Master Thesis

Construction and Performance of Polar Codes for Transmission over the AWGN Channel

Author
Bashar Tahir, B.Sc.
01476041

Supervisor
Univ.Prof. Dipl.-Ing. Dr.techn. Markus Rupp

Co-supervisor
Univ.Ass. Dipl.-Ing. Dr.techn. Stefan Schwarz

Institute of Telecommunications
Technische Universität Wien
Vienna, Austria

October, 2017

Abstract

Polar codes attracted a lot of attention since they were introduced by Arıkan in 2008 [1]. They are the first practical codes that are proven to achieve the channel capacity at infinite length. The field of polar coding is an active field of research, and one of its main topics is the code construction. The construction of polar codes involves finding the set of the most unreliable bit positions, usually called the Frozen set. The complement set is then used to transport information bits.

In this thesis, we look at polar codes operating over the Additive White Gaussian Noise (AWGN) channel and we consider two topics: their construction, and their performance against Turbo and Low-Density Parity-Check (LDPC) codes. In the first part, we start with some construction algorithms that are frequently used, and then we show a new construction algorithm and demonstrate its performance. For the second part, we review the state-of-the-art turbo and LDPC architectures and we benchmark their performance against polar codes.

Contents

List of Figures

List of Tables

List of Abbreviations

5G 5th generation wireless systems.

AWGN Additive White Gaussian Noise.

B-DMC Binary Discrete Memoryless Channel.

BCJR Bahl-Cocke-Jelinek-Raviv.

BEC Binary Erasure Channel.

BER Bit Error Ratio.

BPSK Binary Phase Shift Keying.

CN Check Node.

CRC Cyclic Redundancy Check.

DEGA Density Evolution with Gaussian Approximation.

HARQ Hybrid Automatic Repeat Request.

IEEE Institute of Electrical and Electronics Engineers.

LDPC Low-Density Partiy Check.

List-SC List Successive Cancellation.

LLR Log-Likelihood Ratio.

LTE Long-Term Evolution.

M-DEGA Modified Density Evolution with Gaussian Approximation.

MAP Maximum A Posteriori.

ML Maximum-Likelihood.

PM Path Metric.

QPP Quadratic Permutation Polynomials.

SC Successive Cancellation.

SISO Soft-Input Soft-Output.

SNR Signal-to-Noise Ratio.

VN Variable Node.

1

Introduction

1.1 Channel Coding

The work of Shannon in his 1948 paper [2] marks a groundbreaking milestone in the field of communications; it is his work that established what we now know as Information Theory. One of his main results is that an error-free communication over a noisy channel is possible, if the information transmission rate is below or equal to a specific bound known as the *Channel Capacity*. Since then, enormous efforts were put into finding new transmission techniques starving to get closer and closer to the channel capacity.

Channel coding is one of the fundamental techniques that make such near-capacity operation possible. By introducing a structured redundancy at the transmitter (*encoding*), and exploiting it at the receiver (*decoding*), wide possibilities of error correction and detection can be achieved. The history of practical channel coding has a long run, maybe starting with Binary Golay codes [3] in 1949, Hamming codes [4] in 1950, Convolutional codes [5] in 1955, Reed-Solomon codes [6] in 1960, Low-Density Parity-Check (LDPC) codes [7] in 1960, then to the iterative decoding and capacity-approaching era of Turbo codes [8] in 1993, the rediscovery of LDPC codes [9] in 1996, and the introduction of Polar codes [1] in 2008. A very rich history indeed.

The goal of channel coding can be summarized as providing the receiver with the ability to recover the transmitted information correctly with very high probability.

1.2 Encoding and Decoding

The process of adding redundancy to the transmission is called Encoding. For linear codes, this can be written in the general form of

$$\mathbf{c} = \mathbf{uG}, \tag{1.1}$$

where $\mathbf{c}$ is the $1 \times N$ output (encoded) codeword, $\mathbf{u}$ is the $1 \times K$ information word, and $\mathbf{G}$ is the $K \times N$ *Generator* matrix. As the name suggests, the matrix $\mathbf{G}$ generates the code and so its properties. Usually, the code is defined over a *Galois Field* GF(q) and thus all the entries in (1.1) as well as the operations of addition and multiplication are defined over that field. We restrict ourselves here to $q = 2$, i.e. the binary field, which is the most widely used field in channel coding in specific, and in digital communications in general. Therefore throughout this thesis, we will be dealing with *bits*. Alternatively, the code can be described through its *Parity Check* matrix $\mathbf{H}$, which is defined as

$$\mathbf{GH}^T = \mathbf{0}. \tag{1.2}$$

Matrix $\mathbf{H}$ has a dimension of $(N-K) \times N$, where each row represents a parity check equation. A valid codeword satisfies all the parity check equations, i.e.

$$\mathbf{cH}^T = \mathbf{uGH}^T = \mathbf{0}. \tag{1.3}$$

The encoding can be carried out either in systematic way where the input bits appear directly at output codeword, or non-systematically where they do not. Systematic encoding is usually preferred since it allows for easier rate adaptation.

At the receiver side, the codeword is distorted due to noise and channel impacts, and the goal is to retrieve the original information word $\mathbf{u}$, or at least find out if it is corrupted or not. The process of doing that is Decoding. It encompasses error correction, error detection, or both of them. For example, turbo codes offer great error correction capabilities. On the other hand, Cyclic Redundancy Check (CRC) codes are bad with error correction, but are excellent at error detection. Error correction usually deals with sophisticated algorithms that in many cases suffer from a relatively high complexity.

1.3 The AWGN Channel

The Additive White Gaussian Noise (AWGN) channel is one of the most common approaches to model distortions from many random sources, such as thermal vibrations in matter, shot noise, radiations from objects and space, etc. By the central limit theorem, the sum of many of these random sources tend to be Gaussian distributed. The term additive comes from its additive nature in the sense that it is added on the top of our desired signal, and being white means that at different instants, it is uncorrelated. The model is shown in Figure 1.1 below.

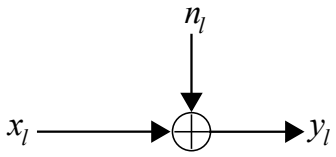

Figure 1.1: The AWGN channel.

At instant l, the channel output is given by

$$y_l = x_l + n_l \tag{1.4}$$

where x_l is the source output at instant l drawn according to some distribution, y_l is the channel output, and $n_l \sim \mathcal{N}(0, \sigma^2)$ is the zero mean AWGN with variance σ^2. The distribution of n_l is equal to

$$f_{n_l}(n_l) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{n_l^2}{2\sigma^2}\Big). \tag{1.5}$$

As we mentioned previously, one of the important results in information theory is the channel capacity. For the AWGN channel, this is given by [2]

$$C_{\text{AWGN}} = \frac{1}{2}\log\Big(1 + \frac{P_x}{\sigma^2}\Big), \tag{1.6}$$

where $P_x = \mathrm{E}\{|x_l|^2\}$ is the mean power of the source. Any transmission rate higher than the channel capacity will produce a non-zero probability of error, and therefore the information is irrecoverable. Our focus in this thesis will be on the transmission over the AWGN channel.

1.4 Soft Information and the L-value

One of the key-enablers for the high coding gain in the modern era of channel coding is the soft information processing. Instead of passing sliced bits (by *hard* decision) to the decoder which are either 0 or 1, pass the reliability of that decision as well. Since bits take only two values, this soft information can be calculated as follows

$$\ell_{x_l} = \frac{\Pr\{x_l = 0|y_l\}}{\Pr\{x_l = 1|y_l\}}, \tag{1.7}$$

where $\Pr\{x_l = i|y_l\}$ is the *posterior* probability of $x_l = i$ given the channel output y_l, and therefore ℓ_{x_l} is the posterior ratio of x_l. For numerical purposes, we usually take the log of that expression

$$L_{x_l} = \log\left(\frac{\Pr\{x_l = 0|y_l\}}{\Pr\{x_l = 1|y_l\}}\right), \tag{1.8}$$

here L_{x_l} is called the posterior L-value. Using *Bayes rule*, it can be written as

$$\begin{aligned} L_{x_l} &= \log\left(\frac{\Pr\{y_l|x_l = 0\}\Pr\{x_l = 0\}}{\Pr\{y_l|x_l = 1\}\Pr\{x_l = 1\}}\right), \\ &= \log\left(\frac{\Pr\{y_l|x_l = 0\}}{\Pr\{y_l|x_l = 1\}}\right) + \log\left(\frac{\Pr\{x_l = 0\}}{\Pr\{x_l = 1\}}\right), \\ &= \mathrm{LLR}_{x_l} + L_{x_l}^{\mathrm{prior}}, \end{aligned} \tag{1.9}$$

where the first term is the log-*likelihood* ratio (LLR) of the bit x_l, and the second term is the *prior* ratio of x_l. In many cases, the bits are uniformly distributed, i.e. $\Pr\{x_l = 0\} = \Pr\{x_l = 1\}$, and therefore

$$L_{x_l} = \mathrm{LLR}_{x_l}. \tag{1.10}$$

That is why it is common that the log posterior ratio and the LLR are both called the L-value, and it is understood from the context which probability it is representing. But as mentioned, in many cases they are equal.

It can be immediately seen that the sign of the L-value represents the hard decision of the bit. If it is positive then the bit is 0, otherwise it is 1. The magnitude of the L-value represent the reliability of that decision. Of our interest is the LLR when operating over the AWGN channel. Assuming Binary

Phase Shift Keying (BPSK) in which 0 is mapped to 1, and 1 is mapped to -1, the LLR is equal to

$$\begin{aligned} \mathrm{LLR}_{x_l} &= \log\left(\frac{\Pr\{y_l|x_l=0\}}{\Pr\{y_l|x_l=1\}}\right), \\ &= \log\left(\frac{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_l-1)^2}{2\sigma^2}\right)}{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_l+1)^2}{2\sigma^2}\right)}\right), \\ &= 2y_l/\sigma^2. \end{aligned} \tag{1.11}$$

This expression becomes more complicated when higher order mapping is used. Accounting for the reliability of decisions is of great importance. For example, consider the simple repetition coding scheme where each bit is transmitted twice. The receiver calculates the L-values and let us assume they are equal to -0.01 and 5.00. If we go for sign decision, the first output says the bit is 1, while the second output says the bit is 0. At this point, the decoder is confused as it cannot tell which output is correct. However, if we account for reliabilities then the decoder will choose the second output since it has much higher reliability and the bit will be decoded as 0.

1.5 Thesis Overview

In Chapter 2, we discuss the principle of polarization and how it can be used as a method for channel coding: polar codes. We look into the non-systematic nature of polar codes and into possible systematic encoding. We finish the chapter considering the decoding algorithms, namely the Successive Cancellation (SC) decoding and its extension of *list* decoding and CRC-aided list decoding. Chapter 3 deals with the construction aspect of polar codes over the AWGN channel. We show a new construction algorithm and compare it against some other constructions. We also consider the topic of polar codes' non-universality. In Chapter 4, we compare the performance of polar codes against turbo and LDPC codes. Finally, we provide some concluding remarks and outlook in Chapter 5.

2

Polar Codes

Polar codes where introduced by Arıkan in 2008 [1] and since then they gained a lot of attention. They are the first practical codes that are proven to achieve the channel capacity at infinite length. Moreover, their finite length performance under list successive cancellation decoding has been shown to be competitive against modern state-of-the-art schemes such as turbo and LDPC codes. They also enjoy low complexity operation encoding-wise and decoding-wise as well. All of these aspects make them a very interesting research topic with respect to both theory and practice.

2.1 Channel Polarization and Polar Codes

The principle of polar coding is based on channel *polarization*. This channel polarization or the polarization transform consists of two steps: channel combining and channel splitting. We provide here a high level description of polarization, since it is pretty technical.

Channel Combining

Assume we have N bits that we wish to transmit over a Binary Discrete Memoryless Channel (B-DMC) W, like the AWGN channel we considered in (1.11). Each transmission accounts for a use of W, or in other words transmitting each bit through a copy of W as shown in Figure 2.1. Channel combining is performed through the mapping $(u_0, u_1) \mapsto (u_0 \oplus u_1, u_1)$, i.e. we start by taking pairs of the input bits and combine them together to form a new 2-inputs 2-outputs channel W_2. This is shown in Figure 2.2. The next step is to combine

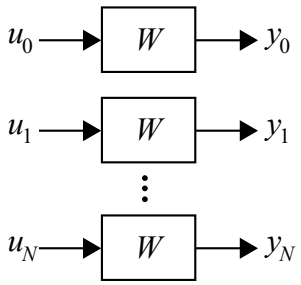

Figure 2.1: N copies of the channel W.

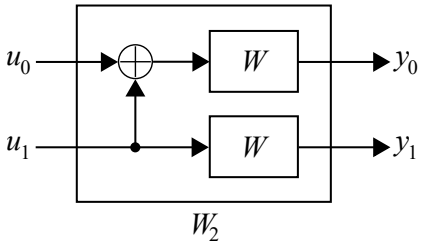

Figure 2.2: Combination of two input bits to form an equivalent channel W_2.

each of these new channels in a similar manner to produce a super channel W_4 (with appropriate selection of the bits) as shown in Figure 2.3. The process continues until the channel W_N is generated.

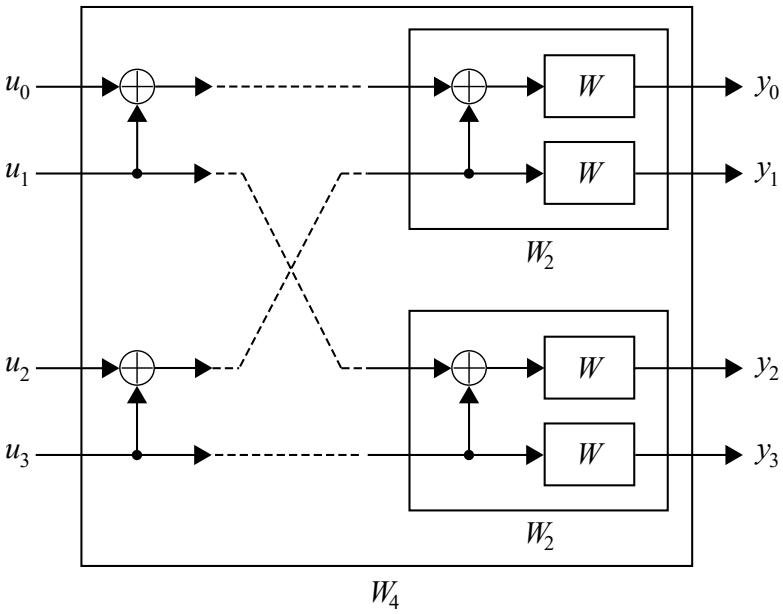

Figure 2.3: Generation of the new channel W_4.

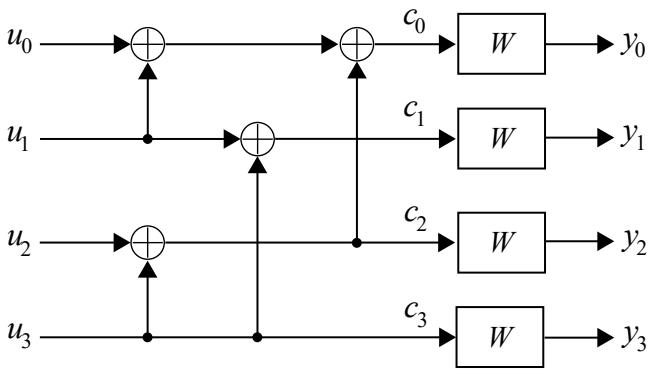

Figure 2.4: Channel splitting for $N = 4$.

Channel Splitting

In the second stage, the combined multi-input channels are split back into set a set of N binary-input channels. This is shown in Figure 2.4

The overall operations of channel combining and splitting can be done recursively using the kernel of length $N = 2$

$$\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.1}$$

For longer lengths, the transform is given by [1]

$$\mathbf{F}^{\otimes n} = \mathbf{F} \otimes \mathbf{F} \ldots \otimes \mathbf{F} \quad (n \text{ times}), \tag{2.2}$$

where $\mathbf{F}^{\otimes n}$ is the *Kronecker* product of $\mathbf{F}$ with itself n times, and $n = \log_2(N)$. This is the polarization transform and as can be seen it exists for lengths that are equal to powers of 2.

Channel Polarization

So how does it work and why is it called polarization? Let us take the transform of length 2 shown in Figure 2.5 below.

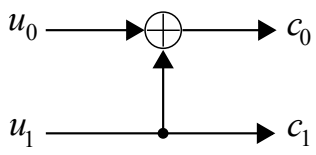

Figure 2.5: Polarization transform for $N = 2$.

The output is mapped to some modulation symbols and transmitted. At the receiver side, the decoder first attempts to decode u_0 using the two channel outputs. This is possible since this is effectively a truncated parity check code with u_0 not being present at the output. Now assume there is a genie that tells the decoder the true value of u_0 irrespectively of what was received. Knowing the value of u_0 from the genie, the decoder can then combine the two channel outputs coherently to generate a more reliable decision for u_1. For the measure of reliability, Arıkan used the *Bhattacharya* parameter $Z(W)$, which is a statistical distance that ranges from 0 (totally reliable) to 1 (totally unreliable). Denoting the Bhattacharya parameter for the upper channel by $Z(W^{'})$ and for the lower channel by $Z(W^{''})$, Arıkan showed that with the aid of such genie, the parameters evolve according to

$$\begin{aligned} &Z(W) \leq Z(W^{'}) \leq 2Z(W) - Z(W)^2, \\ &Z(W^{''}) = Z(W)^2. \end{aligned} \tag{2.3}$$

The terminology for upper and lower channels are shown in Figure 2.6.

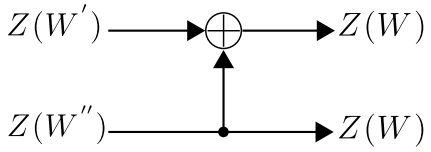

Figure 2.6: Polarization transform for $N = 2$.

The first line in (2.3) tells us that the reliability of the upper channel cannot be better than the original channel, it is at best the same, but in general it is degraded. The second line tells us that with the help of the genie, the reliability of the lower channel is higher than the original channel, i.e. it is upgraded. If we now consider the transformation for longer lengths such as that in Figure 2.4, then there will be more of these upgraded and degraded channels. The important result is that at infinite length, the bit channels will polarize in the sense that a set of the channels will be infinitely upgraded (totally reliable) and the other set will be infinitely degraded (totally unreliable). If we put the information bits only into the reliable set and put foreknown bits (usually zeros) into the unreliable set (*Frozen* set), then the channel capacity can be achieved. The process of doing that is called Polar coding.

2.2 Encoding

2.2.1 The Non-Systematic Nature of Polar Codes

As was mentioned in the previous section, the idea of the polar coding differs from conventional coding schemes in the sense that the code works by increasing the reliabilities of some set of the bit positions at the cost of reducing the reliabilities of the others (the frozen set). Due to that, the input to the polar encoder is the full word of dimension N consisting of the information bits placed at the reliable set together with foreknows bit placed at the frozen set, which leads to the following description of the encoding process

$$\mathbf{c} = \mathbf{uG} = \mathbf{uF}^{\otimes n}, \tag{2.4}$$

where $\mathbf{c}$ is the output codeword, $\mathbf{G}$ is the generator matrix which is equal to the polarization transform, and $\mathbf{u}$ is the input word. Notice that both $\mathbf{c}$ and $\mathbf{u}$ have the same dimension of N. This is due to the discussion above regarding the input word containing both the information bits and the frozen bits. For obvious reasons, the encoded codeword is not necessary systematic. The encoder for a polar code of length 4 is shown in Figure 2.7.

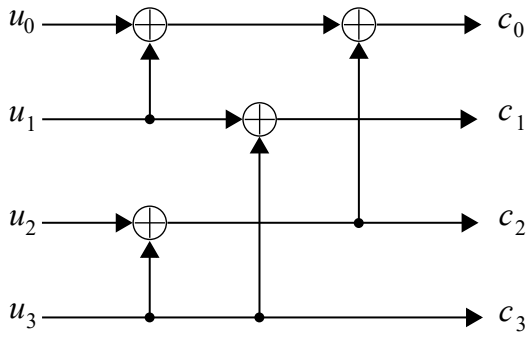

Figure 2.7: Polar encoder of length 4.

2.2.2 Systematic Encoding

It is possible to perform polar encoding in a systematic way. Everything works the same as before, the only difference is that we transform the input word $\mathbf{u}$ in such way that after applying the polarization transform, the information bits appear at the output codeword [10]. This can be performed by writing

the output codeword as follows

$$\mathbf{c} = \mathbf{u}_{\mathcal{F}^c}\mathbf{G}_{\mathcal{F}^c} + \mathbf{u}_{\mathcal{F}}\mathbf{G}_{\mathcal{F}}, \tag{2.5}$$

where $\mathbf{u}_{\mathcal{F}^c}$ is the input part corresponding to information bits, $\mathbf{u}_{\mathcal{F}}$ is the input part corresponding to the frozen bits, $\mathbf{G}_{\mathcal{F}^c}$ is the matrix formed by the rows of the polarization transform that corresponds to the information bits (reliable) positions, and similarly $\mathbf{G}_{\mathcal{F}}$ corresponds to the frozen set rows. Let us break $\mathbf{c}$ into two parts as well

$$\begin{aligned} \mathbf{c}_{\mathcal{F}^c} &= \mathbf{u}_{\mathcal{F}^c}\mathbf{G}_{\mathcal{F}^c\mathcal{F}^c} + \mathbf{u}_{\mathcal{F}}\mathbf{G}_{\mathcal{F}\mathcal{F}^c}, \\ \mathbf{c}_{\mathcal{F}} &= \mathbf{u}_{\mathcal{F}^c}\mathbf{G}_{\mathcal{F}^c\mathcal{F}} + \mathbf{u}_{\mathcal{F}}\mathbf{G}_{\mathcal{F}\mathcal{F}}, \end{aligned} \tag{2.6}$$

where $\mathbf{G}_{\mathcal{F}\mathcal{F}^c}$ is the submatrix formed by the intersection of the rows in $\mathcal{F}$ with columns in $\mathcal{F}^c$, and similarly are the other submatrices. Therefore if $\mathbf{c}_{\mathcal{F}^c}$ is the systematic part of the output codeword, the bits at the reliable positions must satisfy

$$\mathbf{u}_{\mathcal{F}^c} = (\mathbf{c}_{\mathcal{F}^c} - \mathbf{u}_{\mathcal{F}}\mathbf{G}_{\mathcal{F}\mathcal{F}^c})\mathbf{G}^{-1}_{\mathcal{F}^c\mathcal{F}^c}. \tag{2.7}$$

It turns out for $\mathbf{G}_{\mathcal{F}^c\mathcal{F}^c}$ to be invertible, the information bits must appear at the output codeword at the same positions as they appeared at the input [10]. This justifies why we used $\mathbf{c}_{\mathcal{F}^c}$ to refer for the systemic part of the output codeword in the first place. Therefore if the information bits are $\mathbf{b}$, we set $\mathbf{c}_{\mathcal{F}^c} = \mathbf{b}$ and perform (2.7). The result is inserted into the reliable positions of the polarization transform as before. At the receiver side after performing decoding, the decoded bits need to be detransformed in order to obtain the true information bits.

An interesting result is that the Bit Error Ratio (BER) performance of systematic polar codes is actually better ($\sim$ 0.3dB) than the non-systematic counterpart for the same constructed code. An explanation of this is given in [11], where they attributed that gain to the re-encoding process in (2.7). Further techniques for low-complexity operation of systematic polar codes can be found in [12].

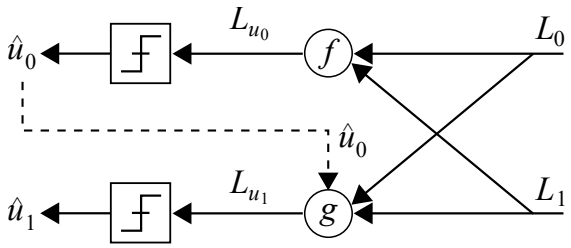

Figure 2.8: SC polar decoder of length 2.

2.2.3 Systematic or Non-Systematic

At this point, one might question whether it pays off to perform systemic encoding for the sake of the extra gain. We need to recall that going for the systematic approach requires extra calculations not only at the transmitter side, but at the receiver side as well (for detransformation). Most of the work and also what has been proposed for the standardization process in 5th generation wireless systems (5G) is the use of non-systematic polar codes. The reasons are mainly because on the one hand, it lacks the extra calculations that are required otherwise by the systematic approach, and on the other hand many of the Hybrid Automatic Repeat Request (HARQ) schemes for polar codes such as Incremental Freezing [13], does not impose any requirements on the polar code systematic-wise. Because of that and since the extra gain is not that substantial, it makes sense to go with non-systematic encoding.

2.3 Decoding

2.3.1 Successive Cancellation (SC)

The recursive structure of the polarization transform permits a very simple decoding scheme based on successive cancellation (SC). If we consider the polarization transform of length 2, we see that it is nothing more than a truncated parity check code, where the truncated part from the output is the bit u_0. Based on this, we can immediately see that the decoding can be performed by replacing the XOR and connection nodes by the probabilistic f and g nodes shown in Figure 2.8. For input LLRs L_a and L_b, those nodes calculate [14]

$$\begin{aligned} f(L_a, L_b) &= \log\left(\frac{e^{L_a+L_b}+1}{e^{L_a}+e^{L_b}}\right), \\ g(L_a, L_b, s) &= (-1)^{u_s} L_a + L_b, \end{aligned} \tag{2.8}$$

where u_s is termed the partial sum, which is the sum of the previously decoded bits that are participating in the current sum (or node g, from the decoder point of view). In this example, $u_s = u_0$ because there is no other bit that is participating in this sum (truncated parity check if you will).

Since u_0 is not present at the output then its L-value is zero and therefore we end up only with the extrinsic information from the parity bit and u_1. This is what node f calculates, and it can be done in a robust manner using the *Min-Sum* approximation, i.,e.

$$f(L_a, L_b) \approx \text{sign}(L_a)\text{sign}(L_b)\min\left(|L_a|, |L_b|\right). \tag{2.9}$$

Node g on the other hand has an L-value for u_1 and therefore it can add it up to the extrinsic information from the other bits. A critical point here, is that node g assumes that u_s is totally correct, meaning that it affects the calculation of the extrinsic information by only changing the sign of parity bit L-value. For polar codes, the first bit u_0 is usually frozen and so it is set to zero. Since the decoder knows the frozen set, it sets u_0 to zero as well and produces an LLR for u_1 equal to

$$L_{u_1} = g(L_0, L_1, u_0) = L_0 + L_1, \tag{2.10}$$

which improves the reliability of u_1, thus producing the coding gain.

For the remaining part of the thesis, we adopt the following terminology shown in Figure 2.9. The LLRs through the decoder are addressed through which stage (columns) s and which bit channel (rows) i they are at, with stage zero being the channel LLRs stage. This is denoted as $L_i^{(s)}$ in the figure. The sequence of decoding a polar code of length 4 is shown in Figures 2.10-2.13. If a specific bit is frozen, then we can directly set it to zero. The partial sum u_s can be calculated recursively as well. There is no specific way, it can be figured out directly from the polarization transform.

The SC decoder suffers from two major drawbacks, the first one is its suboptimality. It can be seen that the way the decoder proceeds is in a stage

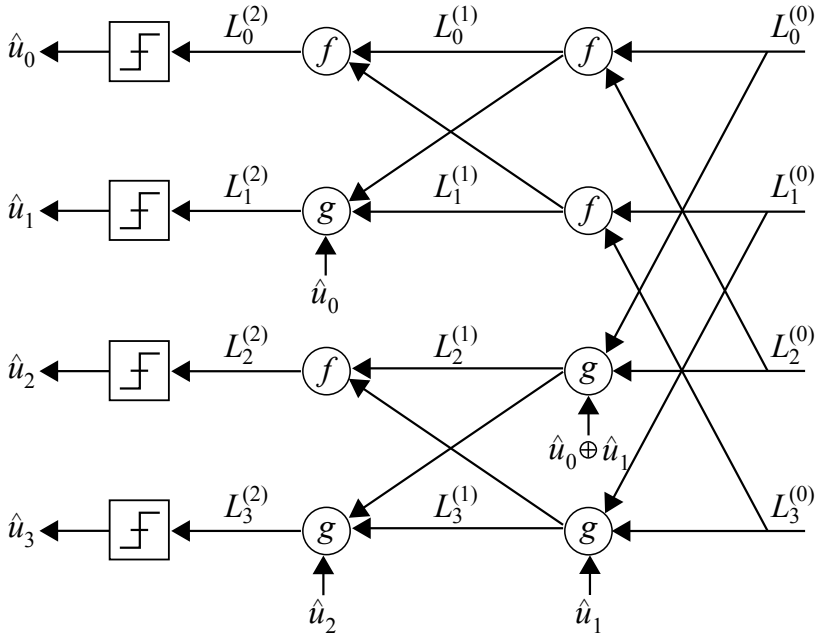

Figure 2.9: SC polar decoder of length 4.

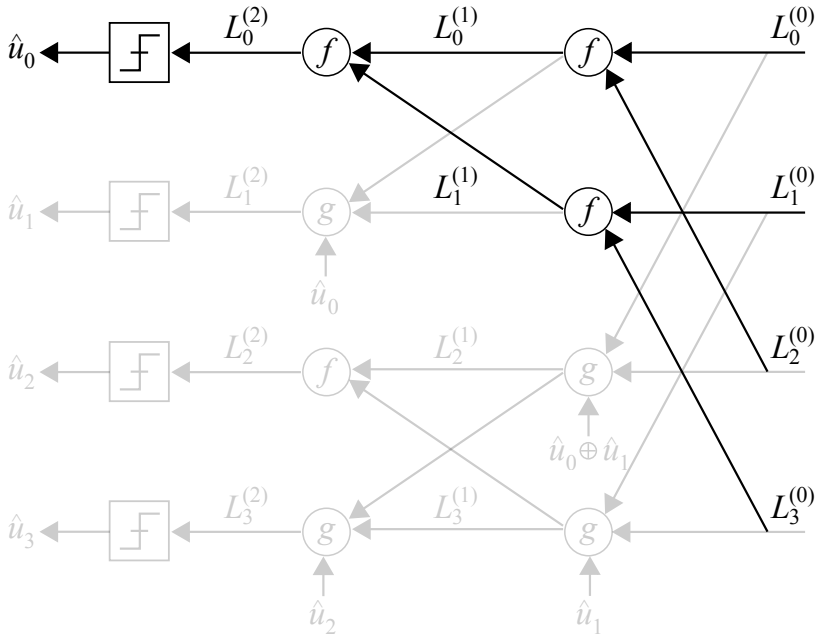

Figure 2.10: SC polar decoding - Step 1: u_0 can be decoded directly by passing the LLRs through appropriate f nodes.

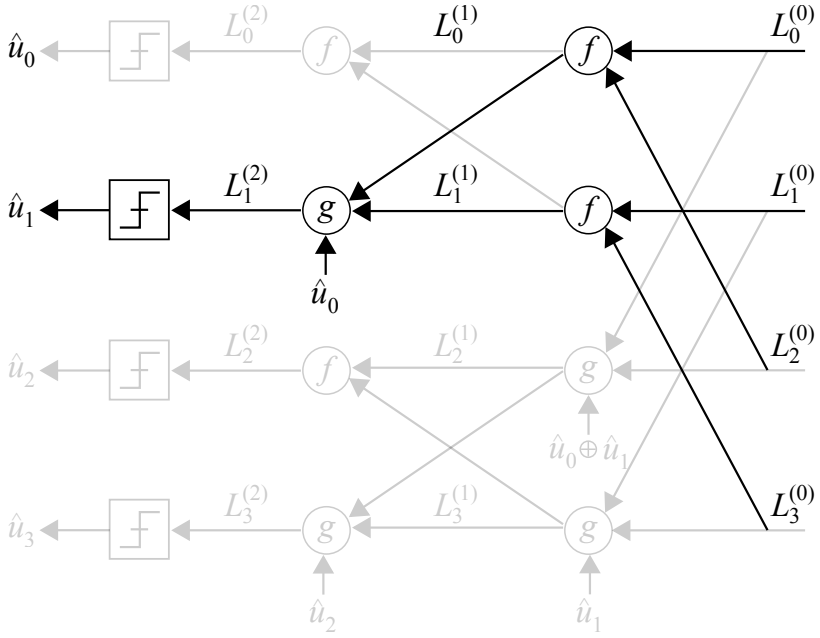

Figure 2.11: SC polar decoding - Step 2: u_1 can be decoded using the already calculated LLRs and the value of $\hat{u}_0$, which is probably frozen (i.e. zero).

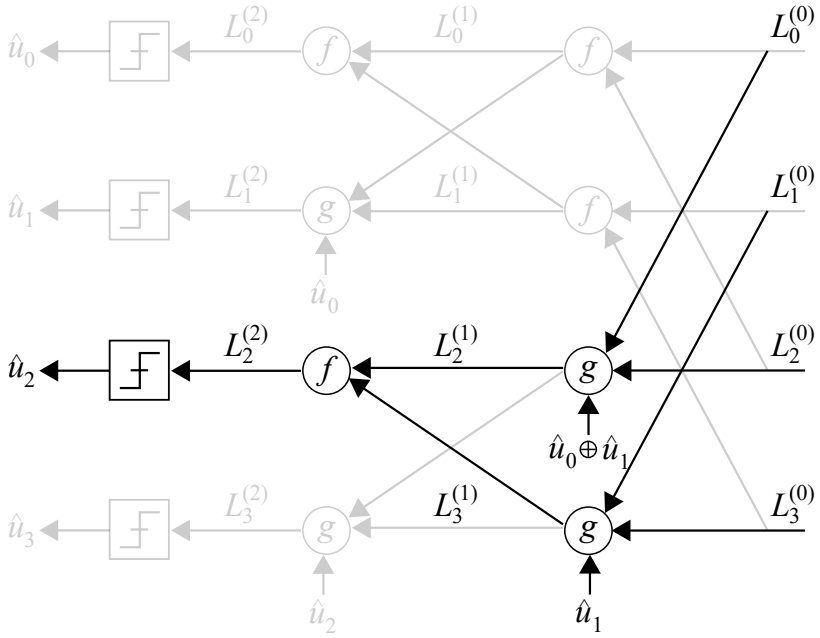

Figure 2.12: SC polar decoding - Step 3: u_3 can be decoded now using $\hat{u}_0$ and $\hat{u}_1$ and their sum $\hat{u}_0 \oplus \hat{u}_1$.

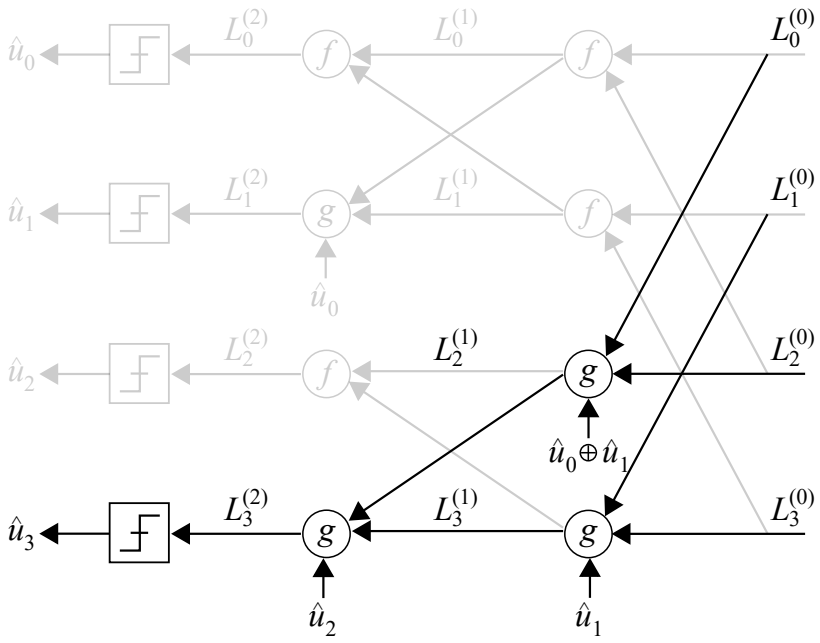

Figure 2.13: SC polar decoding - Step 4: Finally u_3 is decoded in a similar way.

by stage manner. At each stage, and at each node within the stage, two inputs participate in generating the output for the next stage. A near-optimum decoder would calculate/update the L-values across all the stages in a joint manner instead. For example, a joint decoder would attempt to solve a check equation that is defined by the whole sum of $u_0 \oplus u_1 \oplus u_2 \oplus u_3$ for the first output of the polar encoder instead of trying to solve it successively.

The second drawback is with respect to the decoding latency. In general, and especially for long codes, the SC decoder suffers from relatively high latencies that are inevitable due to its successive nature, which limits the possibilities for parallel implementations.

2.3.2 List Successive Cancellation (List-SC)

The first drawback regarding the suboptimality can be addressed by incorporating a list technique into the decoding process [15]. We start by considering the tree structure of the SC decoder. We have seen that the decoder starts by decoding the first bit (if not frozen), then having its value, it proceeds to the second bit, and similarly, it continues to the consecutive bits until reaching the last bit. This can be represented as the tree shown in Figure 2.14. For each decoded bit, we consider both possibilities of whether it has been decoded

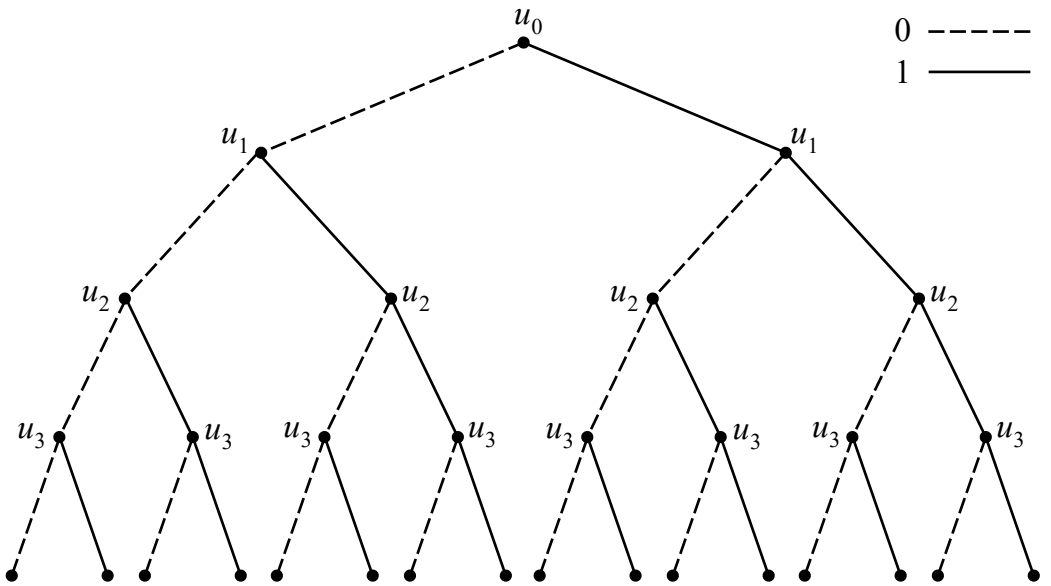

Figure 2.14: Tree representation of the SC decoder of length 4.

correctly or not. In other words, for each bit we take the two possibilities of being zero or one, irrespectively of its LLR. This effectively creates additional decoding branch for each bit. Unfortunately, doing this for each bit is not feasible, since the number of branches will grow up exponentially leading to the *Maximum-Likelihood* (ML) decoder complexity (and performance). In order to keep a reasonable complexity, we limit the number of branches at any level to a maximum of the *list size*. For example, if the list size is 4, then at any level, only 4 decoding-paths are allowed to survive. By looking at Figure 2.14, we see that the first two levels have at most 4 branches, however on the third level we have 8 branches and therefore some of them have to be dropped. For this, a *Path Metric* (PM) is developed based on the LLRs of the bits. These path metrics are tracked down through the tree, and when more branches are available, only those with the smallest path metrics are allowed to survive.

For LLR-based List-SC decoder, this path metric can be calculated as [16]

$$\mathrm{PM}_i[l] = \phi\big(\mathrm{PM}_{i-1}[l], L_i^{(s)}[l], \hat{u}_i[l]\big), \tag{2.11}$$

where $\mathrm{PM}_i[l]$ is the path metric at level i and decoding-path l, and $L_i^{(s)}[l]$ and $\hat{u}_i[l]$ are defined as before but now they have decoding-path dependency as well. The function $\phi()$ is given by

$$\phi\big(a, b, u\big) = a + \log\left(1 + e^{-(1-2u)b}\right), \tag{2.12}$$

which is approximated via [16]

$$\tilde{\phi}(a,b,u) = \begin{cases} a, & \text{if } u = \frac{1}{2}[1-\text{sign}(b)], \\ a+|b|, & \text{else.} \end{cases} \tag{2.13}$$

We continue with the creation and termination of branches until the last bit is reached. At the end, the decoding path with the lowest path metric is chosen.

An interesting observation here is that those decoding paths can be implemented as parallel SC decoders, i.e. if the list size is equal to four, then in principle we could have four SC decoders running in parallel each decoding one of the four paths, and after each level they can exchange information regarding the new branches. This is an important aspect since the extra gain obtained by the list decoder does not have a substantial impact on the decoding latency. It does however impact the memory requirements, since the whole decoded bits and the associated LLRs need to be copied from one path to another, roughly four times the memory for a list size of four, unless some efficient management/addressing techniques are applied.

Figure 2.15 shows the performance of the list decoder for different list sizes L. The simulation is carried out over the AWGN channel with BPSK signaling for code length $N = 4096$ and rate $R = 1/2$.

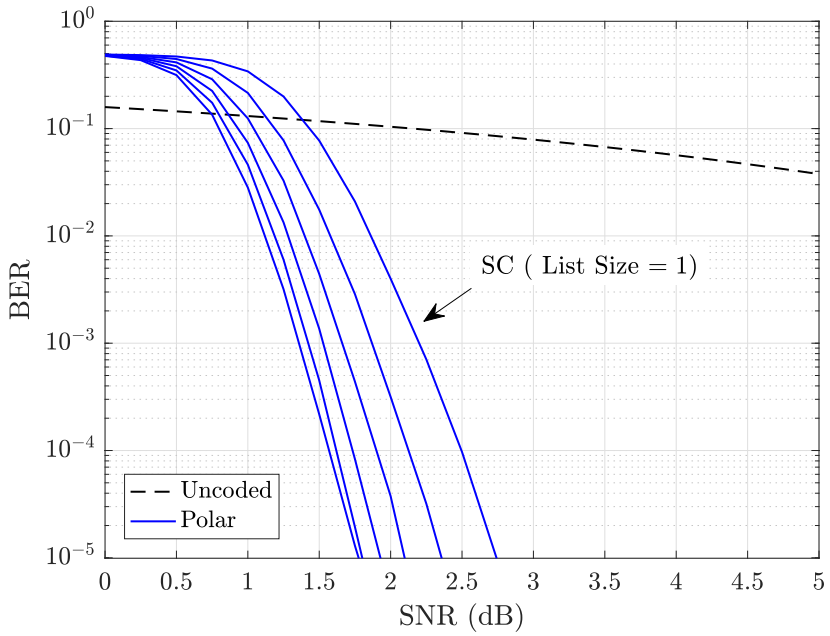

Figure 2.15: List-SC decoding for list sizes of $L = 32, 16, 8, 4, 2, 1$ from left to right, $N = 4096, R = 1/2$.

In [15], the authors observed that it can happen where the selected path at the final stage (i.e. the one with lowest path metric) is not the true path, even though the true path was actually in the final list of survivors. Based on this, a Cyclic Redundancy Check (CRC) extension was proposed for the list decoder. The idea is that at the final list, a CRC detection is performed on the surviving paths, and if one of them satisfies it, then it is the correct path. Otherwise, select the path with the lowest path metric and hope it has the fewest bit errors. Figure 2.16 shows the performance of the CRC-aided list decoder. The CRC used is based on the 16-bit *Long-Term Evolution* (LTE) polynomial [17], for list size $L = 8$, code length $N = 1024$, and rate $R = 1/2$ with BPSK over the AWGN channel.

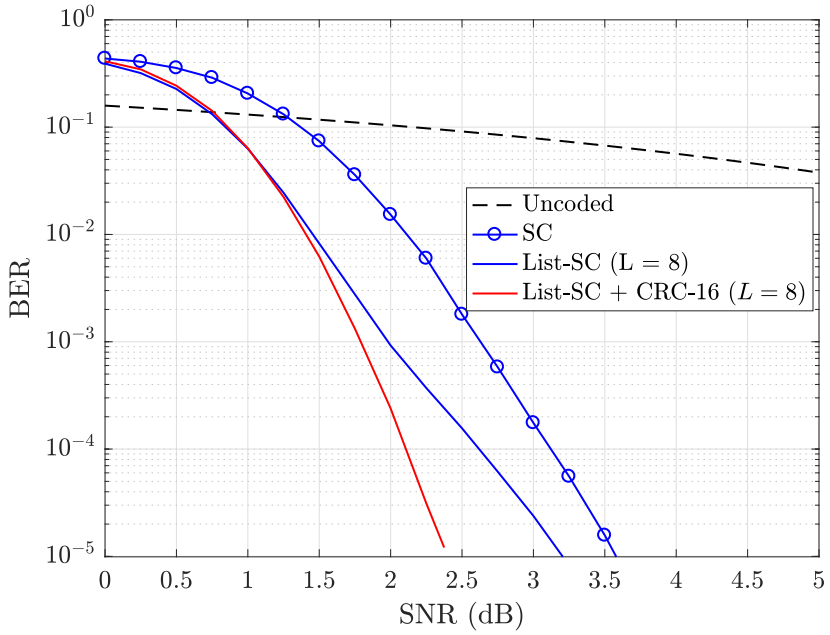

Figure 2.16: List-SC with 16-bit CRC for list size $L = 8$, $N = 1024$, and $R = 1/2$.

One can observe that the list curve is exhibiting a relative saturation towards the SC curve. This is not the case with Figure 2.15, at least in the same range of BER. We attribute this to the code construction, or in other words, there is potential for a better curve slope if a more sophisticated construction is used for this code length. The CRC-aided list decoder seems to be more robust against the construction imperfection and provides much better performance. This of course comes at the cost of running CRC detection on each of the surviving paths of the final list.

3

Construction of Polar Codes for the AWGN Channel

In this chapter we discuss some concepts regarding the construction of polar codes, specialized to the AWGN channel. Polar code construction deals with the problem of finding the set of the most unreliable bit positions, or commonly called the frozen set. These unreliable positions are fed with foreknown bits (zeros usually), while the other set is used to transport information. The receiver has to know the frozen set, otherwise it cannot tell which bits are used for information transport. Unfortunately this set is not only dependent on the code length and rate, but also on the channel conditions as well. Such dependency on the channel categorizes polar codes under the class of non-universal codes. However, it has been shown in [18] that the non-universality is a property of the suboptimal SC decoder and not the code itself. So effectively, we are constructing the code that makes the SC decoder perform the best. In the case of the AWGN channel, this corresponds to the dependency on the receiver *Signal-to-Noise Ratio* (SNR). This is actually a drawback, since the polar code needs to be reconstructed every time the SNR changes in order to guarantee optimum performance. Fortunately, the dependency is not extremely strict in the sense that a code constructed for a specific SNR can maintain a good performance over a range of neighboring SNRs as well. There have been some efforts into the design of universal polar codes such as [19], however it comes at the cost of lower decoding performance, limitations on the code length, or an increase in complexity. We consider here the construction for the SC decoder (non-universal) since it is the low-complexity approach at the moment.

3.1 The Problem of Optimum Construction

As mentioned already, constructing polar codes boils down to finding the set of the most reliable bit positions. One possibility of construction is through the tracking of Bhattacharyya parameter as was considered in Chapter 2. However, it has been shown that the parameters' updates are achieved with equality only for the Binary Erasure Channel (BEC) [1]. For any other channel, it is not exact and therefore an optimal tracking of the channels reliability is not possible with that method. One can attempt to go for the extreme case in which we try to track the whole probability densities of the LLRs through the decoder starting from the channel stage towards the decision stage, and based on those densities we can figure out a way to decide which "density" is more reliable, roughly speaking. Unfortunately, this is a very difficult task, mainly due to the non-linear transformation given by the node f operation in (2.7).

To demonstrate why it is difficult, consider the following example. Let us take the polar code of length 4 and assume an all-zero transmission with BPSK modulation over the AWGN channel of variance $\sigma^2 = 0.5$. Now, let us take a look at the distributions of LLRs of the first bit channel across all the stages (refer to Figure 2.9 for the terminology), i.e. the distribution of $L_0^{(0)}, L_0^{(1)}$, and $L_0^{(2)}$. This is shown in Figure 3.1 below.

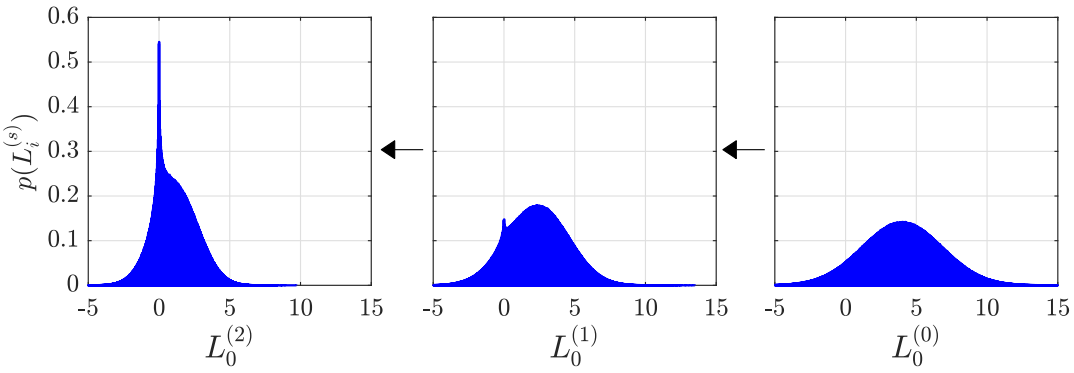

Figure 3.1: LLRs density evolution of the first bit channel through the stages for the SC decoder of length 4.

As can be seen, at the channel stage we start with a Gaussian distributed LLR $L_0^{(0)}$, and as the LLR passes through node f transformations, the Gaussianity is lost and we end up with some different densities. Therefore, it is clear that direct analytical tracking of the densities through the SC decoder is very difficult. For this, we need to find simpler methods. We consider next

some of the possible construction algorithms. Other constructions with varying performance and complexity can be found in [20–23].

3.2 Bhattacharya Parameter Construction

Perhaps the simplest way for constructing polar codes (for the AWGN channel) is to assume that the Bhattacharya parameter updates approximately hold. Following the terminology of Figure 2.9, these can be written as

$$\begin{aligned} Z(b_i^{(s)}) &\approx 2Z(b_i^{(s-1)}) - Z(b_i^{(s-1)})^2, && \text{if node f,} \\ Z(b_i^{(s)}) &= Z(b_i^{(s-1)})^2, && \text{if node g,} \end{aligned} \tag{3.1}$$

where $Z(b_i^{(s)})$ is the Bhattacharya parameter of ith bit channel at the sth stage. Notice that the parameter depends on the parameter at the previous stage of the ith bit channel only, and not on the other $(i + N/2)$ bit channel. The reason for that is any two inputs to any node are processed by the same sequence of nodes at the previous stages as depicted in Figure 2.9. For example, $L_0^{(1)}$ and $L_1^{(1)}$, which are inputs to node f of bit u_0, both result from node f calculation at the previous stage. Therefore, they have the same statistical properties assuming that at the channel stage all inputs have the same statistical properties, which is usually the case. Based on this, we pick the input that is on the same level as the current one in order to simplify notation.

The construction algorithm proceeds as follows: Start at the channel stage with the Bhattacharya parameter of the underlying AWGN channel. Assuming binary transmission with unity power and a noise variance of σ^2, it is given by

$$Z_{\text{AWGN}} = \exp\Big(-\frac{1}{2\sigma^2}\Big) = \exp\Big(-\frac{\text{SNR}}{2}\Big). \tag{3.2}$$

Next, evolve the parameter through the decoder nodes until reaching the last stage (decision stage), then the bit channels with the lowest parameters are the most reliable ones. Because of the similarity between the inputs within the stages statistical-wise, the evolution of the parameter can be done in a recursive way. In other words, at the channel stage, all the inputs have the same Bhattacharya parameter and therefore we only calculate it once. For the next stage, the whole first half of the nodes will have the same parameter, and the other whole half will have another one, therefore we only need to calculate

it twice, at the third stage each whole quarter will have new same parameter, etc. The construction algorithm is given in Algorithm 1 below.

Algorithm 1 Bhattacharya parameter construction

Input: Target SNR (TarSNR) in dB, Code length (N)
Output: Vector of final stage Bhattacharya parameters (Z)
Initialization: $Z[0] = \exp(-10^{\mathrm{TarSNR}/10}/2)$
for $i = 1$ to $\log_2(N)$ **do**
  $j = 2^{i-1}$
  for $k = 0$ to $j - 1$ **do**
    $z = Z[k]$
    $Z[k] = 2z - z^2$
    $Z[k + j] = z^2$
  end for
end for
return Z

The output vector is then sorted from lowest to highest, and the lowest K positions are used for information transport. Depending on the encoder/decoder implementation, bit reversal sorting might be needed as well.

3.3 Density Evolution with Gaussian Approximation (DEGA) Construction

The most popular construction for polar codes operating over the AWGN channel is Density Evolution with Gaussian Approximation (DEGA). As the name suggests, DEGA tries to evolve the densities of the LLRs through the decoder starting from the channel stage towards the decision stage. However, as we have seen in section 3.1, this is very difficult since even if we start with Gaussian densities at the channel stage, the resulting densities at the next stages are no longer Gaussians. DEGA relaxes this by assuming that they are approximately Gaussians, and therefore throughout the decoder it only has to track the mean and the variance of the LLRs. The LLR of the binary AWGN channel is equal to

$$L = 2y/\sigma^2, \tag{3.3}$$

where y is the channel output. Under the assumption of all-zero transmission, the mean value of the LLR above

$$m = \mathrm{E}\{L\} = 2/\sigma^2, \tag{3.4}$$

and the variance

$$\text{var}\{L\} = 4/\sigma^2 = 2m. \tag{3.5}$$

This relationship between the mean and the variance is assumed to hold at all stages [24], i.e. at any stage we have a Gaussian distributed LLR with mean $m_i^{(s)}$ and variance $2m_i^{(s)}$. Therefore we do not have to track the variance, but rather only the mean, and in case we need it, we just invoke the relationship above.

Following our terminology, DEGA performs the following updates [25]

$$m_i^{(s)} = \begin{cases} \phi^{-1}\left(1 - \left(1 - \phi\left(m_i^{(s-1)}\right)\right)^2\right), & \text{if node } f, \\ 2m_i^{(s-1)}, & \text{if node } g, \end{cases} \tag{3.6}$$

where the function $\phi()$ is given by the approximation [26]

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

and the inverse function $\phi^{-1}()$ might be approximated numerically with a piecewise function. The DEGA construction is given in Algorithm 2.

Algorithm 2 DEGA construction

Input: Target SNR (TarSNR) in dB, Code length (N)
Output: Vector of final stage LLRs mean values (M)
Initialization: $M[0] = 2 \times 10^{\text{TarSNR}/10}$
1: **for** $i = 1$ to $\log_2(N)$ **do**
2: $j = 2^{i-1}$
3: **for** $k = 0$ to $j - 1$ **do**
4: $m = M[k]$
5: $M[k] = \phi^{-1}\left(1 - (1 - \phi(m))^2\right)$
6: $M[k + j] = 2m$
7: **end for**
8: **end for**
9: **return** M

Here the larger is the mean of the LLR, the higher is the reliability of the bit channel. The same discussion from last section regarding the sorting applies here as well.

3.4 New Construction and the Modified DEGA

In this section, we show a new construction algorithm based on our work in [27]. The idea here is to characterize the reliability of the bit channels using the probability of bit error. Since it is dealing with the probability of error, this method gives us a performance analysis of the bit channels as well. For the remaining part, we assume the inputs to the encoder are all zeros, and the transmission is carried out using BPSK. Consider the decoding of the bit u_0 in Figure 2.10. It will be decoded correctly if its LLR $L_0^{(2)}$ is positive. This is satisfied if and only if the two input LLRs $L_0^{(1)}$ and $L_1^{(1)}$ to its node f, are both positive or both negative. In either case, the output will be positive and u_0 is decoded correctly. We also notice that the two inputs to any node are processed by the same sequence of nodes in the previous stages, and therefore their statistical properties are the same. Denoting $P_C\{b_i^{(s)}\}$ as the probability of correct decision of the bit channel i at stage s, and similarly $P_E\{b_i^{(s)}\}$ as the probability of error, the probability of decoding u_0 correctly can be calculated as

$$P_C\{b_0^{(2)}\} = P_C\{b_0^{(1)}\}P_C\{b_1^{(1)}\} + P_E\{b_0^{(1)}\}P_E\{b_1^{(1)}\}, \tag{3.7}$$

i.e. u_0 will be decoded correctly if the inputs to its node f are both correct or both incorrect. Since the two inputs have the same statistical properties as discussed above, then

$$P_C\{b_0^{(2)}\} = P_C\{b_0^{(1)}\}^2 + P_E\{b_0^{(1)}\}^2. \tag{3.8}$$

Under our assumption of all-zero transmission, we now notice that the output of the channel $b_0^{(1)}$ will be positive if and only if the two inputs to its corresponding node f, are both positive or both negative, which is the same case considered above. Based on this, we generalize (3.8) to

$$\begin{aligned} P_C\{b_i^{(s)}\} &= P_C\{b_i^{(s-1)}\}^2 + P_E\{b_i^{(s-1)}\}^2, \\ &= (1 - P_E\{b_i^{(s-1)}\})^2 + P_E\{b_i^{(s-1)}\}^2, \\ &= 1 - 2P_E\{b_i^{(s-1)}\} + 2P_E\{b_i^{(s-1)}\}^2, \end{aligned} \tag{3.9}$$

and the probability of error is then

$$\begin{aligned} P_E\{b_i^{(s)}\} &= 1 - P_C\{b_i^{(s)}\}, \\ &= 2P_E\{b_i^{(s-1)}\} - 2P_E\{b_i^{(s-1)}\}^2, \\ &= 2P_E\{b_i^{(s-1)}\}(1 - P_E\{b_i^{(s-1)}\}). \end{aligned} \tag{3.10}$$

We consider now the decoding of bit u_1. Under the all-zero transmission, the output of its node g is equal to

$$L_1^{(2)} = L_0^{(1)} + L_1^{(1)}. \tag{3.11}$$

Unfortunately, this requires invoking the underlying distribution of the LLRs, which is no longer Gaussian as mentioned in the previous sections. At this point, we drop optimality, and similarly to DEGA, we approximate them with Gaussian densities. If the partial sum $u_s = u_0$ is correct, the parameter Z in (3.2) evolves to Z^2 when the bit channel is transformed by a node g according to (2.3), i.e.

$$Z(b_i^{(s)}) = [Z(b_i^{(s-1)})]^2 = \exp(-\mathrm{SNR}_i^{(s-1)}). \tag{3.12}$$

In other words, under correct feedback, node g (in the Gaussian sense) improves the SNR of the transformed bit channel by a factor of two. We now proceed and calculate the probability of error based on the SNRs. For AWGN channels, the tail probability is given by the Q-function

$$P_E\{b_i^{(s)}\} = Q\left(\sqrt{\mathrm{SNR}_i^{(s)}}\right), \tag{3.13}$$

from which the calculation of SNR follows directly as

$$\mathrm{SNR}_i^{(s)} = Q^{-1}\left(P_E\{b_i^{(s)}\}\right)^2, \tag{3.14}$$

where the $Q^{-1}()$ is the inverse Q-function. As shown in (3.12), the SNR will double, and therefore the probability of error at the next stage after passing through node g is given by

$$P_E\{b_i^{(s)}\} = Q\left(\sqrt{2\mathrm{SNR}_i^{(s-1)}}\right), \tag{3.15}$$

which can be expanded using (3.14) into

$$P_E\{b_i^{(s)}\} = Q\left(\sqrt{2}\; Q^{-1}\left(P_E\{b_i^{(s-1)}\}\right)\right). \tag{3.16}$$

Summarizing, the bit error probability of the ith bit channel evolves from stage $(s-1)$ to (s) according to

$$P_E\{b_i^{(s)}\} = \begin{cases} 2P_E\{b_i^{(s-1)}\}(1 - P_E\{b_i^{(s-1)}\}), & \text{if node } f, \\ Q\left(\sqrt{2}\; Q^{-1}\left(P_E\{b_i^{(s-1)}\}\right)\right), & \text{if node } g. \end{cases} \tag{3.17}$$

The construction algorithm is given in Algorithm 3 below.

Algorithm 3 The new construction algorithm

Input: Target SNR (TarSNR) in dB, Code length (N)
Output: Vector of final stage bit error probabilities (P)
Initialization: $P[0] = Q(\sqrt{10^{\text{TarSNR}/10}})$
1: **for** $i = 1$ to $\log_2(N)$ **do**
2: $j = 2^{i-1}$
3: **for** $k = 0$ to $j - 1$ **do**
4: $p = P[k]$
5: $P[k] = 2p(1-p)$
6: $P[k+j] = Q\left(\sqrt{2}\; Q^{-1}\left(p\right)\right)$
7: **end for**
8: **end for**
9: **return** P

Since we have direct access to the probability of bit error, let us check how good is the approximation given by the algorithm. We run a Monte-Carlo simulation with 10^9 repetitions for a polar code of length 16, and estimate the average BER of the individual bit channels (positions). We make sure that all the fedback bits are correct, since the construction algorithm assumes the best-case scenario in which all the fedback bits are correct. The results are then compared against our algorithm. This is shown in Fig. 3.2

The SNR is set to 1 dB here. We can see that the algorithm did very well in approximating the bit error probabilities. The results might vary depending on the SNR and the code length. It might be needed to check the precision of the algorithm for much longer lengths, however performing such simulations would be computationally complex since we are trying to obtain BER estimates for each bit channel.

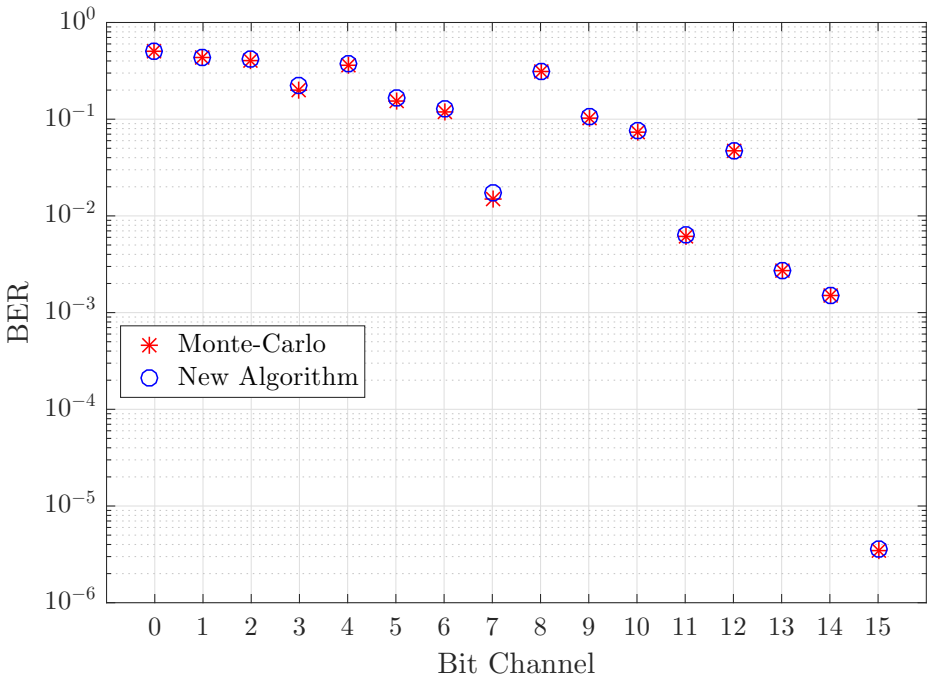

Figure 3.2: Estimated BER using Monte-Carlo simulation vs. bit error probabilities approximated by the new algorithm.

For the binary AWGN channel, the mean value of the LLR is related to the SNR through

$$\mathrm{SNR}_i^{(s)} = \frac{(m_i^{(s)})^2}{2m_i^{(s)}} = \frac{m_i^{(s)}}{2}. \tag{3.18}$$

Using (3.14), we get

$$m_i^{(s)} = 2Q^{-1}\left(P_E\{b_i^{(s)}\}\right)^2. \tag{3.19}$$

We've shown in the probability analysis that passing through node f evolves the error probability as in (3.16), i.e.

$$m_i^{(s)} = 2Q^{-1}\left(2P_E\{b_i^{(s-1)}\}(1 - P_E\{b_i^{(s-1)}\})\right)^2, \tag{3.20}$$

with

$$P_E\{b_i^{(s-1)}\} = Q\left(\sqrt{\frac{m_i^{(s-1)}}{2}}\right). \tag{3.21}$$

The result is the following

$$m_i^{(s)} = \begin{cases} 2Q^{-1}\left(2P_E\{b_i^{(s-1)}\}(1-P_E\{b_i^{(s-1)}\})\right)^2, & \text{if node } f, \\ 2m_i^{(s-1)}, & \text{if node } g. \end{cases} \tag{3.22}$$

We call this the Modified-DEGA (M-DEGA) and different to the original DEGA, the node f update is formulated in terms of the $Q()$ and $Q^{-1}()$ functions. Such formulation should provide better accuracy than DEGA, due to the vast methods out there for the the calculation of the Q function and its inverse in an efficient and accurate manner.

Finally let us see how the four algorithms we considered in the previous three sections stand off against each other. Figure 3.3 shows the performance of the different algorithms for code length $N = 4096$ and rate $R = 1/2$ signaled over the AWGN channel with BPSK. It can be observed that the Bhattacharya construction performs the worst. The other constructions perform similar to each other, with slight gain for the new algorithms compared to DEGA at the low SNR regime.

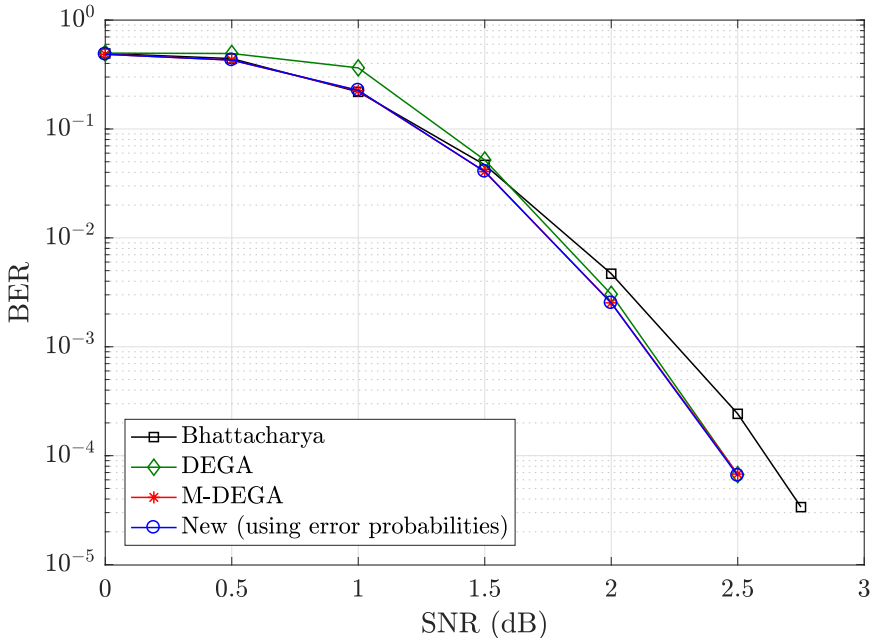

Figure 3.3: Performance of the different construction algorithms for $N = 4096$ and $R = 1/2$.

3.5 Polar Codes "Non-Universality"

Let us end this chapter by showing the effect of non-universality on the performance of polar codes. Recall that at the start, we mentioned that this property turned out to be a property of the SC decoder and not of the polar code itself. So for the following results, it should not be taken as performance measure of the code itself, but rather a limitation of the SC decoder.

In Figure 3.4 the probability of bit error for the 76th and 113th bit channels are obtained using the new construction algorithm for a code length of $N = 128$. What should be observed here is that as long as the SNR is less than 3dB, the 113th bit position is more reliable than the 76th position. For SNRs higher than 3dB, the converse is true and the 76th bit position is much more reliable and should be chosen for information transport. If we plot the whole 128 probability of errors then we will see a lot of these intersections across almost all the bit channels. This is the reason why the design (target) SNR plays a crucial role, and this why the polar code needs to be reconstructed if the current receiver SNR is far away from the design SNR. Remember that this is a limitation of the SC decoder, meaning that if a ML decoder is employed, the non-universal behavior would disappear [28].

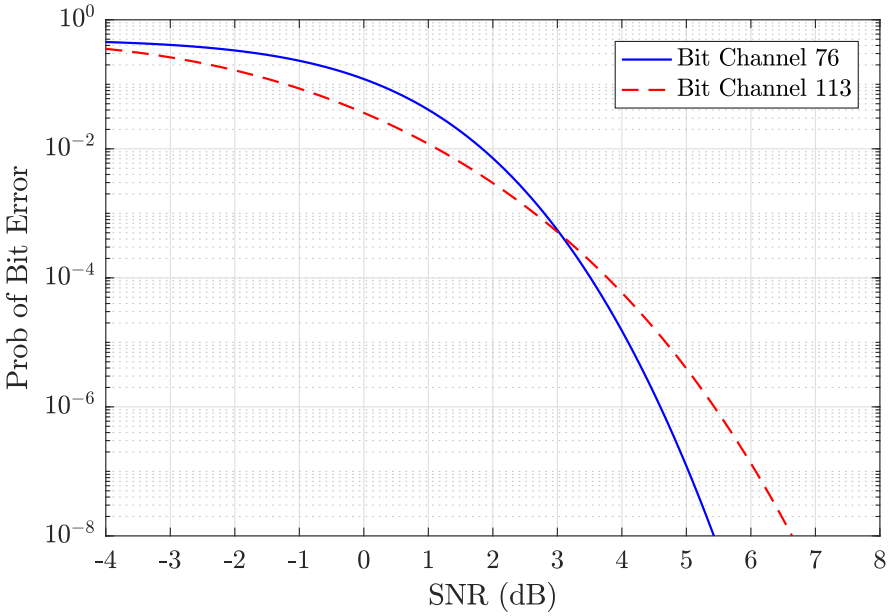

Figure 3.4: Probability of bit error for the 76th and 113th bit positions for a polar code of length $N = 128$, obtained using the new construction algorithm.

Next, let us consider the impact of the target SNR on a fully operational long polar code. In Figure 3.5, the receiver is operating at a fixed SNR of 3dB. We try to transmit with different polar codes of length $N = 2048$ and rate $R = 1/2$ constructed at different target SNRs and plot the resulting average BER for each setting. There is some good news, and that is the polar code maintains a good performance not only at the target SNR but also on a considerable range around it. So as long as the receiver SNR does not change a lot, the polar code will still give good performance. The codes were constructed using the new algorithm and so this figure also shows its good performance in which the curve minimum is obtained when the target SNR almost matches the operating SNR of 3dB.

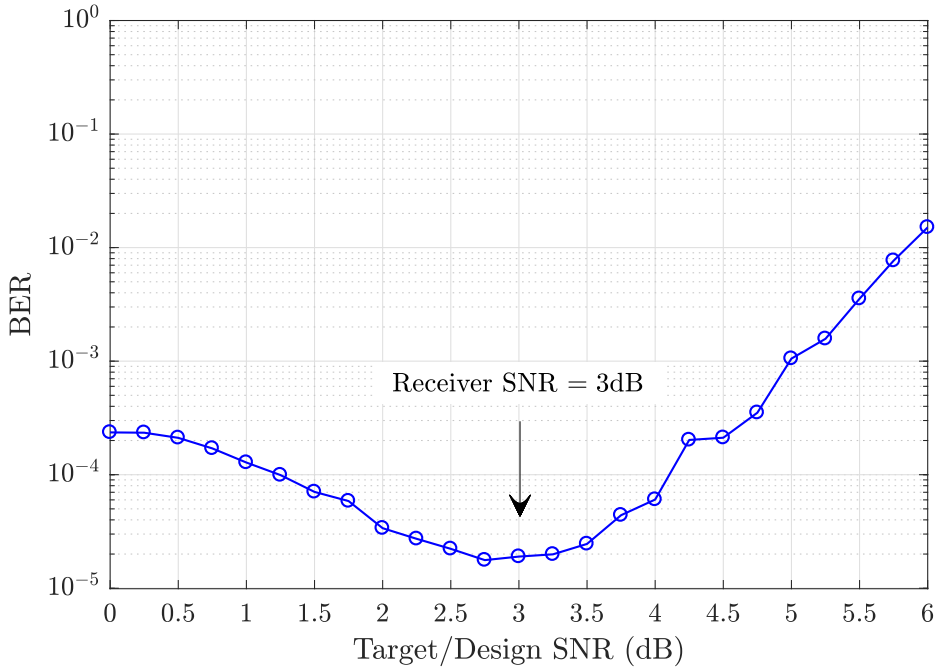

Figure 3.5: Performance of the constructions at different Target SNRs for a receiver operating at 3dB ($N = 2048$ and $R = 1/2$).

4

Performance of Polar Codes versus Turbo and LDPC Codes

In this chapter, we take a look at the performance of polar codes against two of the most popular capacity-approaching codes: Turbo and LDPC codes. The content of this chapter is mainly based on our work in [29], with extensions regarding the scenario of very short code length and the use of CRC-aided list decoding for polar codes. We start by reviewing state-of-the-art architectures for those coding schemes, and finish with a BER comparison against polar codes.

4.1 Turbo Codes

Turbo codes, introduced in 1993 [8], represent a major breakthrough in the field of channel coding. They represent a class of codes that can perform very close to the capacity limit. The iterative turbo decoding scheme itself paved the way for many of current modern near-optimum iterative receivers. In its common form, turbo coding is performed by a parallel concatenation of two recursive convolutional encoders separated by an *interleaver*. The task is then to design the code polynomials for the individual encoders, and to use an appropriate *random-like* interleaver (the reason for this will become clear later). At the receiving side, two decoders are used, each one decodes the streams of the corresponding encoder, and by exchanging probabilistic information, the two decoders can iteratively help each other in a manner similar to a turbo engine, hence the name.

4.1.1 Encoding

Due to the convolutional nature, the encoding can be carried out efficiently using a combination of memory elements and XOR gates. Figure 4.1 shows the turbo encoder used in LTE [17], where a *Quadratic Permutation Polynomials* (QPP) interleaver is used [30]. The outputs of the first encoder are a *systematic* stream u_l and a *parity* stream $p_l^{(1)}$, while the second encoder generates a parity stream $p_l^{(2)}$ only. This makes it a rate 1/3 code.

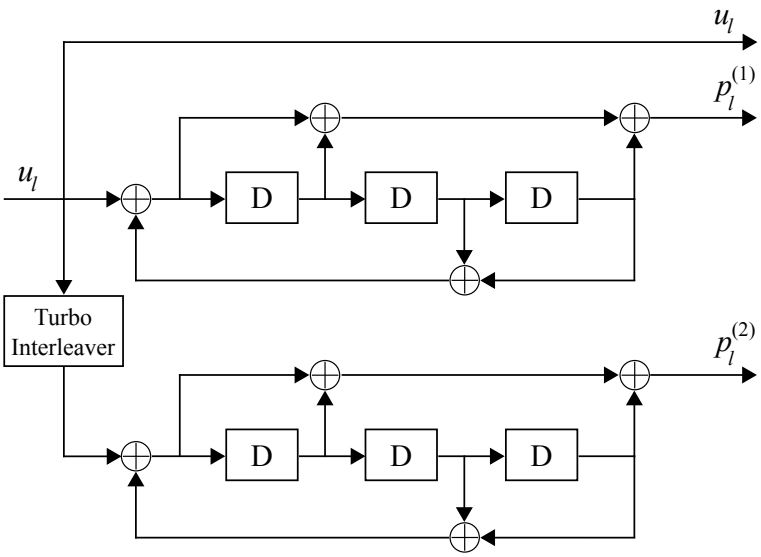

Figure 4.1: LTE rate 1/3 turbo encoder [17].

Such encoding scheme induces a *state* structure in the code based on how the memory elements are used (and the memory length). The knowledge of the *state transitions* is crucial later on for the decoder. Also, the starting and the ending state of both encoders need to be known at the receiver, otherwise performance loss will occur. In LTE, this is handled via *trellis termination*. The design of the interleaver is important as well and it needs to provide random-like behavior rendering the original and interleaved streams as if they were uncorrelated. This is essential for the turbo gain, since it will be unlikely that the original stream and its interleaved counterpart undergo the same encoding, transmission, and/or decoding conditions.

One can observe that the encoding is of low complexity, and combined with the high clock speeds of modern architectures, encoding latency is not a problem.

4.1.2 Decoding

The turbo decoder consists of two *Soft-Input Soft-Output* (SISO) decoders. The soft decoder is basically a bit-wise *Maximum A Posteriori* (MAP) decoder, which is implemented efficiently using the *BCJR* algorithm [31]. Each decoder acts on the streams of the corresponding encoder.

Let us first consider the SISO (BCJR) decoder assuming only one encoder is employed. For information bit u_l at time l, received codeword $\mathbf{y}$, the L-value of u_l is given by

$$L_{u_l} = \log\left(\frac{P\{u_l = 0|\mathbf{y}\}}{P\{u_l = 1|\mathbf{y}\}}\right). \tag{4.1}$$

Due to the *Trellis* structure of the convolutional code, these probabilities can be written as [14]

$$L_{u_l} = \log\left(\frac{\sum_{U_0} P\{s_{l-1} = s', s_l = s, \mathbf{y}\}}{\sum_{U_1} P\{s_{l-1} = s', s_l = s, \mathbf{y}\}}\right), \tag{4.2}$$

where s_l is the state at time l, U_0 is the set of pairs (s', s) for the state transition $s' \to s$ when $u_l = 0$, and U_1 is the set of pairs (s', s) for the transition when $u_l = 1$. Using the BCJR algorithm, these probabilities can be factorized as

$$P\{s_{l-1} = s', s_l = s, \mathbf{y}\} = \alpha_{l-1}(s')\gamma_l(s', s)\beta_l(s). \tag{4.3}$$

In the log-domain, the final expression for the L-value is given by [14]

$$\begin{aligned} L_{u_l} = &\max_{U_0}{}^*[\alpha_{l-1}(s') + \gamma_l(s', s) + \beta_l(s)] \\ &-\max_{U_1}{}^*[\alpha_{l-1}(s') + \gamma_l(s', s) + \beta_l(s)], \end{aligned} \tag{4.4}$$

The $\max^*$ function is equal to

$$\max{}^*(a, b) = \max(a, b) + \log(1 + e^{-|a-b|}), \tag{4.5}$$

and the probabilities α_l and β_l are calculated using the forward and backward recursions [31]

$$\begin{aligned} \alpha_l(s) &= \max_{s'}{}^*[\alpha_{l-1}(s') + \gamma_l(s', s)], \\ \beta_{l-1}(s') &= \max_{s}{}^*[\beta_l(s) + \gamma_l(s', s)], \end{aligned} \tag{4.6}$$

where $\gamma_l(s', s)$ is the *Branch Metric*, in which for the AWGN channel with variance σ^2 is equal to

$$\gamma_l(s', s) = \frac{1}{2}\big(u_l L_{u_l}^{\text{prior}} - \|\mathbf{y}_l - \mathbf{c}_l\|_2^2 / \sigma^2\big). \tag{4.7}$$

Here, $L_{u_l}^{\text{prior}}$ is the prior information about the bit u_l, $\mathbf{y}_l = [y_{u_l}\ y_{p_l}]$ is the channel output of the systematic and the parity stream at time l, and $\mathbf{c}_l = [u_l\ p_l]$ are the generated systematic and parity bits according to the state transition $s' \to s$ at time l. This metric can be shortened as

$$\gamma_l(s', s) = u_l L_{u_l}^{\text{prior}}/2 + (u_l y_{u_l} + p_l y_{p_l})/\sigma^2, \tag{4.8}$$

where the other terms of the ℓ_2-norm were dropped because they are independent of U_0 and U_1 and therefore will cancel each other during the subtraction in (4.4). After some manipulation, (4.4) can be written as

$$\begin{aligned} L_{u_l} = 2y_{u_l}/\sigma^2 + L_{u_l}^{\text{prior}} &+ \max_{U_0}{}^{*}[\alpha_{l-1}(s') + p_l y_{p_l}/\sigma^2 + \beta_l(s)] \\ &- \max_{U_1}{}^{*}[\alpha_{l-1}(s') + p_l y_{p_l}/\sigma^2 + \beta_l(s)]. \end{aligned} \tag{4.9}$$

The first term is basically the channel LLR of the bit u_l, the second term is prior information, and the third term is the new information obtained by the decoding process

$$L_{u_l} = L_{u_l}^{\text{channel}} + L_{u_l}^{\text{prior}} + L_{u_l}^{\text{new}}. \tag{4.10}$$

Given the operation of the BCJR algorithm, the turbo decoder operates as follows: The systematic stream and the first parity stream are fed to the first decoder, while an interleaved version of the systematic stream and the second parity stream are fed to the second one. The first decoder starts, and instead of generating a final LLR, it generates a cleared up version, called *extrinsic* information L^e. The extrinsic information is exactly equal to $L_{u_l}^{\text{new}}$ in (4.10). This is interleaved $\pi(l)$, and sent to the second decoder. The second decoder makes use of this extra information by considering it as prior, therefore the total L-value at the second decoder (2) is given by

$$L_{u_l}^{(2)} = L_{u_{\pi(l)}}^{\text{channel}} + L_{u_{\pi(l)}}^{e(1\to 2)} + L_{u_l}^{\text{new}(2)}. \tag{4.11}$$

This leads to more reliable decoding compared to the case where it does not

have the additional information from the first decoder. In a similar manner, it generates extrinsic information for the first decoder, and instead of interleaving, it performs deinterleaving $\pi^{-1}(l)$, and at this point, an iteration is completed.

On the next iteration, the first decoder starts same as before, but now it has extrinsic information from the second decoder, i.e.

$$L_{u_l}^{(1)} = L_{u_l}^{\text{channel}} + L_{u_{\pi^{-1}(l)}}^{e(2\to1)} + L_{u_l}^{\text{new}(1)}, \tag{4.12}$$

and therefore a more reliable output is calculated. Notice that the new prior information obtained from the other decoder does not only enter in the total L-value in (4.10), but also in the path metric in (4.7). This is why the new information generated by each decoder is improved over the iterations. The decoding continues until a stopping criterion is satisfied, or the maximum number of iterations has been reached. An illustration of the decoder is shown in Figure 4.2 below.

In (4.5), an approximation can be made by neglecting the log term,

$$\max^*(a, b) = \max(a, b). \tag{4.13}$$

The decoding algorithm using this substitution is called *Max-Log-MAP* [32], offering lower complexity at the cost of some performance loss.

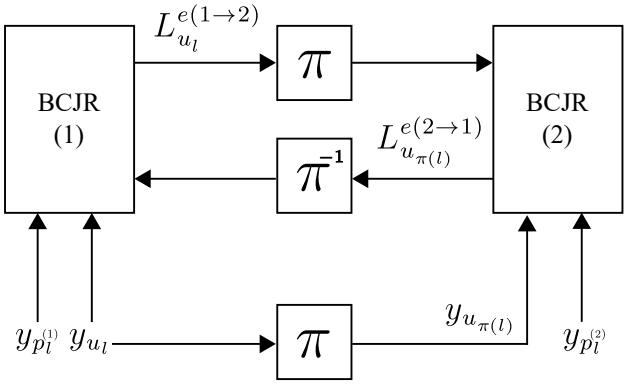

Figure 4.2: The turbo decoder.

4.2 Low-Density Parity-Check (LDPC) Codes

LDPC codes were first proposed by Gallager in 1960 [7]. At that time, they were considered too complex for practical implementation. In 1996 [9], LDPC codes were rediscovered and their capacity-approaching performance was proven. Shortly in [33], the decoding was realized efficiently using the iterative *Sum-Product* algorithm. As the name implies, LDPC codes are block codes with a *sparse* parity check matrix. Such sparsity facilitates low complexity encoding and decoding. An example code is the following

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \tag{4.14}$$

One cannot see the sparsity at this very short length, but we give it here to explain some of the main principles. In general, the main parameters of a parity check matrix are the column and row weights, and the *Girth*. The weight of a column is the number of ones in that column. Similarly, the row weight is the number of ones in that row. If all the columns and rows are of equal weight, then the code is called *regular*, otherwise it is called *irregular*. Irregular codes are known to have better capacity achieving capabilities [34].

An LDPC code can be represented graphically in terms of a *Tanner* graph [35]. Such description is very powerful. Each row, which is a check equation, is represented by a *Check Node* (CN), and each column, corresponding to one of the bits, is represented by a *Variable Node* (VN). The "1"s in the matrix represent the connections between the CNs and VNs. Figure 4.3 shows the Tanner graph of the example code.

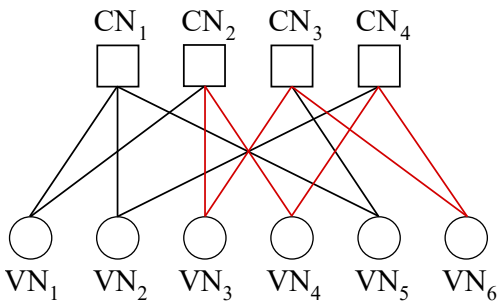

Figure 4.3: Tanner graph of the example code. The red line indicates the girth.

The girth of the code is defined as the shortest cycle in its associated Tanner graph. This in indicated by the red line (of length 6) in the figure. The existence of very short cycles increases the correlation between the extrinsic messages exchanged during the iterative decoding, causing slow convergence and in turn, poor performance.

4.2.1 Encoding

The encoding can be described in the general form of

$$\mathbf{c} = \mathbf{uG}, \tag{4.15}$$

where $\mathbf{c}$ is the output codeword, $\mathbf{u}$ is the input block, and $\mathbf{G}$ is the generator matrix. For LDPC codes, the parity check matrix $\mathbf{H}$ is the design parameter and not the generator matrix $\mathbf{G}$. However, the generator matrix can still be obtained from a given parity check matrix. This is usually done by putting $\mathbf{H}$ into systematic form using *Gauss-Jordan Elimination*, and then the generator matrix is found directly [14].

Two problems exist, first, the parity check matrix is designed for a specific input block length, and therefore using other lengths is not possible. The second problem lies in the transformation of $\mathbf{H}$ into systematic form, since it can get too complicated for long block lengths. The first problem is handled using *Quasi-Cyclic* (QC) LDPC codes, and those can easily support variable input sizes through *Lifting* [36]. The second problem can be mitigated by utilizing a structure similar to *Repeat-Accumulate* (RA) codes [37] or some other upper-triangular based structures. Such structures allow direct encoding from the parity check matrix through fast and low-complexity operations [38] such as *back-substitution*.

Quasi-Cyclic (QC) LDPC Codes

An LDPC code is called Quasi-Cyclic, if the parity matrix can be divided into equal sized square submatrices of some dimension Z_0, that are either the identity matrix, a circular shift of it, or the all zero matrix. This allow us to describe the $\mathbf{H}$ matrix in terms of the circular shifts. Such description is called the *Exponent* matrix $E(\mathbf{H})$, with elements of 0 for the identity matrix, larger than 0 for the circular shifts, or -1 for the all zero matrix.

With QC codes, variable input sizes are easily supported via *Lifting*. In Floor-based lifting [36], the following is performed

$$e_{ij}(\mathbf{H}_{\text{new}}) = \begin{cases} -1, & \text{if } e_{ij}(\mathbf{H}_{\text{old}}) = -1, \\ \left\lfloor e_{ij}(\mathbf{H}_{\text{old}}) \cdot \frac{Z_{\text{new}}}{Z_{\text{old}}} \right\rfloor, & \text{if } e_{ij}(\mathbf{H}_{\text{old}}) \geq 0, \end{cases} \tag{4.16}$$

Where $e_{ij}(\mathbf{H})$ is the element at the ith row and jth column of the exponent matrix $E(\mathbf{H})$. This transforms the old parity matrix with submatrix dimension Z_{old} to a new matrix with submatrix dimension Z_{new}. Therefore, by varying Z_{new} wide range of block lengths can be supported.

Repeat Accumulate (RA) LDPC Codes

The parity check matrix of a RA code can be divided into two submatrices

$$\mathbf{H} = [\mathbf{H}_1 \ \mathbf{H}_2] \tag{4.17}$$

The matrix $\mathbf{H}_1$ is some parity matrix of dimension $(N-K)\times K$. The second matrix $\mathbf{H}_2$ is of dimensions $(N-K)\times(N-K)$, and has a very nice structure. Except of its first column, $\mathbf{H}_2$ has a double diagonal structure that allows for direct encoding from the parity check matrix through back-substitution, without the need to calculate the generator matrix. This is possible since the first K columns of $\mathbf{H}$ are related to the systematic part, and therefore parity check results until the Kth coefficient of the check equations can be readily calculated. Each parity bit is then the sum of its partial parity check result and the previous parity bit.

4.2.2 Decoding

Decoding of LDPC codes is performed with the Sum-Product algorithm [33]. This is based on message passing between the CNs and VNs in the Tanner graph. At the start, the VNs send the channel LLRs L_j to the connected CNs. The CNs then perform the parity checking calculation probabilistically and pass new messages to their connected VNs according to [14]

$$L_{i\to j} = 2\tanh^{-1}\left[\prod_{j'\in N(i)-\{j\}} \tanh(L_{j'\to i}/2)\right], \tag{4.18}$$

where $L_{i\to j}$ is the message passed from the ith CN to jth VN, $L_{j\to i}$ is the message passed from the jth VN to the ith CN, and $N(i)$ is the set of VNs connected to the ith CN. The VNs receive these messages, add them up, and then pass new messages to the connected CNs according to

$$L_{j\to i} = L_j + \sum_{i'\in N(j)-\{i\}} L_{i'\to j}, \tag{4.19}$$

where $N(j)$ is the set of CNs connected to the jth VN. At this point, one iteration is finished, and the total LLR can be calculated as

$$L_{j(\text{total})} = L_j + \sum_{i\in N(j)} L_{i\to j}. \tag{4.20}$$

The sequence in which the nodes are scheduled can affect the performance. The one described above, in which all the CNs, and then all the VNs update their messages in parallel, is called the *Flood* schedule. An improved performance can be achieved if *serial* scheduling is performed. This is called *Layered Belief Propagation* (LBP) [39, 40] and there exist many variants of it. In *Column Message Passing* schedule [41], it starts with the first VN, gets the messages from its connected CNs, processes them, and then generates messages back to those CNs. The second VN starts and does the same. Such schedule creates a turbo effect, since within each iteration, a VN already sees improvement in the CNs messages due to the previous VNs. LBP offers almost double the convergence (half number of iterations) to that of the flood schedule under the same complexity per iteration [42].

An approximation can be made to (4.18) in the form

$$L_{i\to j} = \left(\prod_{j'\in N(i)-\{j\}} \alpha_{j'\to i}\right) \cdot \min_{j'\in N(i)-\{j\}} \beta_{j'\to i}, \tag{4.21}$$

where $\alpha_{j'\to i}$ and $\beta_{j'\to i}$ are the sign and magnitude of $L_{j'\to i}$, respectively. This is the min-sum approximation [43], and offers lower complexity decoding, but again, at the cost of some performance loss.

4.3 The Performance Comparison

In this section we compare the performance of polar codes against turbo and LDPC codes. We transmit using BPSK over the AWGN channel. We consider code rates of $R = 1/3,\ 1/2,\ 2/3$, and $5/6$, and information lengths of $K = 64,\ 256,\ 1024$, and 4096. These settings should be sufficient to show the performance of polar codes across a wide range of high reliability scenarios, as well as scenarios of high throughput. For convolutional and turbo codes, we chose those of LTE [17]. For LDPC, we used the IEEE 802.16 codes [44], and since it does not support codes of rate 1/3, an extension method has been applied to the rate 1/2 code in a fashion similar to [45]. As for polar codes, they were constructed using the new algorithm given in (3.17), and by searching for suitable Target SNRs. The selected SNRs are listed in Table 4.1 for each configuration.

K	$R = 1/3$	$R = 1/2$	$R = 2/3$	$R = 5/6$
64	-0.5	4	6.5	8
256	2	3.5	6	7.5
1024	1.75	3	5.25	7.5
4096	1.5	2.25	4.75	7

Table 4.1: Target SNR in dB for each configuration.

The rate adaptation for convolutional and turbo codes was obviously done by *puncturing*. For LDPC, there is no rate adaption since for each rate we used a different parity check matrix that fits that rate. For polar codes, since the encoder size is limited to powers of 2, we chose the next power of 2 that fits our input length and the extra positions were handled by applying zeros to the encoder bottom positions. As their corresponding outputs do not depend on the upper positions, then they are also equal to zero and can be removed from the output codeword. At the decoder, the LLRs of these positions are set to a very high positive value, reflecting a $+\infty$ LLR.

Regarding the number of iterations and decoding algorithms, we follow our results in [29], in which for turbo codes, the number of iterations are set to 8 and the decoding algorithm is MAX-Log-MAP. For LDPC codes, the number of iterations is 16 and Layered Min-Sum is used for decoding. One should also mention that there exist modified algorithms that try to improve the

approximate algorithms through lookup tables, offsetting, or low complexity functions. These of course come at the cost of extra computations or memory consumption. For polar codes, we used the CRC-based list decoder with list size of 8 and the approximation in (2.9). The CRC used is based on the 16-bit version of LTE [17]. The different code lengths are addressed in terms of the information block length K and the rate R. The reason for using K instead of N is to simplify the encoding process by having K fixed. The results are shown in Figures 4.4-4.7.

At the short information length of $K = 64$, the polar code beats both turbo and LDPC codes at all code rates. This makes it an attractive choice for the control channels in mobile communications, since the transmission of short messages is usually employed there. For fairness, we should keep in mind that CRC-aided decoding was used and therefore the information length is effectively $64 - 16 = 48$ bits. In other words, there is a rate reduction. Once a sufficient length is reached, we see that the schemes start to perform close to each other.

From these results, it is evident that polar codes are an edge-to-edge competitor with the current state-of-the-art coding schemes of turbo and LDPC, even beating them at some point. However, it still has its own problems, namely one needs to consider that the list SC decoding of polar codes suffers from a relatively high decoding latency, and the excellent performance we see here is due to the use of CRC extension on top of that. This places an even extra complexity on the decoder, since the CRC detection needs to be performed on each of the surviving paths of the final list. Nonetheless, with appropriate design of the decoding architectures, one can hope that the impact of those issues are decreased.

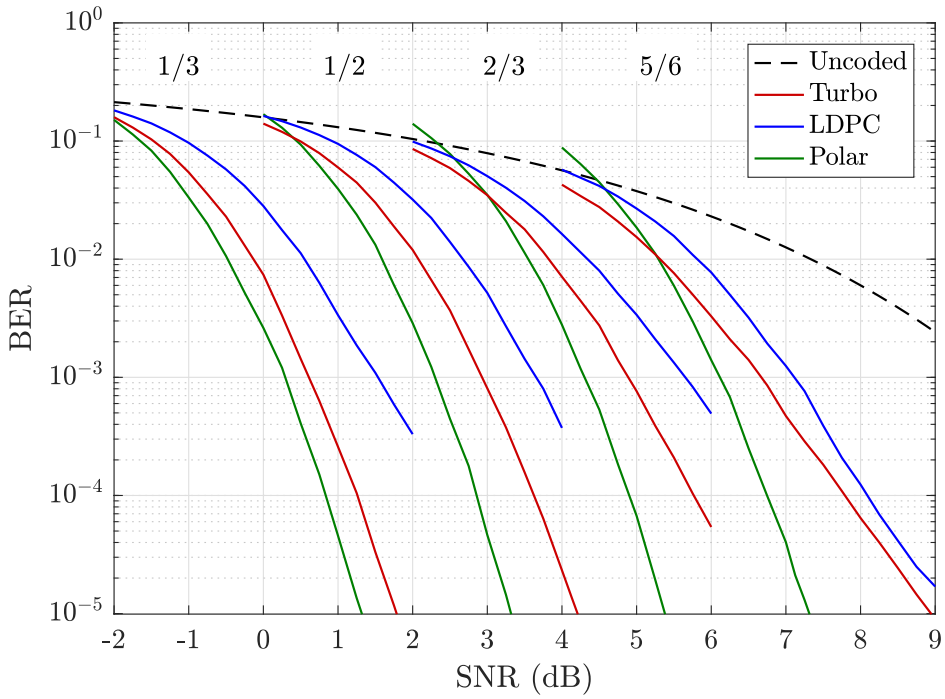

Figure 4.4: BER comparison for different code rates, $K = 64$ (For LDPC, $K = 60$ for $R = 1/2, 1/3,$ and $5/6$.)

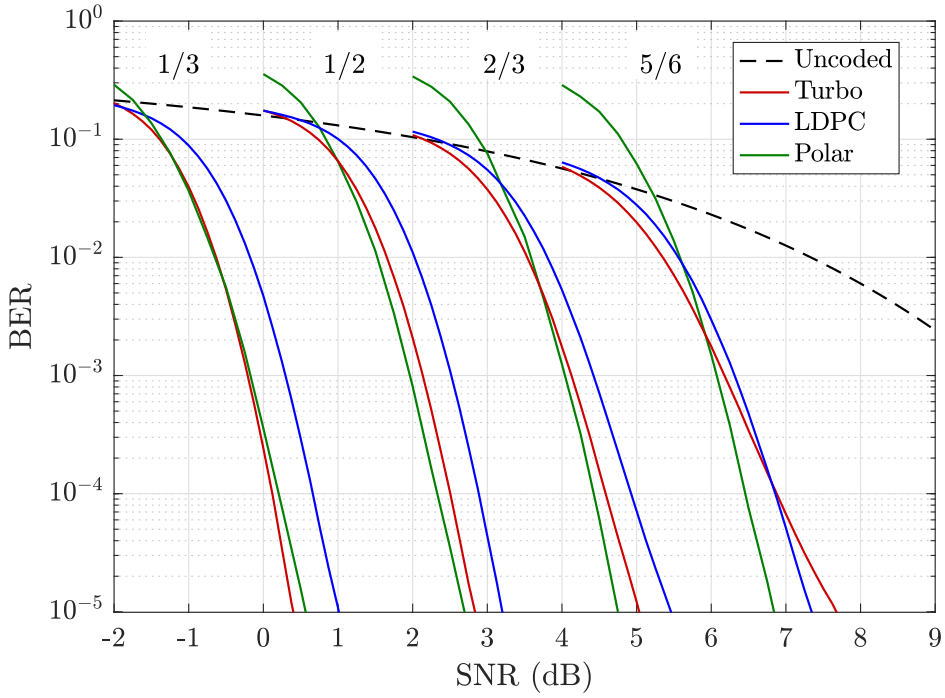

Figure 4.5: BER comparison for different code rates, $K = 256$ (For LDPC, $K = 252$ for $R = 1/2,$ and $1/3$, and $K = 260$ for $R = 5/6$.)

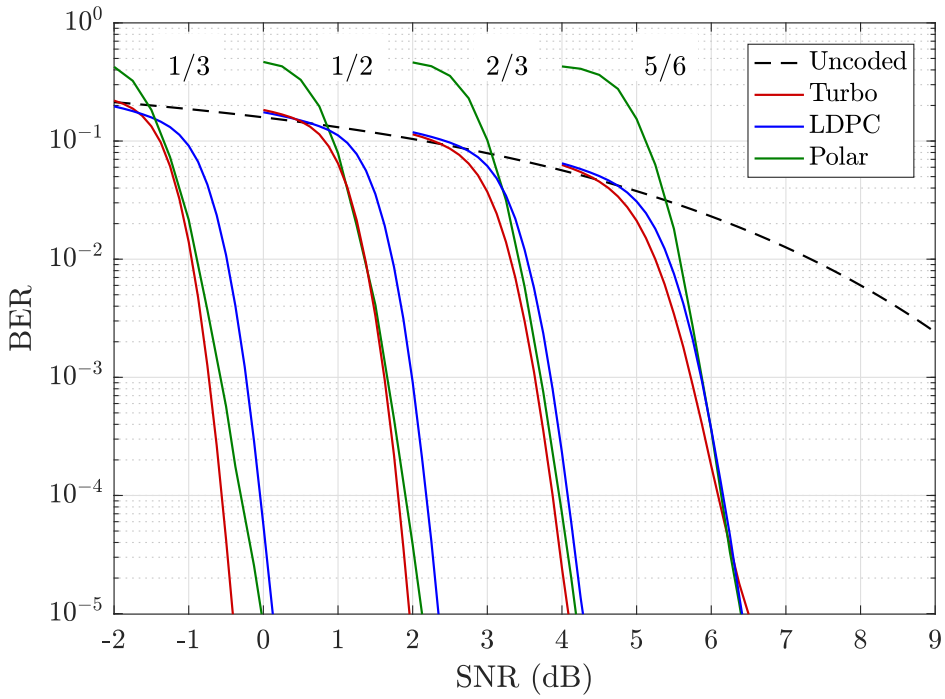

Figure 4.6: BER comparison for different code rates, $K = 1024$ (For LDPC, $K = 1020$ for $R = 1/2, 1/3,$ and $5/6$.)

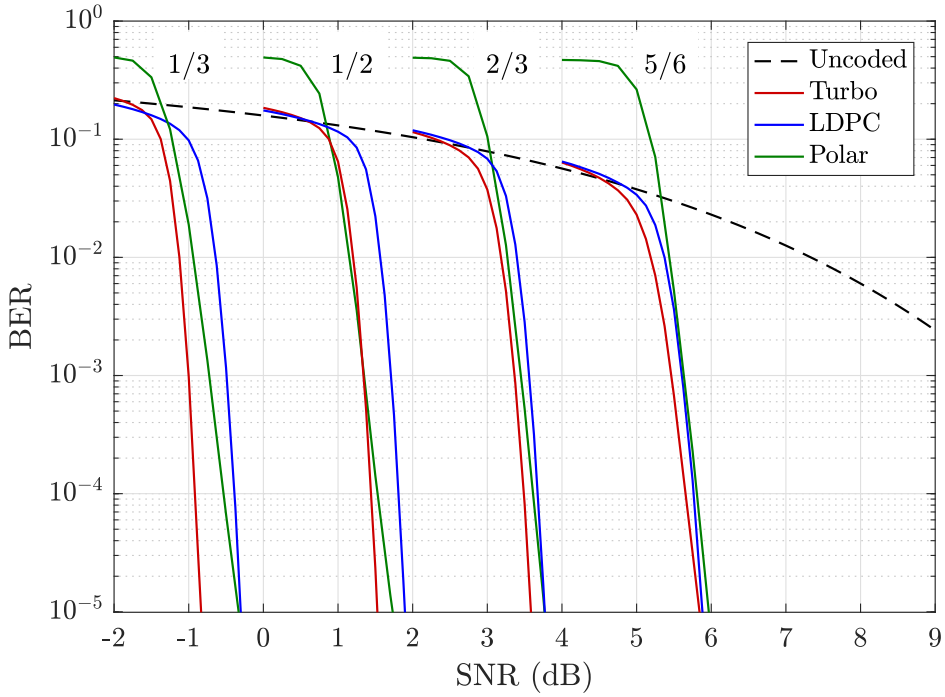

Figure 4.7: BER comparison for different code rates, $K = 4096$ (For LDPC, $K = 4092$ for $R = 1/2,$ and $1/3$, and $K = 4100$ for $R = 5/6$.)

5

Conclusion and Outlook

We considered in this thesis some aspects of polar codes operating over the AWGN channel. We touched on the problem of the construction in Chapter 3 in which we found a new way to construct polar codes and showed that it can deliver very good performance. We also compared the performance of polar codes against the state-of-the-art turbo and LDPC codes and the results showed that polar codes provide a very competitive performance. However, this is not everything, the operation complexity of those coding schemes regarding the encoding and decoding algorithms and the possibility for parallel implementations, is by itself a separate sophisticated topic that shows the other side of the story with respect to the achievable throughput.

The construction of polar codes we considered here is based on the fact that the SC decoder is employed. The codes were optimized to ensure that the SC decoder performs the best. However, under the other types of decoders, it is unclear how a good construction would look like, but at least, we know that the non-universal behavior of polar codes is due to the SC decoder and therefore the use of other decoders might end this problem forever. Unfortunately, no other decoder can provide the same low-complexity that the SC decoder enjoys. Therefore, finding an alternative low-complexity decoder for polar codes can be a very interesting research topic.

References

[1] 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.

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[3] M. Golay, "Notes on digital coding," *Proc. IRE. 37: 657*, 1949.

[4] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, April 1950.

[5] P. Elias, "Coding for noisy channels," *IRE Convention Record*, pp. 37–46, 1955.

[6] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[7] R. G. Gallager, *Low Density Parity Check Codes,*. Sc.D. thesis, MIT, Cambridge, 1960.

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

[9] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[10] E. Arikan, "Systematic Polar Coding," *IEEE Communications Letters*, vol. 15, no. 8, pp. 860–862, August 2011.

[11] L. Li, W. Zhang, and Y. Hu, "On the Error Performance of Systematic Polar Codes," *ArXiv e-prints*, Apr. 2015.

[12] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Flexible and Low-Complexity Encoding and Decoding of Systematic Polar Codes," *IEEE Transactions on Communications*, vol. 64, no. 7, pp. 2732–2745, July 2016.

[13] B. Li, D. Tse, K. Chen, and H. Shen, "Capacity-achieving rateless polar codes," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 46–50.

[14] W. E. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[15] I. Tal and A. Vardy, "List decoding of polar codes," in *2011 IEEE International Symposium on Information Theory Proceedings (ISIT)*, July 2011, pp. 1–5.

[16] 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, Oct. 2015.

[17] "Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding," 3rd Generation Partnership Project (3GPP), TS 36.212, 2016.

[18] E. Sasoglu, *Polar coding theorems for discrete systems*. PhD thesis, EPFL, 2011.

[19] S. H. Hassani and R. Urbanke, "Universal polar codes," in *2014 IEEE International Symposium on Information Theory*, June 2014, pp. 1451–1455.

[20] I. Tal and A. Vardy, "How to Construct Polar Codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, Oct 2013.

[21] R. Pedarsani, S. H. Hassani, I. Tal, and E. Telatar, "On the construction of polar codes," in *2011 IEEE International Symposium on Information Theory Proceedings*, July 2011, pp. 11–15.

[22] D. Kern, S. Vorkper, and V. Khn, "A new code construction for polar codes using min-sum density," in *2014 8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Aug 2014, pp. 228–232.

[23] Y. Zhang, A. Liu, K. Pan, C. Gong, and S. Yang, "A Practical Construction Method for Polar Codes," *IEEE Communications Letters*, vol. 18, no. 11, pp. 1871–1874, Nov 2014.

[24] D. Wu, Y. Li, and Y. Sun, "Construction and Block Error Rate Analysis of Polar Codes Over AWGN Channel Based on Gaussian Approximation," *IEEE Communications Letters*, vol. 18, no. 7, pp. 1099–1102, July 2014.

[25] P. Trifonov, "Efficient Design and Decoding of Polar Codes," *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3221–3227, November 2012.

[26] S.-Y. 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, Feb 2001.

[27] B. Tahir and M. Rupp, "New construction and performance analysis of Polar codes over AWGN channels," in *2017 24th International Conference on Telecommunications (ICT)*, May 2017, pp. 1–4.

[28] E. Arikan, 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, Aug. 2015.

[29] B. Tahir, S. Schwarz, and M. Rupp, "BER comparison between Convolutional, Turbo, LDPC, and Polar codes," in *2017 24th International Conference on Telecommunications (ICT)*, May 2017, pp. 1–7.

[30] J. C. Ikuno, S. Schwarz, and M. Simko, "LTE Rate Matching Performance with Code Block Balancing," in *17th European Wireless 2011 - Sustainable Wireless Technologies*, April 2011, pp. 1–3.

[31] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[32] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data disturbed by time-varying intersymbol interference [applicable to digital mobile radio receivers]," in *Global Telecommunications Conference, 1990, and Exhibition. 'Communications: Connecting the Future', GLOBECOM '90., IEEE*, Dec. 1990, pp. 1679–1684 vol.3.

[33] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[34] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[35] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[36] S. Myung, K. Yang, and Y. Kim, "Lifting methods for quasi-cyclic LDPC codes," *IEEE Communications Letters*, vol. 10, no. 6, pp. 489–491, June 2006.

[37] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for turbo-like codes," *Proc. 36th Annual Allerton Conf. on Communication, Control, and Computing*, pp. 201–210, Sep. 1998.

[38] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[39] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A Statistical Mechanics and its Applications*, vol. 330, pp. 259–270, Dec. 2003.

[40] M. M. Mansour and N. R. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 2, Nov. 2002, pp. 1383–1388 vol.2.

[41] P. Radosavljevic, A. de Baynast, and J. R. Cavallaro, "Optimized Message Passing Schedules for LDPC Decoding," in *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005.*, Oct. 2005, pp. 591–595.

[42] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, Oct. 2004, pp. 107–112.

[43] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[44] "IEEE Standard for Air Interface for Broadband Wireless Access Systems," Institute of Electrical and Electronics Engineers (IEEE), IEEE Std 802.16, 2012.

[45] H. J. Joo, S. N. Hong, and D. J. Shin, "Design of rate-compatible RA-type low-density parity-check codes using splitting," *IEEE Transactions on Communications*, vol. 57, no. 12, pp. 3524–3528, Dec. 2009.