A comparison of Polar Code Constructions and Puncturing methods for AWGN and Fading channels

Jonas Sedin
Abstract

Today 5G and other wireless standards are being developed for the future of our society. The different use-cases of future wireless services are going to be ever-more demanding, whether it is vehicular communication or low-powered sensor networks. High-rate, ultra-reliable and low-power are future requirements that will also affect the coding schemes being used. A relatively recent coding scheme, called polar codes, has the potential to fulfill all of these requirements if the coding scheme applied is well-designed. In this thesis we will be focusing on practical algorithms for implementation of polar codes at medium-sized block-lengths.

Polar codes are very different from other modern coding schemes. The code construction is rather unique in that they are dependent on the underlying channel, where the code construction can change with the Signal-to-Noise-Ratio of the AWGN channel. The puncturing of polar codes is also non-trivial compared to other coding schemes. Since the Polar Codes are dependent on the underlying channel, the fading channel performance is thus important to consider. In this thesis we aim to show through simulations how these different concepts affect the Block Error Rate (BLER) performance. Specifically, we compare how code constructions compare over the AWGN channel, how code construction affects the BLER performance with puncturing and how puncturing affects the performance over fading channels. We find that an appropriate code construction is very important for optimal performance over the AWGN channel with puncturing, in our case using Gaussian Approximation. We also find that different puncturing methods have vastly different performances for different rates over the AWGN and Rayleigh fading channel and that applying an interleaver is very important for optimal performance.
Abstrakt

Idag så är utvecklas och standardiseras 5G och andra trådlösa standarder. De olika applikationerna av framtida trådlösa nätverk kommer att vara mer och mer krävande, allt från kommunikation mellan fordon till små energisnåla sensorer. Högre hastigheter, pål立tlig och energieffektivitet är krav som också kommer att påverka den kanalkodningen som används av standarden. En relativt ny typ av kanalkodning, polar codes, har all potential att kunna uppfylla de framtida kraven. I denna uppsats så kommer vi att undersöka praktiska algoritmer för implementation av polar codes för blocklängder i mediumstorlek.

Acknowledgements

I would like to express my sincerest gratitude to my supervisor at Intel Mobile, Lea Castel, who not only provided daily guidance during the technical work but also took great interest in the work that I did and also spent a great deal of time thoroughly proof-reading this master’s thesis.

I am also very thankful to Intel Mobile Communications Denmark in Aalborg for providing me with the opportunity to learn from them and write my master’s thesis with them, with all of the support and encouragement that followed. Their interest in my work was very encouraging.

Supervisor Tobias Oechtering at KTH who provided me with sound guidance and encouragement during the project.

My mother Annette and brother Nicklas, who assisted and supported me during my years of studies. I am also forever in grateful to KTH for providing me with many opportunities during my studies.
# Contents

ABSTRACT .................................................................................................................1  
ACKNOWLEDGEMENTS ............................................................................................2  
CONTENTS ................................................................................................................4  
LIST OF FIGURES ......................................................................................................6  
LIST OF ACRONYMS AND NOTATION ....................................................................8  

Chapter 1:  Introduction ............................................................................................11  
1.1  Motivation ........................................................................................................11  
1.2  Societal Impact ................................................................................................11  
1.3  The channel coding problem ..........................................................................12  
1.4  Introduction of modern channel coding .......................................................12  
1.5  Work Outline ..................................................................................................14  
1.6  Thesis Outline ................................................................................................14  

Chapter 2:  Communication Theory Basics ............................................................16  
2.1  Probabilistic Channel Models ........................................................................16  
2.2  Basic Information Theory .............................................................................17  
2.3  Modulation ......................................................................................................19  
2.4  Measures of Signal-to-Noise Ratio ...............................................................20  
2.5  Cyclic Redundancy Check ............................................................................20  
2.6  Puncturing ......................................................................................................21  
2.7  Likelihood Ratio and Log Likelihood Ratio ..................................................21  
2.8  Interleaving and permutations ......................................................................22  

Chapter 3:  Basics of Polarization and Polar Codes ...............................................23  
3.1  Polarization basics ........................................................................................23  
3.1.1  Recursive Channel Transformation .......................................................23  
3.1.2  Polarization ..............................................................................................25  
3.1.3  Principles of Polar encoding and Code construction ...............................26  
3.1.4  Binary Erasure Channel .........................................................................27  
3.1.5  Principles of decoding .............................................................................28  
3.2  Properties and practical issues of Polar Codes .............................................29  
3.2.1  General properties of Polar Codes ........................................................29  
3.2.2  Practical problems of Polar Codes ........................................................30  

Chapter 4:  Decoding Algorithms ...........................................................................32  
4.1  LLR-based decoding .......................................................................................32  
4.2  Successive Cancellation Decoder ....................................................................33
4.3 Successive Cancellation List Decoder ..................................................... 36
  4.3.1 Path Management ............................................................................ 41

4.4 CRC-aided Successive Cancellation Decoder ........................................... 41
  4.4.1 CRC for CA-SCL ............................................................................ 44
  4.4.2 Performance of different CRC polynomials ..................................... 44

4.5 SCAN Decoder ...................................................................................... 45

4.6 Selection of Decoding Algorithm ........................................................... 48

Chapter 5: Code Construction Techniques .................................................... 49

  5.1 Code Construction problem ................................................................ 49

  5.2 Channel parameter dependent methods ............................................. 50
    5.2.1 Heuristic ..................................................................................... 50
    5.2.2 Tal-Vardy Degradation Method .................................................... 52
      5.2.2.1 Tal-Vardy Degradation algorithm for Polar Codes .................... 53
      5.2.3 Gaussian Approximation ......................................................... 58

  5.3 Channel parameter independent methods .......................................... 60
    5.3.1 Code Construction ensemble average .......................................... 61
    5.3.2 Polarization Weight .................................................................... 61

  5.4 Code construction comparison ............................................................ 61
    5.4.1 Complexity considerations ......................................................... 62
    5.4.2 BLER comparison of code constructions ...................................... 62
    5.4.3 BLER performance discussion .................................................... 63

Chapter 6: Puncturing .................................................................................... 66

  6.1 Quasi-Uniform Puncturing ................................................................. 66

  6.2 Wang-Liu Method .............................................................................. 69
    6.2.1 Code Construction for the Wang-Liu method ................................. 71

  6.3 Scattered Puncturing ........................................................................... 73

  6.4 Polarization-preserving puncturing ..................................................... 74

  6.5 Performance comparison .................................................................... 75

  6.6 Discussion on performance ................................................................. 77

Chapter 7: Polar Coding for fading channels and higher level modulation ....... 78

  7.1 Polar Codes for higher level modulation ............................................. 78

  7.2 Polar Codes over fading channels ...................................................... 80
    7.2.1 Independent Rayleigh Fading Channels ........................................ 80
    7.2.2 Code Construction for Fading Channels ....................................... 81
    7.2.3 Simulation over the Rayleigh Fading Channels .............................. 81
    7.2.4 Discussion on performance ......................................................... 83

Chapter 8: Discussion and future work ......................................................... 85

  8.1 Conclusion ......................................................................................... 85

  8.2 Future work ....................................................................................... 85

BIBLIOGRAPHY .......................................................................................... 87
List of Figures

Figure 1-1. Channel Coding Problem. 12
Figure 1-2. Turbo Decoder 13
Figure 2-1. Probabilistic Channel 16
Figure 2-2. Binary Symmetric Channel 17
Figure 2-3. Binary Erasure Channel 17
Figure 2-4. BPSK, QPSK and 16QAM modulation. 19
Figure 3-1. Recursive Channel Transformation. 24
Figure 3-2. Polar Encoding. 26
Figure 3-3. Distribution of Bhattacharyya Parameters. 27
Figure 3-4. Bound on the block error probability under successive cancellation decoding. 28
Figure 3-5. Decoding path under Successive Cancellation Decoding 28
Figure 4-1. The basic decoding elements. 33
Figure 4-2. The six first steps of Successive Cancellation decoding. 34
Figure 4-3. Path progression of an SCL-decoder. 37
Figure 4-4. SCL vs SC decoding. 40
Figure 4-5. CRC-aided SCL decoder scheme. 41
Figure 4-6. CA-SCL, SCL vs SC decoding. 43
Figure 4-7. CRCs of different lengths for CA-SCL8. 44
Figure 4-8. Simulation of different CRC polynomials over CA-SCL8. 45
Figure 4-9. The SCAN decoder. 45
Figure 4-10. Performance of SCAN decoder vs. SC and SCL. 47
Figure 5-1. Heuristic Code Construction performance. 51
Figure 5-2. Heuristic code construction performance at fixed SNR. 52
Figure 5-3. Channel degradation. 53
Figure 5-4. Tal-Vardy code construction performance. 57
Figure 5-5. Tal-Vardy code construction performance at fixed SNR. 58
Figure 5-6. GA code construction performance. 60
Figure 5-7. GA code construction performance at fixed SNR. 60
Figure 5-8. Code construction test for CA-SCL. 63
Figure 5-9. The partial order. 65
Figure 6-1. CA-SCL16 performance with Quasi-Uniform Puncturing. 68
Figure 6-2. CA-SCL16 performance with QUP 69
Figure 6-3. Pre-freezing and puncturing. 71
Figure 6-4. Lattice transformation · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · 71
Figure 6-5. CA-SCL16 performance for Wang Liu puncturing · · · · · · · · · · · · · · · · · · · · · · 72
Figure 6-6. Pre-freezing and puncturing · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · · 73
Figure 6-7. CA-SCL16 performance for Scattered Puncturing · · · · · · · · · · · · · · · · · · · · · · 74
Figure 6-8. CA-SCL16 performance for Polarization-preserving puncturing · · · · · · · · · · · · · · 75
Figure 6-9. CA-SCL8 performance for rates 5/6, 2/3 and 1/2 · · · · · · · · · · · · · · · · · · · · · · · · 76
Figure 6-10. CA-SCL8 performance for rates 1/2, 1/3 and 1/4 · · · · · · · · · · · · · · · · · · · · · · · 76
Figure 6-11. CA-SCL8 performance for rates 1/3, 1/6 and 1/10 · · · · · · · · · · · · · · · · · · · · · · · 76
Figure 7-1. CA-SCL performance for different list-sizes · · · · · · · · · · · · · · · · · · · · · · · · · · 79
Figure 7-2. CA-SCL performance for different list-sizes · · · · · · · · · · · · · · · · · · · · · · · · · · 79
Figure 7-3. CA-SCL8 performance for different design SNRs · · · · · · · · · · · · · · · · · · · · · · · · 80
Figure 7-4. CA-SCL8, Rayleigh fading QPSK channel without interleaver · · · · · · · · · · · · · · · 82
Figure 7-5. CA-SCL8, Rayleigh fading QPSK channel without interleaver · · · · · · · · · · · · · · · 82
Figure 7-6. CA-SCL8, Rayleigh fading QPSK channel with interleaver · · · · · · · · · · · · · · · · 83
Figure 7-7. CA-SCL8, Rayleigh fading QPSK channel with interleaver · · · · · · · · · · · · · · · · 83
Figure 7-8. How the QPSK signal is demodulated in to LLRs · · · · · · · · · · · · · · · · · · · · · · · · 84
# List of Acronyms and Notation

<table>
<thead>
<tr>
<th>Acronyms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWGN</td>
<td>Additive White Gaussian Noise</td>
</tr>
<tr>
<td>BEC</td>
<td>Binary Erasure Channel</td>
</tr>
<tr>
<td>BLER</td>
<td>Block Error Rate</td>
</tr>
<tr>
<td>BSC</td>
<td>Binary Symmetric Channel</td>
</tr>
<tr>
<td>BPSK</td>
<td>Binary Phase Shift Keying</td>
</tr>
<tr>
<td>CA-SCL</td>
<td>CRC-aided Successcive Cancellation List decoder</td>
</tr>
<tr>
<td>CSI</td>
<td>Channel Side Information</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>GA</td>
<td>Gaussian Approximation</td>
</tr>
<tr>
<td>LR</td>
<td>Likelihood Ratio</td>
</tr>
<tr>
<td>LLR</td>
<td>Log Likelihood Ratio</td>
</tr>
<tr>
<td>PP</td>
<td>Polarization-preserving puncturing</td>
</tr>
<tr>
<td>PW</td>
<td>Polarization Weight code construction</td>
</tr>
<tr>
<td>SCAN</td>
<td>Soft CANcellation decoder</td>
</tr>
<tr>
<td>SC</td>
<td>Successive Cancellation decoder</td>
</tr>
<tr>
<td>SCL</td>
<td>Successive Cancellation List decoder</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>QAM</td>
<td>Quadrature Amplitude Modulation</td>
</tr>
<tr>
<td>QUP</td>
<td>Quasi-Uniform Puncturing</td>
</tr>
<tr>
<td>QPSK</td>
<td>Quartenary Phase Shift Keying</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td>random variable</td>
</tr>
<tr>
<td>$\mathcal{Y}$</td>
<td>alphabet</td>
</tr>
<tr>
<td>$y$</td>
<td>realization of $Y$</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{Y}</td>
</tr>
<tr>
<td>$\bar{y}$</td>
<td>conjugate symbol of $y$</td>
</tr>
<tr>
<td>$p(Y = y</td>
<td>X = x)$</td>
</tr>
<tr>
<td>$W(y</td>
<td>x)$</td>
</tr>
<tr>
<td>$Q(y</td>
<td>x)$</td>
</tr>
<tr>
<td>$C(W)$</td>
<td>capacity of channel $W(y</td>
</tr>
<tr>
<td>$I(W)$</td>
<td>symmetric capacity of channel $W(y</td>
</tr>
</tbody>
</table>
$G_N$ decoder matrix of block-length $N$
$\mathcal{A}, \mathcal{A}_c$ set of frozen and non-frozen indices
$Z(W)$ Bhattacharyya parameter
$W_N^{(i)}$ bit-channel
$W^N$ $N$ channel uses
$W_N$ vector channel of size $N$
$x$ codeword or symbol
$y$ received symbol
$u$ information bit
$N$ block-length of a code
$K$ information length
$R$ code rate
$M$ punctured block-length
$L_{\text{list}}$ list size
$L_{\text{CRC}}$ length of CRC
$n$ polarization steps
$B_i(j, k)$ soft or hard decision array
$L_i(j, k)$ LLR array
$\gamma_{LR}, \gamma_{\text{LLR}}$ likelihood ratio and log likelihood ratio
$p$ puncturing pattern
$p_e$ probability of error
$\sigma^2$ variance
$N_0$ noise spectral density
$E_s$ power per symbol
$E_b$ power per bit
$\mathcal{N}(0, \sigma^2)$ normal distribution with mean $\mu$ and variance $\sigma^2$
$\mathcal{CN}(0, \sigma^2)$ complex normal distribution with mean $\mu$ and variance $\sigma^2$
$\delta$ fidelity parameter
$\log\Omega$ logarithm of the base of two
$[y], \lfloor y \rfloor$ ceiling and floor of $y$
$O(N)$ asymptotic Complexity of $N$
Chapter 1: 
Introduction

1.1 Motivation

Wireless communication systems are today being developed to provide a large range of use cases that will transform our society to become a much more connected society. An example is 5G, which currently has three different use-cases:

- **eMBB** – enhanced Mobile Broadband
- **URLLC** – Ultra Reliable Low Latency Communication,
- **mMTC** – Massive Machine Type Communications

Each of these three use cases has varying requirements, from the up to 10Gbps rates of eMBB to less than 10^-5 packet loss of URLLC and to the more than 50 billion low-power devices envisioned through mMTC. These vastly varying requirements and specifications put enormous demands on the standards and algorithms that fit should each use-case.

At the heart of modern digital communication lies channel coding, or also called Forward Error Correction (FEC), and with the requirements of future communication systems, channel coding also has to satisfy the requirements posed by future communication systems, such as being high rate, ultra-reliable or low power.

Turbo Codes and LDPC codes are today the leading channel codes used by most wireless communication standards, and until only a few years ago, this dominant position was expected to continue. But with the new use-cases, there is a need to explore other channel coding schemes that can satisfy the requirements of the future wireless communication standards, and one of the newest codes to compete with the former codes are the polar codes.

1.2 Societal Impact

The impact of wireless communication has already had profound impacts on society, but it is believed that the biggest impact is yet to come. It is projected that there will be more than 50 billion wirelessly connected devices by year 2020 and they will have the potential for huge cost savings within businesses such as healthcare, autonomous driving, manufacturing, agriculture, shipping and entertainment [1]. 5G is projected to employ 22 million workers globally by 2036 and enable $12.3 trillion global economic output by 2035 [2]. And 5G is just one wireless standard of many currently being developed for the future.
To see how the field of channel coding has had a large impact on society, there is a very clear example from the past. In the late 70’s, the joint modulation and coding scheme, Trellis-Coded Modulation was invented by Ungerboeck [3]. This scheme was alone able to increase the throughput of modems on normal telephone-lines from 23kbps to 45kbps, which is considered to be a big contributor to making internet more wide-spread in the early ages of the internet and for instance made it viable to download media-files over the internet [4].

1.3 The channel coding problem

The basic channel coding problem is depicted below. We send message $x$ described by $K$ bits through a noisy channel and we want to recover this signal by introducing some type of redundancy so that we end up with $N > K$ bits.

![Channel Coding Problem](image)

Figure 1-1. Channel Coding Problem.

Our objective would be to find codes or algorithms that, 1) allows us to recover the information with low probability of error, 2) that reduces the amount of redundancy that we send over the channel and, 3) does the two above under acceptable computational complexity.

Shannon, in his milestone paper published in 1948 [5], showed that we can achieve rates with vanishingly low error rates if we allow a sufficiently long block-length and that the best rate that we can achieve is called the capacity $C$.

1.4 Introduction of modern channel coding

Ever since Shannon showed that there is an upper bound on the information rate achievable over a noisy channel, coding theorists have been trying to find efficient ways of achieving capacity as Shannons work did not really give any hint towards how to achieve capacity or even come close to capacity.
Figure 1-2. Turbo Decoder

It took more than 40 years of research before capacity could be approached using channel codes, when Turbo Codes [6] were invented in 1993, which uses an iterative decoding algorithm seen in Figure 1-2. The Turbo Codes work by having two decoders produce so-called a-priori probabilities to each other and for every iteration the probability of a correct decision becomes larger.

The Turbo Codes were a major breakthrough in coding theory and a few years later the Low-Density Parity-Check Codes [7] were rediscovered after being invented in the 60’s. The Low-Density Parity-Check works by having an encoder matrix which is very sparse and this sparseness is easily represented by a graph. At the decoder, the decoder exchanges a-priori probabilities in the graph iteratively, to make decisions more robust and correct errors. This makes the LDPC code highly parallelizable, which is suitable for high-rate applications.

Although capacity has been approached, which can be empirically proved, it is still not possible to prove that any of the above mentioned codes are able to achieve capacity on an arbitrary channel. Furthermore, although the performance of the codes are very close to capacity, channel decoding is still one of the most energy and area consuming tasks for any modern chip designer [8]. Therefore, there is still a lot of room for improvements both in theoretical as well as practical aspects of channel coding.

Polar codes, invented by Erdal Arikan in 2008, which can be proven to achieve capacity, were first seen as a theoretical curiosity with limited practical applications. Recently they have been shown to have practical performance that can rival and sometimes outperform [9] the most recent LDPC and Turbo Codes under similar computational complexity. Due to this performance, polar codes were accepted to be used in the 5G standard for eMBB traffic [10]. Polar codes are however very different from Turbo Codes and LDPC in that the codes are not iteratively decoded and although many methods of polar coding are borrowed from LDPC coding theory, the polar codes differ in many aspects such as code construction and puncturing, which will be the focus of this thesis.
1.5 Work Outline

Here we present the outline of the work that produced this thesis.

- Step 1: Bibliographic research on the academic state-of-the-art of polar codes in order to get an understanding of what is required of polar codes and what methods could be interesting to study.
- Step 2: Study the basics of polar codes and polarization, in order to be prepared for possible future problems and from this understanding implement a basic polar encoding and Decoding.
- Step 3: In order to completely verify the performance of the basic polar codes with other academic research papers, we in this step implemented a set of code construction methods.
- Step 4: Implement other more complex decoders that are incrementally more complex and select one/two of them to compare the code constructions and puncturing methods.
- Step 5: Revise the work so far and make a bibliographic research on the industrial state-of-the-art of the polar codes in 5G New Radio and make adjustments to previous selections of algorithms.
- Step 6: Implement and compare different algorithms for code construction and puncturing presented in academia and industry.
- Step 7: Implement the algorithms over more complicated channels such as higher-level modulation and Rayleigh fading simulations to see how well the algorithms perform over more complicated channels.
- Step 8: Report-writing and deep-diving in to academia to give a possible explanation to some of the results, along with preparing for final presentation.

The algorithms were implemented in Matlab except for some implementations that were written in C++ to speed up simulations.

1.6 Thesis Outline

The report is organized as follows.

- In Chapter 2, we introduce some digital communication basics needed to fully understand the theory of polar codes. In this chapter we will discuss probabilistic channel models, information theory and other more practical aspects such as SNR.
- In Chapter 3 we will introduce the basics of polarization and polar codes. Some of the properties and key issues of polar codes will be introduced here.
- In Chapter 4, we first introduce the basic decoder for the polar codes called Successive Cancellation decoder and then we implement more powerful decoders to improve the medium block-length performance. These decoders are the Soft CANCELLation (SCAN) decoder, Successive Cancellation List decoder(SCL) and the CRC Aided Successive Cancellation List(CA-SCL) decoder. We compare the performance of these algorithms
and introduce some practical aspects, such as CRC concatenation that we will use later on.

- In Chapter 5 we look at the important problem of code construction and compare their performance and complexity.
- In Chapter 6 we look at puncturing of the polar codes and their performance under different code construction methods.
- In Chapter 7 we discuss polar codes for the fading channel and show performance for polar codes over both higher order modulations as well as over the fading channel to show that polar codes can work over these channels with the code constructions developed earlier.
- Finally in Chapter 8 the thesis is concluded with a brief discussion on the results obtained, challenges and future work on polar codes
Chapter 2: Communication Theory Basics

In this section we will discuss some basics that will be needed in order to develop the polar codes.

2.1 Probabilistic Channel Models

As seen in Figure 2-1, in order to build and describe a communication system there is a need to have a model of the channel that we are transmitting information over. In this section we will briefly discuss some important concepts of channel models that we will use later on.

![Figure 2-1. Probabilistic Channel](image)

A channel has an input alphabet $\mathcal{X}$ and an output alphabet $\mathcal{Y}$. A channel is conveniently described by a set of conditional probability mass functions that relates the input and the output:

$$P[\text{output}|\text{input}] = P[Y = y|X = x]$$

In this thesis we will mostly focus on memoryless channels, defined as:

$$P[y^n|x^n] = \prod_{i=1}^{N} P[y_i|x_i]$$

This means that each "use" of the channel is independent of the other uses.

The binary Symmetric Channel, in Figure 2-2, has two input and output symbols $\mathcal{Y} \in \{0,1\}$ and $\mathcal{X} \in \{0,1\}$, and its conditional probabilities:

$$P[Y = 0|X = 0] = P[Y = 1|X = 1] = 1 - p_e$$

$$P[Y = 1|X = 0] = P[Y = 0|X = 1] = p_e$$

16
Another simple channel model is the Binary Erasure Channel, in Figure 2-3, defined as:

\[ P[Y = 0|X = 0] = P[Y = 1|X = 1] = 1 - p_e \]
\[ P[Y = \varepsilon|X = 1] = P[Y = \varepsilon|X = 0] = p_e \]

Where the \( \varepsilon \)-symbol is an erasure symbol which is neither 0 nor 1, but a lost symbol.

The Binary Input Additive White Gaussian Noise Channel Model is another useful and simple model, where the input is binary, but the output alphabet is continuous \( y \in \mathbb{R} \):

\[ p(Y = y|X = x) = W(Y = y|X = x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y-x)^2}{2\sigma^2}\right) \]  

(2.3)

### 2.2 Basic Information Theory

In this section we will explain some basic concepts of information theory.

If we have two discrete random variables \( X \) and \( Y \), and we want to observe the outcome of \( Y \) to know what \( X \) is, then we need to know how much information \( Y \) provides of \( X \). This can be conveniently described by:
\[ I(x; y) = \log \left( \frac{P[x|y]}{P[x]} \right) \] (2.4)

Where \( I(x; y) \) is the mutual information between \( x \) and \( y \) and the mutual information between the two random variables \( X \) and \( Y \) is given by averaging (2.4) over the whole alphabet:

\[ I(X; Y) = \sum_{x \in X} \sum_{y \in Y} P[X = x, Y = y] I(x; y) \] (2.5)

In the case where \( Y = y \), the input \( X = x \) is completely determined, i.e \( P[X = x|Y = y] = 1 \), and the mutual information is reduced to:

\[ I(X; Y) = - \sum_{x \in X} P[X = x] \log(P[X = x]) = H(X) \]

Where \( H(X) \) is defined as the entropy of the random variable \( X \).

The capacity of a channel \( C \) is defined as the maximum rate where reliable communication is possible, which for an arbitrary Discrete Memoryless Channel is given by:

\[ C = \max_P I(X; Y) \]

The symmetric capacity of a Binary Input Discrete Memoryless Channel is given by:

\[ I(W) = \sum_{y \in Y} \sum_{x \in X} \frac{1}{2} W(y|x) \log \left( \frac{W(y|x)}{\frac{1}{2} W(y|0) + \frac{1}{2} W(y|1)} \right) \] (2.6)

Where the symmetric capacity denotes a channel that is symmetric such that every output symbol \( y \) has a corresponding symbol \( \bar{y} \) such that \( W(y|0) = W(\bar{y}|1) \) and the input probability \( P(X = x) \) is assumed to be uniform \([54]\).

The capacity of the BSC channel is the following (the channel is symmetric thus it is also the symmetric capacity):

\[ C(BSC) = 1 - H(p_e) \]

Where \( H(p) \) is the binary entropy function \([37]\):

\[ H(p) = -p \log_2(p) - (1 - p) \log_2(1 - p) \]

The capacity of a BEC channel is:

\[ C(BEC) = 1 - p_e \]

18
Another important parameter is the Bhattacharyya parameter of $Z(W)$, here defined for a single channel use of $W : X \rightarrow Y$:

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

(2.7)

The Bhattacharyya parameter, which is a measure of reliability of the channel, is an upper bound on the probability of error for one single use of the channel $W$ under ML-decoding, i.e. evaluating $W(y|0)$ and $W(y|1)$ to decide $x \in \{0,1\}$ [13]. For Binary Input Discrete Memoryless Channels, the Bhattacharyya parameter can be proven to be related to the symmetric capacity as [13]:

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

$$I(W) \geq \log \left( \frac{2}{1 - Z(W)} \right)$$

Thus, when $Z(W) = 0$, $I(W) = 1$ and when $Z(W) = 1$, $I(W) = 0$. For the AWGN channel, the Bhattacharyya parameter is:

$$Z(W) = \exp(1/2\sigma^2)$$

2.3 Modulation

In this thesis we will be using a set of modulation techniques, namely BPSK, QPSK and $M$-ary QAM. In general, modulation is a mapping of a string of $m$ bits in to one of $M = 2^m$ signal waveform.

![BPSK, QPSK and 16QAM modulation](image)

Figure 2-4. BPSK, QPSK and 16QAM modulation.
BPSK (Binary Phase Shift Keying) is the most simple one and is useful when analyzing coding schemes. BPSK modulation is done as follows:

\[ \{0,1\} \rightarrow \{\sqrt{E_s}, -\sqrt{E_s}\} \]

This corresponds to a mapping of 1 bit to 1 signal point. Another popular modulation that maps two bits to one of four signal points is QPSK, seen in Figure 2-4, which is done as:

\[ \{11,01,10,00\} \rightarrow \{\sqrt{E_s} + i\sqrt{E_s}, -\sqrt{E_s} + i\sqrt{E_s}, \sqrt{E_s} - i\sqrt{E_s}, -\sqrt{E_s} - i\sqrt{E_s}\} \]

M-QAM is a higher-level modulation that maps \(m\) bits to one out of \(M\) signal points. In Figure 2-4 the gray-coded 16QAM can be seen.

### 2.4 Measures of Signal-to-Noise Ratio

When measuring performance of different decoding techniques we need to look at the Block Error Rate (BLER) performance versus the Signal-to-Noise ratio, SNR. One convenient measure to use, in the context of coding, is to look at the SNR per bit \(E_b/N_0\), where \(N_0 = 2\sigma^2\). When simulating communication systems, it is common to set \(E_b = 1\) and vary \(N_0\) to achieve different values of the SNR. Another SNR measure is the SNR per symbol \(E_s/N_0\). The SNR per symbol \(E_s/N_0\), is related to SNR per bit \(E_b/N_0\) in the following way with \(m\) being the modulation order:

\[ \frac{E_b}{N_0} = \frac{E_s}{N_0 mR} \]

Which in decibel is:

\[ \frac{E_b}{N_0} (dB) = \frac{E_s}{N_0} (dB) - 10 \log_{10}(mR) \]

### 2.5 Cyclic Redundancy Check

The Cyclic Redundancy Check, mostly referred to as CRC, is an error detection code. This means that it is designed to detect errors rather than correcting errors as Forward Error Correcting codes are designed to do. This is useful as it can alert a receiver that a message is in error.

A CRC of length \(L_{CRC}\) can be represented by a polynomial of the form:

\[ z^{L_{CRC}} + z^k \ldots + z^k + 1 \]

Another convenient representation is the implicit 1, which is best shown through an example:
The length 8 CRC \( z^8 + z^7 + z^6 + z^4 + z^2 + 1 \) in binary form is \((1, 1, 1, 0, 1, 0, 1, 0, 1)\). The implicit 1 notation is 0xea, formed by grouping the binary form \((1, 1, 1, 0, 1, 0, 1, 0, 1)\) = \((s_2, s_1, 1)\) where \(s_2\) and \(s_1\) is converted into hexagonal numbers \(e\) and \(a\), thus forming the implicit-1 notation 0xea.

### 2.6 Puncturing

Puncturing is a commonly used technique to reduce the rate of a code. It is done by systematically or randomly removing bits from transmission. The receiver then has to know where in the code the puncturing takes place, in order to be able to correctly decode the code. Most codes should be able to handle puncturing given that the rate \( R \geq C(W) \).

A puncturing pattern is described by:

\[
p_i = \begin{cases} 
0, & \text{if } i \text{ punctured} \\
1, & \text{else} 
\end{cases}
\]

As an example, if we have the puncturing pattern \( p = (1, 1, 1, 0, 1, 0, 1, 0, 1) \), the codeword \( c = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}) \) will be punctured to \( c_p = (c_1, c_2, c_3, c_5, c_6, c_8, c_{10}) \).

### 2.7 Likelihood Ratio and Log Likelihood Ratio

In many communication systems, we have to deal with conditional distributions such as \( p(y|x) \) and this tends to be somewhat difficult as distributions tend to have small probabilities for certain values of \( y \). Instead of dealing with \( p(y|x) \), where \( x \in \{0,1\} \), it tends to be more convenient to deal with Likelihood Ratios(LR) or Logarithmic Likelihood Ratios(LLR):

\[
\gamma_{LR} = \frac{p(y|x = 0)}{p(y|x = 1)} \tag{2.8}
\]

\[
\gamma_{LLR} = \ln \left( \frac{p(y|x = 0)}{p(y|x = 1)} \right) \tag{2.9}
\]

The LLR is especially convenient for the following reason:

If we plug in the conditional distribution of the AWGN-channel into (2.9) then the BPSK mapping becomes:

\[
\gamma_{LLR} = \ln \left( \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left( -\frac{(y+1)^2}{2\sigma^2} \right) \right) = \ln \left( \frac{\exp\left( -\frac{(y+1)^2}{2\sigma^2} \right)}{\exp\left( -\frac{(y-1)^2}{2\sigma^2} \right)} \right) = \frac{2y}{\sigma^2}
\]
Thus the received $y$ is converted to an LLR-value.

2.8 Interleaving and permutations

Many communication systems will experience so-called bursty errors, which is when many errors occur in succession. Although there are some codes that are well equipped to deal with this, it is common to employ an interleaver in order to spread these errors. An interleaver is defined as a permutation of indices:

$$\pi(n) = k$$

Where $N$ is the interleaver length. The most obvious interleaver is the random interleaver whose permutation sequence $\pi$ is random. One type of permutation/interleaving that is seldom used as an interleaver is the bit-reversal permutation, defined as:

$$\pi(n) = (n_b, n_{b-1}, ..., n_0)_2$$

Where $(n_0, n_1, ..., n_b)_{10} = n$ is the binary representation of the integer $n$. The bit-reversal flips the order of the binary representation of the integer $n$. As an example, for the bit-reversal of length $N = 8$, the binary representation of $n = 4 = (1,0,0)_2$ is permuted to $\pi(4) = (0,0,1)_2 = 1$. 
Chapter 3: 
Basics of Polarization and Polar Codes

In this section we will go through the basics of polarization and polar codes that will be needed for later sections.

3.1 Polarization basics

3.1.1 Recursive Channel Transformation

Polarization is an effect where a set of channel uses $W^N$, each with symmetric capacity $I(W) \in (0,1)$ are transformed to a set of channels $W^N_i$, whose capacities $I(W^N_i)$ tend to 0 or 1, i.e either completely noiseless or completely noisy. This polarization has been proven for memoryless channels and channels with memory [11], and under binary and non-binary input alphabets [12], but here we will only discuss Arikans original binary scheme [13].

Arikans polarization scheme is recursive and starts by combining two independent channels $W(y|x)$ with the same statistics to a vector channel, using our information bits $u$:

$W_2: x^2 \rightarrow y^2$, as:

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2)$$

(3.1)

and for arbitrary $N$:

$W_N: x^N \rightarrow y^N$, as

$$W_N(y_1^N|u_1^N) = W_N(y_1^N|u_1^N G_N)$$

(3.2)

Where $G_N$ is defined as:

$$G_N = B_N F^\otimes n$$

(3.3)

Where $N = 2^n$ and $B_N$ is the bit-reversal permutation matrix and the Kronecker product is defined as:

$$F^\otimes n = F^1 \otimes F^\otimes n-1 = \begin{pmatrix} F^\otimes n-1 & 0 \\ F^\otimes n-1 & F^\otimes n-1 \end{pmatrix}$$

(3.4)
\[ F^i = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \]

This transformation can be seen in the Figure 3-1 for \( W_2 \) and \( W_4 \). The relation between \( u_i \) and \( x_i \) can be seen to be \( G_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \).

Figure 3-1. Recursive Channel Transformation.

A vector channel in itself is not very useful and we would like to have the channel, which in the rest of the thesis, will be called bit-channels, on the form \( W_N^{(i)} : \mathcal{X} \rightarrow \mathcal{Y} \). Therefore, the channel is “split” in the following way for arbitrary \( N \):

\[
W_N^{(i)}(y^n_1, u^n_{1,i-1} | u_i) = \sum_{u_{i+1} \in \mathcal{X}^{n-i}} \frac{1}{2^{n-1}} W_N(y^n_1 | u^n_i) \tag{3.5}
\]

The splitting operation of (3.5) and combining operation of (3.1) can be combined to one single operation, which for \( N = 2 \) is:

\[
W_2^{(1)}(y^2_1 | u_1) = \sum_{u_2 \in \{0, 1\}} \frac{1}{2} W(y_1 | u_1 \oplus u_2) W(y_2 | u_2) \tag{3.6}
\]

\[
W_2^{(2)}(y^2_1, u_1 | u_2) = \frac{1}{2} W(y_1 | u_1 \oplus u_2) W(y_1 | u_2) \tag{3.7}
\]

and for arbitrary \( N \):

\[
W_{2N}^{(2i-1)}(y^{2N}_1, u^{2i-2}_1 | u_{2i-1}) = \sum_{u_{2i} \in \{0, 1\}} \frac{1}{2} W_N^{(i)}(y^{2i-2}_1, u^{2i-2}_1, u_{2i-1} \oplus u_{2i-2} | u_{2i-1} \oplus u_{2i-2}) W_N^{(i)}(y^{2N}_{2i+1}, u^{2i-2}_{2i+1} | u_{2i}) \tag{3.8}
\]

\[
W_{2N}^{(2i-1)}(y^{2N}_1, u^{2i-2}_1 | u_{2i-1}) = W_N^{(i)}(y^{2i-2}_1, u^{2i-2}_1, u_{2i-1} \oplus u_{2i-2} | u_{2i-1} \oplus u_{2i-2}) W_N^{(i)}(y^{2N}_{2i+1}, u^{2i-2}_{2i+1} | u_{2i}) \tag{3.9}
\]

As an example, for the channels of \( N = 4 \), the splitting/combining operations would result in:
\[ W_4^{(1)}(y_1^4 | u_1) = \sum_{u_2 \in \{0,1\}} \frac{1}{2} W_2^{(1)}(y_1^2 | u_1 \oplus u_2) W_2^{(1)}(y_3^4 | u_2) \quad (3.10) \]

\[ W_4^{(2)}(y_1^4, u_1 | u_2) = \frac{1}{2} W_2^{(1)}(y_1^2 | u_1 \oplus u_2) W_2^{(1)}(y_3^4 | u_2) \quad (3.11) \]

\[ W_4^{(3)}(y_1^4, u_1^2 | u_3) = \sum_{u_4 \in \{0,1\}} \frac{1}{2} W_2^{(2)}(y_1^2, u_1 \oplus u_2 | u_3 \oplus u_4) W_2^{(2)}(y_3^4, u_2 | u_4) \quad (3.12) \]

\[ W_4^{(4)}(y_1^4, u_1^3 | u_4) = \frac{1}{2} W_2^{(2)}(y_1^2, u_1 \oplus u_2 | u_3 \oplus u_4) W_2^{(2)}(y_3^4, u_2 | u_4) \quad (3.13) \]

In these four equations (3.10)-(3.13), we can observe some important properties. In equation (3.10), we are not dependent on any decisions \( u_i \), and the decision on \( u_i \) is done as \( \hat{u}_1 = \arg\max_{u_1 \in \{0,1\}} W_4^{(1)}(y_1^4 | u_1) \). In the subsequent equations (3.11)-(3.13), we are dependent on previous decisions \( \hat{u}_1, \hat{u}_2, \hat{u}_3 \). Thus the decoding can be seen to be sequential as it processes bit-by-bit.

### 3.1.2 Polarization

Polarization means that the channels \( W_2^{(1)} \) and \( W_2^{(2)} \) synthesized from \( W \) using equation (3.6) and (3.7) satisfies the following:

\[ I(W_2^{(1)}) \leq I(W) \leq I(W_2^{(2)}) \]

This means that from the \( W(y|x) \), we have synthesized one channel that is better, and one channel that is worse. If we continue this by synthesizing two new channels, say \( W_4^{(1)} \) and \( W_4^{(2)} \) from \( W_2^{(1)} \), and then synthesize two new channels \( W_4^{(1)} \) and \( W_4^{(2)} \) from \( W_2^{(2)} \), then we can see that the capacities have driven away from the original capacity \( I(W) \):

\[ I(W_4^{(1)}) \leq I(W_2^{(1)}) \leq I(W) \leq I(W_2^{(2)}) \leq I(W_4^{(2)}) \]

This comes from the fact that the polarizing operations of (3.6) and (3.7) preserve capacity, meaning that when we apply (3.8) and (3.9) one time:

\[ I(W) = \frac{I(W_2^{(1)}) + I(W_2^{(2)})}{2} \]

And when we apply this recursively in \( n \) steps to create \( N \) bit channels \( W_N^{(i)} \), the capacity is preserved:

\[ I(W) = \frac{\sum_{i \leq N} I(W_N^{(i)})}{N} \]

25
Arikan showed that as $N$ grow [13]:

- The fraction of bit-channels with $I(W_{N}^{(i)}) \approx 1$, will converge to the original capacity of the channel $I(W_{0}^{0})$
- The fraction of bit-channels with $I(W_{N}^{(i)}) \approx 0$, will converge to $1 - I(W_{0}^{0})$
- The fraction of bit-channels that are not equal to 1 or 0 will converge to 0

3.1.3 Principles of Polar encoding and Code construction

Channel coding is trivial in the following two cases:

- $I(W) = 0$, i.e. no communication is possible, no point of employing channel codes
- $I(W) = 1$, i.e. the channel is perfect and no channel codes are needed

The encoding operation for a set of $N$ polarized channels is thus simple:

- If $I(W_{N}^{(i)}) = 0$, we “freeze” this bit-channel, i.e. no information and set $u_{i} = 0$
- If $I(W_{N}^{(i)}) = 1$, we send our information $u_{i} \in \{0, 1\}$ completely un-coded

At the decoder side, the positions of the frozen indices need to be known at the encoder and decoder side and therefore we only need to focus on the non-frozen indices.

![Figure 3-2. Polar Encoding. Shadowed indices of $u$ are frozen, i.e. $u_{i} = 0$](image)

The algorithm or method for choosing which of the indices to put information on is usually referred to as “code construction”, which is the same term that we will use throughout this thesis. The idea is to find the set of indices with the lowest probability of error of the channels $W_{N}^{(i)}$, as $arg\min_{\mathcal{A}_{c}, |\mathcal{A}_{c}|=N-K} \sum_{\mathcal{A}_{c}} P_e(W_{N}^{(k)})$, where $\mathcal{A}_{c}$ is the set of channel indices with high probability of error. An example of the code construction for $N = 8$ and $R = 0.5$ can be seen in Figure 3-2. Here 4 indices will be frozen, denote by shadowed indices. The encoding is equal to $x = uG_{8}$ where $u_{1,2,3,5} = 0$. 

26
3.1.4 Binary Erasure Channel

The BEC is an easy channel to analyze, and for polar codes it has the special property that the code construction can be expressed precisely. The reliability $Z(W)$ of each bit-channel can be shown to be recursive:

$$Z\left(W_n^{(2i-1)}\right) = 2Z\left(W_n^{(i)}\right) - Z(W_{n-1}^{(i)})^2$$  \hspace{1cm} (3.14)

$$Z(W_n^{2i}) = Z(W_{n-1}^{(i)})^2$$  \hspace{1cm} (3.15)

Where the recursion starts with $Z(W_0^{(0)}) = Z(W) = \varepsilon$, where $\varepsilon$ is the erasure probability and $N = 2^n$ is the block length of the code. In Figure 3-3 the recursion of some BECs with different erasure probabilities are shown where the starting Bhattacharyya of the channel is a) $Z(W_0^{(0)}) = 0.2$, b) $Z(W_0^{(0)}) = 0.35$, c) $Z(W_0^{(0)}) = 0.5$ and d) $Z(W_0^{(0)}) = 0.9$. It can be seen as the recursion step $n$ increases, the distributions become more extreme.

![Distribution of Bhattacharyya Parameters](image)

Figure 3-3. a), b), c) and d) Distribution of Bhattacharyya Parameters. After a few steps, the distribution becomes more extreme and the probability of a channel having a capacity not 0 or 1 is very low.

The set of good channels, denoted $\mathcal{A}$ is selected by taking the $K$ lowest values of $Z(W_n^{(i)})$. As we mentioned in Section 2.3, the Bhattacharyya parameter is an upper bound on the error probability of a single use of the channel $W$. In (3.14) and (3.15), the Bhattacharyya parameter will be an upper bound on the probability of error of the bit-channel $W_n^{(i)}: \mathcal{X} \rightarrow \mathcal{Y}$.

By seeing each error event of the bit-channel $W_n^{(i)}: \mathcal{X} \rightarrow \mathcal{Y}$ as independent of other error events $W_n^{(j)}: \mathcal{X} \rightarrow \mathcal{Y}$ for the same use of the channels $W_N$, we can derive an upper bound on the Block Error Probability by summing the Bhattacharyya parameters. This will be an upper bound due to the fact that the error events of channel $W_n^{(i)}$ and $W_n^{(j)}$ are likely to be dependent since if we make an error in our decoding, this error is likely to propagate due to the Successive Cancellation decoder.
Using these Bhattacharyya parameters we can get an upper on the block error rate as:

\[ P_{\text{upper}}(\varepsilon) = \sum_{i \in \mathcal{A}} Z(W_N^{(i)}) \]  

(3.16)

In Figure 3-4, the upper bound of the block error probability (BLER) vs. the rate \( R \) of the code is plotted for block lengths \( N = 2^6, 2^{10}, 2^{15} \) under Successive Cancellation decoding and \( p_e = 0.5 \). As the block length increases, the Bhattacharyya parameters of the non-frozen indices become lower and lower, which decreases the block error rate greatly. The Binary Erasure Channel is a channel that is easier than the AWGN channel that we will focus on in later sections, but Figure 3-4 shows how if the polarization increases, then the performance will greatly increase.

![Binary Erasure Channel](image)

**Figure 3-4. Bound on the block error probability under successive cancellation decoding.**

### 3.1.5 Principles of decoding

![Decoding path](image)

**Figure 3-5. Decoding path under Successive Cancellation Decoding**
Arikan formulated the polar coding scheme using the Successive Cancellation decoder. The successive cancellation decoder is a greedy search algorithm that sequentially makes decisions for each bit using the genie-aided algorithm:

\[
\hat{u}_i = \arg\max_{u_i \in \{0, 1\}} W(y_i^N, u_i^{l-1} | u_i)
\]  

(3.18)

It is genie-aided in the sense that it requires the correct decisions \(u_i^{l-1}\). These are usually unavailable to the decoder, so it can be approximated by setting \(u_i^{l-1} = \hat{u}_i^{l-1}\) as seen in Figure 3-5. This however makes it prone to error propagation as when an error is made, the error will affect future decisions.

### 3.2 Properties and practical issues of Polar Codes

#### 3.2.1 General properties of Polar Codes

- **Error floor**

Error floor is an effect that occurs in many different decoding schemes where an increase in SNR does not notably improve the performance. It can be proven that polar codes exhibit no error floors under Successive Cancellation decoding and to our knowledge [14], no error floors have been reported to be seen through empirical observations. In Figure 3-6, for block length \(N = 256, 1024\) over the AWGN channel, no error floor can be seen down to block error rates of \(10^{-9}\).

![AWGN BPSK, R = 0.5](image_url)

*Figure 3.6. Successive Cancellation decoder BLER under AWGN BPSK channel.*

Arikan and Telatar [15] showed that the block error probability at high SNR scales as \(O(2^{-2n^\beta})\), where \(n = \log_2 N\) and some constant beta \(\beta \approx 0.5\).

- **Recursive structure**
Unlike LDPC and Turbo Codes, Polar Codes have no random parts, meaning that codes are very easy to construct and requires no random or pseudorandom parts. When analyzing LDPC or Turbo Codes, one often analyze an ensemble of codes, but for polar codes this is not necessary.

- **Full rate-compatibility**

  Polar Codes can achieve any rate by unfreezing and freezing more bit-channels \( R = \frac{K}{N}, \)
  \( K = 1, 2, \ldots, N. \)

### 3.2.2 Practical problems of Polar Codes

The polar codes have some obvious issues that need to be solved and here we will list some.

- **Medium-length performance**

  Although it can be shown that polar codes achieve capacity as \( N \) grows, the medium-length performance is not very good compared to other modern codes such as LDPC and Turbo Codes. In Chapter 3 we will discuss and show how to improve the medium-length block error rate performance.

- **Complexity and latency**

  The decoding complexity of the Successive Cancellation is \( O(N \log(N)) \). Which for small \( N \) is roughly linear as \( \log(N) \ll N \). But when the block-length is increased, which is needed to achieve capacity, the complexity becomes an issue since the complexity is no longer linear in \( N \). The other issue is that the Successive Cancellation decoding is a sequential algorithm that processes bit by bit, which makes the algorithm very slow. In this thesis, we will not focus on these issues, but a lot of research has been dedicated to improve both the complexity and latency of the polar codes and remarkable advances have already been made in this field.

- **Rate Compatibility**

  As seen in the beginning section of this chapter, the polar coding matrix \( G_N \) has the size of \( N = 2^N \), which means that the block-length of polar codes are limited to a power of two. This problem will be treated in Chapter 6.

- **Code Construction**

  The problem of code construction, where the construction of the codes are dependent on the channel \( W: X \to Y \) is a relatively unique property. This phenomena thus needs to be fully understood in order to make the codes practical. We will be treating this problem in Chapter 5 and 6.
Chapter 4: Decoding Algorithms

In this chapter, we will implement practical decoding algorithms for polar codes. One of the problems of the original polar codes mentioned in Section 3.2, was that, despite polar codes provably achieving capacity as $N$ grows using the Successive Cancellation decoder, the medium block-length performance was not very good compared to Turbo Codes and LDPC. We will thus first implement the original decoder and then propose and analyze decoders that significantly improve the medium block-length performance.

We will explain, implement and assess the performance of a set of different decoders. We will focus on LLR-based decoders as these are considerably simpler compared to LR and nominal likelihood-based decoders and avoid some of the numerical issues that are present in other decoders. We also assume that we have the frozen and non-frozen sets, respectively $\mathcal{A}$ and $\mathcal{A}_c$.

There are bounds on the block error probability of Successive cancellation decoder similar to Equations 3.16 and 3.17, that can be approximated by methods introduced in Chapter 5 but only for the successive cancellation decoder, and these bounds are not very tight. Thus we are forced to rely on simulations for the evaluation of all of our decoding algorithms.

4.1 LLR-based decoding

For the SC decoder, the definition of LLR-based decoding is as follows:

for $i \in \mathcal{A}_c$, $u_i = 0$

for $i \in \mathcal{A}$:

$$\hat{u}_i = \begin{cases} 0, & \ln \frac{w(y_0, a_0^{-1} | i_0)}{w(y_1, a_1^{-1} | i_1)} > 0 \\ 1, & \text{otherwise} \end{cases} \quad (4.1)$$

Meaning that if the index is frozen, then we set the decoding output to zero and if the index is not frozen, then we will use the LLR in (4.1) to decide the output. Since the construction of polar codes are entirely recursive, we will show the principles by deriving the decoding operations for $N = 2$. 

32
We define the likelihood ratio $\gamma_{LR}$ as $\gamma_{LR} = \frac{w(y, a^{i-1}|0)}{w(y, a^{i-1}|1)}$ and use the recursive formulas (3.6) and (3.7) to derive the LLR denoted $L_i$:

$$L_1 = \ln \frac{w_1^{(i)}(y_1|0)}{w_1^{(i)}(y_1|1)} = \ln \frac{\gamma_{y_1} \gamma_{y_2+1}}{\gamma_{y_1}+\gamma_{y_2}}$$

(4.2)

$$L_2 = \ln \frac{w_2^{(i)}(y_2, a_i|0)}{w_2^{(i)}(y_2, a_i|1)} = \begin{cases} \ln(\gamma_{y_2} * \gamma_{y_1}), & u_1 = 0 \\ \ln(\frac{\gamma_{y_2}}{\gamma_{y_1}}), & u_1 = 1 \end{cases}$$

(4.3)

If $L_{y_1}$ and $L_{y_2}$ are the received LLRs from the channel, then the LLR-based update equations are the following, using suitable approximations [16]:

$$L_1 = sgn(L_{y_1})sgn(L_{y_2}) \min(|L_{y_1}|, |L_{y_2}|)$$

(4.4)

$$L_2 = \begin{cases} L_{y_2} + L_{y_1}, & u_1 = 0 \\ L_{y_2} - L_{y_1}, & u_1 = 1 \end{cases}$$

(4.5)

It can be seen that the equations here are very simple, consisting of sum, sign and comparison operators, which are very suitable for hardware implementations. This makes the LLR-based decoding more robust compared to using nominal distributions values $W(y|0), W(y|1)$ or LR-functions where multiplication of small numbers might be needed. (4.4) and (4.5) are sometimes referred to as f- and g-functions seen in Figure 4-1.

![Figure 4-1. The basic decoding elements. a) is the g-function, equation (4.4), b) is the f-function, equation (4.5).](image)

### 4.2 Successive Cancellation Decoder

For the SC (Successive Cancellation) decoder, the decoding process is done sequentially step by step. The decoder has to make decisions as decoding is on-going and those decisions are then used during the decoding. In the case of frozen indices, the decoder will set the decisions to zero regardless of the value of the LLR. In addition to the LLR-computation (4.4) and (4.5), the decisions in the graph have to be propagated in the graph, through the following equations:

$$u_{2i-1}(k-1) = u_{2i-1}(k) + u_{2i}(k)$$

(4.6)
$u_{2i}(k - 1) = u_{2i}(k)$  \hspace{1cm} (4.7)

Figure 4-2. The six first steps of Successive Cancellation decoding of $N = 8$. Red arrows towards the right indicate computation of LLRs through equation (4.4) and (4.5) and blue arrows to the left indicate decision-propagation through equations (4.6) and (4.7).

The decoding graph for $N = 8$ polar codes and the six first steps of decoding are seen in Figure 4-2. In Figure 4-2 a) the decoder first go down to the deepest level and the first decision-LLRs through the f-function (4.4) from lowest levels up to the first decision-LLRs, but the index is already frozen. b) the LLR of the next index is calculated based on the LLR and decision made earlier through the g-function. c) the decisions a) and b) are propagated through (4.6) and (4.7). d) here we calculate the LLRs first using g-function and then f-function. e) using g-function we calculate a decision-LLR. f) using the previous decision-LLR we make a decision on this index.

The algorithm for decoding is presented below. This decoding algorithm largely follows [17]. Since the decoding is recursive, there will be two recursive functions, `recursivelyCalculateLLRs` and `recursivelyPropagateBits`. `recursivelyCalculateLLRs` computes the LLR computations from left to right and `recursivelyPropagateBits` propagates the decisions made by the decoder in line 11 to 15 in Algorithm 4.1.
**Input:** the received vector $y$, with size $N$

**Output:** approximated information sequence $u$

1. for $\beta = 0, 1, \ldots, N - 1$
2. \[ L(\beta, 0) = 4 \frac{y_\beta}{N_0} \]
3. for $\phi = 0, 1, \ldots, N - 1$
4. recursivelyCalculateLLR($m, \phi$)
5. if $u$ is frozen
6. \[ B_m(\phi, 0) = 0 \]
7. \[ u(\phi) = 0 \]
8. else
9. if $L(0, \phi) > 0$
10. \[ B_m(\phi, 0) = 0 \]
11. \[ u(\phi) = 0 \]
12. else
13. \[ B_m(\phi, 0) = 1 \]
14. \[ u(\phi) = 1 \]
15. if $\phi \mod 2 = 1$
16. recursivelyPropagate($\phi, m$)
17. Return: $u(\phi)_{n=\phi}$

In Algorithm 4.1, the first two lines load the channel LLRs on to the variable $L(\beta, \lambda)$ which holds the LLRs in the graph. Then the loop goes through all of the bits, where lines 6 – 8 set the output to zero if frozen. The lines 10 – 15 are the decision functions that makes the estimate decisions $u_i$ and $B_m(\phi, 0)$, where $B_m(\phi, 0)$ is variable that holds the bit-decisions.

**Algorithm 4.2: recursivelyCalculateLLR($\lambda, \phi$)**

**Input:** layer $\lambda$ and phase $\phi$

1. $\psi = \lfloor \phi/2 \rfloor$
2. if $\lambda = 0$
3. return
4. if $\phi \mod 2 = 1$
5. recursivelyCalculateLLR($\lambda - 1, \psi$)
6. for $\beta = 0, 1, \ldots, 2^{m-\lambda}$
7. if $\phi \mod 2$
8. \[ L(\beta, \lambda) = f[L(2\beta, \lambda - 1), L(2\beta - 1, \lambda - 1)] \]
9. else
10. \[ L(\beta, \lambda) = g[L(2\beta, \lambda - 1), L(2\beta - 1, \lambda - 1)] \]
11. end
In Algorithm 4.2, the LLRs are being calculated recursively at each \( \phi \) through the \( f \) and \( g \)-functions. And in Algorithm 4.3 the propagation of (4.6) and (4.7) are done at odd indices.

**Algorithm 4.3: recursivelyPropagateB(\( \lambda \), \( \phi \))**

**Input:** layer \( \lambda \) and phase \( \phi \)

1. \( \psi = \lfloor \phi/2 \rfloor \)

2. 

3. \( \text{for } \beta = 0, 1, ..., 2^{m-\lambda} \)

4. \( B(2 \cdot \beta) = B(\beta, 0) \oplus B(\beta, 1) \)

5. \( B(2 \cdot \beta + 1, 1) = B(\beta, 1) \)

6. 

7. \( \text{if } \psi \mod 2 = 1 \)

8. \( \text{recursivelyPropagateB}(\lambda - 1, \psi) \)

9. 

The performance vs the other decoders can be seen in Figure 4-6 and 4-7.

### 4.3 Successive Cancellation List Decoder

The main issue of the SC decoder in terms of performance is that once a wrong bit-decision is made, there is no way of correcting that faulty decision, thus a block error is certain. To keep the successive cancellation decoder from making these mistakes, the Successive Cancellation List (SCL) decoder was presented in [17], which drastically improved the performance. It is however worthwhile to notice that list decoders for Reed Muller-codes [18], which are very similar to polar codes [19] in their structure, have been known for a long time.

The list-decoder uses the same decoding operations as the successive cancellation decoder, but instead of only making one decision in line 13 of algorithm 4.4, the decoder will make both decisions and split the decoding paths in to two decoding paths and at the next index, it will split these two paths into four paths. This is then continued until the paths have to be killed off due to too many paths. This killing is based on a path metric that is given to each path. This path metric is an approximation of the likelihood of the path being the correct information sequence \( u \). A low metric denotes a good path and high metric a bad path. Six decoding steps from index two to seven are shown for decoding of length \( N = 8 \).
In Figure 4.3 a) the iteration is at $\phi = 1$, b) The first unfrozen index, which is split in to two decisions, $u_2(l = 0) = 0$ and $u_2(l = 1) = 1$. c) This index is frozen, thus there is no decision to be made but $u_3(0) = 0$ and $u_3(1) = 0$. d) We split our two paths in to four paths, but our list size is $L = 3$, thus one of the split paths will not be extended. e) We again split our paths and then decide which path to use again, according to our path metrics of the split paths. f) Same procedure is repeated at every unfrozen index, but if none of the paths are extended, then the whole path up to root node will be killed off.

The complexity of this decoding algorithm is $O(LN\log(N))$, but it also needs a large amount of memory and a sorting operation to sort the paths for every non-frozen index. Below, in Algorithm 4.4, is the SCL decoding algorithm.

<table>
<thead>
<tr>
<th>Algorithm 4.4: Successive Cancellation List Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong>: the received vector $y$, with size $N$</td>
</tr>
<tr>
<td><strong>Output</strong>: approximated information sequence $u$</td>
</tr>
</tbody>
</table>

1. $l_1 =$ assignInitialPath
2. for $\beta = 0,1,...,N - 1$
   3. \[ L_l(\beta, 0) = 4 \frac{y_{\beta}}{N_0} \]
4. for $\phi = 0,1,...,N - 1$
   5. recursivelyCalculateListLLR($m, \phi$)
   6. for $l = 0,1,...,L - 1$
      7. if $u_\phi$ is frozen
         8. if path $l$ is active
            9. $B_{m}(\phi, 0, l) = 0$
            10. $PM_l = CalcPathMetric(L_l, PM_0)$
            11. $u(\phi, l) = 0$
      12. else
\[
\text{pathPruning}(\phi)
\]
\[
\begin{align*}
&\quad \text{if } i \text{ mod } 2 \neq 1 \\
&\quad \text{recursivelyUpdateListB}(\phi, m) \\
&\quad l_{\min} = \arg \min_{l \in L} PM_l \\
&\quad \text{Return: } \mathbf{u}(\phi, l_{\min})_{n=\phi}
\end{align*}
\]

After the list has been produced, the path with the best path metric is chosen as the output information sequence as seen by line 18. The performance of this decoder for a different set of block lengths and list sizes L are shown in Figure 4-4 a)-d). Below is the algorithm for \text{pathPruning}. We have included \text{recursivelyUpdateListB} and \text{recursivelyCalculateListLLR} in the appendix for reference. They are very similar to the ones for the Successive Cancellation algorithm, except that they always operate over multiple lists.

In Algorithm 4.5, the path-pruning operation is shown. This is one of the most important operations of the List decoder. In line 1 to line 9, all active paths \( p_{\text{active}} \) are extended and their path metrics are computed. In line 10 to line 16, we analyze and decide which of these extended paths \( p_{\text{extended}} \) we should continue on, and which should be killed. This is done by sorting the paths according to their metric and choosing the \( L \) best \( p_{\text{extended}} \). If both extended paths are not chosen as the one of the \( L \) surviving paths \( p_{\text{extended}} \), then we will kill off the root path \( p_{\text{active}} \).

**Algorithm 4.5:** pathPruning

\[
\begin{align*}
&\quad \text{for } l = 0, 1, \ldots, L - 1 \\
&\quad \quad \text{if path } l \text{ is active} \\
&\quad \quad \quad \text{pathMetricForks}(l, 0) = \text{CalcPathMetric}(L^l_{\phi}, PM_l, 0) \\
&\quad \quad \quad \text{pathMetricForks}(l, 1) = \text{CalcPathMetric}(L^l_{\phi}, PM_l, 1) \\
&\quad \quad \quad i = i + 1 \\
&\quad \quad \text{else} \\
&\quad \quad \quad \quad \text{pathMetricForks}(l, 0) = \infty \\
&\quad \quad \quad \quad \text{pathMetricForks}(l, 1) = \infty \\
&\quad \quad \quad \sigma = \min(2i, L) \\
&\quad \quad \quad \text{continueProbability} = \text{sort pathMetricForks} \text{ and set } L \text{ last indices to 0, and } L \text{ first indices to 1} \\
&\quad \quad \text{for } l = 0, 1, \ldots, L - 1 \\
&\quad \quad \quad \text{if path } l \text{ is active} \\
&\quad \quad \quad \quad \text{if both continueProbability of } l \text{ is zero} \\
&\quad \quad \quad \quad \quad \text{killPath}(l) \\
&\quad \quad \quad \text{for } l = 0, 1, \ldots, L - 1 \\
&\quad \quad \quad \quad \text{if both continueProbabilities}(l) = 1 \\
&\quad \quad \quad \quad \quad B_m(0, \phi \text{ mod } 2, l) = 0 \\
&\quad \quad \quad \quad \quad u(\phi, l) = 0 \\
&\quad \quad \quad \quad \quad PM_l = \text{CalcPathMetric}(L^l_{\phi}, PM_l, 0)
\end{align*}
\]
\[ l_{\text{clone}} = \text{clonePath}(l) \]
\[ B_m(0, \phi \mod 2, l_{\text{clone}}) = 1 \]
\[ u(\phi, l_{\text{clone}}) = 1 \]
\[ PM_{l_{\text{clone}}} = \text{CalcPathMetric}(l_{\text{clone}}, PM_{l_{\text{clone}}}, 0) \]

else

if \( \text{continuePaths}(l) = 1 \)
\[ B_m(0, \phi \mod 2, l) = 0 \]
\[ u(\phi, l) = 0 \]

else
\[ B_m(0, \phi \mod 2, l) = 1 \]
\[ u(\phi, l) = 1 \]
Figure 4-4. SCL vs SC decoding for block-lengths a) 128, b) 256, c) 512 and d) 1024 where the rate is 0.5. The SCL performs much better than SC decoding. It can be seen that there is a so-called diminishing return when increasing list-size does not yield a large performance increase and at high SNR, the decoders with different list-sizes will usually perform equally good. Also important to mention is that in d) the performance of the different decoders seem to converge which happens at higher SNRs. This is also observed in [12].
4.3.1 Path Management

We use the same type of path management as described in [16]. A set of important operations that are needed for path management is the killing and copying/opening of new paths. For LLR-based implementations, this is done by having each path represented by a path metric. The path metric in line 11 of Algorithm 4.4 is updated as follows:

$$PM^I_{\phi}(L^I_{\phi}, PM^I_{\phi-1}, \tilde{u}^I_{\phi}) = PM^I_{\phi-1} + \ln(1 + \exp(-(1 - 2\tilde{u}^I_{\phi})L^I_{\phi}))$$ (4.8)

Where \( l \) is the path number, \( \phi \) is the bit index.

4.4 CRC-aided Successive Cancellation Decoder

The SCL decoder has much better performance compared to the SC decoder, but the performance is still not competitive compared to LDPC and Turbo Codes. It was noticed that in many error-cases, the correct code word was in the final output list but not chosen as the corresponding path metric was not the minimum one [17]. The idea is to implement a genie that is capable of finding the correct code word if present in the list. This “genie” can be implemented by attaching a CRC and then selecting the code word that 1) passes the CRC-check and 2) has the best path metric. The scheme is shown in Figure 4-8 and the CASCL algorithm with modification in line 18-20 is shown in Algorithm 4.6. The CRCs are appended on the indices with lowest error probability to make sure that the CRC is very well-protected.

![CRC-aided SCL decoder scheme](image)

**Figure 4-5.** CRC-aided SCL decoder scheme. We attach a CRC and at the output of the SCL decoder we choose the codeword that pass the CRC check.

<table>
<thead>
<tr>
<th>Algorithm 4.6: CRC-aided Successive Cancellation List Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> the received vector ( y ), with size ( N )</td>
</tr>
<tr>
<td><strong>Output:</strong> approximated information sequence ( u )</td>
</tr>
</tbody>
</table>

1. \( l_1 = \text{assignInitialPath} \)
2. for \( \beta = 0,1,...,N - 1 \)
3. \( L_1(\beta, 0) = \frac{y_{\beta}}{N_0} \)
4. 
5. for \( \phi = 0,1,...,N - 1 \)
6. \( \text{recursivelyCalculateListLLR}(m, \phi) \)
7. for \( l = 0,1,...,L - 1 \)
if u is frozen
    if path \( l \) is active
        \( B_m(\phi, 0, l) = 0 \)
        \( PM_i = \text{CalcPathMetric}(L_m, PM_i, 0) \)
        \( u(\phi, l) = 0 \)
    else
        \( \text{pathPruning}(\phi) \)
        if \( i \mod 2 = 1 \)
            \( \text{recursivelyUpdateList}(\phi, m) \)
        for \( l = 0, 1, ..., L - 1 \)
            if path \( l \) does not pass CRC
                \( PM_l = \infty \)
        \( l_{\min} = \arg \min_{l \in L} PM_l \)
    Return \( u(\phi, l_{\min}) \), \( k = \phi \)

The performance is improved considerably, which is shown for a different set of block lengths and list sizes \( L \) in Figure 4-6 a)-d). Notable is that in some cases, by attaching a CRC, CA-SCL4 can outperform SCL64. At a block-length of \( N = 128 \), the BLER performance can be improved more than 1.5 dB compared to SC decoding and roughly 1 dB compared to SCL decoding. At \( N = 1024 \), the improvement over SC decoding can almost be 1 dB and 0.5 dB over SCL decoding. These BLER improvements are very large improvements, especially at low SNR.
Figure 4-6. CA-SCL, SCL vs SC decoding for block-lengths, CRC and rates a) 128, CRC6, 0.45 b) 256, CRC8, 0.47 c) 512, CRC10, 0.48 and d) 1024, CRC24, 0.48. The CA-SCL performs much better than SCL and SC. Different CRCs were used to make a fair comparison between the coding schemes. Simulations were done over AWGN BPSK channel.
4.4.1 CRC for CA-SCL

Attaching a CRC has been shown very beneficial to improve the performance of the channel coding. In order to preserve the information rate, \( R \), the CRC concatenation forces the decoding algorithm to unfreeze more indices \( K + L_{CRC} \), which causes a rate-loss. It is therefore important to not use a CRC which is too long, but is still long enough to keep its error correcting abilities. This effect is shown below in Figure 4-7 where 3 different CRC lengths are used for the same block length.

Figure 4-7. CRCs of different lengths for CA-SCL8 with rates 0.5, block-length \( N = 1024 \) for AWGN BPSK. Short CRCs work best at low SNR and Long CRCs work better at high SNR.

In Figure 4-7 it can be seen that the length of the CRC affects performance greatly. At lower SNR, the rate-loss that is caused by using a bigger CRC affects the performance more than using a large CRC with good error detecting performance, while at high SNR the error detection performance becomes more important and the rate-loss becomes negligible. It is expected that at high SNR, the CRC30 will outperform CRC16, but this is difficult to simulate due to the low block error rates where this is expected to occur.

4.4.2 Performance of different CRC polynomials

Using a good CRC-code according to [20] is very important. In Figure 4-8 a), b) the performance of these different CRCs under CA-SCL8 decoding can be seen. These CRCs are taken from [21] and according to the reference the CRCs have a guaranteed error-detecting performance for certain block-length that it is appended with. However, this error-detecting performance is only guaranteed at error probabilities lower than \( 10^{-5} \), thus it is expected that the performance of the CRCs will become more distinct at very high SNRs. Seeing that CRC24 0xed93bb has the best performance, we will continue to use this CRC throughout the rest of the thesis unless stated otherwise. There might be more powerful CRCs, but since we will not focus on performance in the high SNR region, this CRC is considered good for our purposes.

44
Figure 4-8. Simulation of different CRC polynomials over CA-SCL. a) $N = 512$, b) $N = 1024$. The performance differences are very small at low SNR but can be seen at higher SNR and low error probabilities.

4.5 SCAN Decoder

Figure 4-9. The SCAN decoder with $B$ being a-priori soft bits from previous iterations and $I$ is the amount of iterations.
One major issue of the SC decoder is that it does not output soft bits, nor is it able to process a priori soft bits. This problem is solved by using the Soft CANcellation decoder [22]. The SCAN decoder is capable of processing the output of the channel recursively, which can be used for joint equalization and decoding. This is seen in Figure 4.9 where the decoder has two inputs, one from the channel, and one a-priori input from the previous decoding iteration.

The LLR-based decoding formulas of (4.4) is changed to:

\[ L_{1}(0,0) = L_{0}(0,0) \boxplus [B_{1}(1,0) + L_{0}(0,1)] \]  \hspace{1cm} (4.9)

And (4.5) is changed to:

\[ L_{1}(1,0) = L_{0}(0,1) + [B_{1}(0,0) \boxplus L_{0}(0,0)] \]  \hspace{1cm} (4.10)

Where the bits \( B_{1}(1,0) \) and \( B_{1}(0,0) \) are some type of side-information and \( A \boxplus B = \text{sgn}(A) \text{sgn}(B) \min(|A|,|B|) \)

If we substitute \( B_{1}(0,0) \) with a quantized value \( \pm \infty \), and always set \( B_{1}(1,0) = 0 \), then we get exactly (4.4) and (4.5) as \( \pm \infty \boxplus L_{y} = \pm L_{y} \)

The algorithm that we use for the SCAN decoder is shown in Algorithm 4.7, which largely follows [22].

<table>
<thead>
<tr>
<th>Algorithm 4.7: Soft Cancellation decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> the received LLR vector ( y ), with size N</td>
</tr>
<tr>
<td><strong>Output:</strong> approximated information sequence ( u )</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

46
The performance is shown in Figure 4-10. It can be seen that the SCAN decoder does not exhibit the same type of performance compared to the SCL or CA-SCL decoder. The SCAN decoder has a complexity of $O(I \log N)$, where $I$ is the number of iterations with, which is typically much less complex compared to the SCL-decoder, but the iterative approach incurs more latency in general as we apply the Successive Cancellation decoding many times in succession.

![BLER vs EbNo](image)

*Figure 4-10. Performance of SCAN decoder vs. SC and SCL (no CRC attached) decoder. The first iteration of the SCAN decoder has the same performance as the SC decoder, which is expected as there are no a-priori LLRs at the first iteration. There are so-called diminishing returns where there is no benefit in doing more iterations of the SCAN decoder after around eight iterations.*

**Algorithm 4.8: recursivelyPropagateSoftB**

**Input:** layer $m$ and index $\phi$

1. $\psi = \lfloor \phi/2 \rfloor$
2. 
3. if $\phi \mod 2 = 1$
4.     for $\omega = 0, \ldots, 2^{m-\lambda} - 1$
5.         $B_{2,\omega}(\psi, 2\omega) = B_{2,\omega}(\phi - 1, \omega) \boxplus [B_{2,\omega}(\phi, \omega) + L_{2,\omega-1}(\psi, 2\omega + 1)]$
6.         $B_{1,\omega}(\psi, 2\omega + 1) = B_{1,\omega}(\phi, \omega) + B_{1,\omega}(\phi - 1, \omega) \boxplus L_{1,\omega-1}(\psi, 2\omega)$
7.     if $\psi \mod 2 = 1$
8.         recursivelyPropagateSoftB($\psi, m$)
9.

**Algorithm 4.9: recursivelyCalculateSoftLLR**

**Input:** layer $m$ and index $\phi$

1. if $\lambda = 0$
2. return
3. $\psi = \lfloor \phi/2 \rfloor$
if $\phi \mod 2 = 0$

\[
\text{recursivelyCalculateSoftLLR}(m, \phi)
\]

for $\omega = 0, ..., 2^{m-1} - 1$

if $\phi \mod 2 = 0$

\[
L_\lambda(\phi, \omega) = L_{\lambda-1}(\psi, 2\omega + 1) + B_\lambda(\phi + 1, \omega)
\]

else

\[
L_\lambda(\phi, \omega) = L_{\lambda-1}(\psi, 2\omega + 1) + L_{\lambda-1}(\phi, 2\omega) \oplus B_\lambda(\phi - 1, \omega)
\]

4.6 Selection of Decoding Algorithm

In this chapter we have simulated and shown the performance of the SC, SCL, CA-SCL and SCAN decoder. In Table 1 we have listed the asymptotic complexities of these decoders. Even though CA-SCL and SCL-decoders have a large complexity compared to SCAN and SC, their better performance makes up for the larger complexity and we can also see that at medium-sized block-lengths, this complexity is acceptable if the list-size is not too large. It is also clear that the CA-SCL largely outperforms SCL through the help of a concatenated CRC, thus going forward, we will mostly be using the CA-SCL as a standard decoding algorithm while comparing different methods for code construction, puncturing and fading channels with low list-sizes.

<table>
<thead>
<tr>
<th>Decoding algorithm</th>
<th>SC</th>
<th>SCAN</th>
<th>SCL</th>
<th>CA-SCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymptotic Complexity</td>
<td>$O(N\log(N))$</td>
<td>$O(N\log(N))$</td>
<td>$O(LN\log(N))$</td>
<td>$O(LN\log(N))$</td>
</tr>
</tbody>
</table>

Table 1. Asymptotic complexity of the decoding algorithms.
Chapter 5:  
Code Construction Techniques

In this chapter we will explain and assess the performance of a set of different techniques for code constructions. In Section 3.1.3 we defined the problem of code construction and in this chapter we will discuss methods for the BPSK AWGN channel.

5.1 Code Construction problem

As mentioned in Section 3.1.3, the problem of code construction is the problem of selecting the good indices on which to put information on:

\[
\mathcal{A}_c = \arg\min_{\mathcal{A}_c \subseteq \mathbb{Z}_2^{N-K}} \sum_{k \in \mathcal{A}_c} P_e(W_N^{(k)})
\]

The only channel for which the code construction can be exactly described is the BEC channel as mentioned in Section 3.1.3. For arbitrary channels, the problem is a lot more difficult and we will show the issue of gaining the exact error probability for arbitrary channels from the following example:

Let \( X = \{0,1\} \) and \( Y = \{0,1,2,3\} \) with \(|Y| = 4\).

Then with the two transform techniques (3.6) and (3.7), the conditional probabilities will be:

\[
p(y_1, y_2 | x_1) = p(Z_1 | x_1)
\]

\[
p(y_1, y_2, x_1 | x_2) = p(Z_2 | x_2)
\]

Which will have an output alphabet that will have the size of:

\[
|Z_1| = |Y^2| = |\{(1,1), (1,2), ..., (2,1), (2,2), ..., (4,4)\}| = 16
\]

\[
|Z_2| = 2|Y^2| = |\{(1,1,0), ..., (4,4,1)\}| = 32
\]

It is clear that if we continue this recursion, even for moderate sized output alphabets \(Y\), not to mention output alphabets on \(\mathbb{R}\), the explosive growth is too complex and the memory requirements are too large.
From this example, it is clear that we have to find some suitable approximations and that there will be some computational complexity involved in doing a suitable approximation. In this chapter we will show the performance of a couple of code construction techniques at medium block-lengths. And in the last section we will also look into their complexities. As mentioned in the beginning of Chapter 4 and in Section 3.1.4, there are ways to put bounds on the block error probability if we can obtain the individual error probabilities of each bit-channel. But our methods use different metrics and approximations for the error probability, such as Bhattacharyya parameter and approximations of bit error probability, so we will resort to simulating all of the code construction methods.

5.2 Channel parameter dependent methods

As explained in the previous section, polar code constructions, in a theoretical sense, need to be tailored to a specific channel [23]. For example, the BPSK AWGN channel with $EbN0 = 2dB$ and the same channel with $EbN0 = 3 dB$ might require different code construction. As we will see in Section 5.4.1, to reconstruct the code construction for every channel parameter is considered infeasible and in most modern systems, the SNR-estimation is not accurate enough to ensure that both transmitter and receiver observe the same SNR, which would be a requirement for such a construction method.

However, as it turns out, there are methods for constructing codes that are independent of the channel parameter. In Section 5.2 we will explore some channel parameter dependent methods and in Section 5.3 common channel parameter independent methods will be shown.

As will be seen, in order to construct the codes for our channel, we will need to decide a channel $\mathcal{W}(y|x)$ over which to construct the code for. Since we are only interested in the AWGN channel, we need to decide an SNR on which to design the code around. In order to show how this design-SNR affects the BLER performance, we simulate the performance of a range of design-SNRs to 1) see how the performance is affected by the code construction and 2) find the optimal code construction.

All simulations are done considering a BPSK AWGN channel.

5.2.1 Heuristic

One of the first proposed methods is the Heuristic Method [19]. The idea is to use exactly the same recursion as that of the BEC channel:

$$Z(W_n^{2i-1}) = 2Z(W_n^i) - Z(W_{n-1}^{i+1})^2 \quad (5.1)$$

$$Z(W_n^{2i}) = Z(W_{n-1}^i)^2 \quad (5.2)$$

However, instead of using the Bhattacharyya parameter $Z(W_0^0) = \varepsilon$ of the BEC as the beginning point of the recursion, the suggestion was to use either the Bhattacharyya parameter of the channel or the approximation:
\[ Z(W'_0) = 1 - C(W) \]

For the AWGN channel in our case, we have for the AWGN channel:

\[ Z(W'_0) = \exp\left( -\frac{1}{2} \frac{E_b}{N_0_{\text{design}}} \right) \quad (5.3) \]

The performance of this code construction over the AWGN channel can be seen for a set of different design-SNRs in Figure 5-1 a) with CA-SCL8 decoding, CRC24, \( R = 0.5 \) and in Figure 5-1 b) with SC decoding and \( R = 0.5 \). It can be seen that using the wrong design-SNR can cause large performance degradations at higher SNRs.

In Section 5.4, the different schemes are compared.

![Figure 5-1](image)

**Figure 5-1.** In a) CA-SCL8 at \( N = 1024 \), b) SC at \( N = 1024 \), both with \( R = 0.5 \) over the AWGN BPSK channel using Heuristic code construction. Performance is roughly the same for lower SNR while at high SNR there is a larger difference.
In both Figure 5-1 a) and 5-1 b), it can be seen that at lower SNRs, the BLER stays constant across the whole range of design SNRs. However, the performance starts to differ at higher SNRs. In Figure 5-2 a) and b) we can see the BLER performance at different design SNRs when the EbN0 is fixed. The best design SNR for CA-SCL8 at this EbN0 is 4 dB and around 4.5 dB for SC.

![Figure 5-2. Heuristic code construction, a) the BLER performance of CA-SCL8 with fixed EbN0 = 2.5 dB, b) BLER performance of SC with fixed EbN0 = 3.5 dB.](image)

### 5.2.2 Tal-Vardy Degradation Method

One of the earliest and most popular techniques for code construction of Polar Codes is the so-called Tal-Vardy method. In the paper [24], the authors lay out a set of techniques where the channel \( W(y|x) \) is degraded/upgraded with respect to the original channel, which creates a channel that has a smaller output alphabet. Worth to mention is that the idea of channel degradation is nothing new and this type of channel degradation has been used in the construction of LDPC codes [25].

The problem of the exponentially growing output alphabet \( Y \) is solved by degrading the channel \( W(y|x) \) to a channel \( Q(z|x) \), where the size of the output alphabet \( |Z| < |Y| \), which will cause a small capacity loss \( C(Q) = C(W) - \epsilon \), but make the channel more easy to deal with.

The operation of reducing the channel output alphabet, with \( x \in \{0,1\} \), is done as follows:

\[
Q(z|x) = \begin{cases} 
W(z|x), & \text{if } z \notin \{z_{1,2}, \bar{z}_{1,2}\} \\
W(y_1|x) + W(y_2|x), & \text{if } z = z_{1,2} \\
W(\bar{y}_1|x) + W(\bar{y}_2|x), & \text{if } z = \bar{z}_{1,2}
\end{cases}
\]  

(5.4)

Where \( \bar{y} \) is the conjugate symbol of \( y \), which for AWGN channel implies \( W(\bar{y}|0) = W(y|1) \).

This algorithm takes the output symbols \( y_1 \) and \( y_2, \bar{y}_1 \) and \( \bar{y}_2 \) and merges these four symbols into two symbols in the output alphabet \( Z \) as seen in Figure 5.3. Applying this degrading merge once reduces the output alphabet by 2 symbols. This is repeatedly applied until one has reached the wanted output alphabet size \( \mu \). The question is now which output
symbols \( y_{1,2} \) and \( \bar{y}_{1,2} \) to merge. This is done by selecting symbols \( y_i \) that creates the smallest capacity difference [24]:

\[
\Delta C[z_i] = \Delta C[W(y_i|0), W(\bar{y}_i|0), W(y_{i+1}|0), W(\bar{y}_{i+1}|0)] = \\
C(W(y_i|0), W(\bar{y}_i|0)) + C(W(y_{i+1}|0), W(\bar{y}_{i+1}|0)) - \\
C(W(y_i|0) + W(y_{i+1}|0), W(\bar{y}_i|0) + W(\bar{y}_{i+1}|0))
\]

Where the capacity \( C \) for two output symbols is calculated as:

\[
C(p_a, p_b) = -(p_a + p_b) \log \left( \frac{p_a + p_b}{2} \right) + p_a \log(p_a) + p_b \log(p_b)
\]

<table>
<thead>
<tr>
<th></th>
<th>( W )</th>
<th>( Q_1 )</th>
<th>( Q_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W(y_4</td>
<td>0) )</td>
<td></td>
<td>( Q(z_1</td>
</tr>
<tr>
<td>( W(y_3</td>
<td>0) )</td>
<td>( Q(z_2</td>
<td>0) = W(y_3</td>
</tr>
<tr>
<td>( W(y_2</td>
<td>0) )</td>
<td>( Q(z_1</td>
<td>0) = W(y_1</td>
</tr>
<tr>
<td>( W(y_1</td>
<td>0) )</td>
<td>( Q(z_1</td>
<td>1) = W(y_1</td>
</tr>
<tr>
<td>( W(y_1</td>
<td>1) )</td>
<td>( Q(z_2</td>
<td>1) = W(y_3</td>
</tr>
<tr>
<td>( W(y_2</td>
<td>1) )</td>
<td>( Q(z_3</td>
<td>1) = W(y_4</td>
</tr>
<tr>
<td>( W(y_3</td>
<td>1) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( W(y_4</td>
<td>1) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>\bar{y}</td>
<td>= 8 )</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-3. Channel degradation \( W \gg Q_1 \gg Q_2 \) where the alphabet output size is reduced by two symbols for each degradation but the capacity is reduced for each step.

5.2.2.1 Tal-Vardy Degradation algorithm for Polar Codes

In this section, we will explain the details of the Tal-Vardy implementation as well as some problems that have not yet been explained in the previous section. We will here restrict ourselves to the AWGN channel.

The AWGN channel \( W(y|x) \) has a continuous output \( y \in \mathcal{R} \), which means that we have to degrade the AWGN channel to a specific output alphabet size \( \mu \). First we define the following:

Let \( f(y|1) \) and \( f(y|0) \) be the pdf functions of the output given 0 and 1, \( y \in \mathcal{R} \) and \( \lambda(y) = \frac{f(y|0)}{f(y|1)} \), and the set partitions:

\[
A_i = \left\{ y \geq 0 : \frac{i - 1}{\mu} \leq \lambda(y) < \frac{i}{\mu} \right\}
\]

(5.5)

\[
A_v = \left\{ y \geq 0 : \frac{v - 1}{\mu} \leq \lambda(y) \leq 1 \right\}
\]

(5.6)
where \( v = \mu / 2 \) and

\[
C[\lambda] = 1 - \frac{\lambda}{\lambda + 1} \log_2 \left( 1 + \frac{1}{\lambda} \right) - \frac{1}{\lambda + 1} \log_2 (\lambda + 1)
\]  

(5.7)

The degrading operation on the AWGN channel is then defined as:

\[
Q(z_i|0) = \int_{A_i} f(y|0)dy
\]  

(5.8)

\[
Q(z_i|1) = \int_{A_i} f(-y|0)dy
\]  

(5.9)

Integrating (5.8) and (5.9), we obtain:

\[
Q(z_i|0) = \int_{A_i} f(y|0)dy = \frac{1}{2} \text{erf} \left( \frac{A_i^d - 1}{\sqrt{2}\sigma} \right) - \text{erf} \left( \frac{A_i^d - 1}{\sqrt{2}\sigma} \right)
\]  

(5.10)

\[
Q(\tilde{z}_i|0) = \int_{A_i} f(y|0)dy = \frac{1}{2} \text{erf} \left( \frac{A_i^d + 1}{\sqrt{2}\sigma} \right) - \text{erf} \left( \frac{A_i^d + 1}{\sqrt{2}\sigma} \right)
\]  

(5.11)

Where:

\[
A_i^d = \left\{ y : \frac{i}{v} \right\}, \quad i \in [0,v]
\]  

(5.12)

are a set of discrete points instead of a partition as in (5.5) and (5.6) and \( \text{erf} \) is the complementary error function \( \text{erf}(x) = \frac{1}{\sqrt{\pi}} \int_x^\infty \exp(-t^2)dt \)

The Algorithms 5.1 and 5.4 are largely based on the algorithm in [24], but here we provide some more details of our implementation.

<table>
<thead>
<tr>
<th>Algorithm 5.1: Tal-Vardy AWGN degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( E_b N_0 ) optimization parameter, fidelity parameter ( \mu ), block length ( N = 2^n )</td>
</tr>
<tr>
<td><strong>Output:</strong> a ranking of channel parameters based on reliability</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
else
    \( W = \text{circleConvolution}(Q) \)
    \( Q = \text{degradingMerge}(W, \mu) \)

for \( k = 1, 2, \ldots, \mu / 2 \)
    \( P_e(i) = P_e(i) + \frac{1}{2} \min(Q(k|0), Q(k|1)) \)

channel ranking = indices of sorted \( P_e \)

Return: channel ranking

In line 1 to 3, we initiate the algorithm by degrading the continuous output AWGN-channel. Line 5 to 16 is the main loop. Line 9, \textit{squareConvolution} is the convolution defined by (3.6) in Section 3.1 and \textit{circleConvolution} is the convolution defined by (3.7). The Algorithm \textit{degradingMerge} of line 12 follows the outline of [20], and we will for briefness not list it here. The string \( b \) defines whether channel is polarized through equation (3.6) or (3.7). For instance, \( W_N^{(1)} \) the bit channel distribution will only go through \textit{squareConvolution} \( n \) times, making it a very bad bit-channel. In line 15, we approximate the error probability of the bit channel by using the approximation: \( P_e \left( W_N^{(i)} \right) = \sum_{k=1}^{\mu / 2} \min(W_N^{(0)}(y_k|0), W_N^{(i)}(y_k|1)) \)

And lastly, the channel is sorted according to the error probabilities of the bit-channels and the \( N - K \) indices are selected as frozen channels.

Below, Algorithm 5.2 and 5.3 explains how to easily calculate the convolutions in line 9 and 11.

**Algorithm 5.2: SquareConvolution**

**Input:** channel \( Q \)

**Output:** square-convoluted \( W_{\text{sorted}} \)

1
2

// \( W_{\text{N}^2} \) is an array of size \( N^2 \)
2
3
4

// \( \text{LLR}_W \) is an array of size \( N^2 \)
3
4
5
6
7
8
9
10
11
12

for all \( y_1, y_2 \) of \( Q(y|0) \)
5
6
7
8

\( W_{\text{N}^2} = \frac{1}{2} (Q(y_1|1) \cdot Q(y_2|1) + Q(y_1|0) \cdot Q(y_2|0)) \)
9
\( W = \text{vec}(W_{\text{N}^2}) \) // stacking of the column vectors of \( W_{\text{N}^2} \)
7
8
9
10

for all \( y_1, y_2 \) of \( Q(y|0) \)
5
6
7
8

\( \text{LLR}_{\text{N}^2} = \log \left( \frac{Q(y_1|1) \cdot Q(y_2|1) + Q(y_1|0) \cdot Q(y_2|0)}{Q(y_1|0) \cdot Q(y_2|1) + Q(y_1|1) \cdot Q(y_2|0)} \right) \)
9
\( \text{LLR}_W = \text{vec}(\text{LLR}_{\text{N}^2}) \)
8
9
10
11

\( i_{\text{sorted}} = \text{sorted LLR}_W \)
10
11
12

\( W_{\text{sorted}} \) is sorted by the indices of line 10
12
13
14
15

Return: \( W_{\text{sorted}} \)
**Algorithm 5.4: degradeAWGN**

**Input:** $\sigma$ and fidelity parameter $\mu$  
**Output:** degraded AWGN $Q$

1. $v = \mu/2$
2. $A_0 = \text{small number}$
3. $A_{v+1} = \infty$
4. for $k = 1, 2, ..., v$
   5. find $y$ where $C \left[ \frac{w(y|0)}{w(y|1)} \right] = k/v$
   6. $A_k = y$
7. for $k = 1, 2, ..., v$
   8. $Q(k|0) = 1/2 \left( \frac{\text{erf}(A(k + 1) - 1)}{\sqrt{2\sigma}} - \frac{\text{erf}(A(k + 1) - 1)}{\sqrt{2\sigma}} \right)$
   9. $Q(k|1) = 1/2 \left( \frac{\text{erf}(A(k + 1) + 1)}{\sqrt{2\sigma}} - \frac{\text{erf}(A(k + 1) + 1)}{\sqrt{2\sigma}} \right)$
10. $\text{Return: } Q(y|0)$ and $Q(y|1)$, $|y| = v$
Some practical issues arising with the Tal-Vardy method are that the distributions of the intermediate steps become too extreme for the absolute worst and best channels. This can be solved by making sure that the bit-channels with extreme distributions are automatically put in the set of perfect/noisy channels. This happens at medium-length for the bit-channels close to $i = 0$ and $i = N - 1$.

![Graph](image)

**Figure 5-4.** In a) CA-SCL8 at $N = 1024$, b) SC at $N = 1024$, both with $R = 0.5$ over the AWGN BPSK channel using Tal-Vardy code construction.

In Figure 5-4 a) the BLER performance of the Tal-Vardy degradation method under CA-SCL8, CRC24 and $R = 0.5$ and in Figure 5-4 b) the SC decoding performance under the same code construction algorithm can be seen. The performance does not start to diverge until lower SNRs.

In Figure 5-5 a) the EbN0 is constant at $2.5dB$ while the design SNR is varied for the CA-SCL algorithm and b) the EbN0 is constant at $3.5dB$ while the design SNR is varied. The interesting thing to note, is that the best design SNR for the two different decoding algorithms are different: with CA-SCL having the best performance at around 2.5dB and SC somewhere around 3.75dB. This phenomena of different decoders having different optimal code constructions is also noted in [26].
5.2.3 Gaussian Approximation

The Gaussian approximation [27] was proposed as a way of making the Density Evolution [28] code construction less complex by only evaluating one parameter, the expectation of the received channel LLR.

Given that we send the all-zero code-word across the channel, the distribution of the LLR over the AWGN channel is:

\[ LLR_i^{(1)}(y_i) \sim \mathcal{N}\left(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right) \]

By approximating the distribution of the intermediate LLRs in the decoding graph as Gaussian, we only need to concern ourselves with the mean LLR in between each step. The recursion then becomes:

\[ E[LLR^{2l-1}_n] = \phi^{-1}(1 - (1 - \phi(E[LLR^l_n])^2)) \]

\[ E[LLR^{2l}_n] = 2E[LLR^l_{n-1}] \]  \hspace{1cm} (5.12)

\[ \phi(x) = \begin{cases} 
1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{\infty} \tanh\left(\frac{u}{2} \exp\left(-\frac{1}{4x}\right)\right) du, & x > 0 \\
1, & x = 0 
\end{cases} \]  \hspace{1cm} (5.13)

(5.14) can be approximated as [29]:

58
\[ \phi(x) = \begin{cases} \frac{1}{2} e^{-\frac{x}{\sqrt{\pi}}} \left( \frac{3}{x} \right)^{3/2} \exp(-\frac{x}{4}) (1 - \frac{3}{x}) + \frac{1}{\sqrt{\pi}} e^{-\frac{x}{4}} (1 - \frac{1}{7x}), & x > 10 \\ \exp(\alpha x^\gamma + \beta), & x < 10 \end{cases} \] (5.15)

\[ \alpha = -0.4527, \beta = 0.0218 \quad \text{and} \quad \gamma = 0.86 \]

Algorithm 5.5 shows the algorithm for the Gaussian Approximation.

<table>
<thead>
<tr>
<th>Algorithm 5.5: Gaussian Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( EbN_0 ) optimization parameter</td>
</tr>
<tr>
<td><strong>Output:</strong> a ranking of channel parameters based on reliability and LLRs</td>
</tr>
<tr>
<td>1. ( \sigma^2 = (2R10^{EbN_0/10})^{-1} )</td>
</tr>
<tr>
<td>2. ( E[LLR^2] = \frac{2}{\sigma^2} )</td>
</tr>
<tr>
<td>3.</td>
</tr>
<tr>
<td>4.</td>
</tr>
<tr>
<td>5. for ( k = 1,2,...,N )</td>
</tr>
<tr>
<td>6. for ( j = 1,2,...,N/2 )</td>
</tr>
<tr>
<td>7. ( E[LLR^2_{k,j}] = \phi^{-1}(2 \phi(E[LLR^1_{k-1}]) - \phi(E[LLR^1_{k,j}]))^2 )</td>
</tr>
<tr>
<td>8. ( E[LLR^1_{k,j}] = 2E[LLR^1_{k-1}] )</td>
</tr>
<tr>
<td>9.</td>
</tr>
<tr>
<td>10. channel ranking by ( E[LLR^1_{n}] \text{=} N )</td>
</tr>
<tr>
<td>11.</td>
</tr>
<tr>
<td>12. <strong>Return:</strong> channel ranking</td>
</tr>
</tbody>
</table>

In line 7 of Algorithm 5.5 (5.12) is implemented by expanding the square product as the \( \phi \)-values, which occasionally become very small. The \( \phi \) and \( \phi^{-1} \) functions are implemented as lookup-table and this lookup table needs a large range and precision. However, the approximation (5.15) still has numerical issues when LLRs become too big. This issue can be apparent even at medium-length codes, but is easily dealt with by clipping the LLRs and including the bit-channels among the good bit-channels. This usually only happens for the LLRs at higher indices where the channels can be considered to be “infinitely good”.

a)
Figure 5-6. In a) CA-SCL8 at $N = 1024$, b) SC at $N = 1024$, both with $R = 0.5$ over the AWGN BPSK channel using GA code construction.

The Gaussian approximation BLER performance, as seen in Figure 5-6 a) for CA-SCL8, CRC24 and $R = 0.5$ and in Figure 5-6 b) for SC decoding and $R = 0.5$, have similar performance as the Tal-Vardy algorithm. In Figure 5-6 b) we can see that at low design SNR the performance is bad and the reason for this is unknown.

In Figure 5-7 we simulate the GA BLER performance for a fixed EbN0 with design SNR varied. The best performance for CA-SCL is at 2.5 $dB$, and around 3.5 $dB$ for SC.

Figure 5-7. Gaussian Approximation code construction, a) BLER performance for CA-SCL8 with fixed $EbN0 = 2.5 \, dB$, b) SC with fixed $EbN0 = 3.5 \, dB$.

5.3 Channel parameter independent methods

As explained in Section 5.2, there have been methods developed that are independent of channel parameters.
5.3.1 Code Construction ensemble average

To show that channel parameter independent code constructions are possible and also perform well, we here present a simple novel algorithm for generating a Code construction based on ensemble average of Gaussian Approximation. Similar constructions can be done based on Tal-Vardy or Heuristic technique.

Algorithm 5.6: Code Ensemble average

<table>
<thead>
<tr>
<th>Output: a ranking of channel parameters based on reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 fidelity parameter $\mu$</td>
</tr>
<tr>
<td>2 $designEbN0 = (0, \mu, 2\mu, ..., EbN0_{\text{max}})$</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4 for $j = 0, ..., \frac{EbN0_{\text{max}}}{\mu} - 1$</td>
</tr>
<tr>
<td>5 [ LLR = \text{GaussianApproximation}(designEbN0(j)) ]</td>
</tr>
<tr>
<td>6 $LLR_{\text{avg}} = LLR_{\text{avg}} + LLR / (\text{sum}(LLR))$</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8 channel ranking = sort $LLR_{\text{avg}}$</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10 Return: channel ranking</td>
</tr>
</tbody>
</table>

The performance in comparison with the other techniques can be seen in Figures 5-5 a)-c).

5.3.2 Polarization Weight

One simple code construction technique was suggested in [30] also called $\beta$-expansion. The channel ranking is done through a channel metric as follows:

$$W_i = \sum_{j=0}^{n-1} b_j * 2^{j/4}, \text{ where } (b_1, b_2, ..., b_n) = i \text{ is the binary expansion of an integer } i$$

For instance, for $N = 8$; $W_1^B = (0, 1.00, 1.1892, 2.1892, 1.1421, 2.1421, 2.6034, 3.6034)$, thus for $R = 3/8$, the frozen indices would be $A_c = (1,2,3,5,6)$.

The channels are then sorted according to their channel metrics where a larger weight implies a more reliable channel. The performance in comparison with the other techniques can be seen in Figure 5-5 a)-c).

5.4 Code construction comparison

In this section, we will compare the performance of different code construction techniques for different rates, different block lengths and decoding methods.
5.4.1 Complexity considerations

In this section, we will look into some complexity consideration of the code constructions and some practical issues. We will not include our Code-Average construction method as it is only a proof-of-concept. It is important to mention that in order for the polar codes to work, both receiver and transmitter need the exact same code construction, otherwise the decoding is certain to fail.

In Table 2, we have compared the computation time of our code construction algorithms. This is done on Matlab on an Intel Xeon CPU E5-2687W with 128GB RAM memory. It can be seen that the Tal-Vardy algorithm is very slow compared to the fastest algorithm, the heuristic code construction or the Polarization Weight.

<table>
<thead>
<tr>
<th>Block length ( N )</th>
<th>64</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic</td>
<td>1.8224e-4</td>
<td>1.2273e-4</td>
<td>2.5483e-4</td>
<td>5.3775e-4</td>
<td>0.0012</td>
</tr>
<tr>
<td>PW</td>
<td>0.0031</td>
<td>0.0127</td>
<td>0.0253</td>
<td>0.0510</td>
<td>0.1023</td>
</tr>
<tr>
<td>GA</td>
<td>0.0471</td>
<td>0.1851</td>
<td>0.3686</td>
<td>0.7350</td>
<td>1.4748</td>
</tr>
<tr>
<td>Tal-Vardy</td>
<td>0.9962</td>
<td>5.1754</td>
<td>11.1978</td>
<td>24.5210</td>
<td>53.4213</td>
</tr>
</tbody>
</table>

Table 2. Computation time in seconds for block length \( N \). Tal-Vardy can be seen to be a very slow algorithm compared to all other code construction techniques.

For a practical implementation of the algorithm, such as in hardware, there are some very important considerations and problems. First we need an algorithm that does not require a high amount of precision as this would cost a lot of hardware resources. Secondly, the algorithm should not require large amount of memory. Thirdly, the computations involved should not be very complex. For instance, in the Gaussian approximation, there is a rather complicated \( \phi \) function that needs to be evaluated.

Due to the performance and flexibility offered by the Gaussian Approximation (which we will see in Chapter 6, we will henceforth use the Gaussian Approximation to represent channel independent methods and use the polarization weight as a channel-independent code construction.

5.4.2 BLER comparison of code constructions

The performance for SC and CA-SCL for \( N = 512, N = 1024 \) and \( N = 2048 \) with \( R = 0.5 \) can be seen in Figures 5-8 a)-c). The reason why we compare under these different block-lengths is that at lower block-lengths the difference in performance tends be very small, explained in Section 5.4.3 and that we want to stay at medium block-lengths. At lower SNRs, the performance different is not more than 0.05 \( dB \) but as one starts to approach higher SNR and lower block-error rates, the performance difference can be up to 0.20 \( dB \). Simulating at SNRs higher than 2.5 \( dB \) tends to require a very large amount of blocks to reliably determine the BLER, thus we have not included simulations at very high SNR, where we expect the difference to be larger.

62
Figure 5-8. Code construction test for CA-SCL at block-lengths a) 512, b) 1024, c) 2048.

5.4.3 BLER performance discussion

As can be seen in Figure 5-8 a)-c), the code construction methods has performance that is very close despite using similar techniques and we will here discuss this.
In [31], it is found that there is an order of the capacities of the bit-channels that is independent of the SNR of the channel. As an example, for $N = 16$ and the bit-channel $W_{16}^{(1)}$ will be degraded compared to $W_{16}^{(2)}$, which is denoted as:

$$W_{16}^{(1)} \preceq W_{16}^{(2)}$$

which means that $I(W_{16}^{(1)}) \leq I(W_{16}^{(2)})$ and $Z(W_{16}^{(1)}) \geq Z(W_{16}^{(2)})$.

The reason why this relationship holds is because $W_{16}^{(1)}$ will go through four polarization-steps, all of them being (3.6) of Section 3.1, at each time making the channel worse, while $W_{16}^{(2)}$ will go through three steps of (3.6), at each step making the channel worse and then one step of (3.7), which will make the channel better compared to $W_{16}^{(1)}$. And this relationship holds for any channel $W(y|x)$

There are two different relations based upon the binary expansion of the integer $i = (i_0, i_1, ..., i_b)$ that can make a channel degraded regardless of the channel:

- $W_{16}^{(k)}$ is degraded to $W_{16}^{(j)}$ if the binary expansion of $j$ has a larger Hamming weight than the binary expansion of $k - 1$. We call this degradation 1.
  For instance, for $k = 11$, the binary expansion is $k - 1 = 10 = (1, 0, 1, 0)_2$ and $j = 15$, $j - 1 = 14 = (1, 1, 1, 0)_2$, $W_{16}^{(11)} \preceq W_{16}^{(15)}$.

- $W_{16}^{(k)}$ is degraded to $W_{16}^{(j)}$ if the binary expansion of:
  $$k - 1 = (k_0, ..., k_{n-2}, k_{n-1} = 0, k_n = 1, k_{n+1}, ..., k_b)_2$$
  and
  $$j - 1 = (j_0, ..., j_{n-2}, j_{n-1} = 1, k_n = 0, j_{n+1}, ..., j_b)_2.$$
  Where $j_i = k_i$, for all $i \in N, i \neq n, n - 1$. We call this degradation 2. For instance, for $k - 1 = 6 = (0, 1, 1, 0)_2$ and $j - 1 = 10 = (1, 0, 1, 0)_2$, $W_{16}^{(7)} \preceq W_{16}^{(11)}$.

These two degradation relations lead to an order of degradations $W_{16}^{(1)} \preceq \cdots \preceq W_{16}^{(k)}$, that is called a “partial order”. This order can be visualized using a Hasse diagram where an arrow between to indices denotes a degradation order. An example for this for $N = 16$ can be seen in Figure 5-9. In here, a single-lined arrow denotes a degradation through the first relation and a double-lined arrow denotes a degradation through the second relation. As can be seen in Figure 5-9, a set of indices are already ordered, but the relationship between some bit-channels are unclear, such as indices three and four. For the already ordered channels, there would be no need to evaluate their error probabilities, since they are independent of the channel, thus also any channel parameter.
Figure 5-9. The partial order where the number indicates the index of the bit channel $W_N^{(i)}$. Double-lined arrow denotes degradation 1 and one-lined arrow denotes degradation 2. If there exist no arrow between two nodes (or indices), the degradation order is unknown.

For the AWGN channel, it is shown in [30] that the degrading ordering of the channels are even more obvious. For instance, for $N = 16$, the ordering is completely determined and will not change with the SNR, but at $N = 1024$ the order is not completely determined for a subset of channels. This offers a possible explanation why the code construction methods that we have compared in this section performs roughly similar.
Chapter 6: Puncturing

For modern communication systems, where spectral efficiency needs to be utilized to the max, there is a strong need for flexible block lengths and rates. As an example of the latest communications systems currently being discussed, 5G, the code rate have to be able to vary from rate 1/12 up to 8/9 with 1-bit granularity, which in other words means that the block-length should be able to be any output block-length of integer $M$. As explained earlier, the rate of polar codes can be varied as $R = K/N$, $K = 1, 2, ..., N - 1$, but the block length of the polar codes are however constrained to what we in this section will call a mother code-length $N = 2^n$, where $n$ is an integer, $N = 2, 4, ..., 128, 256, 512, 1024, ...$. The polar codes therefore need to be made rate-compatible in order to be practical for most modern communication systems.

In this section we will explain a couple of popular puncturing methods, and show their performance and how the code construction impacts the performance of the overall system. We will show that for some puncturing schemes, the code construction can be very important.

For polar codes, there has recently been a large amount of research on methods for making rate-compatible polar codes and the methods that we will explore are some of the most popular methods out of many that has been proposed in academia and industry.

We introduced puncturing in Section 2.6 and we will use the same conventions in the following sections.

6.1 Quasi-Uniform Puncturing

Quasi-Uniform Puncturing (also referred to QUP) [32] is a rather simple technique to puncture the polar codes. The algorithm is the following:

1. Set $p_{\text{initial}} = (0_{1\rightarrow N-M}^{1-M}, 1_{1}^{N})$
2. Set the puncturing pattern $p = \text{bitreversal}(p_{\text{initial}})$
3. Calculate the frozen indices

The QUP is very simple, but in order to successfully apply it, there is a need to take the puncturing into account when calculating the frozen indices. This is done by setting the expected LLRs to 0 and the LLRs are calculated as follows:
\[
E[L_{\text{LLR}}] = \begin{cases} 
0, & \text{if } p_k = 0 \\
\frac{2}{\sigma^2}, & \text{if } p_k = 1
\end{cases}
\] (6.1)

\[
E\left[L_{\text{LLR}}^{k+\beta}\right] = \phi^{-1}(1 - (1 - \phi(E[L_{\text{LLR}}^{k+\beta}]))(1 - \phi(E[L_{\text{LLR}}^{k+\beta+\delta}])))
\] (6.2)

\[
E\left[L_{\text{LLR}}^{k+\beta}\right] = E\left[L_{\text{LLR}}^{k+\beta} + L_{\text{LLR}}^{k+\beta+\delta}\right]
\] (6.3)

For the input to the decoder, the LLRs of the punctured indices will be \(y_p = \ln\left(\frac{W(y|0)}{W(y|1)}\right) = 0\), since we do not know anything about \(y\), \(W(y|0) = W(y|1)\). Equation (6.1), (6.2) and (6.3) are implemented as Algorithm 6.1.

**Algorithm 6.1: QUP AWGN**

**Input:** \(EbN_0\) optimization parameter

**Output:** a ranking of channel parameters based on reliability and LLRs

\[
\sigma^2 = (2R10^{EbN_0/10})^{-1}
\]

1. \(N = 2^{\log(M)} \quad / / \ N \text{ is referred to the mother block-code}
2. \(E[L_{\text{LLR}}] = \frac{2}{\sigma^2}
3. \) for \(j = 0, 1, ..., N - 1
4. \quad if \(p_j = 1
5. \quad \quad E[L_{\text{LLR}}] = \frac{2}{\sigma^2}
6. \quad else
7. \quad \quad E[L_{\text{LLR}}] = 0
8. \)
9. for \(i = 2, 3, ..., n + 1
10. \delta = 2^{i-2}
11. for \(\beta = 1, 2^{i-1}, ..., N
12. \quad \) for \(k = 0, 1, ..., \delta - 1
13. \qquad E\left[L_{\text{LLR}}^{k+\beta}\right] = \phi^{-1}(1 - (1 - \phi(E[L_{\text{LLR}}^{k+\beta}]))(1 - \phi(E[L_{\text{LLR}}^{k+\beta+\delta}])))
14. \qquad E\left[L_{\text{LLR}}^{k+\beta+\delta}\right] = E\left[L_{\text{LLR}}^{k+\beta}\right] + E[L_{\text{LLR}}^{k+\beta+\delta}]
15. \)
16. channel ranking by \(E[L_{\text{LLR}}]_{n+1}\)
17. \)
18. **Return:** channel ranking

67
Figure 6-1. CA-SCL16 performance with Quasi-Uniform Puncturing using Gaussian Approximation and Polarization Weight for output block length $M = 400$, mother block-code $N = 512$.

To see the importance of code construction when puncturing, we simulate some high and low rates $R = 1/8, 1/2$ and $3/4$ with the same output block length $M = 400$ using CA-SCL16 with CRC24 in Figure 6-1. Using the same output block length $M$ means that the same puncturing pattern is used for all rates, where only the code construction and rate changes. It can be seen that very large performance losses are observed at higher block lengths if channel-independent code construction is used. The reason for this large performance loss is due to the fact that the code construction is not taking the puncturing into account when calculating code constructions. An empirical proof of this can be seen in Figure 6-2 where PW and GA code constructions, without accounting for the puncturing through (6.1), (6.2) and (6.3), are compared with GA when we do account for the puncturing for $R = 3/4$, using CA-SCL16 and CRC24.
Figure 6-2. CA-SCL16 performance with QUP, $M = 400$, $N = 512$. We compare the performance when taking the puncturing into account, "GA" and when not taking the puncturing into account when doing the code construction, "PW" and "GA non-recalc". Large performance degradations are seen when not taking the puncturing into account.

6.2 Wang-Liu Method

Wang-Liu shortening method [33] is another simple puncturing method that is based on puncturing indices that are highly polarized.

Given the polar encoding matrix $G_N = F^\otimes m$ (we describe it without the bit-reversal here) and each column vector of $G_N$ denoted $g_i$ and the Hamming weight of the vector column vector as $w(g_i)$, the shortening is done through Algorithm 6.2.

<table>
<thead>
<tr>
<th>Algorithm 6.2: Wang Liu Puncturing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Code length $M$</td>
</tr>
<tr>
<td><strong>Output:</strong> Shortened encoding matrix $G_M$ or $p$</td>
</tr>
</tbody>
</table>

1. $N = 2^{\lceil \log(M) \rceil}$ // $N$ is referred to the mother code
2. initialize $p$ to all ones
3. index each column $l = \{1, 2, ..., N\}$
4. for $i = 0, ..., N - M - 1$
5. calculate the column weight $g$ of each column of $G_{N-i}$
6. find the index $j$ of a column with column weight $1$
7. set $p(i) = 0$
8. delete the column $j$ and the row where the $1$ is located in $G_{N-i}$
9. delete $j$ from $l$
10. **Return:** $G_M$ or the puncturing pattern $p$

The algorithm is best explained through an example for $N = 8$. 

69
First we calculate the Hamming weight of the column vectors as \( w(g_i) = (8, 4, 4, 4, 2, 2, 1) \). Then we can see that \( i = 8 \) has the lowest value. Thus we therefore delete column 8 and the row 8 where the 1 is located:

\[
G_7 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{pmatrix}
\]

The column weight vector for \( G_7 \) is \( w(g_i) = (7, 3, 3, 1, 3, 1, 1) \), thus we can choose between three different indices to delete, and in this case we choose \( i = 4 \), giving us:

\[
G_6 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1
\end{pmatrix}
\]

\( G_6 \) is thus our encoding matrix.

It was realized that the last column of \( G_N \) will always have one of the smallest column vector weights \( w(g_i) \). One simple rule for deleting columns from the encoding matrix is to always remove the last index. This leads to the following algorithm in [33]:

1. Set \( p_{\text{initial}} = (1_1^M, 0_1^{N-M}) \)
2. \( p = \text{bitreversal of } p_{\text{initial}} \) - this is because \( G_N \) is described without bitreversal \( B_N \)
3. Set indices \( \{M + 1, M + 2, ..., N\} \) as part of pre-frozen indices and select the other frozen indices using GA
4. Encode by using \( x = uG_N \) and then puncture using \( p \)

An example of the puncturing and pre-freezing of the Wang-Liu puncturing can be seen in Figure 6-3. In this case the output block-length is \( M = 6 \).
The encoding is similar to the QUP, but the decoding has an important distinction, which can be seen by looking at the left lattice of Figure 6-4. In this figure the bit-reversal matrix in the decoding matrix $G_N$ has been left out, thus $G_N = R^\otimes n$. Since we know that the output of the punctured indices is zero, regardless of what information is encoded, we can set the received LLRs to the decoder to $LLR = \ln \left( \frac{W(y|0)}{W(y|1)} \right) = +\infty$, since $W(y|0) = 1$ and $W(y|1) = 0$.

In this sense, the Wang-Liu method does not puncture, but rather shorten the code, as no information is lost. This method is also sometimes referred to as “Known-bit puncturing”.

6.2.1 Code Construction for the Wang-Liu method

Code construction for the Wang-Liu method is slightly different compared to QUP. In order to compute the code construction for the Wang-Liu method we need to calculate the code construction for the right lattice of Figure 6-4. It is similar to the normal case, but certain indices will not go through as many steps of polarization through (3.6) and (3.7) of Chapter 3.1. This is done through Algorithm 6.3.
Algorithm 6.3: Gaussian Approximation Wang Liu AWGN

**Input:** $EbN_0$ optimization parameter  
**Output:** a ranking of channel parameters based on reliability and LLRs

\[
\sigma^2 = (2R_110^{EBNO_{40}})^{-1}
\]

for $j = 0, 1, ..., N - 1$

if $p_j = 1$

\[
E[LLR^j_j] = \frac{2}{\sigma^2}
\]

else

\[
E[LLR^j_j] = 0 \quad // This one will be automatically frozen
\]

for $i = 2, 3, ..., n + 1$

$\delta = 2^{i-2}$

for $\beta = 1, 2^{l-1}, ..., N$

for $k = 0, 1, ..., \text{delta} - 1$

if $E[LLR^k_{i-1}] = 0$

\[
E[LLR^{k+\beta}_{i}] = E[LLR^{k\beta}_{i-1}]
\]

\[
E[LLR^{k+\beta+\delta}_{i}] = E[LLR^{k\beta+\delta}_{i-1}]
\]

else

\[
E[LLR^{k\beta}_{i}] = \phi^{-1}(1 - \phi(E[LLR^{k\beta}_{i-1}]))(1 - \phi(E[LLR^{k\beta+\delta}_{i-1}]))
\]

\[
E[LLR^{k+\beta+\delta}_{i}] = E[LLR^{k\beta}_{i}] + E[LLR^{k\beta+\delta}_{i-1}]
\]

channel ranking by $E[LLR^j_{n+1}] = N$

**Return:** channel ranking

---

**Figure 6-5.** CA-SCL16 performance for Wang Liu puncturing using Algorithm 6.3, Gaussian Approximation and Polarization Weight for output block length $M = 400$ and $N = 512$. "GA" denotes when applying GA code construction when not applying the puncturing pattern and "GA recal" means that we account for the puncturing as in Algorithm 6.3.
One option would be to simply not re-calculate the code construction but first freeze $\{M + 1, M + 2, ..., N\}$ and then use the same code construction for the rest of the bit-channels as for the non-punctured case $M = N = 2^n$. We simulate this in Figure 6-5 along with code construction in Algorithm 6.3 and PW using CA-SCL16 and CRC24. At lower rates, the performance becomes more dependent on taking the puncturing into account through Algorithm 6.3. This is directly the opposite of QUP where the performance degradations due to not accounting for the puncturing, occurs at higher rates.

6.3 Scattered Puncturing

The Scattered puncturing (also sometimes referred to as bit-interleaved(BIV) puncturing), presented in [34], is derived from the Wang-Liu method with some variations.

1. Set the puncturing pattern $p = (1^M, 0^{N-M})$.
2. Bit reverse the indices $\{M + 1, M + 2, ..., N\}$ and set them as part of frozen indices and select the other frozen indices using GA.

An example of the puncturing of the Scattered puncturing can be seen in Figure 6-6. In this case the output block-length is $M = 6$. It can be seen that it is basically the reverse of the Wang-Liu algorithm.

![Figure 6-6. Depiction of the pre-freezing of $u$ and puncturing of $x$. This is similar to the Wang-Liu method.](image)

Just like for the Wang-Liu method, re-calculating the code construction given the puncturing pattern is very important and we use Algorithm 6.3, but using the scattered puncturing pattern $p$. If the index is punctured then the LLR to the decoder will be set to $+\infty$, just like in the Wang-Liu method.
Figure 6-7. CA-SCL16 performance for Scattered Puncturing using Gaussian Approximation and Polarization Weight for output block length $M = 400$.

In Figure 6-7 we simulate the rates $R = 1/8, 1/2$ and $3/4$ for CA-SCL16 and CRC24. As can be seen in Figure 6-7, scattered puncturing does not have any large performance degradations at low/high rates due to not taking puncturing into account in the code construction as in QUP and Wang Liu puncturing. The scattered puncturing is therefore very suitable to use with PW code construction as the performance difference is very small.

6.4 Polarization-preserving puncturing

Polarization-preserving [35] is a relatively simple method that revolves around dividing up the puncturing into two blocks, roughly $p = \{0_1^{N_1}, 1_1^{N_2}, 0_1^{N_3}, 1_1^{N_4}\}$. The idea is to try to puncture the indices with low reliability to preserve the polarization of the unfrozen indices, as the name indicates.

The puncturing pattern is given by the following steps:

1. Lower block puncturing; set $p_i = 0$ for indices $i = \{1, 2, ..., \min(N - M, N/4)\}$

2. Higher block puncturing; If $N - M > N/4$ set $p_i = 0$ for:

   $i = \left\lfloor \frac{N}{4} + 1, ..., \frac{N}{4} + \left\lfloor \frac{(N - M) - N/4)}{2} \right\rfloor \right\rfloor$

   $i = \left\lfloor \frac{N}{2} + 1, ..., \frac{N}{2} + \left\lfloor \frac{(N - M) - N/4)}{2} \right\rfloor \right\rfloor$

3. Bit-reverse the puncturing pattern $p_b = \text{bitreverse}(p)$ and use this pattern for puncturing.

4. Freeze all punctured bits $p$ and use any code construction technique to find the best indices.
For decoding, the LLRs of the punctured bit-channels will be set to zero.

The code construction can be done in the same way as for the QUP, but using the puncturing pattern $p$ described above. The performance comparison of using (6.1), (6.2) and (6.3) and channel-independent code construction can be seen in Figure 6-8 for rates $R = 1/8, 1/2, 3/4$ using CA-SCL16 and CRC24.

![Diagram](image)

**Figure 6-8.** CA-SCL16 performance for Polarization-preserving puncturing using Gaussian Approximation and Polarization Weight for output block length $M = 400, N = 512$.

From Figure 6-8, it can be seen that polarization preserving has exactly the same performance as QUP. This is due to the fact that at $M = 400$, the amount of punctured indices is $N - M = 112 < N/4$, which means that the puncturing pattern will be the same. For comparison, when $N - M > N/4$, as shown in Figure 6-9, the performance of the two methods will differ.

### 6.5 Performance comparison

In this section we show simulations of the BLER performance for all four puncturing methods at different rates and block-lengths. In order to get a good understanding on how the different code construction perform we have simulated rates from $R = 5/6$ to $R = 1/10$. In each graph we keep the information rate $K$ constant in order to assess the puncturing pattern, which is different from the previous when we assessed the code construction with a constant output block-length $M$. The code construction used in these simulations is GA.
Figure 6-9. CA-SCL8 performance for rates 5/6, 2/3 and 1/2 with $K = 1000$.

Figure 6-10. CA-SCL8 performance for rates 1/2, 1/3 and 1/4 with $K = 500$.

Figure 6-11. CA-SCL8 performance for rates 1/3, 1/6 and 1/10 for $K = 200$. 

76
In Figure 6-9, for higher rates we can see that the Wang Liu method and Scattered method have the best performance while the QUP and polarization-preserving have performance degradations up of to 1dB at $R = 5/6$. As we go to lower rates in Figure 6-10 and 6-11, QUP and Polarization-preserving can be seen to have better performance. For rates such as $R = 0.5$, the puncturing methods tend to have similar performance and only when we go to higher or lower rates the performance of the different methods tend to diverge. This is also observed in [36].

What can be concluded from this is that QUP and Polarization-preserving tends to work better at lower rates and that Scattered and Wang Liu work better for high rates, and that re-calculating the code construction given a puncturing pattern is very important for optimal and stable performance.

6.6 Discussion on performance

One important question that arises is why some code constructions have really bad performance at certain rates when not recalculating the code construction. For instance, for Polarization-preserving and QUP, seen in Figure 6-1 and Figure 6-8, the performance degradation can be more than 0.5dB.

A possible explanation is that the Scattered and Wang Liu puncturing optimize the minimum distance of the code, which would explain why these methods work better at high SNR, since for minimum distance is important for good performance at high SNR [37]. On the other hand, the QUP and Polarization-preserving tend to puncture indices that correspond to lower indices, where the bit-channels are considered to be very bad. Exactly how the puncturing affects the low SNR performance is unclear, but it seems as if there is a tradeoff between keeping the channels polarized and having good distance properties of the codes.

Another possible explanation is that the puncturing changes the universal partial ordering mentioned in Section 5.4.3. The partial ordering should be universal but the question remains if this holds if a bit-channel is punctured either by setting the input LLR of the decoder to 0 or $\infty$. 
Chapter 7:
Polar Coding for fading channels and higher level modulation

Any practical wireless system will at some point experience some type of fading. There are many techniques for combatting wireless fading, such as multi-antenna diversity, detection etc. Channel codes, which are employed by most wireless communication systems today, thus also need to be examined over fading channels.

In this chapter we will introduce channel coding for the fading channel and over higher-order modulation. A question that has arisen is whether polar codes can actually work over fading channels due to the fact that during the construction of polar codes in Chapter three, we assumed that the channel that we construct the codes for $W(y|x)$ is independent and identically distributed.

We will in this section show that polar codes can work very well over fading channels through simulations.

7.1 Polar Codes for higher level modulation

So far, we have only focused on BPSK modulation over the AWGN channel. Before we show results of polar codes over the fading channel, there is a need to clarify and show that polar codes work also for higher order modulation, such as QPSK and QAM.

In the previous chapter, when we defined the channels, we use $W(y|x)$, where for BPSK $x \in \{0,1\} \rightarrow \{-1,1\}$. In the case of higher order modulation $x \in \{x_1,...,x_M\}$, so it would be natural to assume that we would have to deal with the conditional probabilities $W(y|x_1), W(y|x_2), ..., W(y|x_M)$. However, analyzing these probabilities and creating a decoder for this, is considered to be very computationally complex. There is research on non-binary construction and polarization of polar codes in [12] and [37]. We will however assume that the higher order modulation is demodulated after receiving, to a set of binary variables $x_{mod} \in \{1,...,M\} \rightarrow X^m$. Even though we can no longer assume that the channel is i.i.d, we will still use the same decoding and code construction as before.

We however need to make sure that the same performance “relations” hold as before. We therefore do similar simulations as in Section 3.3 and 4.4 to make sure that the same “relations” hold. For these simulations, it is more convenient to use the $E_sN_0$ as the Signal-
to-Noise Ratio, defined in Section 2.5. In Figure 7-1 and Figure 7-2 the performance of different list-sizes are shown. In the 64QAM modulation, high SNRs are however required due to the higher-order modulation, but the improvements for different list-sizes are still the same. In Figure 7-3, the simulation is over a set of design-SNRs and we can see that it performs very similar to the BPSK case where the best design-SNR is around 2.0 to 2.5dB.

Figure 7-1. CA-SCL performance for different list-sizes over the AWGN QPSK channel at block-length $N = 512$, $R = 0.5$ using GA to do the code construction.

Figure 7-2. CA-SCL performance for different list-sizes over the AWGN 64QAM channel at block-length $N = 512$, $R = 0.5$ using GA to do the code construction.
7.2 Polar Codes over fading channels

Channel coding for the fading channel is a very open research topic. Typically, channel codes are analyzed over more simple channels such as BEC or AWGN channels. Even for the easiest models of fading channels, even for slow fading channels, the mathematical analysis tends to become difficult. As polar codes are dependent on the underlying channel, it was unclear whether they would work for fading channels. There have nevertheless been some research on polar codes for fading channels. In [38], a code construction for Rayleigh fading channels is presented and in [39] and [40], block fading channels for polar codes are discussed.

For the decoding in fading channels, the performance strongly depends on the minimum Hamming distance of the codes, which for polar codes can be approximated as [32]:

$\text{d}_{\text{min}} \leq \min_{i \in \mathcal{A}} \text{wt}(i-1)$

Where the $\text{wt}(i)$ is the weight of row $i$ in the encoding matrix $G_N$. For example, for $N = 1024$, $R = 0.5$ and code construction design SNR 2.5dB, the lowest non-frozen index in $\mathcal{A}$ is $i = 128$, thus $\text{wt}(127) = 16$, $\text{d}_{\text{min}} \leq 16$. However, when concatenating a CRC as in the CA-SCL decoder, the minimum distance is increased.

7.2.1 Independent Rayleigh Fading Channels

The independent AWGN Rayleigh fading channel is defined as:

$y_k = r_k x_k + n_k, 1 \leq k \leq N \quad (7.1)$
Where $r_k = r_k^l + j r_k^q$, $r^l$ and $r^q$ are $\mathcal{N}(0,1/\sqrt{2})$ distributed and $n_k$ is $\mathcal{N}(0,\sigma^2)$ distributed (like in the AWGN case). Let us assume that, at the receiver, we have full Channel Side Information (CSI), meaning that we know all of the $R_k$. Demodulation is then done by multiplying (7.1) with the conjugate $\bar{r}_k$:

$$z_k = y_k \bar{r}_k = |r_k|^2 x_k + \bar{r}_k n_k \tag{7.2}$$

The reason we have chosen the Rayleigh fading channel is because it is a rather tough channel to handle since every symbol experiences independent fading. In terms of practical systems, the independent Rayleigh fading channel is not considered very realistic compared to other modern channel models, but as a measure to test the polar codes, it is a very good channel because its statistics are will not be in the same as in previous sections, as we assumed in earlier sections. Thus if the polar codes can work over the Rayleigh fading channel with an assumption of Gaussian channels, we expect them to work well over any channel.

In (7.2), we can see that the symbols $x_k$ all experience independent fading. However, when $y_k$ are demodulated, the LLRs $y_j = \ln \frac{W(y_1)}{W(y_0)}$ will not experience independent fading. An effective way to make the LLRs experience more independent is to employ an interleaver to spread out the fading across the LLRs. We will perform simulations using both no interleaver and with a random interleaver.

In [41], the Turbo Codes under Rayleigh fading with and without CSI is investigated and in [42] LDPC are investigated under the Rayleigh fading channel where it is seen that even with CSI both cases cause large performance degradations compared to AWGN. This is also seen in [43] under Maximum Likelihood-decoding.

### 7.2.2 Code Construction for Fading Channels

By for instance using the Tal-Vardy algorithm, we can apply the code construction to an arbitrary channel if we have the conditional probabilities $W(y|x)$, but for most channels they are either difficult to compute or otherwise unavailable. And through our simulations, it is clear that polar codes are capable of working well without applying any specialized code constructions tailored to the fading channel, since the noise is assumed to be Gaussian. As mentioned, there are code constructions for the fading channels as in [38], where the mean LLR is tracked as in the GA code construction, but it require us to numerically solve some complex functions and it is not clear if they would work well over the AWGN channel. We therefore use the GA code construction.

### 7.2.3 Simulation over the Rayleigh Fading Channels

In this section we show some simulations over the fading channels to prove that they can work over the fading channels.
Figure 7-4. CA-SCL8 performance over Rayleigh fading QPSK channel without interleaver for the rates 5/6, 2/3 and 1/2 and $K = 1000$. Scattered puncturing can be seen to have very bad performance.

Figure 7-5. CA-SCL8 performance over Rayleigh fading QPSK channel without interleaving for the rates 1/3 and 1/4 for $K = 500$.

In Figure 7-4 we simulate four different puncturing methods for the rates $R = 5/6, 2/3, 1/2$ under CA-SCL8 without an interleaver. And in Figure 7-5 we simulate the same puncturing methods for the rates $R = 1/3, 1/4$ without an interleaver. As we would expect, there is a considerable higher SNR required for decoding with low error probability due to the fading. As can be seen here, the puncturing methods perform completely different here, with scattered puncturing having bad performance and polarization-preserving being the method with the best performance across all rates.
In Figure 7-6 and 7-7 we simulate the exact same as above but with a random interleaver. The random interleaver is simply generated as a sequence of uniform unique random integers \( \pi = (\pi_1, \pi_2, ..., \pi_N) \). It can here be seen that the interleaver provides an almost 2dB gain at rate \( R = 5/6 \). This gain decreases as the code rate decreases, to about 0.5 dB at rate \( R = 1/4 \). In Figure 7.6 the relations in performance between the different puncturing methods are similar to that of the AWGN channel.

![Figure 7-6. CA-SCL8 performance over Rayleigh fading QPSK channel with interleaving for the rates 5/6, 3/4 and 1/2 for \( K = 1000 \).](image1)

![Figure 7-7. CA-SCL8 performance over Rayleigh fading QPSK channel with interleaving for the rates 1/3 and 1/4 for \( K = 500 \).](image2)

### 7.2.4 Discussion on performance

In last section we showed the performance of different code construction techniques over the AWGN QPSK Rayleigh fading channel. The most notable difference in the simulation is that the scattered puncturing with no interleaving performs has bad performance across all rates. This is also clearly observed in the comparison in [44], but no comparison between different puncturing methods are done here. The probable explanation why the scattered puncturing performs considerably worse is due to the fact that the scattered puncturing is
the only method which punctures a large block of indices. This is can be seen as a problem with the modulation of polar codes which is better explained through the example in Figure 7-8. In Figure 7-8, the D-block represents the demodulation of the faded QPSK signals $y_k$. $y_k$ are the LLR input the decoder. When puncturing in a block, the LLRs tend to be more correlated when they enter the first processing steps of the decoder, the $f$-function. And when the puncturing is spread out, the puncturing is less correlated when they enter the first processing steps of the decoder. Since taking advantage of this correlation is very difficult as we would most likely have to change the decoding steps of Equations (4.6) and (4.7), it is easier to reduce the correlation by introducing a interleaver that reduces correlation, which improves the performance seen in Figures 7-4 to 7-7, especially for the scattered puncturing where the puncturing is done in a block as in Figure 7-8 a).

![Figure 7-8. How the QPSK signal is demodulated into LLRs to the polar decoder. a) represents the scattered puncturing and b) represents the Wang Liu puncturing for $N = 8$. The shadowed indices are the punctured indices set to either 0 or $\infty$.](image)

84
Chapter 8: Discussion and future work

8.1 Conclusion

In this thesis we have compared the performance of different set of important techniques for polar codes; code construction and puncturing and its performance under AWGN and fading channels. We have compared several candidate code construction and puncturing methods from both academia and industry which are currently being considered for 5G.

After analyzing the results of our simulations, we can see that depending on what technique is used, the performance will vary greatly, especially with respect to the puncturing methods. We have shown that for optimal performance without puncturing, the code construction is very important. We have also shown that puncturing which is independent of a channel parameter can work for AWGN channels but the performance will be greatly degraded for certain rates and methods, especially QUP and polarization-preserving puncturing.

We have shown that the polar codes can work over higher-level modulation and Rayleigh fading channels without employing a code construction tailored to those channels. Our simulations have however shown that there are significant performance degradations, especially at higher rates if no interleaver is employed. This performance degradation is even more significant for certain puncturing methods.

8.2 Future work

The last one to two years have seen an explosion of research on Polar Codes, which can be attributed to the fact that the industry has taken a very large interest on Polar Codes now that it is being adopted in to the 5G standards. There are however some problems that are still open within puncturing and code construction:

- **Optimal puncturing** – It is possible that there are puncturing pattern not yet discovered that have better performance than the ones compared in this thesis.
- **Joint code construction and puncturing** – In all of our puncturing methods that we have investigated, the puncturing is specified first and then the code construction is calculated, thus the topic of code construction and puncturing are often treated
independently. It is possible that optimal performance can be given if we jointly optimize the puncturing and the code construction.

- **Compound channel code constructions** – As polar codes are not universal, meaning that the optimal code constructions are not the same for different channels, it would be interesting to see if there is a way to for instance create a code construction which is jointly optimized for the AWGN and Rayleigh fading channels and see how these would compare in comparison on other more complex channels such as the Rician Fading Channel or modern channel models such as EPA, ETA and ETU. The joint optimization could be done by weighing the code construction of the Rayleigh Fading Channel in [39] and the AWGN channel code constructions presented in this thesis.

- **Interleaver design** – We employed a random interleaver for simplicity, but employing a simple interleaver that have good spreading properties and can have any size \( M \) is very important.

- **Reducing the complexity of code constructions** – In Section 5.4.3 we discussed the partial order of code constructions. Since a lot of channels are ordered naturally, we can focus on code constructions on a subset of the channels. In [45], code construction can be shown to have sub-linear complexity, meaning that the complexity of code construction is reduced from \( \mathcal{O}(N\log(N)) \) to complexity less than \( \mathcal{O}(N) \) for any code construction technique over any Binary Symmetric Channel. Specifically, it was shown that the code construction is only needed for a fraction \( \frac{3}{\log^2(N)} \) of the bit-channels. This can open up for channel dependent code constructions implemented in hardware since only a small fraction of channels need to be computed.

Apart from puncturing, code construction and polar codes for fading channels, there are several very hot topics within polar codes that would be interesting to look into. These include:

- **Decoders** - such as Sequential decoders [45], Sphere decoders [46] and adaptive list decoders [47], some of which promise better performance and lower complexity.

- **Latency reductions** - such as Simplified SCL decoders [48].

- **Achieving any block-length \( N \)** - using mixed non-binary kernels \( F^1 \) as in [49] and [50].
Bibliography


[29] D. Wu, Y. Li och Y. Sun, "Construction and Block Error Rate Analysis of Polar Codes over AWGN Channel based on Gaussian Approximation," *IEEE Communications Letters*, vol. 18, nr 7, 2014.


