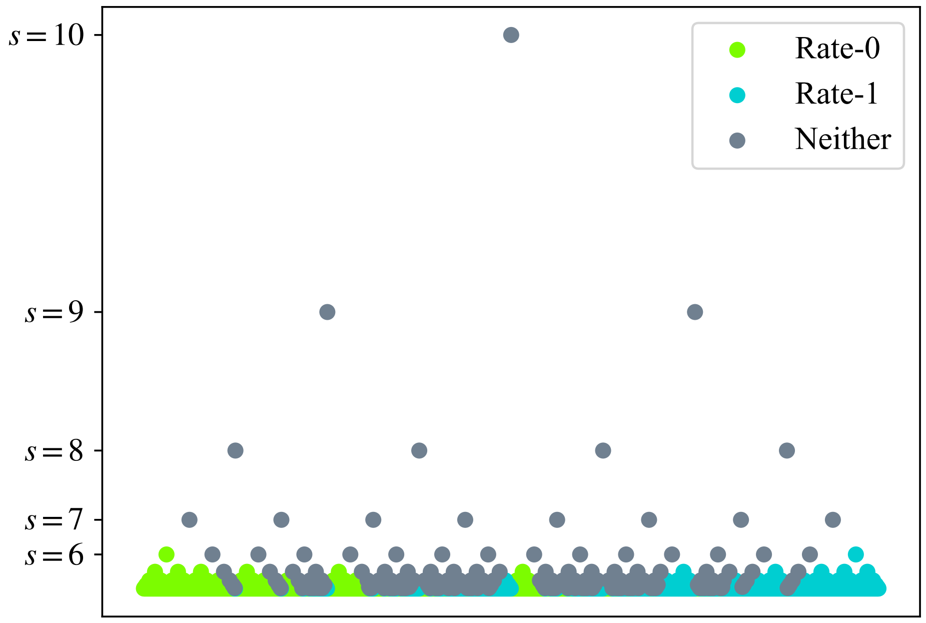


Figure 4.9 – Example of decoding tree for SSC decoding with $N = 1024$, $K = 512$ and constructed using the Density Evolution - Gaussian Approximation (DEGA) method with design $E_s/N_0 = 0$ dB.

4.5 FAST SIMPLIFIED SUCCESSIVE CANCELLATION

Two other node types stand out for providing simple direct decoding without any performance loss, reducing further the decoding latency. The technique described next is the Fast Simplified Successive Cancellation (Fast-SSC) [13] decoding.

4.5.1 REP Nodes

We define REPetition (REP) nodes as nodes in which only the rightmost leaf is an information bit, that is, $\boldsymbol{\beta}(V_i^k) = \{0, ..., 0, \beta_\nu\}$. For this case, the likelihoods of the frozen leaves do not matter and since all the bits before the decoding of β_ν are zero, at every computation of the right likelihoods using g, the likelihoods will be summed, in a manner that the value of β_ν can be decided by

$$\beta_\nu = \begin{cases} 0, & \text{if } \sum_{j=0}^{2^k-1} \alpha_{i,j}^k > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4.14}$$

The node bits are obtained by encoding the leaf bits using $\mathbf{F}_{2^k}$ and, since the last row of this matrix is always 1, we have that $\boldsymbol{\beta}_i^k = \{\beta_\nu, ..., \beta_\nu\}$. This makes clear that rate $1/N$ polar codes are very large repetition codes.

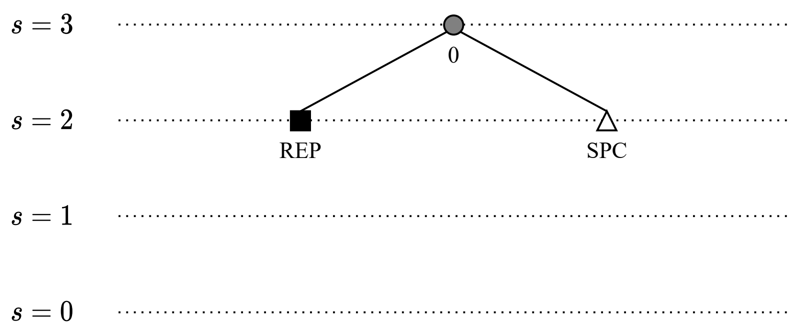


Figure 4.10 – Pruned decoding tree, considering REP and SPC nodes.

4.5.2 SPC Nodes

The REP nodes counterparts are Single Parity Check (SPC) nodes, in which the only frozen leaf is the leftmost one, that is, $\boldsymbol{\beta}(V_i^k) = \{0, \beta_\nu^0 ..., \beta_\nu^{2^k-2}\}$. The node betas $\boldsymbol{\beta}_i^k$ are computed in two steps. At first, hard decision is made just like in Rate-1 nodes

$$\beta'_j = \begin{cases} 0, & \text{if } \alpha_{i,j}^k > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4.15}$$

Then, we compute the parity of this hard decision with

$$\gamma = \bigoplus_{j=0}^{2^k-1} \beta'_j, \tag{4.16}$$

along with the index of the least reliable likelihood

$$m = \arg\min_j |\alpha_{i,j}^k|. \tag{4.17}$$

Finally, we flip the least reliable bit if the parity is odd

$$\beta_{i,j}^k = \begin{cases} \beta'_j, & \text{if } j \neq m \\ \beta'_j \oplus \gamma, & \text{if } j = m. \end{cases} \tag{4.18}$$

One can see that in Figure 4.7 the node ν_0^2 is a REP node and that the node ν_1^2 is a SPC node. Figure 4.10 shows the tree from Figure 4.7 but pruned considering the REP and SPC nodes. Figure 4.11 shows a Fast-SSC tree example with $N = 1024$ and $K = 512$, using the DEGA construction method. We can see that we have REP and SPC nodes as large as 128 bits, at $s = 7$.

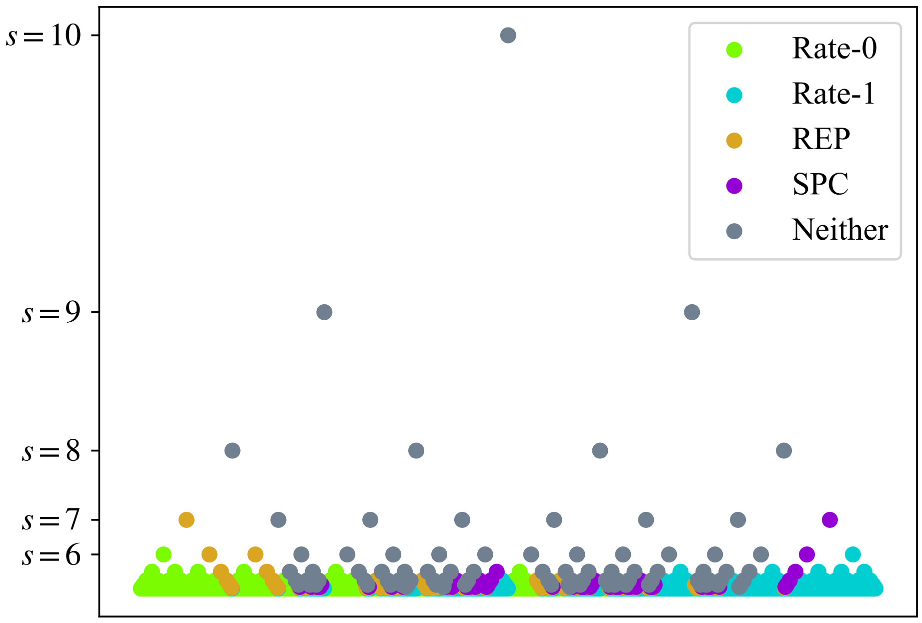


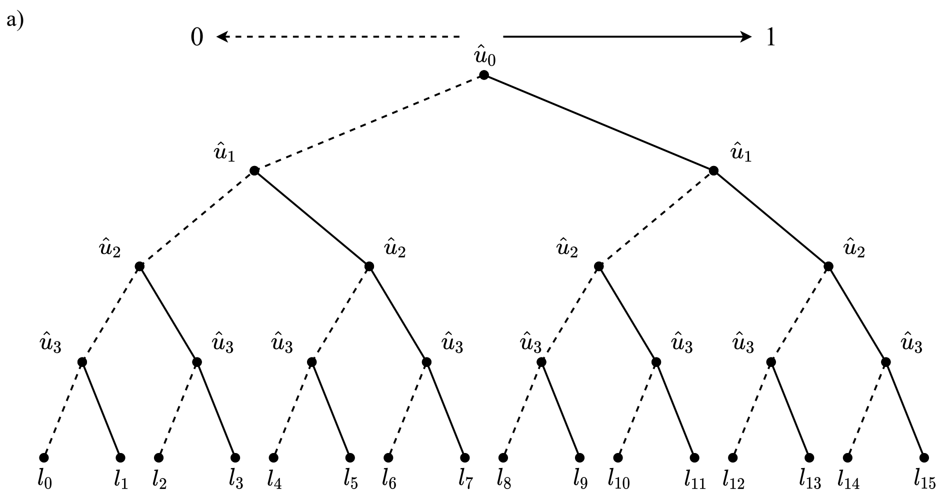
Figure 4.11 – Example of decoding tree for Fast-SSC decoding with $N = 1024$, $K = 512$ and constructed using the DEGA method with design $E_s/N_0 = 0$ dB.

4.6 SUCCESSIVE CANCELLATION LIST DECODING

What if, instead of only choosing $\hat{u}_i$ according to its likelihood, we considered both 0 and 1 options to decode the next bits? This is the basic idea of list decoding. However, if we consider every bit possibility, this problem scales up exponentially and corresponds to the Maximum Likelihood (ML) decoding. To keep the number of paths limited, the list decoding is done for a certain list size, in a manner that if the number of paths exceeds the list size some paths are pruned. In order to decide which paths to keep, a path metric is used, and the paths with the smallest metrics are allowed to continue decoding. Figure 4.12 a) shows all 16 possible paths for the decoding of $\hat{u}_0^3$, and Figure 4.12 b) shows an example of how the tree is pruned to keep the list size at $L = 2$.

If both possibilities for $\hat{u}_i$ are considered then at some point a g node will compute different likelihoods for each path. In fact, each path corresponds to a different SC decoder. However, for a list decoder with size L and code size N, efficient hardware implementations [5, 14, 15] allow space complexity $O(L \cdot N)$ and time complexity $O(L \cdot N \log N)$.

In order to perform list decoding we will use the LLR based [14] scheme instead of the original scheme proposed by [5]. The decoder is initiated with the path metric $\text{PM}_l^{(-1)} = 0$ and begins traversing the decoding tree to compute $L_N^{(0)}$. Once this value is available, each


a)
0
1
$\hat{u}_0$
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
l_0 l_1 l_2 l_3 l_4 l_5 l_6 l_7 l_8 l_9 l_{10} l_{11} l_{12} l_{13} l_{14} l_{15}
b)
$\hat{u}_0$
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
l_0 l_1

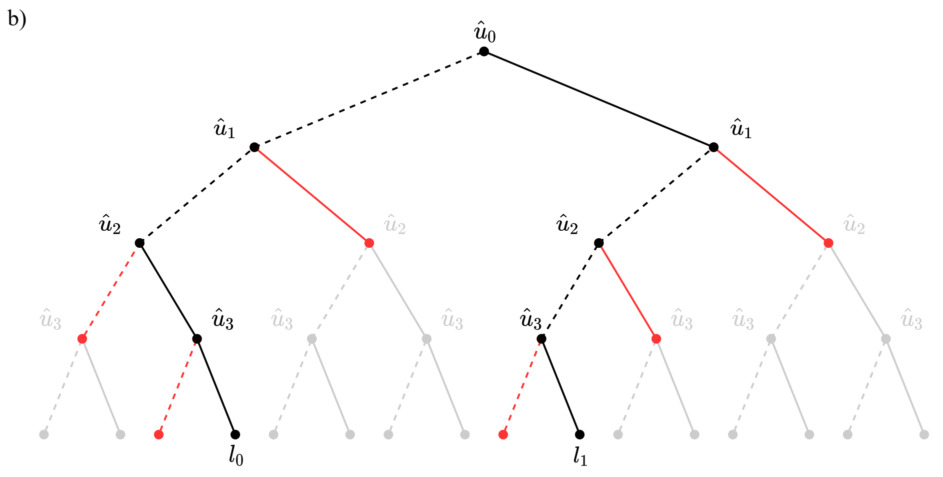


Figure 4.12 – a) Tree depicting the decoding paths l_j as the bits $\hat{u}_i$ are decoded. b) Example of pruned path tree with list size $L = 2$, where pruned paths appear in red.

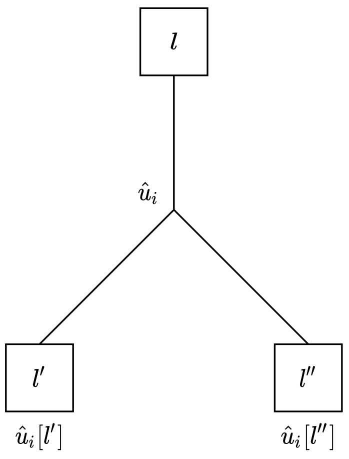


Figure 4.13 – Upon reaching bit i, a path l splits into two paths l' and l'', each having a different value for $\hat{u}_i$.

possibility for $\hat{u}_0$ originates a new path with different metrics, as depicted in Figure 4.13. The rest of the decoding is then carried out separately for each hypothesis of $\hat{u}_0$, which will lead to $L_N^{(1)}[l']$ being computed differently as $L_N^{(1)}[l'']$.

As illustrated in Figure 4.13, when bit i is reached the path l is split into two other paths with different choices for $\hat{u}_i$, and the path metrics are updated using [14]

$$\text{PM}_{l'}^{(i)} = \phi(\text{PM}_l^{(i-1)}, L_N^{(i)}[l], \hat{u}_i[l']), \tag{4.19}$$

where $\text{PM}_l^{(i-1)}$ is the path metric at stage $i-1$ and path l, $L_N^{(i)}[l]$ is the log-likelihood ratio of the bit u_i at path l and $\hat{u}_i[l']$ is the choice of $\hat{u}_i$ at path l'. The function ϕ is defined as [14]

$$\phi(\mu, \lambda, u) = \mu + \log(1 + e^{-(1-2u)\lambda}). \tag{4.20}$$

If the number of paths reaches the list size L at bit k, then at bit $k+1$ we will have $2L$ path metric values after computing Eq. (4.19), and only the paths with smallest metrics will be allowed to continue. Figure 4.12 b) shows this process, where the red branches had their metrics computed, but were discarded.

Instead of using ϕ, we can use the approximated version [14]

$$\tilde{\phi}(\mu, \lambda, u) = \begin{cases} \mu, & \text{if } u = \frac{1}{2}[1 - \text{sgn}(\lambda)] \\ \mu + |\lambda| & \text{otherwise,} \end{cases} \tag{4.21}$$

where $\text{sgn}(x)$ is the sign function that returns 1 if $x \geq 0$ and -1 if $x < 0$. When applied to the path metrics, the above expression translates to

$$\text{PM}_{l'}^{(i)} = \begin{cases} \text{PM}_l^{(i-1)}, & \text{if } \hat{u}_i[l'] = \frac{1}{2}[1 - \text{sgn}(L_N^{(i)}[l])] \\ \text{PM}_l^{(i-1)} + |L_N^{(i)}[l]|, & \text{otherwise.} \end{cases} \tag{4.22}$$

Note that the path metric remains the same when $\hat{u}_i[l'] = 0$ and $L_N^{(i)}[l] \geq 0$, or when $\hat{u}_i[l'] = 1$ and $L_N^{(i)}[l] < 0$. In other words, when the bit estimate $\hat{u}_i[l']$ agrees with the sign of $L_N^{(i)}[l]$, the path metric stays the same. When the bit estimate $\hat{u}_i[l']$ disagrees with the sign of $L_N^{(i)}[l]$, a penalty of $|L_N^{(i)}[l]|$ is added to the metric. This is somehow intuitive, since paths on which we choose the opposite of what $L_N^{(i)}[l]$ tells us to choose receive a penalty. The greater the value of $|L_N^{(i)}[l]|$ is, the larger the penalty is for going against the certainty of $L_N^{(i)}[l]$.

If it weren't for the frozen bits, we could always choose $\hat{u}_i[l']$ so that the metric doesn't receive any penalty. At frozen bits, a path that had the smallest metric can receive a large penalty for having $L_N^{(i)}[l'] < 0$.

Figure 4.14 shows the SCL decoding performance for various list sizes and block length

$N = 1024$, where the code was constructed using DEGA optimized for $E_b/N_0 = 2$ dB. We observe that $L = 32$ shows much better performance than $L = 1$, which corresponds to SC decoding. However, the performance difference between $L = 16$ and $L = 32$ is very narrow. In fact, the authors in [5] showed a way of computing the ML bound of polar codes. They essentially simulated the behavior of $L \to \infty$ without actually simulating infinite lists. It was found that one could approach ML decoding performance closely for list sizes small as $L = 32$.

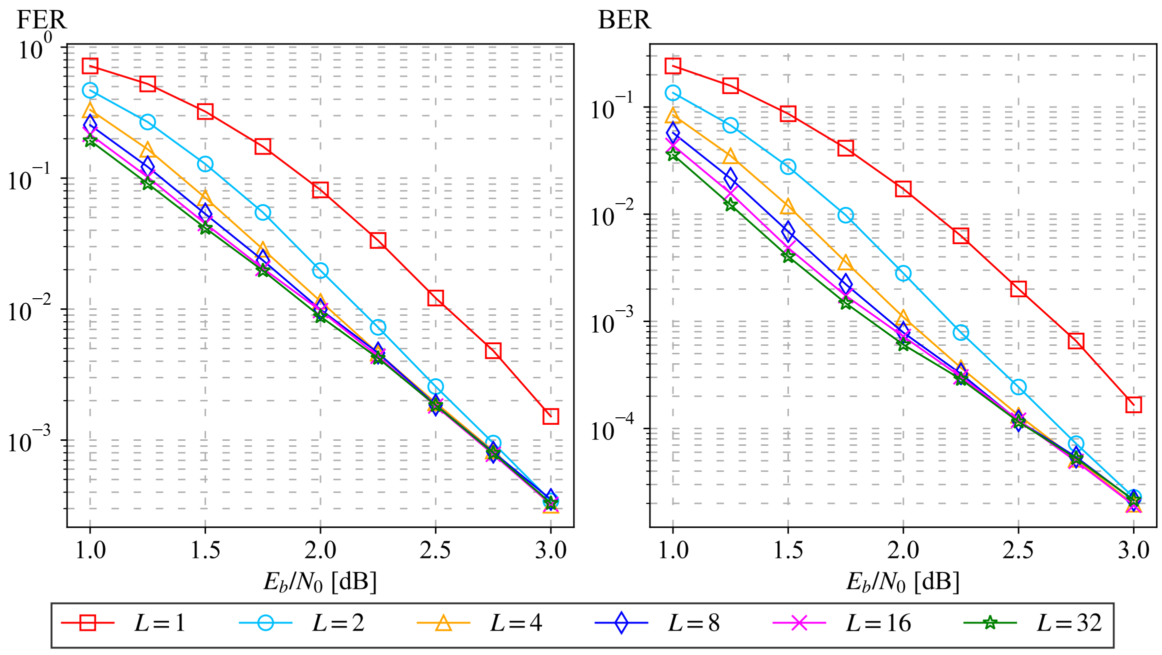


Figure 4.14 – Successive cancellation list decoding performance for $N = 1024$, $L = 1$ (SC) to $L = 32$, using a fixed DEGA construction method optimized for $E_b/N_0 = 2$ dB.

4.6.1 Comparison with Turbo Codes

Recall that in Section 3.7.3 we compared the SC decoding with the DVB-RCS2 Turbo codes. Figure 4.15 shows the comparison between the same Turbo codes and polar codes decoded by a list decoder with $L = 32$, both having a block size of $N = 1024$. It is clear that polar codes, even with list decoding, show poor performance when compared to this state-of-the-art technique. This is in principle a huge setback to polar codes, since [5] showed that the ML bound lies not far from the $L = 32$ performance. This means that polar codes alone can only achieve smaller block error probabilities by increasing N, which is guaranteed by the polar coding theorems presented in Chapter 3. If polar codes alone are bounded by this poor performance, we must look for another solutions involving modifications on polar codes. On the next section we explore one such modification.

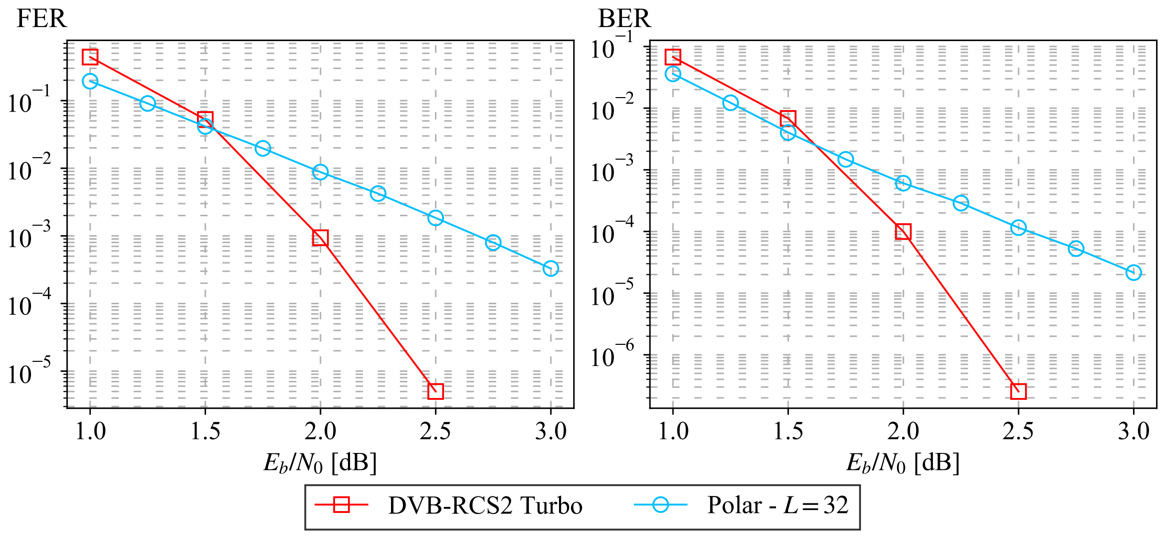


Figure 4.15 – Comparison between the DVB-RCS2 Turbo codes and polar codes with successive cancellation list decoding, $L = 32$ and constructed using fixed DEGA construction optimized for $E_b/N_0 = 2$ dB, for block length $N = 1024$ and code rate $R = 1/2$.

4.7 CRC CONCATENATED POLAR CODES

The authors of [5] noted that even though the path with the smallest metric resulted on a wrong path, the final list contained the transmitted code word. We need to implement a way of detecting the right path among the final paths without depending on the path metrics. This turns out to be easily accomplished by inserting a redundancy check on polar codes.

By proposing a concatenation of polar codes with Cyclic Redundancy Check (CRC), the polar coding performance can be improved significantly. CRC is a type of parity check defined by generator polynomials $g(x)$ [7]. Given any bit sequence $(b_0, ..., b_m)$, a polynomial $g(x)$ with degree T will generate a T-bit redundancy word. This type of redundancy check is used to detect errors on the sequence, since a vast list of error patterns are guaranteed to result in different CRC words [7].

Firstly, for a polar code with length N and K unfrozen bits, one uses $K - T$ information bits to generate a T-bit CRC word. Then, CRC concatenated polar codes are obtained by setting the T most reliable bits to the T-bit CRC, and the remaining $K - T$ positions to information bits. Note that the code rate

$$R = \frac{K - T}{N} \tag{4.23}$$

is affected by this operation.

The decoding is done identically to the list decoding described in the previous section, differing only when the final paths list is obtained:

a) if at least one final path has a correct CRC, then we choose the path with the smallest metric and matching CRC;

b) if no final path has a correct CRC, then we choose the path with the smallest metric.

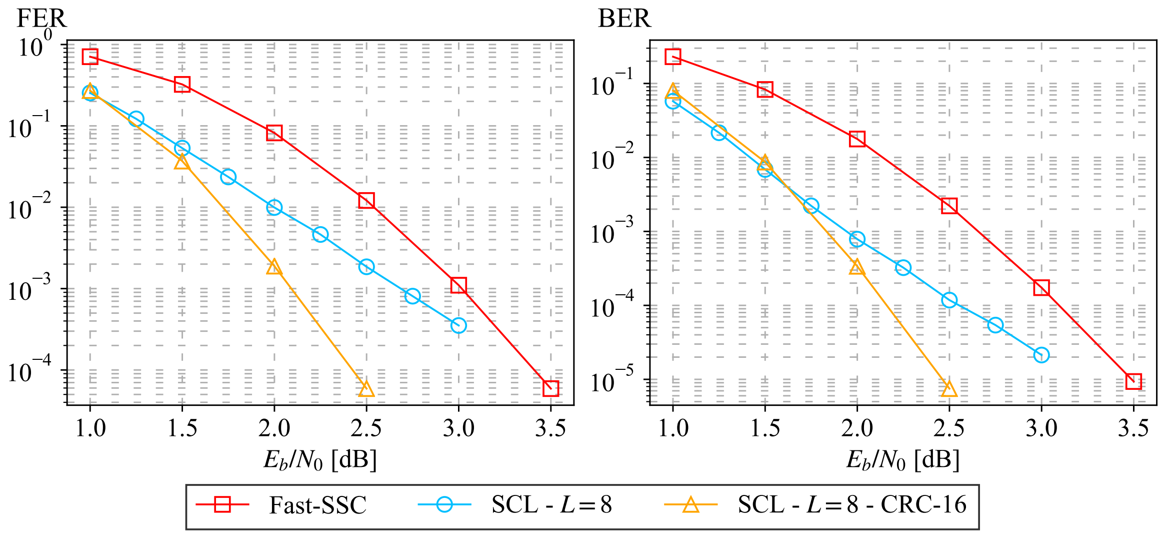


Figure 4.16 – A comparison between the Fast-SSC, SCL and SCL-CRC decoding schemes, with $L = 8$, for polar codes with block size $N = 1024$ and $K = 512$. The CRC used is CRC-16.

Figure 4.16 shows the comparison between polar codes decoded with Fast-SSC, SCL and SCL-CRC for $L = 8$, using block size $N = 1024$ and $K = 512$. The CRC used has the polynomial $1 + x^2 + x^{15} + x^{16}$. Recall that Fast-SSC presents the same performance as the original SC decoding. We see a dramatic performance gain when using CRC concatenated polar codes. At BER $\approx 10^{-5}$, the encoding gain is near 1 dB, meaning that SCL-CRC was able to achieve BER of 10^{-5} 1 dB before Fast-SSC. The reader may notice that for low E_b/N_0 the SCL scheme performs slightly better than SCL-CRC. This may sound contradictory, since a block successfully decoded by SCL would also have a matching CRC, and the performance of SCL-CRC is expected to be better or equal than that of SCL. However, recall that introducing the CRC causes a reduction on the effective code rate, which in turn causes a shift on the SCL-CRC curve, making it perform slightly worse than SCL in terms of information bit-energy-to-noise ratio E_b/N_0. Complete simulation results with other list sizes will be presented later, since they were obtained using a simplified list decoding algorithm.

4.8 SIMPLIFIED SUCCESSIVE CANCELLATION LIST DECODING

The previous algorithm describes a procedure to perform list decoding that requires one to obtain the leaf likelihoods $L_N^{(i)}$. However, there are strategies to prune the decoding tree in a manner similar to the techniques presented in Sections 4.4 and 4.5. The authors in [15] proposed the Simplified Successive Cancellation List (SSCL) decoder capable of decoding Rate-0, Rate-1 and REP nodes directly without any performance loss. In the same work, an approximation is made to decode SPC nodes directly, which introduces no performance loss for $L \leq 2$ and introduces negligible performance loss for $L > 2$. Such decoding is called SSCL-SPC. Further optimizations were carried out [24] resulting in Fast-SSCL and Fast-SSCL-SPC. This section approaches SSCL and SSCL-SPC only.

4.8.1 Rate-0 Nodes

Consider that the decoding reaches a Rate-0 node ν_i^k. We know that the leftmost leaf of this node has index $t_l = i \cdot 2^k$ and that its rightmost leaf has index $t_r = (i+1)2^k - 1$. We also know that the last path split occurred in bit $t_l - 1$. Lets denote the list of all surviving paths after the splitting at bit $t_l - 1$ as $\mathcal{L}^{t_l - 1}$. At Rate-0 nodes, we can update all paths $l \in \mathcal{L}^{t_l - 1}$ using [15]

$$\mathrm{PM}_l^{(t_r)} = \mathrm{PM}_l^{(t_l - 1)} + \frac{1}{2} \sum_{j=0}^{2^k - 1} |\alpha_{i,j}^k[l]| - \alpha_{i,j}^k[l]. \tag{4.24}$$

Note that $\frac{1}{2}(|x| - x)$ equals to 0 for $x > 0$ and to $|x|$ for $x \leq 0$, in a manner that each term of the sum in Eq. (4.24) is either 0 or $|\alpha_{i,j}^k[l]|$. We can then see that the penalty each path $l \in \mathcal{L}^{t_l - 1}$ receives is the sum of $|\alpha_{i,j}^k[l]|$ for all j in which $\alpha_{i,j}^k[l]$ is negative. This is intuitive since any negative node likelihood $\alpha_{i,j}^k[l]$ goes against the fact that all node bits in $\boldsymbol{\beta}_i^k$ should be zero. Also, note that there is no path splitting in Rate-0 nodes, the metrics are just updated.

4.8.2 REP Nodes

At a REP node ν_i^k we showed earlier that $\boldsymbol{\beta}_i^k = \{\beta_\nu, ..., \beta_\nu\}$. Using the auxiliary variable $\eta_\nu[l'] = 1 - 2\beta_\nu[l']$ and also using the same notation used for Rate-0 nodes, all paths $l \in \mathcal{L}^{t_l - 1}$ are updated using [15]

$$\mathrm{PM}_{l'}^{(t_r)} = \mathrm{PM}_l^{(t_l - 1)} + \frac{1}{2} \sum_{j=0}^{2^k - 1} |\alpha_{i,j}^k[l]| - \eta_\nu[l']\alpha_{i,j}^k[l]. \tag{4.25}$$

If at path l' we have that $\beta_\nu[l'] = 0$, then $\eta_\nu[l'] = 1$ and the path metrics are updated just like in Rate-0 nodes. This makes sense, since in this case $\boldsymbol{\beta}_i^k[l'] = \{0, ..., 0\}$. If $\beta_\nu[l'] = 1$, then $\eta_\nu[l'] = -1$ and, since $\frac{1}{2}(|x| + x)$ equals to $|x|$ for $x > 0$ and to 0 for $x \leq 0$, we have that the path penalty is the sum of $|\alpha_{i,j}^k[l]|$ for all j in which $\alpha_{i,j}^k[l]$ is positive. This also makes sense, since in this case $\boldsymbol{\beta}_i^k[l'] = \{1, ..., 1\}$ and any positive likelihood $\alpha_{i,j}^k[l]$ goes against this hypothesis.

4.8.3 Rate-1 Nodes

If the decoding reaches a Rate-1 node ν_i^k with leftmost leaf index t_l and rightmost leaf index t_r, the path metrics are updated for $0 \leq j \leq 2^k - 1$ as [15]

$$\mathrm{PM}_{l'}^{(t_l+j)} = \begin{cases} \mathrm{PM}_l^{(t_l+j-1)}, & \text{if } \beta_{i,j}^k[l'] = \frac{1}{2}[1 - \operatorname{sgn}(\alpha_{i,j}^k[l])] \\ \mathrm{PM}_l^{(t_l+j-1)} + |\alpha_{i,j}^k[l]|, & \text{otherwise.} \end{cases} \tag{4.26}$$

For Rate-1 nodes the path metrics are updated just like they would be at leaf nodes, but instead of using the leaf likelihoods and leaf bits one uses node likelihoods and node bits.

4.8.4 SPC Nodes

The following procedure implements the approximated SPC list decoding found in [15]. For an SPC node ν_i^k with leftmost leaf t_l, rightmost leaf t_r and paths $\mathcal{L}^{t_l-1}$, we first begin by finding the indexes of the least reliable likelihood for every path $l \in \mathcal{L}^{t_l-1}$

$$m[l] = \arg\min_j |\alpha_{i,j}^k[l]|. \tag{4.27}$$

The decoding in SPC nodes will follow a different order from the natural one, starting at $m[l]$ instead of 0. To accomplish this, we use a permutation $\pi_l[j]$ that maps

$$\pi_l[j] = \begin{cases} m[l], & \text{if } j = 0 \\ j - 1, & \text{if } j \leq m[l] \\ j, & \text{if } m[l] < j \leq 2^k - 1. \end{cases} \tag{4.28}$$

An example with $k = 3$ and $m[l] = 5$ yields $\pi_l = (5, 0, 1, 2, 3, 4, 6, 7)$. The next step is to compute the hard decision for each path $l \in \mathcal{L}^{t_l-1}$

$$\beta_j'[l] = \begin{cases} 0, & \text{if } \alpha_{i,j}^k[l] > 0 \\ 1, & \text{otherwise,} \end{cases} \tag{4.29}$$

and use these values to compute the parity of each path with

$$\gamma[l] = \bigoplus_{j=0}^{2^k-1} \beta_j'[l]. \tag{4.30}$$

The least reliable bit is decoded first, without any path splitting, by updating the metric with

$$\mathrm{PM}_l^{(t_l)} = \begin{cases} \mathrm{PM}_l^{(t_l-1)} + |\alpha_{i,m}^k[l]|, & \text{if } \gamma[l] = 1 \\ \mathrm{PM}_l^{(t_l-1)}, & \text{otherwise.} \end{cases} \tag{4.31}$$

For the remaining bits, with $1 \leq j \leq 2^k - 1$, we split the paths according to

$$\mathrm{PM}_{l'}^{(t_l+j)} = \begin{cases} \mathrm{PM}_l^{(t_l+j-1)}, & \text{if } \beta_{i,\pi_l[j]}^k[l'] = \\ & \frac{1}{2}[1 - \mathrm{sgn}(\alpha_{i,\pi_l[j]}^k[l])] \\ \mathrm{PM}_l^{(t_l+j-1)} + |\alpha_{i,\pi_l[j]}^k[l]| + (1-2\gamma[l])|\alpha_{i,m}^k[l]|, & \text{otherwise.} \end{cases} \tag{4.32}$$

Notice that we use $\pi_l[j]$ to index the node likelihoods and bits instead of j. This ensures that the path metrics are indexed in the natural order, from t_l to t_r, but also ensures that at the same time the node bits are decoded starting with $m[l]$, followed by the natural order with a gap where $m[l]$ should be.

After all bits are estimated, we set the less reliable bit of each surviving path $l \in \mathcal{L}^{t_r}$ to preserve the even-parity constraint

$$\beta_{i,m[l]}^k[l] = \bigoplus_{\substack{j=0 \\ j \neq m[l]}}^{2^k-1} \beta_{i,j}^k[l]. \tag{4.33}$$

This approximation to decode SPC nodes is exact for $L \leq 2$ and introduces negligible performance loss for $L > 2$ [15]. The permutation choice π_l is not the only one possible. In fact, any permutation π_l such that $\pi_l[0] = m[l]$ is valid. This fact is explored in [24] to decode SPC and Rate-1 nodes with reduced latency. The basic intuition for this fact will be given in Subsection 4.8.5.

4.8.5 General Case

Based on the decoding result for Rate-1 nodes in [15], we show next how the path metrics for any node can theoretically be computed.

For convenience, lets use the notation $\eta_{i,j}^k = (1 - 2\beta_{i,j}^k)$. At a Rate-1 node ν_i^k with leftmost leaf index t_l, rightmost leaf index t_r and paths $l \in \mathcal{L}^{t_l-1}$ we have that [15]

$$\mathrm{PM}_{l'}^{(t_r)} = \mathrm{PM}_l^{(t_l-1)} + \frac{1}{2}\sum_{j=0}^{2^k-1} |\alpha_{i,j}^k[l]| - \eta_{i,j}^k[l']\alpha_{i,j}^k[l], \tag{4.34}$$

where l' is a path obtained after the last bit decision $j = 2^k - 1$ was made. We see that, when

$\text{sgn}(\alpha_{i,j}^k[l]) \neq \eta_{i,j}^k[l']$, the corresponding term in the sum is different from zero, meaning that at step j the splitting that eventually led to path l' had $\beta_{i,j}^k$ chosen differently from what the sign of $\alpha_{i,j}^k[l]$ told us to choose, and a penalty was applied.

The catch here is that the only thing that restricts Eq. (4.34) to be applied for every node ν_i^k is that at Rate-1 nodes one has the ability to freely choose $\beta_{i,j}^k$ for every j, since there are no frozen bits. If we have a Rate-0 node, the frozen bits constrain $\boldsymbol{\beta}_i^k$ to $\{0, ..., 0\}$, and Eq. (4.34) becomes equivalent to Eq. (4.24). If we have a REP node, the frozen bits constrain $\boldsymbol{\beta}_i^k$ to $\{0, ..., 0\}$ or $\{1, ..., 1\}$, and Eq. (4.34) becomes equivalent to Eq. (4.25).

Then, Eq. (4.34) is valid for any node ν_i^k, as long as the leaf bit vector $\boldsymbol{\beta}(V_i^k)$ has the bits in frozen positions set to zero, which means we can't choose $\boldsymbol{\beta}_i^k$ freely; these bits can only be code words of a polar code with $N = 2^k$.

For a Rate-1 node, imagine that at step t the paths in $\mathcal{L}^t$ remained, and that a particular path metric $\text{PM}_{l'}^{(t)}$ is left out. We can guarantee that for the remaining of this Rate-1 node decoding this path would remain left out. This is because the other L paths $\mathcal{L}^t$ that survived after step t can be decoded so as to receive no penalty, since the remaining bits $\beta_{i,j}^k$ can be decoded according to $\alpha_{i,j}^k$ without any frozen bit restriction. By doing this, if the metric $\text{PM}_{l'}^{(t)}$ or any of its children were to be in the final list, then the L paths in $\mathcal{L}^t$ with smaller metrics should also be, a contradiction.

For other nodes, however, sequentially splitting the paths and choosing the smallest metrics doesn't guarantee that the final paths have the smallest metrics on that node, since a restriction on $\beta_{i,j}^k$ may force a large penalty on a surviving path.

A great result obtained from Eq. (4.34) is explored in [24], which states that if the path splitting is made on the sorted metrics, from smaller absolute values to larger, then one doesn't necessarily need to split paths $N = 2^k$ times. At the present work, we don't apply this extra sorting step and use only the algorithms SSCL and SSCL-SPC in [15].

4.9 SIMULATION RESULTS

4.9.1 List Size Effect

We begin by showing the effect of the list size L on the code performance. Figure 4.17 shows the code performance as we increase the list size, with $N = 1024$, $K = 512$, CRC-16 with polynomial $1 + x^2 + x^{15} + x^{16}$, decoded using SSCL-SPC and constructed using DEGA optimized for $E_b/N_0 = 2$ dB. We chose the DEGA construction algorithm for reasons that will be clarified later in this section. These results show a dramatic performance increase by introducing the CRC.

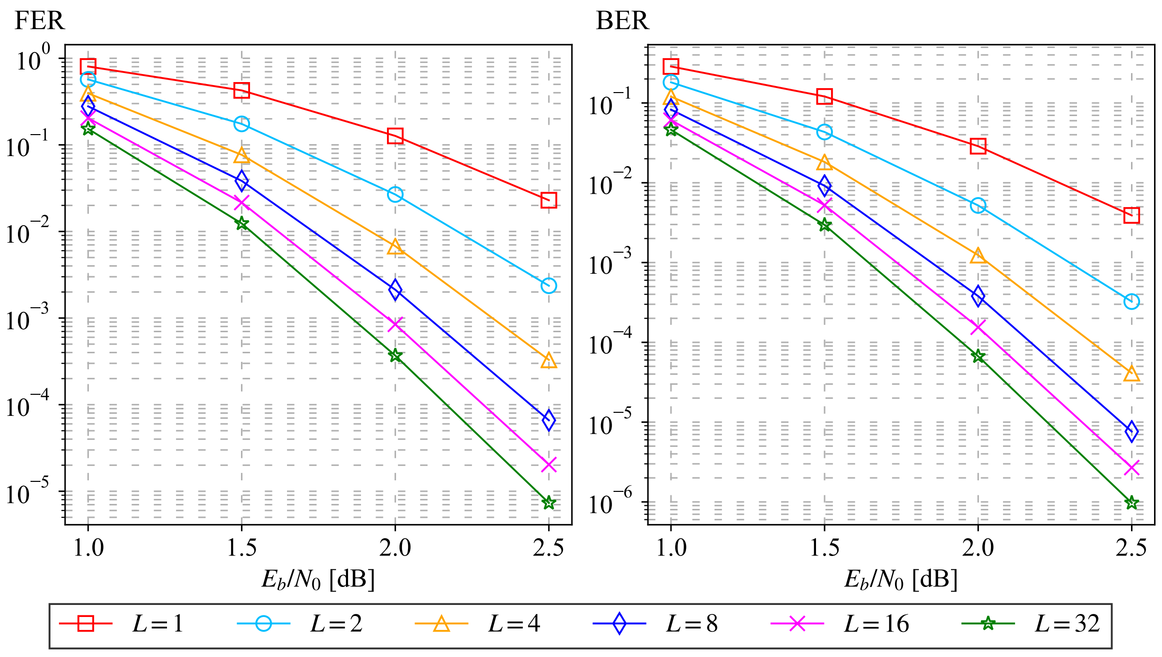


Figure 4.17 – Polar coding performance for different list sizes, with $N = 1024$, $K = 512$ and CRC-16, decoded using SSCL-SPC and constructed using DEGA optimized for $E_b/N_0 = 2$ dB.

4.9.2 SSCL-SPC Performance Loss

Lets now justify the claim [15] that SSCL-SPC introduces negligible performance loss to SCL decoding. Figures 4.18-4.19 show the performance comparison between SSCL and SSCL-SPC for $N = 1024$ and rates $1/2$ and $3/4$. For both list sizes and code rates, the performance loss caused by SSCL-SPC is in fact negligible. We put this data into another separate plot in Figure 4.20, on which we can see how the code rate affects the decoding performance. It can be seen that using $L = 8$ instead of $L = 4$ presents similar gains at both rates of $1/2$ and $3/4$.

4.9.3 Comparison with Turbo Codes

The previous sections and chapters had many comparisons between polar codes and the state-of-the-art Turbo codes. It was shown earlier that even with list decoding the performance was far worse than that of the DVB-RCS2 Turbo codes. We repeat this comparison with CRC concatenated polar codes. Figure 4.21 shows a comparison between the DVB-RCS2 Turbo code and a polar code with SSCL-SPC decoding, CRC-16 and constructed using DEGA optimized for the channel E_b/N_0. Both codes have $N = 1024$ and $K = 512$, but the polar codes have a slightly smaller code rate due to the CRC bits. We see that after improving the polar coding performance with the concatenation of CRC polar codes were able to outperform Turbo codes, at least for lower E_b/N_0 values. We also see that the polar

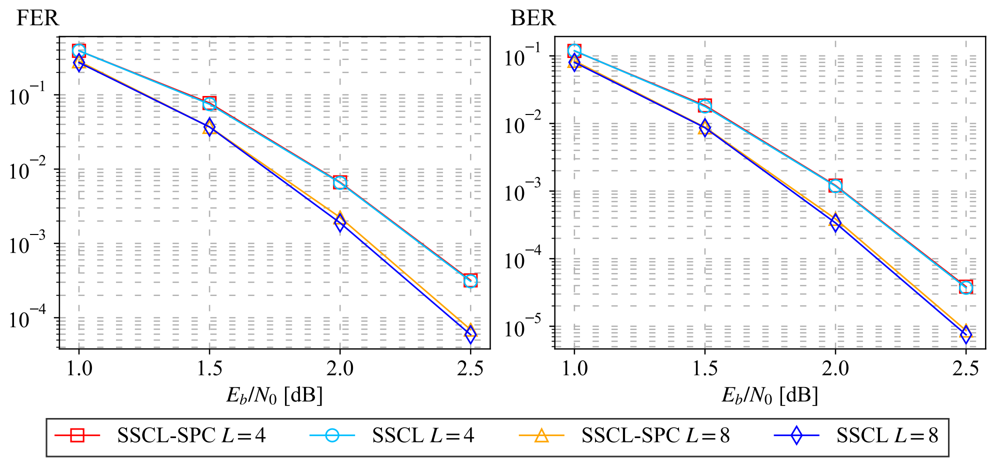


Figure 4.18 – Polar coding performance comparison between SSCL and SSCL-SPC, with $N = 1024$, $K = 512$ and CRC-16, for list sizes $L = 4$ and $L = 8$ and constructed using DEGA optimized for $E_b/N_0 = 2$ dB.

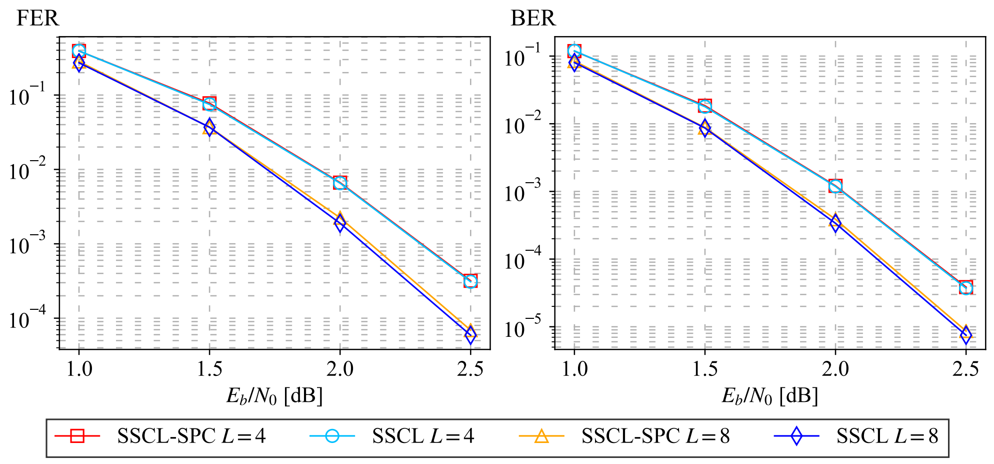


Figure 4.19 – Polar coding performance comparison between SSCL and SSCL-SPC, with $N = 1024$, $K = 768$ ($R = 3/4$) and CRC-16, for list sizes $L = 4$ and $L = 8$ and constructed using DEGA optimized for $E_b/N_0 = 2$ dB.

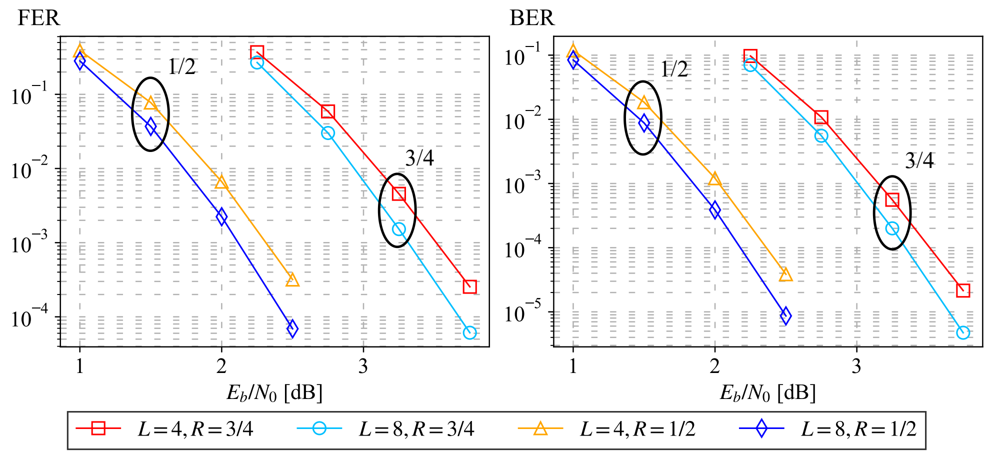


Figure 4.20 – Polar coding performance for $R = 1/2$ and $R = 1/3$, with $N = 1024$ and CRC-16, for list sizes $L = 4$ and $L = 8$, decoded using SSCL-SPC and constructed using DEGA optimized for $E_b/N_0 = 2$ dB.

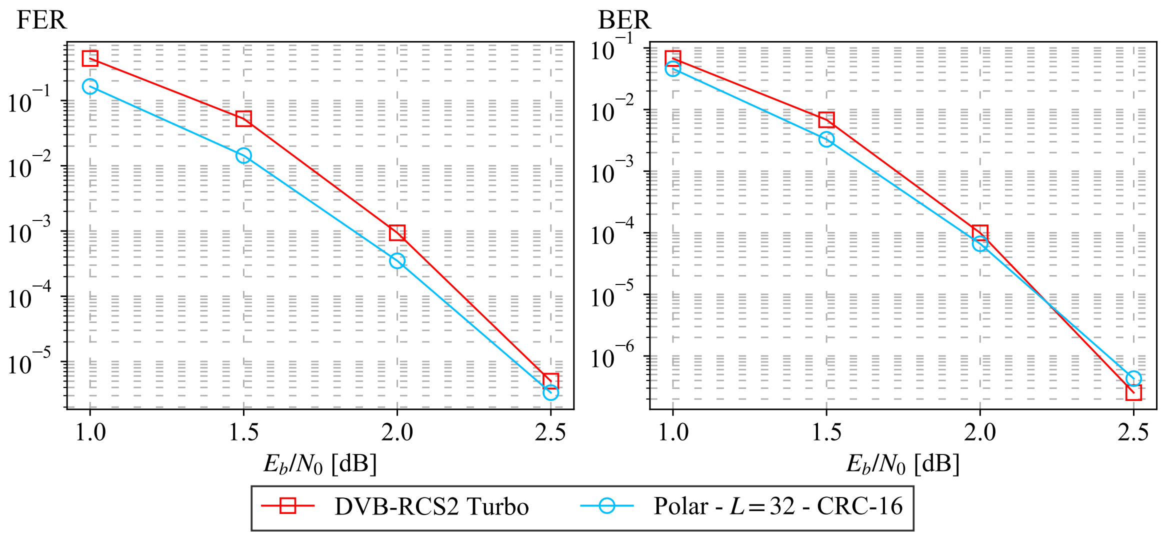


Figure 4.21 – Performance comparison between the DVB-RCS2 Turbo code and polar code with SSCL-SPC decoding, $L = 32$, CRC-16 and constructed using DEGA dynamic construction, both codes with $N = 1024$. The polar code has $K = 496$ information bits and the Turbo code has $K = 512$ information bits.

code curve falls slower than the curve of Turbo codes. We will see next that this may be caused by suboptimal code construction.

4.9.4 Code Construction

Now, we will analyze how the construction method affects code performance. We focus our analysis on the list size $L = 8$, since they can be simulated quicker than $L = 32$ and has a list size big enough to show excellent performance improvement. Remember that the construction methods presented in Section 3.6 were developed under the context of SC decoding, and that for ML decoding the code construction is universal, being valid for every channel [18]. Also, introducing CRC to the decoding modifies the definition of polar codes. These facts imply that it is unknown how optimal construction looks like for SCL-CRC decoding, and we illustrate this with the following results.

Figure 4.22 shows how the different construction methods perform for polar codes with $N = 1024$, $K = 512$ and $L = 8$, CRC-16, decoded using SSCL and optimized for $E_b/N_0 = 2$ dB. Remember that in Chapter 3 all four construction methods had similar performance for $N = 1024$ and SC decoding, with the Bhattacharyya method performing slightly worse. Surprisingly, for the SSCL-CRC decoding scheme the difference between the methods is considerable and the DEGA method performed better than the other methods for the entire E_b/N_0 range simulated. When using the SSCL-SPC decoding scheme, the results are almost the same, as it is seen in Figure 4.23. This motivates us to investigate how different the node classification is for these different methods.

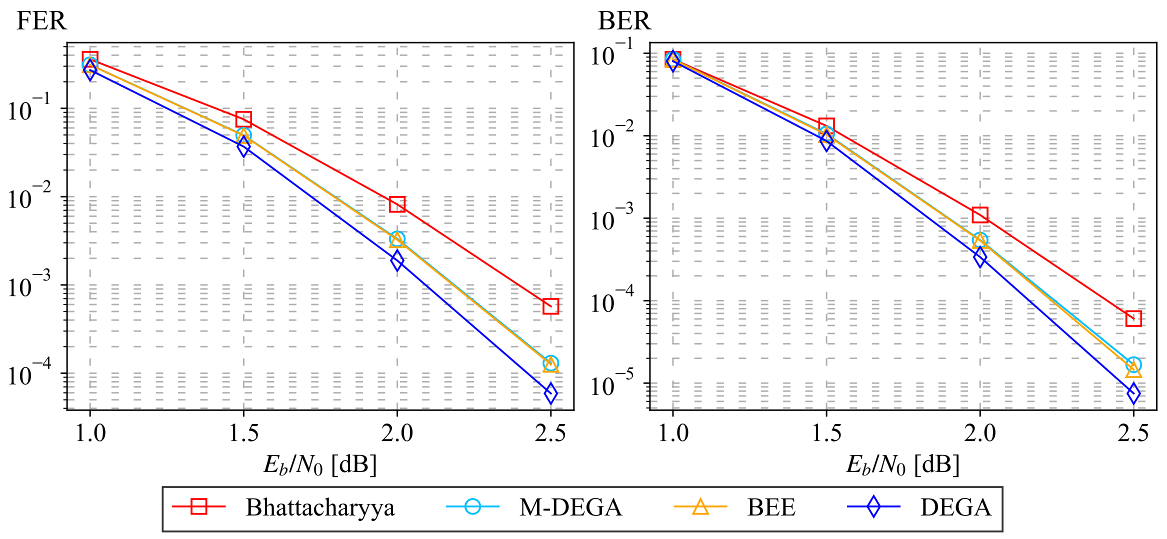


Figure 4.22 – A comparison between the different construction methods for polar codes with $N = 1024$, $K = 512$, $L = 8$, CRC-16, decoded using SSCL and optimized for $E_b/N_0 = 2$ dB.

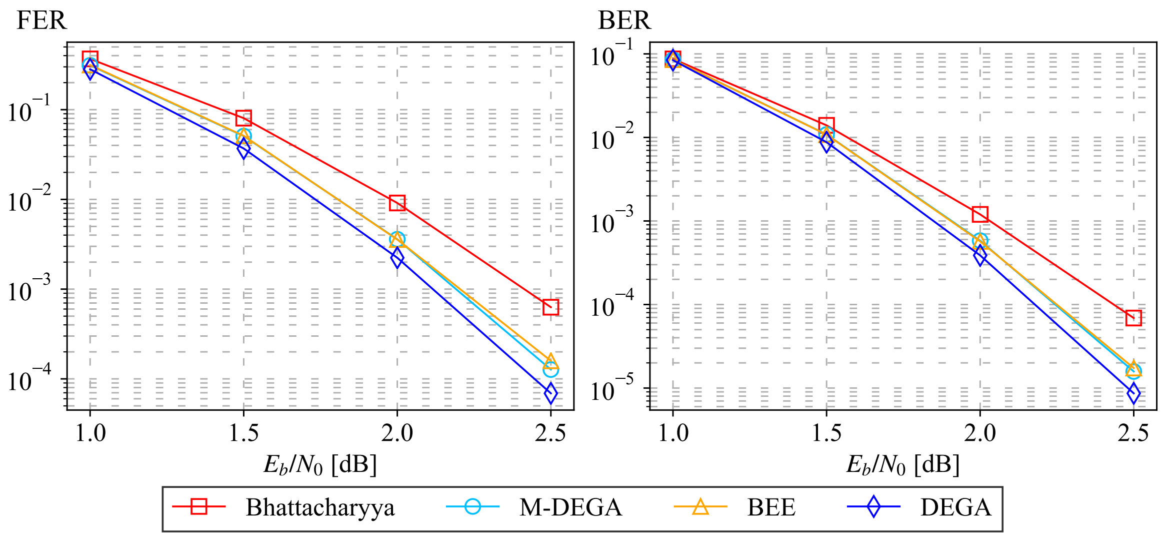


Figure 4.23 – A comparison between the different construction methods for polar codes with $N = 1024$, $K = 512$, $L = 8$, CRC-16, decoded using SSCL-SPC and optimized for $E_b/N_0 = 2$ dB.

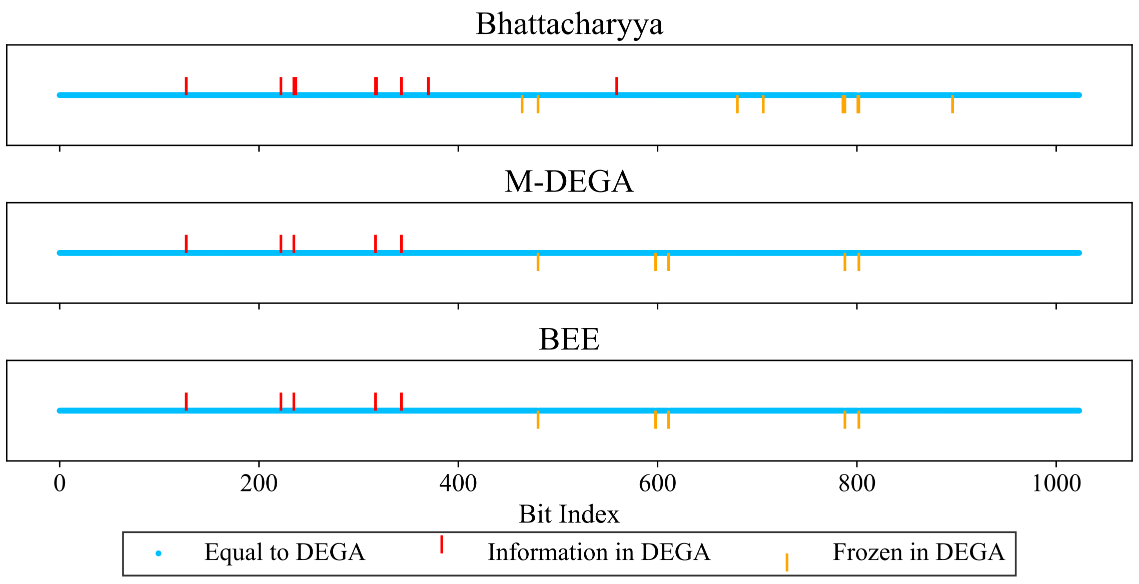


Figure 4.24 – Information and frozen bit differences comparing the shown methods to the DEGA approximation, with $N = 1024$, $K = 512$ and $E_b/N_0 = 2$ dB. A $496 = 512 - 16$ information length was considered to convert E_b/N_0 to E_s/N_0.

Figure 4.24 shows the node classification differences between the shown methods and the DEGA mehtod for $N = 1024$, $K = 512$ and $E_b/N_0 = 2$ dB. We see that the DEGA method differs only in 10 positions from the M-DEGA and BEE methods and in 16 positions for the Bhattacharyya method. Namely, 5 bit indexes that carry information bits in DEGA are switched to frozen in M-DEGA and BEE. We can see that these information positions are transferred to higher index positions. It is surprising that only 10 different positions in DEGA are responsible for the performance difference observed. This result shows how sensitive the list decoding performance is to the construction method. Further investigation must be carried out on the direction of finding an optimal construction method for list decoders.

Next, we analyse how sensitive to the design E_b/N_0 the list decoding performance is. Figure 4.25 shows the code performance for a fixed channel E_b/N_0 of 2 dB, while varying the design E_b/N_0, for $N = 1024$, $K = 512$, $L = 8$ and CRC-16, decoded using SSCL and constructed using the DEGA construction method. We see that the decoding performance is best not only when the design E_b/N_0 matches the channel E_b/N_0 at 2 dB; the design $E_b/N_0 = 0$ dB shows also a similar performance. Figure 4.26 shows the bit positions on which these two scenarios are different and we see that only 8 classifications are changed. From Figure 4.25 we can see an increase in BER and FER when the design E_b/N_0 changes from 0 to 0.5 dB. Inspired by this, we show Figure 4.27, which depicts the bit positions on which the scenarios 0.5 and 2 dB are different. Surprisingly enough, the number of switched positions on both scenarios 0 and 0.5 dB is also 8. In a future work, these results can be further investigated to show why changing only 8 positions in the $E_b/N_0 = 0.5$ dB scenario causes performance loss, while doing the same in the $E_b/N_0 = 0$ dB scenario doesn't.

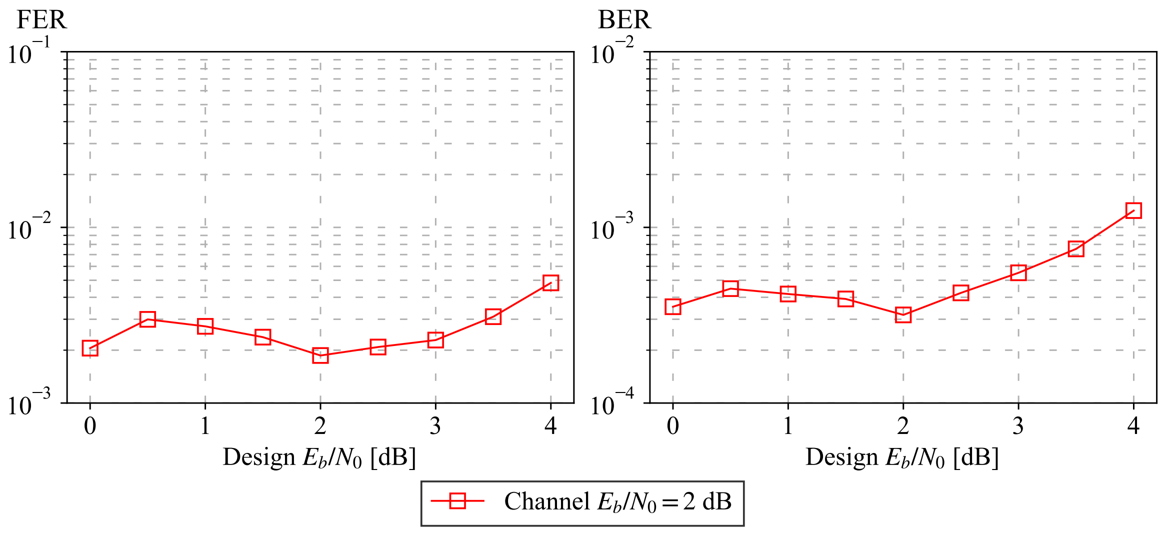


Figure 4.25 – Polar code performance for fixed channel $E_b/N_0 = 2$ dB while varying the design E_b/N_0, for $N = 1024$, $K = 512$, $L = 8$, CRC-16 and decoded using SSCL and constructed using the DEGA construction method.

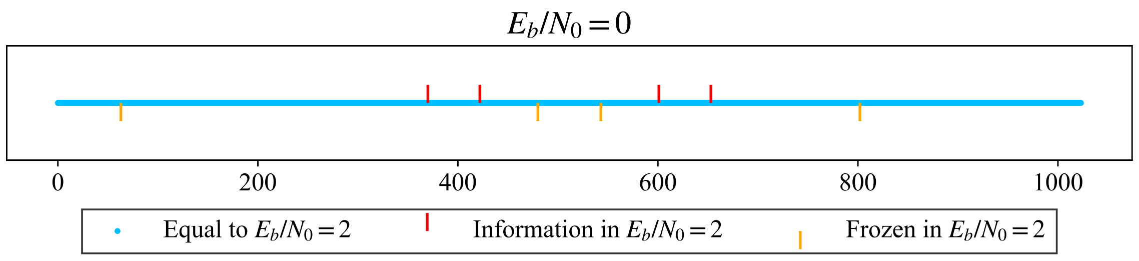


Figure 4.26 – Information and frozen bit differences comparing the DEGA construction method at 0 and 2 dB E_b/N_0 design, with $N = 1024$ and $K = 512$. A $496 = 512 - 16$ information length was considered to convert E_b/N_0 to E_s/N_0.

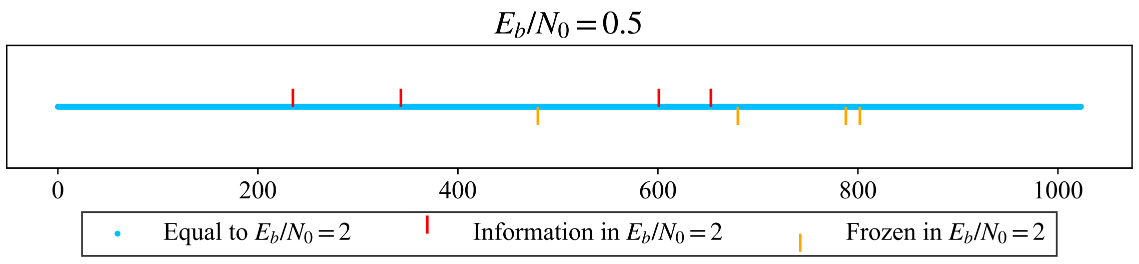


Figure 4.27 – Information and frozen bit differences comparing the DEGA construction method at 0.5 and 2 dB E_b/N_0 design, with $N = 1024$ and $K = 512$. A $496 = 512 - 16$ information length was considered to convert E_b/N_0 to E_s/N_0.

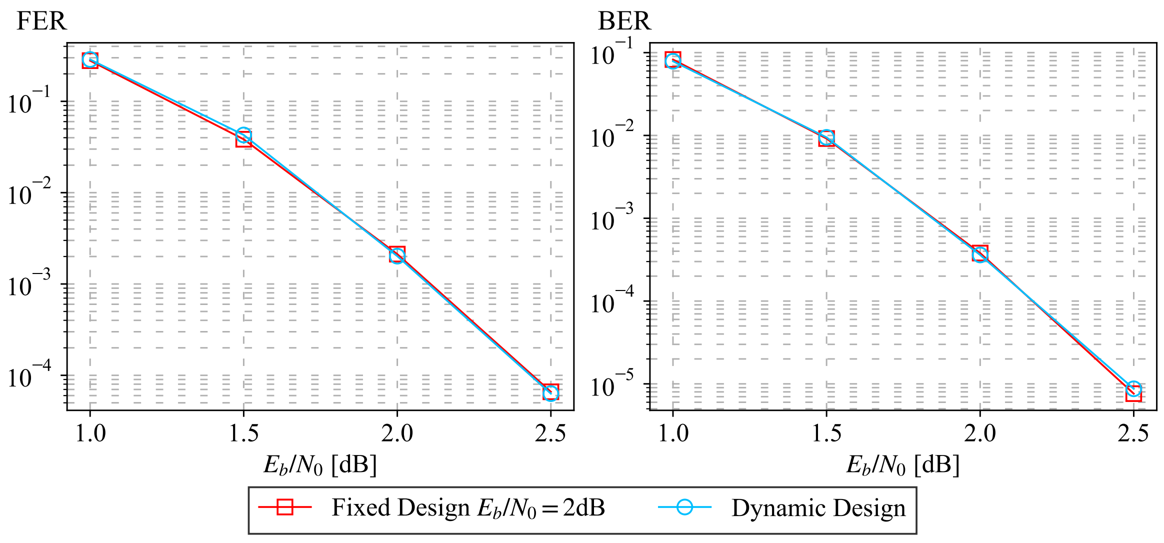


Figure 4.28 – Performance comparison between fixed design at $E_b/N_0 = 2$ dB and dynamic design to match the channel E_b/N_0, with $N = 1024$, $K = 512$, $L = 8$ and CRC-16, decoded using SSCL-SPC and constructed using DEGA.

On the same line of thinking, we removed the fixed design $E_b/N_0 = 2$ dB and simulated $L = 8$ and $L = 32$ by changing the design to match the channel E_b/N_0. Figures 4.28-4.29 show the simulation results, which show that for $L = 8$ there is no visible performance improvement obtained by changing the design E_b/N_0 at every simulated channel E_b/N_0. On the other hand, for $L = 32$, the dynamic design performs better at high E_b/N_0.

4.9.5 Systematic Encoding and QPSK Transmission

We now test how systematic encoding affects the SCL-CRC performance. Figure 4.30 shows the comparison between between systematic and non-systematic polar encoding for $N = 1024$ and $L = 8$, and we see a similar behavior to the systematic encoding under SC decoding: both encoding types have the same FER performance with better BER performance for the systematic codes.

Finally, we consider polar codes under QPSK transmission instead of BPSK. Polar codes were shown to achieve channel capacity for memoryless discrete channels. Apart from AWGN not being a discrete channel, transmission using QPSK is not memoryless. This is because in QPSK each symbol transmits two bits at a time, and both bits are bonded statistically. Considering these two facts alone, it is expected that polar codes have poorer performance under QPSK, which is confirmed in Figure 4.31. Another interesting behavior is that for QPSK the M-DEGA construction performs better than DEGA. A possible strategy to approximate QPSK with a memoryless channel is to introduce a bit interleaver before introducing the encoded bits to the constellation mapping. However, this isn't explored in this work.

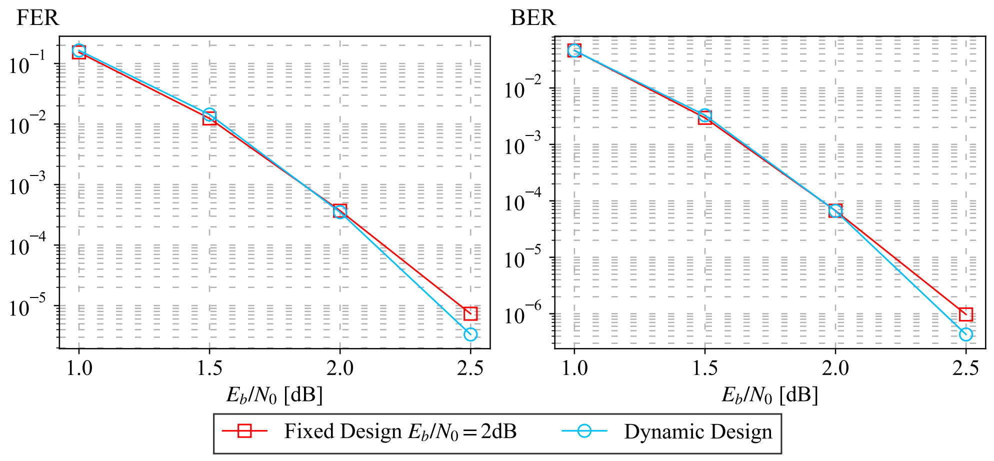


Figure 4.29 – Performance comparison between fixed design at $E_b/N_0 = 2$ dB and dynamic design to match the channel E_b/N_0, with $N = 1024$, $K = 512$, $L = 32$ and CRC-16, decoded using SSCL-SPC and constructed using DEGA.

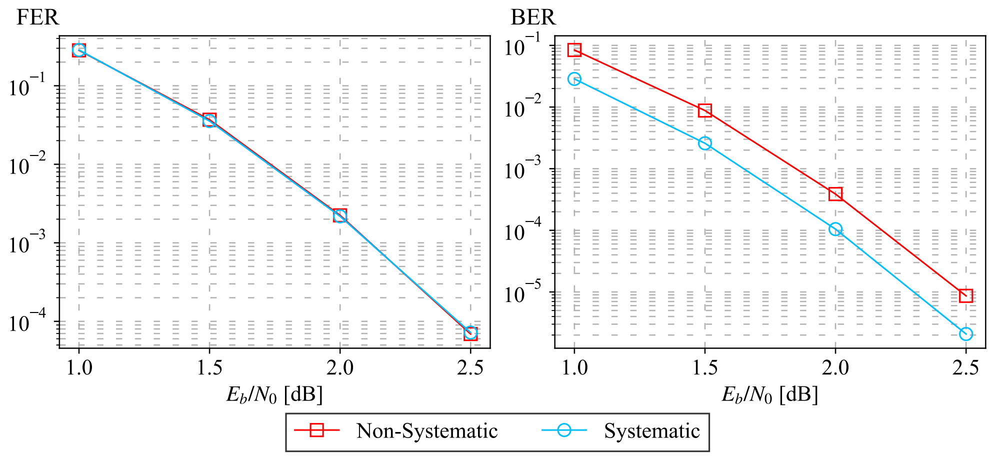


Figure 4.30 – Comparison between systematic and non-systematic polar encoding for $N = 1024$, $K = 512$, $L = 8$ and CRC-16, decoded by SSCL-SPC and constructed using DEGA optimized for $E_b/N_0 = 2$ dB.

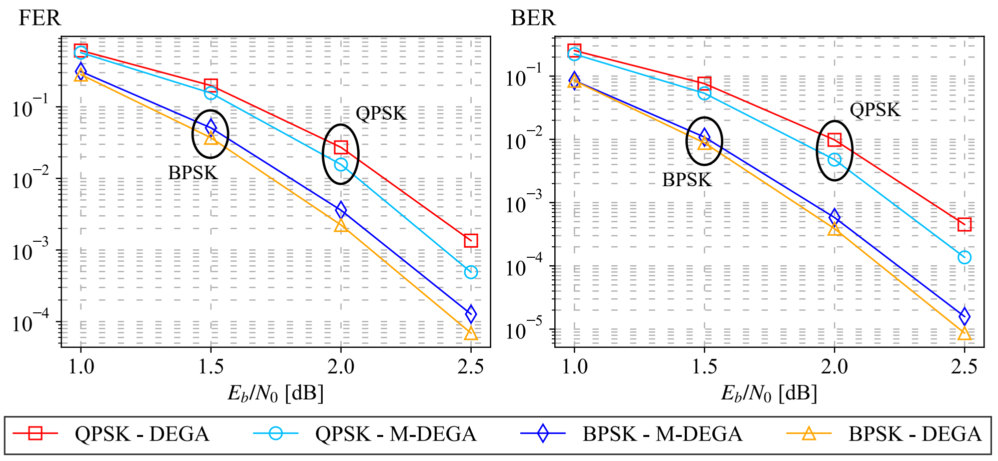


Figure 4.31 – Comparison between polar encoding performance under BPSK and QPSK transmission, for $N = 1024$, $K = 512$, $L = 8$ and CRC-16, decoded by SSCL-SPC and constructed using DEGA optimized for $E_b/N_0 = 2$ dB.

4.10 CONCLUSION

We started the chapter by formalizing a decoding algorithm based on a binary tree. This algorithm makes intuitive the idea that polar decoding is carried out recursively. Then, we proceeded by showing how the original SC decoding can be optimized without any performance loss, originating the Simplified SC (SSC) and Fast-SSC schemes [12, 13]. Then, we proceeded by introducing the Successive Cancellation List (SCL) decoding [5, 14], which allows one to approach the ML decoding of polar codes. We showed that even with performance near ML, polar codes can't compete with state-of-the-art DVB-RCS2 Turbo codes, which motivates the introduction of CRC concatenated polar codes. A preliminary analysis for list size $L = 8$ presented a 1 dB gain when compared to SC decoding. Simplifications to the SCL decoding, similar to those introduced to SC decoding, were introduced, which led to the SSCL and SSCL-SPC [15] decoding.

The first simulation results presented showed how CRC-aided performance increases dramatically when increasing the list size L. We explored the effects of code rate in list decoding and confirmed that the performance loss introduced by SSCL-SPC is in fact negligible [15]. CRC-aided list decoding allowed polar codes to achieve competitive performance when compared to same length and same rate Turbo codes.

We resumed the simulation results section by exploring the effects of code construction and design E_b/N_0 on the code performance, which showed how sensitive the code performance is to these settings. We concluded by analysing systematic list decoding and by performing an experiment using QPSK transmission.

5 CONCLUSION

We began this work in Chapter 2 by introducing and illustrating some fundamental concepts in communications and coding theory. These concepts included the mathematical modeling of the channel, where we reviewed *discrete memoryless channels*, *Binary Erasure Channels (BEC)*, *AWGN channels*, and definitions such as the *symmetric capacity* and the *Bhattacharyya parameter*. We introduced *block codes*, *linear codes* and *coset codes*, and showed how optimal decoding is closely related to minimizing distances and is a computationally unfeasible problem.

Next, in Chapter 3, we introduced virtual channels, a concept in the foundation of polar codes, and then proceeded to the underlying theoretical aspects of this coding technique. While on the theoretical realm, we showed that the Successive Cancellation (SC) decoding can be performed in the natural order without the need of the permutation matrix $\mathbf{R}_N$ proposed by Arikan [4], which was accomplished using equivalent representations of the polar decoder. We were also able to derive recursive relations, similar to those obtained by Arikan [4], that validate previously proposed tree algorithms [12, 13].

While still in Chapter 3, we reviewed several approximated construction methods for the AWGN channel, namely DEGA [10], M-DEGA [11], BEE [11] and Bhattacharyya [4] approximations. We performed computer simulations that allowed us to visualize the differences between these techniques and also allowed us to assess their performance. Our results show that under SC decoding the DEGA, M-DEGA and BEE methods had similar performance, while the Bhattacharyya approximation led to a poorer performance.

We devoted Chapter 4 to the detailed description of other decoding schemes and their performance. We first began by introducing lower latency versions of the SC decoding, namely the Simplified Successive Cancellation (SSC) [12] and Fast-SSC [13] decoders, which have the exact same performance as SC decoders [12, 13]. Even though polar codes are proven to achieve channel capacity, our simulation results showed that polar codes with SC decoding have poor performance when compared to same length and same rate state-of-the-art Turbo codes used on the Digital Video Broadcasting - Return Channel via Satellite - Second Generation (DVB-RCS2) standard. To address this issue, list decoding of polar codes was proposed by [5]. Our simulation results showed that even by introducing list decoding, polar codes also had a poor performance when compared to Turbo codes. Further improvement can be obtained by concatenating polar codes with a Cyclic Redundancy Check (CRC) word, as proposed by [5]. As our simulation results confirm, this modification improved drastically the performance of polar codes, making them a competitive alternative to DVB-RCS2 Turbo codes.

Also in Chapter 4, we presented the lower latency versions of the Successive Cancellation List (SCL) decoder, namely the Simplified Successive Cancellation List (SSCL) and SSCL-SPC decoders [15]. Our simulation results confirmed the claim of [15] that the approximated technique SSCL-SPC has negligible performance loss when compared to SSCL.

Optimal construction methods for CRC-aided polar codes are currently unknown: the same methods developed for SC decoding are also used for list decoding. To investigate further, a comparison between the four construction methods presented showed that the DEGA method has the best performance for CRC-aided polar codes. This comparison also revealed how sensitive to the construction method this performance is: only a few different bit positions between the methods caused the difference in performance observed. Also, by simulating the construction methods optimized to different Signal-to-Noise Ratios (SNRs), we showed that the CRC-aided list decoding performance is very sensitive to the SNR setting of the construction method.

We also considered SCL-CRC decoding using systematic polar codes. Our results show that the same behavior shown for SC decoding of systematic codes by Arikan [21] also happens for SCL-CRC decoding: both non-systematic and systematic schemes present the same Frame Error Rate (FER), while the systematic scheme has a better Bit Error Rate (BER) performance. Finally, we considered SCL-CRC decoding of QPSK modulated polar codes. We showed that polar codes perform worse in this case, as it was expected, since the conditions on which polar codes are modeled disappear. Also, the DEGA method performs better than M-DEGA in this scenario.

5.1 FUTURE WORK

Polar codes are still an active area of research. Throughout this work, we encountered several opportunities for future work. They are:

- search for better code construction approximations for SC decoding;
- search for optimal or approximated construction methods for CRC-aided polar codes;
- search for further latency improvements on the decoding algorithms;
- search how to adapt polar codes and their construction methods to other constellations, such as QPSK;
- simulate polar codes for higher order constellations such as 8PSK and 16QAM;
- investigate the performance of polar codes in satellite communications.

[1] Ericsson, “Ericsson mobility report,” https://www.ericsson.com/4adc87/assets/local/mobility-report/documents/2020/november-2020-ericsson-mobility-report.pdf, 2020, [Online; accessed in 03/12/2020].

[2] U. Madhow, *Introduction to Communication Systems.* Cambridge University Press, 2014.

[3] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 4, pp. 623–656, 1948.

[4] E. Arikan, “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, July 2009.

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

[6] 3GPP, “LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding,” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.212, 01 2010, version 10.0.0.

[7] T. Moon, *Error Correction Coding: Mathematical Methods and Algorithms.* Wiley, 2005.

[8] 3GPP, “5G; NR; Multiplexing and channel coding,” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.212, 07 2018, version 15.2.0.

[9] S. Guelton, P. Brunet, M. Amini, A. Merlini, X. Corbillon, and A. Raynaud, “Pythran: Enabling static optimization of scientific python programs,” *Computational Science & Discovery*, vol. 8, no. 1, p. 014001, 2015.

[10] P. Trifonov, “Efficient design and decoding of polar codes,” *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3221–3227, 2012.

[11] B. Tahir, “Construction and performance of polar codes for transmission over the awgn channel,” Master’s thesis, Technische Universität Wien, 2017.

[12] A. Alamdar-Yazdi and F. R. Kschischang, “A simplified successive-cancellation decoder for polar codes,” *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[13] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, “Fast polar decoders: Algorithm and implementation,” *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, 2014.

[14] 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.

[15] S. A. Hashemi, C. Condo, and W. J. Gross, “A fast polar code list decoder architecture based on sphere decoding,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2368–2380, 2016.

[16] B. Sklar, *Digital Communications: Fundamentals and Applications*, ser. Prentice Hall Communications Engineering and Emerging Techno. Prentice-Hall PTR, 2001.

[17] E. Ankan, N. ul Hassan, M. Lentmaier, G. Montorsi, and J. Sayir, “Challenges and some new directions in channel coding,” *Journal of Communications and Networks*, vol. 17, no. 4, pp. 328–338, 2015.

[18] E. Sasoglu, “Polar coding theorems for discrete systems,” Ph.D. dissertation, EPFL, 2011.

[19] Sae-Young Chung, T. J. Richardson, and R. L. Urbanke, “Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657–670, 2001.

[20] W. Tranter, K. Kosbar, T. Rappaport, and K. Shanmugan, *Principles of Communication Systems Simulation with Wireless Applications*, ser. Prentice Hall Communications E. Prentice Hall, 2004. [Online]. Available: https://books.google.com.br/books?id=3AcfAQAAIAAJ

[21] E. Arikan, “Systematic polar coding,” *IEEE Communications Letters*, vol. 15, no. 8, pp. 860–862, 2011.

[22] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, 2013.

[23] M. Mondelli, S. A. Hashemi, J. Cioffi, and A. Goldsmith, “Sublinear latency for simplified successive cancellation decoding of polar codes,” *IEEE Transactions on Wireless Communications*, pp. 1–1, 2020.

[24] S. A. Hashemi, C. Condo, and W. J. Gross, “Fast and flexible successive-cancellation list decoders for polar codes,” *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5756–5769, 2017.

APPENDIX

A RECURSIVE RELATION

Here we show how the recursive relations

$$W_{2N}^{(2i-1)} = \sum_{u_{2i}} \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}) \tag{A.1}$$

and

$$W_{2N}^{(2i)} = \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}) \tag{A.2}$$

are obtained.

We begin by noting that the combined channel transition probabilities $W_{2N}(y_1^{2N} \mid u_1^{2N})$ can be written recursively using

$$W_{2N}(y_1^{2N} \mid u_1^{2N}) = W_N(y_{1,o}^{2N} \mid u_{1,o}^{2N} \oplus u_{1,e}^{2N}) W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}). \tag{A.3}$$

Here, we used the fact that the two copies of W_N used to build W_{2N} are independent, and that the inputs to these channels are obtained as depicted in Figure A.1. Next, we substitute A.3 in the split channel definition

$$\begin{aligned}
W_{2N}^{(2i-1)}(y_1^{2N}, u_1^{2i-2} \mid u_{2i-1}) &\triangleq \frac{1}{2^{2N-1}} \sum_{u_{2i}^{2N}} W_{2N}(y_1^{2N} \mid u_1^{2N}) \\
&= \frac{1}{2^{2N-1}} \sum_{u_{2i}^{2N}} W_N(y_{1,o}^{2N} \mid u_{1,o}^{2N} \oplus u_{1,e}^{2N}) W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}) \\
&= \frac{1}{2^{2N-1}} \sum_{u_{2i,o}^{2N}, u_{2i,e}^{2N}} W_N(y_{1,o}^{2N} \mid u_{1,o}^{2N} \oplus u_{1,e}^{2N}) W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}) \\
&= \sum_{u_{2i}} \frac{1}{2} \sum_{u_{2i+1,e}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}) \sum_{u_{2i+1,o}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,o}^{2N} \mid u_{1,o}^{2N} \oplus u_{1,e}^{2N}).
\end{aligned} \tag{A.4}$$

Until now, we only separated the sums. Now, note that in the last sum the vector $u_{1,o}^{2N} \oplus u_{1,e}^{2N}$ has its lower part $u_{1,o}^{2i} \oplus u_{1,e}^{2i}$ fixed, while the top part $u_{2i+1,o}^{2N} \oplus u_{2i+1,e}^{2N}$ varies. For each $u_{2i+1,e}^{2N}$ fixed, determined by the outer sum, varying $u_{2i+1,o}^{2N}$ on the inner sum results in the top part $u_{2i+1,o}^{2N} \oplus u_{2i+1,e}^{2N}$ assuming every possible value in $\mathcal{X}^{N-i}$. Using the split channel definition again, we rewrite the last sum as

$$\sum_{u_{2i+1,o}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,o}^{2N} \mid u_{1,o}^{2N} \oplus u_{1,e}^{2N}) = W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}). \tag{A.5}$$

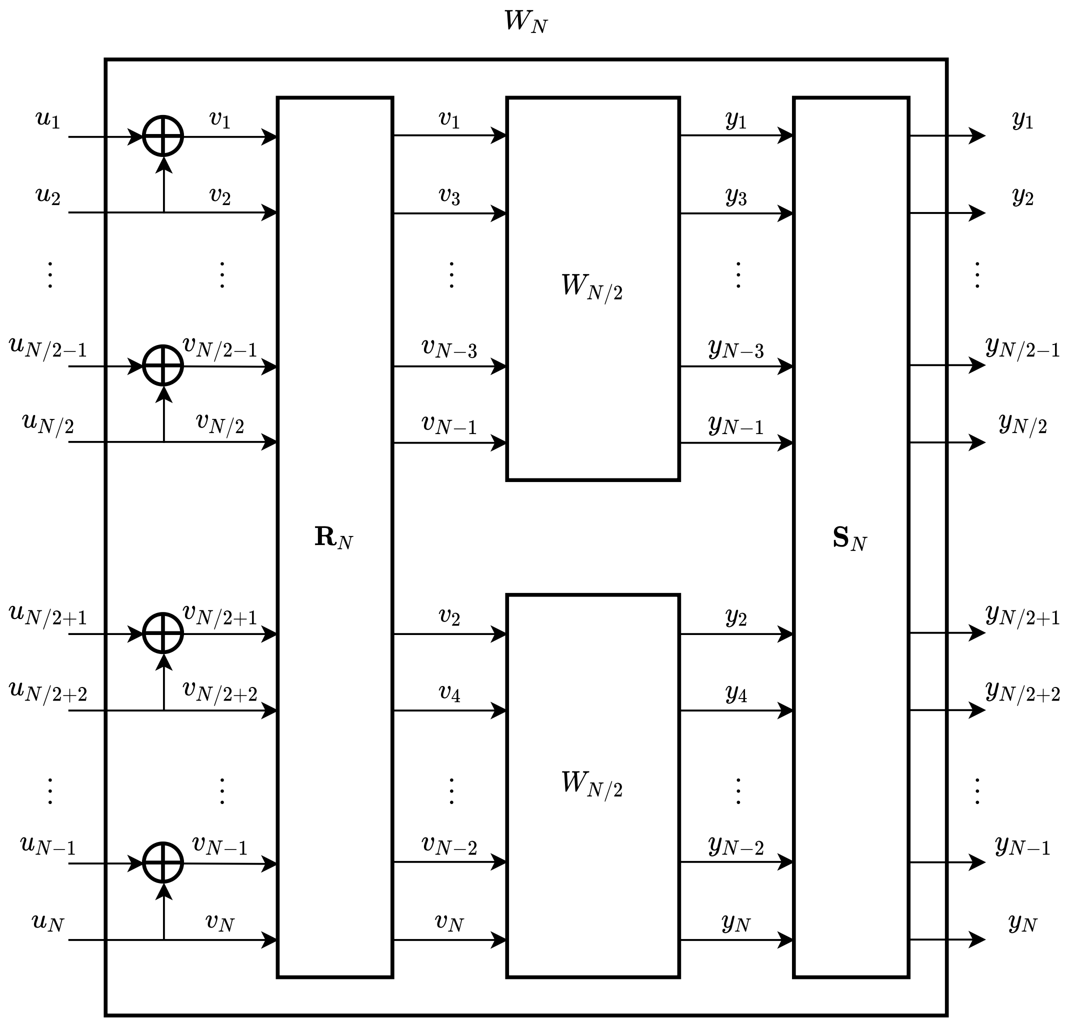


Figure A.1 – An equivalent representation of the combined channel using a recursive construction suitable to obtain the recursive relations.

Substituting A.5 in A.4 we get

$$W_{2N}^{(2i-1)}(y_1^{2N}, u_1^{2i-2} \mid u_{2i-1}) \\ = \sum_{u_{2i}} \frac{1}{2} \sum_{u_{2i+1,e}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}) W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) \tag{A.6}$$

and, factoring out the $W_N^{(i)}$ term we get

$$W_{2N}^{(2i-1)}(y_1^{2N}, u_1^{2i-2} \mid u_{2i-1}) \\ = \sum_{u_{2i}} \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) \sum_{u_{2i+1,e}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}). \tag{A.7}$$

Using the split channel definition again, we get to the final result

$$W_{2N}^{(2i-1)}(y_1^{2N}, u_1^{2i-2} \mid u_{2i-1}) \\ = \sum_{u_{2i}} \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}). \tag{A.8}$$

The second relation can be obtained by writing

$$W_{2N}^{(2i)}(y_1^{2N}, u_1^{2i-1} \mid u_{2i}) = \frac{1}{2} \sum_{u_{2i+1,e}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,e}^{2N} \mid u_{1,e}^{2N}) \sum_{u_{2i+1,o}^{2N}} \frac{1}{2^{N-1}} W_N(y_{1,o}^{2N} \mid u_{1,o}^{2N} \oplus u_{1,e}^{2N}) \tag{A.9}$$

and by carrying out the rest of the development in the same way as was for the first relation.

B AWGN BHATTACHARYYA PARAMETER

If $W : \{-\sqrt{E_s}, \sqrt{E_s}\} \to \mathbb{C}$ is the AWGN channel with variance $\sigma^2 = N_0/2$ under BPSK transmission, we have that the Bhattacharyya parameter is computed by

$$Z(W) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \sqrt{W(r + j \cdot q \mid \sqrt{E_s})W(r + j \cdot q \mid -\sqrt{E_s})}\, dr\, dq, \tag{B.1}$$

where

$$W(r + j \cdot q \mid \pm\sqrt{E_s}) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{q^2 + (r \mp \sqrt{E_s})^2}{2\sigma^2}\right), \tag{B.2}$$

and double integrals are used instead of sums, since we have 2-dimensinal probability density functions $W(\cdot \mid \cdot)$. Then,

$$\begin{aligned}
Z(W) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2}\sqrt{\exp\left(\frac{-2q^2 - (r - \sqrt{E_s})^2 - (r + \sqrt{E_s})^2}{2\sigma^2}\right)}\, dr\, dq \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2}\sqrt{\exp\left(\frac{-2q^2 - r^2 + 2r\sqrt{E_s} - E_s - r^2 - 2r\sqrt{E_s} - E_s}{2\sigma^2}\right)}\, dr\, dq \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2}\sqrt{\exp\left(\frac{-2q^2 - 2r^2 - 2E_s}{2\sigma^2}\right)}\, dr\, dq \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2}\exp\left(\frac{-q^2 - r^2 - E_s}{2\sigma^2}\right)\, dr\, dq \\
&= \exp\left(\frac{-E_s}{2\sigma^2}\right)\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{2\pi\sigma^2}\exp\left(\frac{-(q^2 + r^2)}{2\sigma^2}\right)\, dr\, dq.
\end{aligned} \tag{B.3}$$

Since the two-dimensional Gaussian function above has area 1, we get the final result

$$\begin{aligned}
Z(W) &= \exp\left(-\frac{E_s}{2\sigma^2}\right) \\
&= \exp\left(-\frac{E_s}{N_0}\right).
\end{aligned}$$

C RESUMO ESTENDIDO EM LÍNGUA PORTUGUESA

Título: Estudo e Análise de Desempenho de Algoritmos de Construção e Decodificação de Códigos Polares em Canais de Comunicação com Ruído do Tipo Branco Gaussiano (AWGN)
Autor: Rodrigo Andres Rodrigues Fischer
Orientador: Prof. Dr. João Paulo Leite
Monografia de Trabalho de Conclusão de Curso em Engenharia Elétrica
Brasília, 11 de dezembro de 2020

Palavras-chave: Códigos Polares, AWGN, Algoritmos de Construção, Algoritmos de Decodificação.

INTRODUÇÃO

O correto funcionamento da maioria dos sistemas de comunicação depende de usarmos técnicas para corrigir os erros introduzidos pelo canal de comunicação. Ruído térmico, por exemplo, presente nos circuitos dos dispositivos de comunicação, pode causar erros na recepção do sinal enviado. Em 1948, Shannon [3] mostrou com o Teorema Fundamental da Teoria da Informação que os esquemas de codificação de erro tem desempenho limitado por um certo limite, conhecido como capacidade do canal. Entretanto, foi apenas em 2009 que Arikan [4] propôs os *códigos polares*, a primeira família de códigos com construção explícita a provadamente atingir a capacidade do canal. Esses códigos tiram proveito de um fenômeno denominado *polarização de canal*, no qual alguns bits são transmitidos com grande confiabilidade e outros não.

Os códigos polares são atraentes por possuírem esquemas de codificação e decodificação com baixa complexidade e por possuírem uma construção explícita que os torna capazes de atingir a capacidade do canal. Porém, essa capacidade é atingida apenas quando o tamanho do bloco tende ao infinito. De fato, verificou-se que os códigos polares, da maneira como definidos por Arikan, apresentam desempenho de erro inferior àquele dos códigos LDPC [5]. Além disso, sua construção ótima, apesar de explícita, é um problema computacionalmente intratável. Outro desafio dessa técnica é a sua alta latência, que possui dependência linear com o tamanho de bloco. Estes e outros desafios tornaram os códigos polares uma área ativa de pesquisa na última década.

OBJETIVOS

O primeiro objetivo deste trabalho é apresentar uma revisão abrangente de três tópicos principais: a teoria básica dos códigos polares, métodos de construção aproximados, e algoritmos de decodificação eficientes que permitem a redução da latência de decodificação. Antes de entrar em códigos polares, fornecemos ao leitor os conceitos necessários em teoria de comunicações e codificação para que o leitor seja capaz de ganhar intuição sobre as técnicas abordadas à medida que descrições detalhadas dos algoritmos são apresentadas.

Neste trabalho, abordamos as técnicas de decodificação Successive Cancelation (SC), proposta originalmente por [4], e Successive Cancelation List (SCL) [5, 14], proposta a fim de tornar o desempenho dos códigos polares comparável com outras técnicas estado-da-arte. Vertentes de menor latência de ambos os decodificadores são tratadas, como o Simplified SC [12] e Fast-SC [13], Simplified SCL e SSCL-SPC [15]. Além disso, o desempenho desses decodificadores é avaliado com diferentes técnicas de construção, entre elas a construção Bhattacharyya [11], BEE (Bit Error Evolution) [11], DEGA (Density Evolution with Gaussian Approximation) [10] e, por fim, Modified DEGA [11].

O segundo objetivo deste trabalho é desenvolver implementações próprias em software dos algoritmos revisados, a fim de gerar resultados de simulação abrangentes. Utilizamos a linguagem de programação Python, com partes do código otimizadas utilizando o compilador estático Pythran [9].

Em terceiro lugar, apresentamos ao leitor os resultados de simulações mencionados acima. Comparamos os métodos de construção apresentados para múltiplos algoritmos de decodificação e mostramos uma análise sobre as semelhanças entre os métodos de construção, o que destaca potenciais áreas futuras de pesquisa. Também exploramos como a mudança da modulação utilizada causa a perda de condições ótimas dos códigos polares, e destacamos possíveis soluções para esse problema.

Como outra contribuição, também mostramos que a decodificação da ordem natural dos códigos polares não é uma consequência da matriz de codificação, mas da forma como definimos a estrutura recursiva dos códigos polares. Obtivemos novas relações recursivas, que são similares às obtidas por Arikan [4], para a nova estrutura de recursão proposta.

RESULTADOS E DISCUSSÃO

Decodificação em Ordem Natural

Primeiramente, mostramos que o esquema de codificação mostrado na Figura C.1 é equivalente à codificação feita por

$$\mathbf{x} = \mathbf{uF}, \tag{C.1}$$

mas que permite, porém, a decodificação na ordem natural, sem o uso da matriz de permutação introduzida por Arikan. Utilizando o argumento da indução matemática, podemos mostrar que, assumindo a decodificação na ordem natural em $W_{N/2}$, temos decodificação em ordem natural em W_N. Mostramos as seguintes relações recursivas para o esquema proposto:

$$W_{2N}^{(2i-1)} = \sum_{u_{2i}} \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}), \tag{C.2}$$

$$W_{2N}^{(2i)} = \frac{1}{2} W_N^{(i)}(y_{1,o}^{2N}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}) W_N^{(i)}(y_{1,e}^{2N}, u_{1,e}^{2i-2} \mid u_{2i}). \tag{C.3}$$

Análise da Decodificação SC e Métodos de Construção

Mostramos gráficos que permitem visualizar a distribuição dos parâmetros utilizados para construir os códigos para as diferentes técnicas. Na Figura C.2 observamos como os parâmetros Bhattacharyya aproximados estão distribuídos no intervalo $[0, 1]$, e como essa distribuição se reflete nos índices de informação (*information*) e congelados (*frozen*). Um gráfico análogo para o método M-DEGA é mostrado na Figura C.3.

Em seguida, comparamos os diferentes métodos de construção para blocos de diversos tamanhos e diversas taxas de código. Nas Figuras C.4 e C.5 podemos ver para $N = 4096$ e $N = 256$, respectivamente, o efeito do método de construção no desempenho de erro dos códigos, medido em taxa de erro de quadro (FER) e taxa de erro de bit (BER). Podemos ver que para $N = 4096$ o método Bhattacharyya apresenta pior desempenho, enquanto para $N = 256$ todos os métodos apresentam desempenho similar.

Uma análise extra foi feita para o caso $N = 4096$, onde, na Figura C.6, vemos as posições de bit onde os métodos mostrados foram diferentes do método Bhattacharyya. Podemos ver um deslocamento de bits de informação das posições superiores para as posições inferiores.

Outra análise importante é mostrada na Figura C.7, que mostra o que acontece quando variamos a SNR de projeto, enquanto mantemos a SNR do canal fixa. Vemos claramente um mínimo local quando a SNR de projeto é a mesma que a SNR do canal.

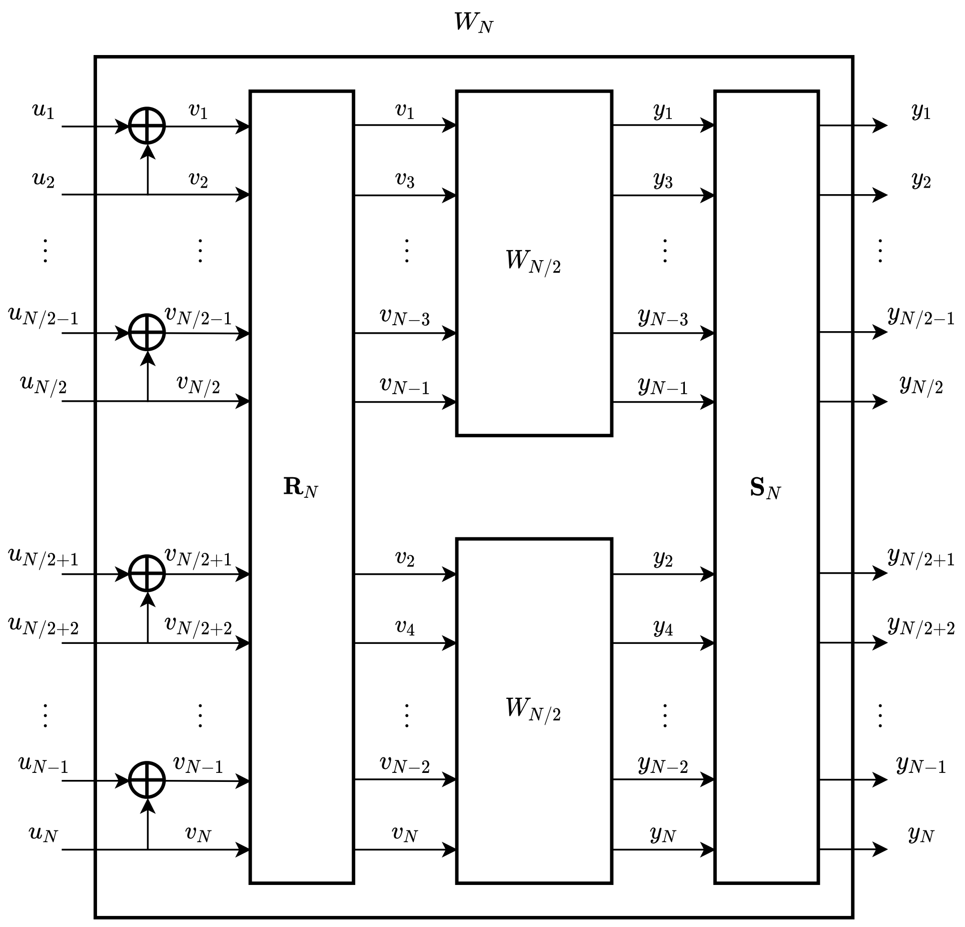


Figura C.1 – Uma representação equivalente dos canais combinados adequada para a decodificação em ordem natural.

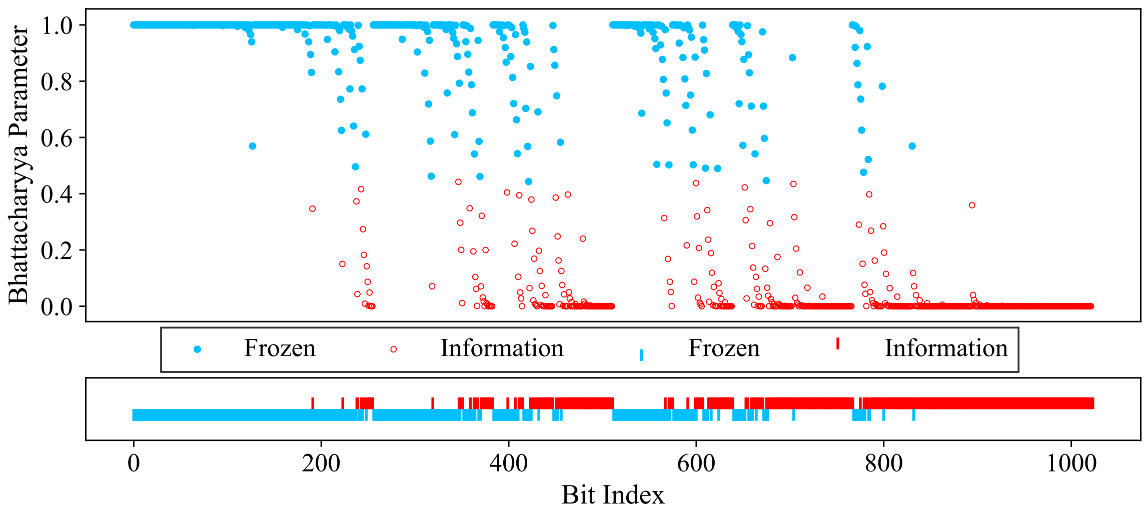


Figura C.2 – Gráfico dos parâmetros Bhattacharyya obtidos com o método de construção Bhattacharyya para $N = 1024$ e $E_s/N_0 = -1.5$ dB. Os índices de informação e congelados foram escolhidos para $K = 512$.

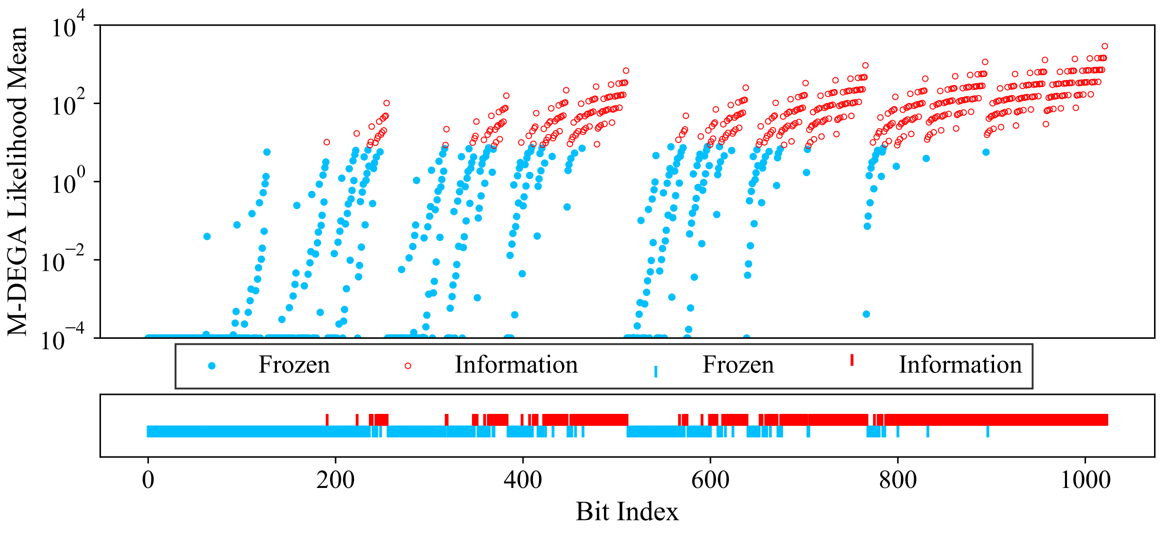


Figura C.3 – Gráfico das médias das verossimilhanças m_N^i obtidas com o método de construção M-DEGA para $N = 1024$ e $E_s/N_0 = -1.5$ dB. Valores menores que o intervalo mostrado foram representados em 10^{-4}. Os índices de informação e congelados foram escolhidos para $K = 512$.

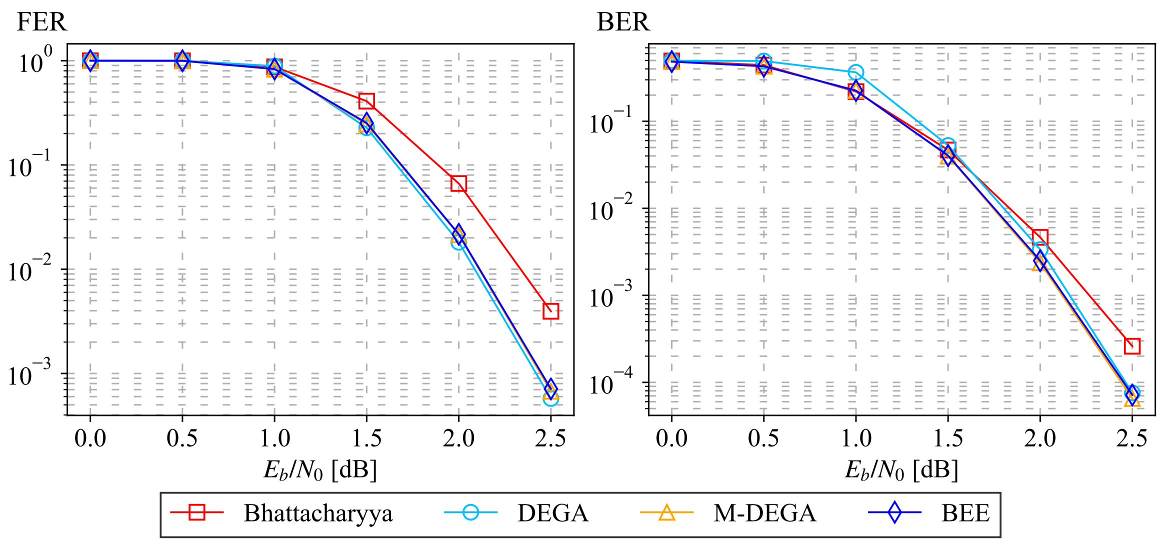


Figura C.4 – Resultados da simulação para $N = 4096$ e $K = 2048$ ($R = 1/2$).

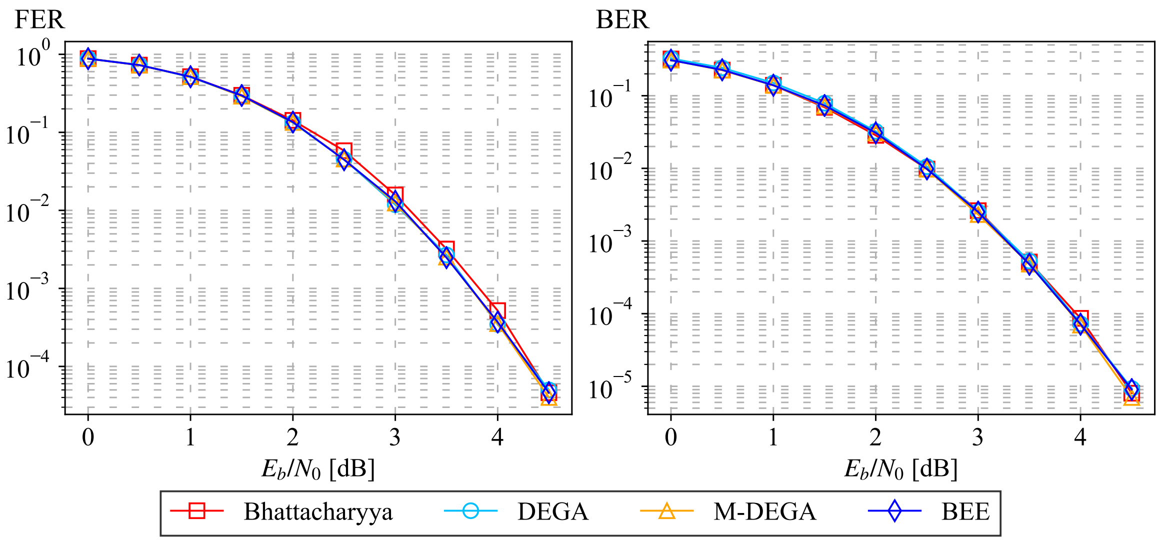


Figura C.5 – Resultados da simulação para $N = 256$ e $K = 128$ ($R = 1/2$).

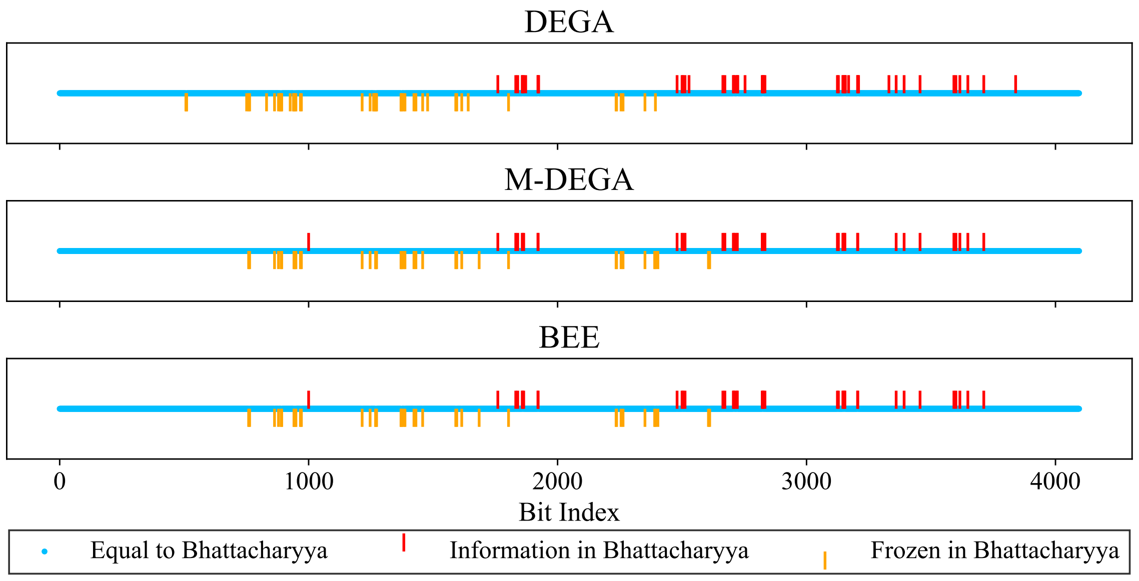


Figura C.6 – Diferença entre os bits de informação e congelados para os métodos mostrados, comparando-se com o método Bhattacharyya, com $N = 4096$, $K = 2048$ e $E_b/N_0 = 2.5$ dB.

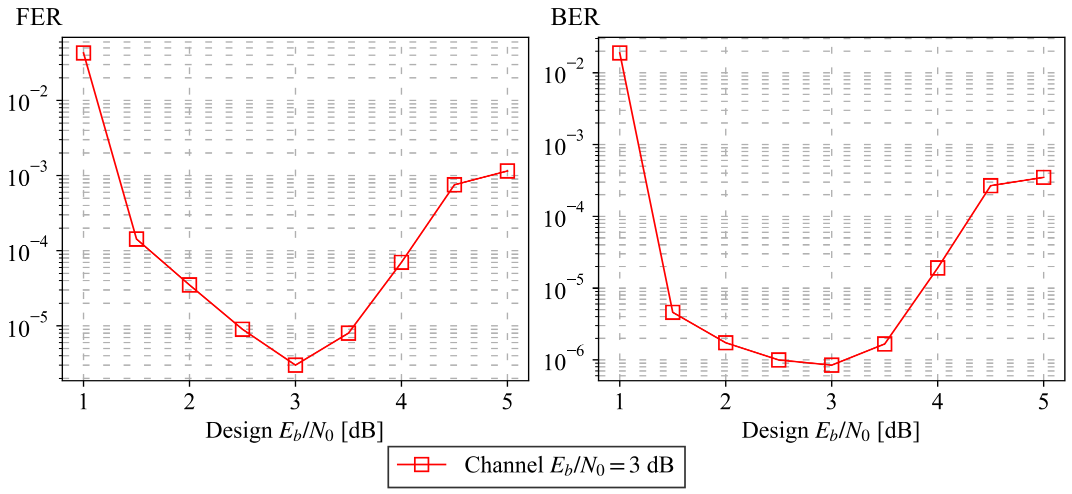


Figura C.7 – Desempenho para $E_b/N_0 = 3$ dB do canal fixa, enquanto variou-se a E_b/N_0 de projeto, para $N = 4096$, $K = 2048$ e utilizando o método de construção DEGA.

Como foi observado na Figura C.7, o projeto dos códigos polares varia de acordo com as condições do canal. Um gráfico que deixa visível o porquê isso acontece é mostrado na Figura C.8, onde vemos que, por exemplo, o canal 13 era o menos confiável de todos para E_s/N_0 baixo, enquanto para E_s/N_0 mais alta foi o mais confiável.

Por fim, mostramos o efeito do aumento do tamanho de bloco no desempenho dos códigos polares. Na Figura C.9 vemos que ao se aumentar o valor de N de 256 para $16\ 384$, um fator de 64x, obtivemos um ganho de aproximadamente 2 dB para BER de 10^{-4}. Além disso, comparamos os códigos polares com $N = 1024$ e $K = 512$ com os códigos Turbo do padrão DVB-RCS2 de comunicações satelitais com a mesma taxa de código e tamanho de bloco. Podemos ver que os códigos polares apresentam desempenho insatisfatório em comparação com os códigos Turbo.

Análise da Decodificação SCL e Métodos de Construção

Primeiramente, mostramos como tanto o algoritmo SC ou o algoritmo SCL podem ter sua latência reduzida utilizando-se esquemas de decodificação simplificados. Na Figura C.11 vemos como a árvore de decodificação pode ser podada ao se introduzir tipos especiais de nós que permitem a decodificação direta sem prosseguir a diante para as folhas. Vemos na Figura C.11 que nós simplificados com tamanho até $2^7 = 128$ foram obtidos para essa configuração.

Em seguida, introduzimos o método SCL proposto por [5], cujo objetivo é melhorar o desempenho dos códigos polares. Na Figura C.12 observamos o desempenho dessa técnica para diversos tamanhos de lista. Fica claro que aumentar o tamanho da lista oferece uma melhora no desempenho desses códigos, mas também que há um certo limite que é rapidamente atingido.

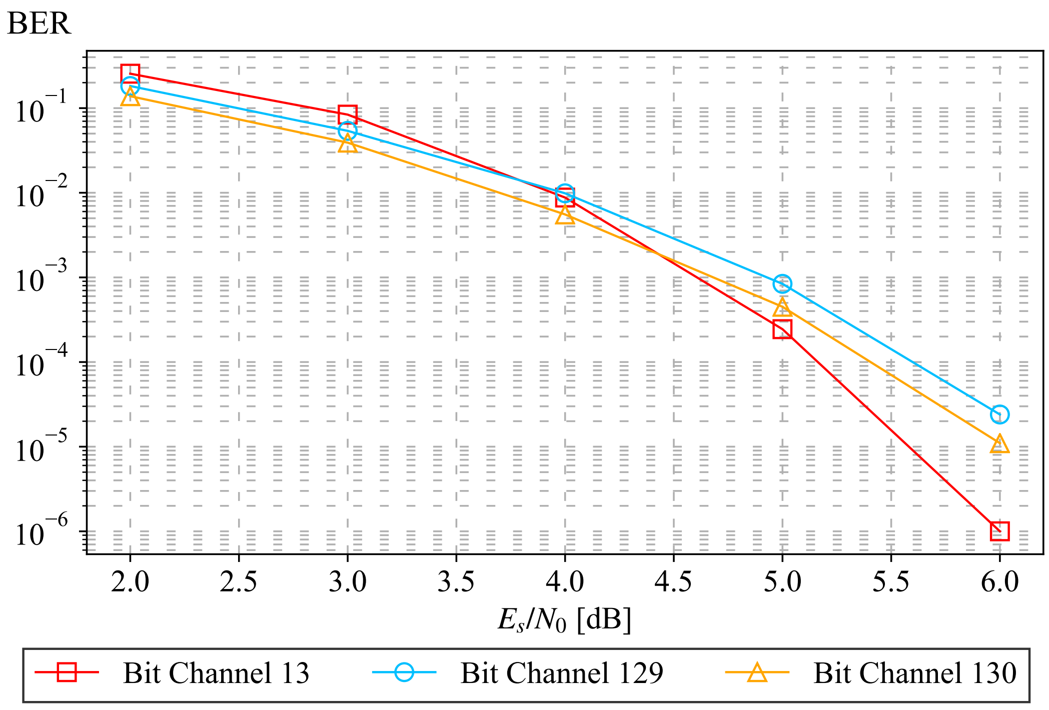


Figura C.8 – Taxa de erro de canal de bit simulada para $N = 256$ para os canais 13, 129 e 130.

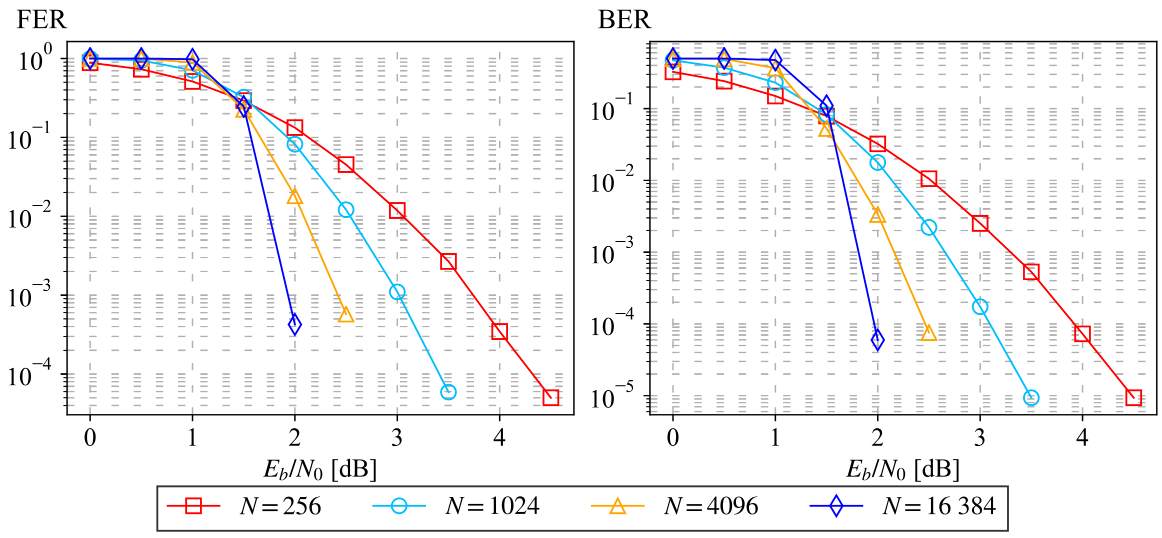


Figura C.9 – Resultados da simulação para $R = 1/2$ utilizando-se o método de construção DEGA.

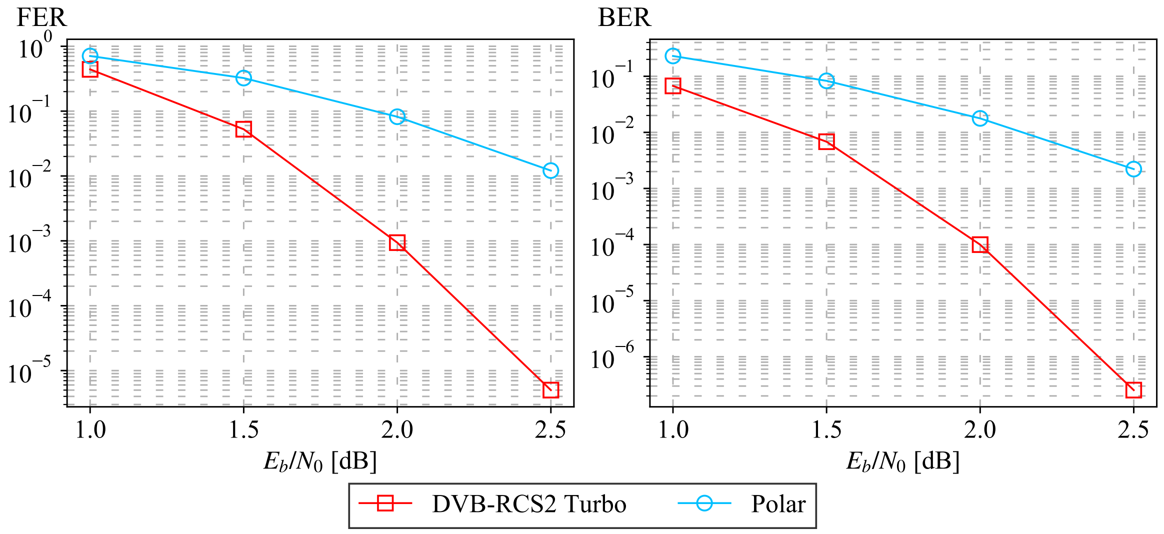


Figura C.10 – Resultados da simulação para $R = 1/2$ onde compara-se os códigos Turbo do DVB-RCS2 com os códigos polares construídos usando DEGA para $N = 1024$.

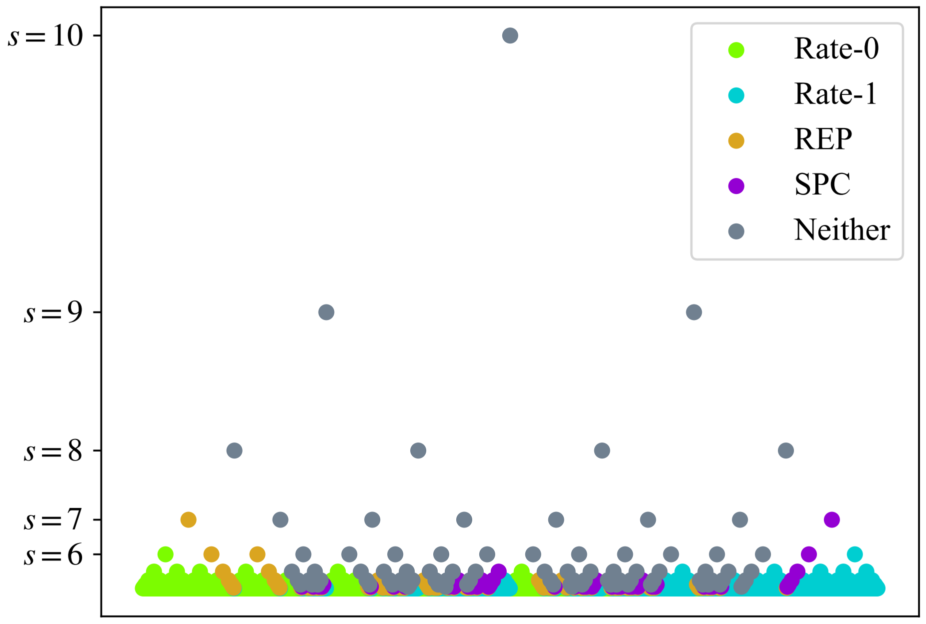


Figura C.11 – Exemplo de árvore de decodificação para o Fast-SSC/SSCL-SPC com $N = 1024$, $K = 512$ e construído utilizando-se o DEGA com $E_s/N_0 = 0$ dB.

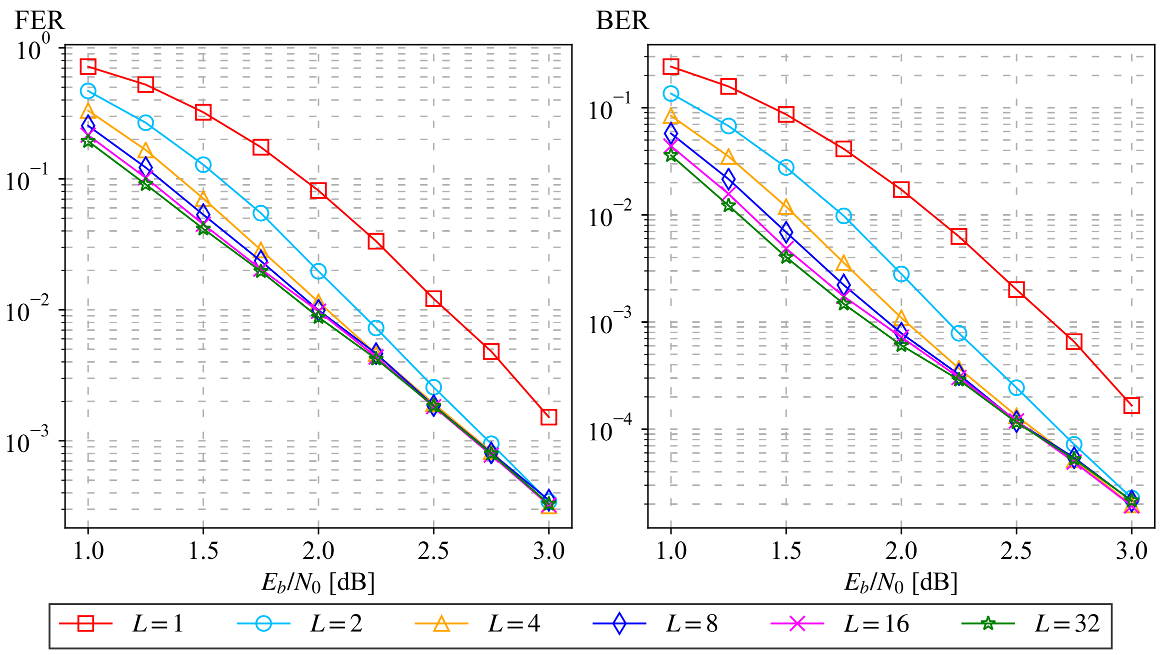


Figura C.12 – Desempenho da decodificação SCL para $N = 1024$, $L = 1$ (SC) até $L = 32$, utilizando-se o método DEGA otimizado para $E_b/N_0 = 2$ dB.

Após introduzir a verificação de paridade CRC [5], fomos capazes de obter uma melhora significativa no desempenho dos códigos polares, como é mostrado na Figura C.13. Na Figura C.14 vemos como os códigos polares concatenados com CRC se comportam com outros tamanhos de lista. Utilizando-se essa configuração, conseguimos mostrar que os códigos polares são capazes de atingir desempenho de erro comparável aos códigos Turbo do DVB-RCS2, como apontado na Figura C.15.

Comparamos também o desempenho dos códigos polares com SLC para os métodos de construção abordados. Vemos na Figura C.16 que o método DEGA foi o que apresentou melhor desempenho, enquanto o M-DEGA e BEE apresentaram desempenho similar, e o método Bhattacharyya apresentou o pior desempenho. Esses vão contra o que foi observado antes, que os métodos M-DEGA, BEE e DEGA apresentavam desempenho similar. De fato, métodos otimizados para SCL-CRC ainda são uma área ativa de pesquisa. Outro indício da não otimalidade desses métodos para o caso SCL-CRC é apresentado na Figura C.17, onde vemos que temos dois pontos de mínimo local, e em um desses pontos a SNR de projeto é diferente da do canal.

Por fim, analisamos o comportamento dos códigos polares com o QPSK. No QPSK, perdemos a hipótese de que o canal é sem memória, uma vez que dois bits são transmitidos de uma vez em um mesmo símbolo. Podemos ver uma piora de desempenho entre o QPSK e o BPSK. Surpreendentemente, o método M-DEGA apresentou melhor desempenho que o DEGA para o QPSK.

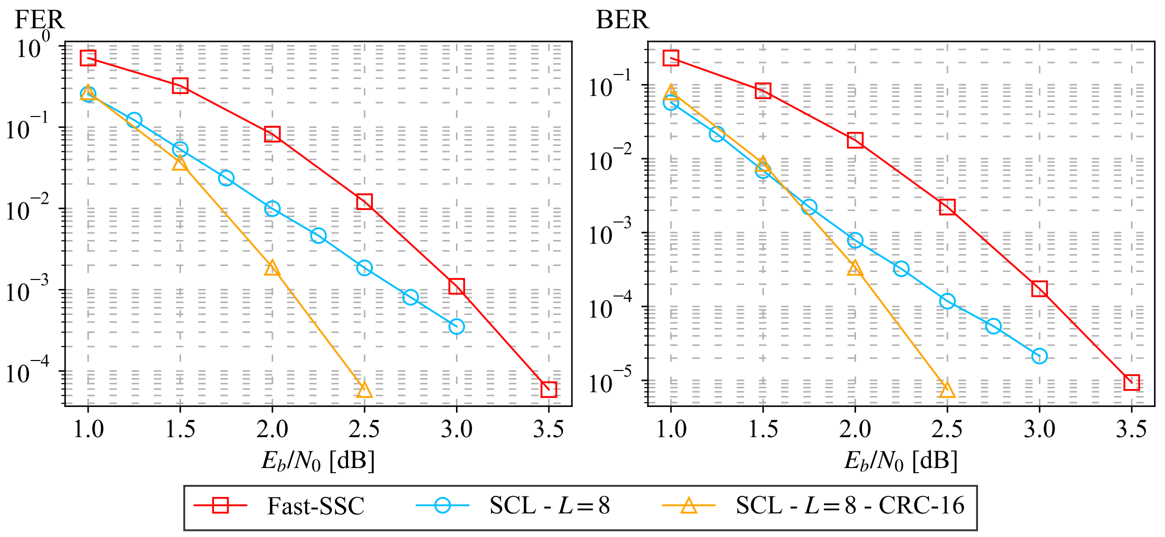


Figura C.13 – Uma comparação entre o Fast-SSC, SCL e SCL-CRC, com $L = 8$, para tamanho de bloco $N = 1024$ e $K = 512$. Utilizamos o CRC-16.

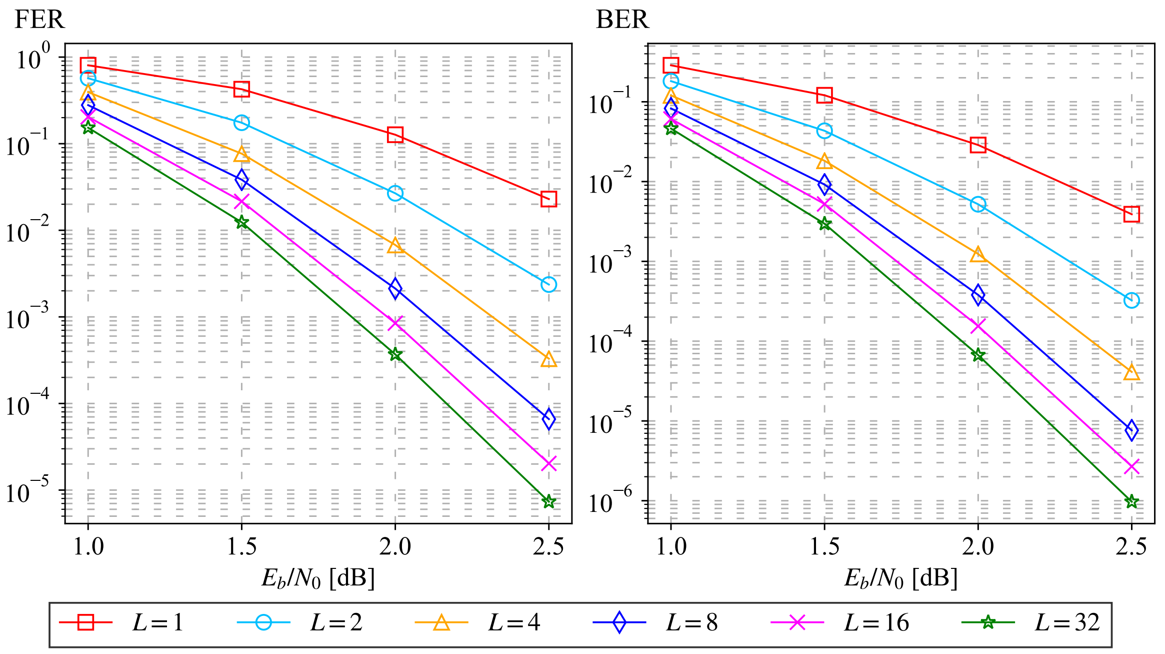


Figura C.14 – Desempenho dos códigos polares com diferentes tamanhos de lista, com $N = 1024$, $K = 512$ e CRC-16, decodificados com SSCL-SPC e construídos com DEGA otimizado para $E_b/N_0 = 2$ dB.

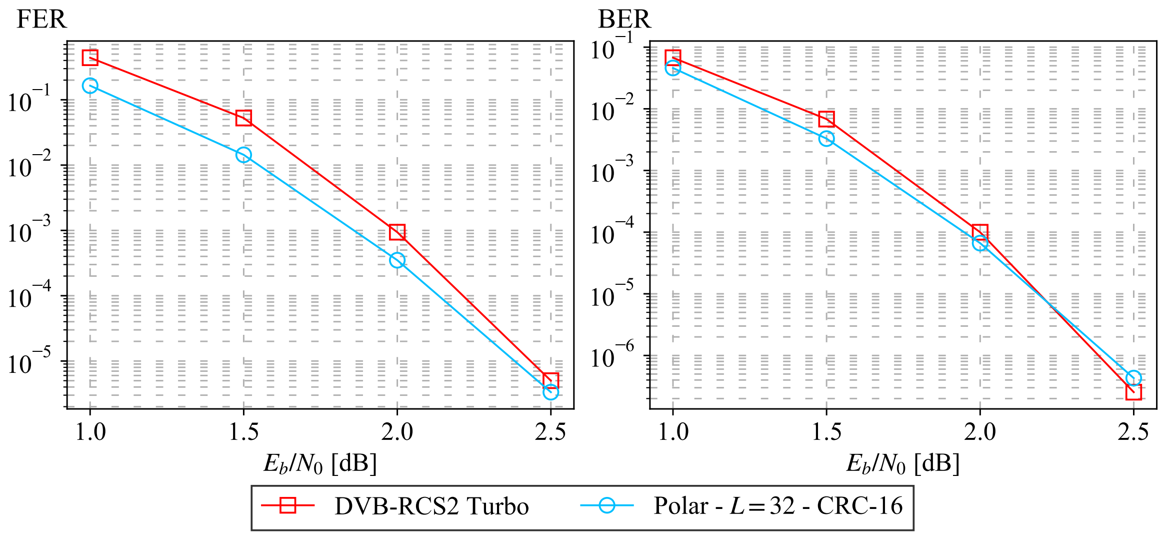


Figura C.15 – Comparação de desempenho de erro entre os códigos Turbo do DVB-RCS2 e códigos polares decodificados com SSCL-SPC, com $L = 32$, CRC-16 e construídos usando o DEGA otimizado com a SNR do canal. Ambos os códigos possuem $N = 1024$. Os códigos polares possuem $K = 496$ bits de informação e os códigos Turbo $K = 512$.

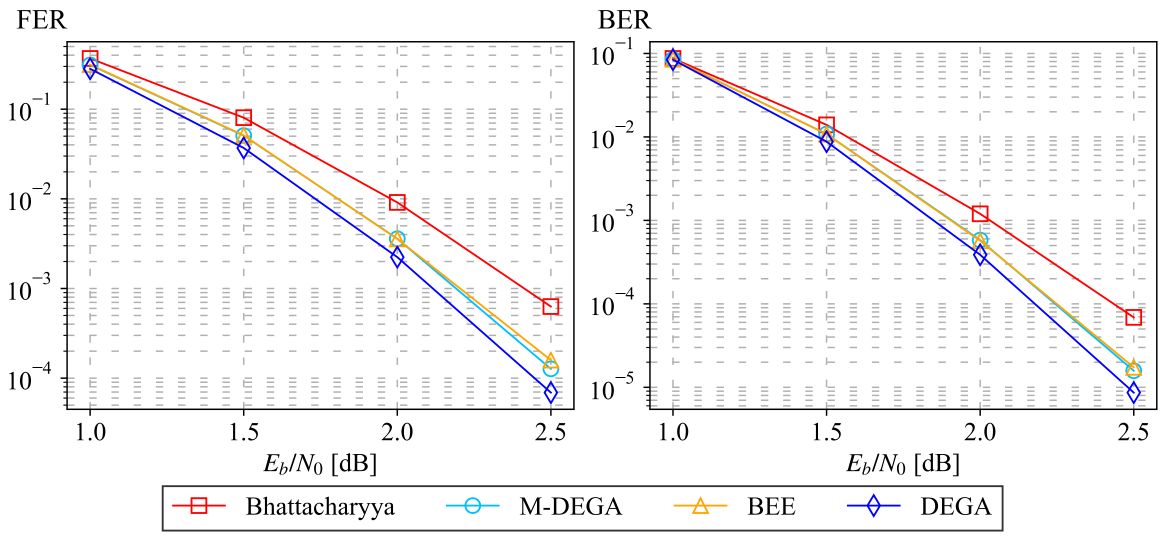


Figura C.16 – Uma comparação entre os diferentes métodos de construção para códigos polares com $N = 1024$, $K = 512$, $L = 8$, CRC-16, decodificados com SSCL-SPC e otimizados para $E_b/N_0 = 2$ dB.

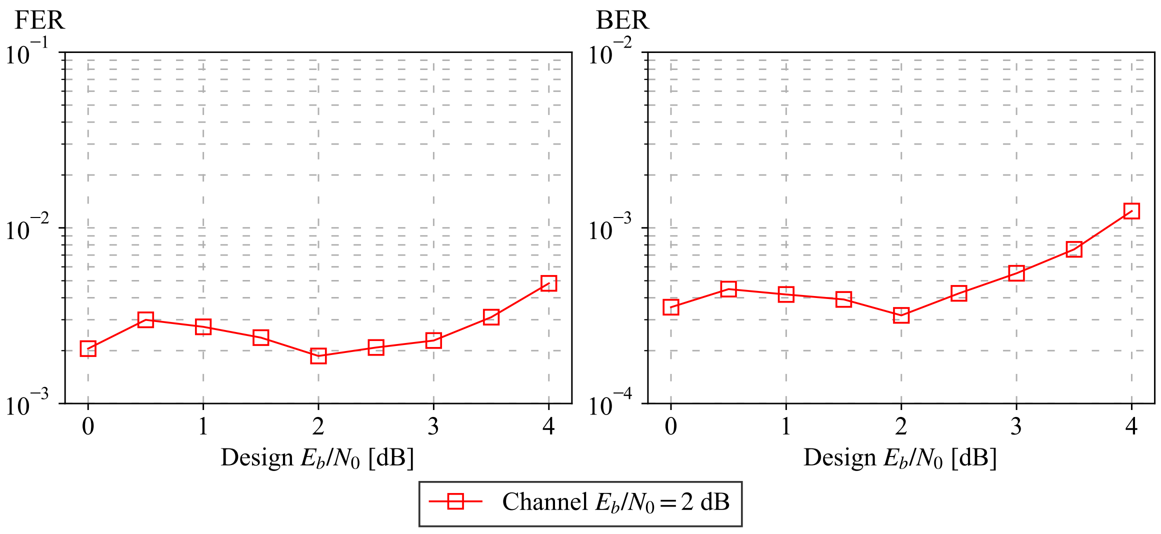


Figura C.17 – Desempenho dos códigos polares para $E_b/N_0 = 2$ dB fixa, enquanto variou-se a E_b/N_0 de projeto, para $N = 1024$, $K = 512$, $L = 8$, CRC-16, decodificados utilizando-se SSCL e construídos utilizando-se o DEGA.

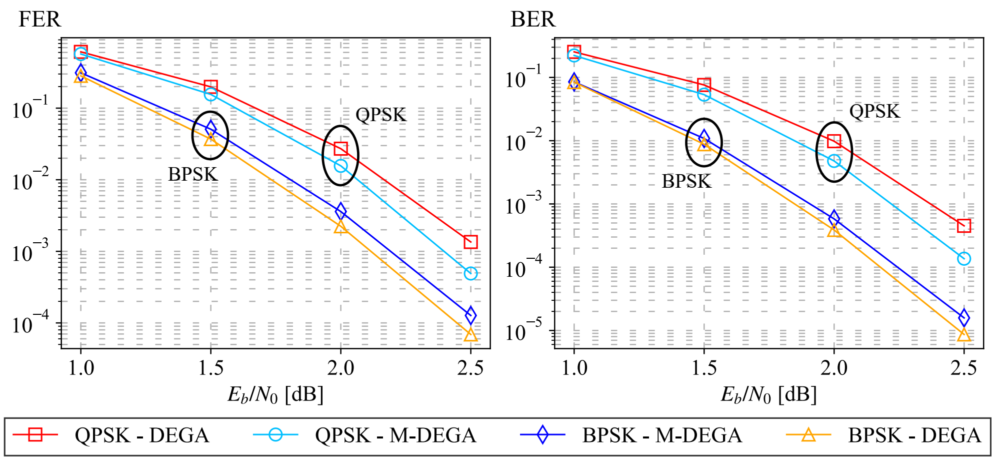


Figura C.18 – Comparação de desempenho dos códigos polares para transmissão BPSK e QPSK, para $N = 1024$, $K = 512$, $L = 8$ and CRC-16, decodificados com SSCL-SPC e construídos utilizando-se DEGA para $E_b/N_0 = 2$ dB.

CONCLUSÕES

No campo teórico, mostramos que a decodificação SC pode ser realizada na ordem natural sem a necessidade da matriz de permutação proposta por Arikan, o que foi realizado utilizando representações equivalentes dos canais combinados. Também conseguimos derivar relações recursivas, semelhantes às obtidas por Arikan, que validam algoritmos de árvores previamente propostos em [12, 13].

Realizamos simulações computacionais nos permitiram visualizar as diferenças entre os diferentes métodos de construção e também nos permitiram avaliar seu desempenho. Nossos resultados mostram que sob a decodificação SC os métodos DEGA, M-DEGA e BEE tiveram desempenho semelhante, enquanto a aproximação de Bhattacharyya teve o pior desempenho.

Embora os códigos polares sejam comprovadamente capazes de atingir a capacidade de canal, nossos resultados de simulação mostraram que esses códigos com decodificação SC têm desempenho ruim quando comparados com códigos Turbo de mesmo comprimento e a mesma taxa. Para resolver este problema, a decodificação em lista foi proposta por [5]. Nossos resultados de simulação mostraram que mesmo introduzindo a decodificação por lista, os códigos polares também tiveram um desempenho ruim quando comparados aos códigos Turbo. É possível obter melhorias adicionais ao concatenar códigos polares com uma palavra CRC, como proposto por [5]. Como confirmam nossos resultados de simulação, esta modificação melhorou drasticamente o desempenho desses códigos, tornando-os uma alternativa competitiva aos códigos Turbo DVB-RCS2.

Os métodos de construção ótimos para o SCL-CRC são atualmente desconhecidos. Para investigar mais, uma comparação entre os quatro métodos de construção apresentados para o SC mostrou que o método DEGA tem o melhor desempenho para os códigos polares concatenados com CRC. Ao simular o método DEGA otimizado para diferentes SNR de projeto, mostramos que o desempenho de decodificação do SCL-CRC é muito sensível a esse parâmetro.

Finalmente, consideramos os códigos polares com modulação QPSK. Mostramos que esses códigos têm pior desempenho neste caso, como era esperado, uma vez que as condições nas quais os códigos polares são projetados desaparecem.