A low-complexity successive cancellation decoder for polar codes

Orion Afisiadis

Supervised by: Prof. Andreas Peter Burg - EPFL,
Assist. Prof. Christos Dimakis - AULT

Assisted by: Alexios Balatsoukas-Stimming - EPFL

Submitted for the degree of THMMY
Aristotle University of Thessaloniki
2013-14
Acknowledgements

From EPFL, I owe special thanks to my supervisor, Professor Andreas Burg for the acceptance of working my diploma thesis in his lab and his excellent supervision guidance. Special thanks goes to Alexios Balatsoukas-Stimming for his everyday guidance, supervision and support throughout the thesis and not only. I owe big gratitude to TCL secretary Ioanna Paniara for all the help with the administrative issues. Thanks to all guys and girls(?) of TCL.

From AuTH, I would like to acknowledge with gratitude my co-supervisor Assistant Professor Christos Dimakis and Konstantinos Arkoudogiannis for the unending guidance and all the life-changing discussions. I owe my deepest gratitude to Associate Prof. Traianos Yioultsis, Professor George Karagiannidis and Professor Antonis Papagiannakis for their unceasing support and the inspiration they still provide me with. I am grateful to Professor Alkiviadis Hatzopoulos for managing the exchange between AuTH and EPFL. A special word of gratitude goes to Eirini Savvidou for the hours she lost helping me. Special thanks to the whole secretary unit, present and past, of Electrical Engineering department of AUTh for the help with the administrative issues.

This diploma thesis would not be done the same way if it was not for my “eternal” roommate Tatiana Fountoukidou. I owe a big gratitude to my parents and my sisters for supporting me in all possible ways all these years of my studies. Last but not least, I would like to express my deepest gratitude to Eirini Manolaki for sharing all the difficulties and the joys of all this time with me.
Abstract

Polar codes are particularly attractive from a theoretical point of view because they are the first codes that are both highly structured and provably optimal for a wide range of applications (in the sense of optimality that pertains to each application). Moreover, they can be decoded using an elegant, albeit suboptimal, successive cancellation (SC) algorithm, which has low computational and memory complexity. Even though the SC decoder is suboptimal, it is sufficient in order to prove that polar codes are capacity achieving in the limit of infinite blocklength.

Unfortunately, the error correcting performance of SC decoding at finite blocklengths is not as good as that of other modern codes, such as LDPC codes. To improve the finite blocklength performance, more sophisticated algorithms, such as SC list decoding and SC stack decoding, were introduced recently. These algorithms use SC as the underlying decoder, but improve its performance by exploring multiple paths on a decision tree simultaneously, with each path resulting in one candidate codeword. The computational and memory complexities of SC list decoding are though much higher than simple SC decoder.

In this diploma thesis, we describe a new SC-based decoding algorithm, called SC flip, which retains the memory complexity of the original SC algorithm and has an average computational complexity that, at high SNR, is practically the same as the computational complexity of the simple SC decoder, while still providing a significant gain in terms of error correcting performance.
# Contents

1 Introduction to Polar Codes .......................................................... 3
  1.1 Preliminaries ........................................................................... 3
  1.1.1 Notation ........................................................................... 3
  1.1.2 Rate and Reliability Parameters ........................................... 4
  1.2 Channel Polarization ................................................................. 5
  1.2.1 Channel Combining ............................................................. 6
  1.2.2 Channel Splitting ............................................................... 6
  1.2.3 Transformation of Rate and Reliability ............................... 7
  1.2.4 Blockwise Channel Transformation ..................................... 8
  1.3 Polar Coding ........................................................................... 11
  1.3.1 Coset Codes ...................................................................... 12
  1.3.2 Idea of SC Decoder ............................................................ 12
  1.3.3 Polar Codes Construction Rule .......................................... 13
  1.3.4 Theorems on Rate of Polarization - Performance of Polar Codes ........................................ 14
  1.4 Encoding ................................................................................ 14
  1.4.1 Encoding Operation $G_N$ ................................................... 14
  1.4.2 Encoding Complexity ......................................................... 16
  1.5 Decoding ................................................................................ 16
  1.5.1 Decoding Complexity ........................................................ 16
  1.5.2 Recursive SC Decoding Algorithm ..................................... 17
  1.6 Construction of Polar Codes ..................................................... 19

2 Improved SC Decoders for Polar Codes ........................................... 21
  2.1 Space-Efficient Implementation of SC Decoding ....................... 21
  2.2 SC as a path searching process on a code tree ......................... 22
  2.3 List SC Decoding of Polar Codes ............................................. 24
    2.3.1 List SC Decoding Algorithm .............................................. 25
    2.3.2 Complexity of List SC Decoder ......................................... 25
    2.3.3 Performance of List SC Decoder and CRC-Aided Decoder ........ 26
  2.4 Stack SC Decoding of Polar Codes ........................................... 27
    2.4.1 Stack SC Decoding Algorithm .......................................... 27
    2.4.2 Complexity of Stack SC Decoder ...................................... 28

3 Flip SC Decoder ............................................................................. 30
  3.1 Error Propagation in SC Decoding ............................................ 30
    3.1.1 Effect of Error Propagation .............................................. 30
    3.1.2 Significance of Error Propagation ..................................... 32
    3.1.3 Oracle-Assisted SC Decoder ............................................. 32
  3.2 SCFlipDecoder ......................................................................... 33
    3.2.1 SC Flip Decoding Algorithm ............................................ 33
    3.2.2 Complexity of SC Flip Decoding ...................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.3 Error Correcting Performance of SC Flip Decoding</td>
<td>35</td>
</tr>
<tr>
<td>3.2.4 Average Computational Complexity of SC Flip Decoding</td>
<td>35</td>
</tr>
<tr>
<td>3.2.5 Implementation Considerations</td>
<td>37</td>
</tr>
<tr>
<td>4 Conclusion</td>
<td>38</td>
</tr>
<tr>
<td>5 Bibliography</td>
<td>39</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction to Polar Codes

Channel coding has been an ever growing field since Shannon’s seminal work in 1948 [13]. The existence of capacity achieving codes was shown; these codes are random codes with exponential encoding and decoding complexity. Since then, the goal of achieving capacity with relatively low complexity has been a major issue for coding society. Arıkan has reached this goal for the class of binary input discrete memoryless channels (B-DMC) [1]. He discovered the method of “channel polarization” for constructing capacity achieving codes for B-DMC.

In this chapter we briefly introduce the basic concepts of channel polarization and polar codes as well as the main ideas about encoding and decoding operations. We end the chapter with some words on code construction issues. Throughout this chapter we will restate the main results in [1], so references will not be given separately, except if needed.

1.1 Preliminaries

We start with presenting the notation that is going to be used throughout this thesis. Afterwards, we introduce the main parameters which capture the notion of channel rate and reliability, namely symmetric capacity and Bhattacharyya parameter, as well as their main properties.

1.1.1 Notation

We are mainly following the notation conventions of [1]. Sets are denoted with script-style uppercase letters, like $A$. The complementary of a set is denoted with superscript $c$, like $A^c$. The number of elements in a set $A$ is denoted by $|A|$. For the indicator function of a set $A$ we have that

$$1_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise.} \end{cases} \quad (1.1)$$

Random variables (RVs) are denoted by upper case letters like $X, Y$, and their sample values by lower case letters like $x, y$. The probability distribution of a RV $X$ is denoted by $P_X$ and for a joint ensemble of RVs $(X, Y)$, the joint probability distribution is $P_{X,Y}$.

With $a_1^N$ we denote a row vector $(a_1, ..., a_N)$ and with $a_i^j$ the subvector $(a_i, ..., a_j)$. If $j < i$, $a_i^j$ is an empty vector. Alternatively, $a_I$ also denotes a subvector containing the elements of the vector $a$ with indices in $I$. $0_1^N$ is used for determining the all-zero vector. We use $a_i^j_{e, o}$ to denote the subvector that contains the even indices of $a_i^j$ and we use $a_i^j_{o, o}$ to denote the subvector that contains the odd indices of $a_i^j$. For $a_1^N, b_1^N$ vectors over GF(2), we denote their componentwise mod-2 sum by $a_1^N \oplus b_1^N$.

If $A$ is an $m \times n$ matrix and $B$ is an $p \times q$ matrix, then the Kronecker product $A \otimes B$ is the $mp \times nq$ matrix

$$A \otimes B = \begin{bmatrix} A_{11}B & \ldots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \ldots & A_{mn}B \end{bmatrix}. \quad (1.2)$$
The Kronecker power is defined as $A^\otimes n = A \otimes A^\otimes(n-1)$, with the convention that $A^\otimes 0 \triangleq [1].$

A B-DMC is denoted by $W : \mathcal{X} \to \mathcal{Y}$. The input alphabet is $\mathcal{X}$ and is always $\{0, 1\}$, while the output alphabet is $\mathcal{Y}$ and can be arbitrary. The transition probabilities are $W(y|x)$ with $x \in \mathcal{X}, y \in \mathcal{Y}$. We write $W^N : \mathcal{X}^N \to \mathcal{Y}^N$ to denote the vector channel that corresponds to $N$ independent uses of $W$ with input alphabet $\mathcal{X}^N$, output alphabet $\mathcal{Y}^N$ and transition probabilities $W^N(y^N_1|x^N_1) = \prod_{i=1}^N W(y_i|x_i)$.

All logarithms used throughout this thesis are base-2 logarithms; any different case will be stated explicitly.

### 1.1.2 Rate and Reliability Parameters

The first important channel parameter is the symmetric capacity, which is defined as

\[
I(W) \triangleq \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2} W(y|0) + \frac{1}{2} W(y|1)}. \tag{1.3}
\]

The symmetric capacity is a measure of the rate of information transmission. It is actually the maximum possible rate of reliable information transmission. The second important channel parameter is the Bhattacharyya parameter, which is defined as

\[
Z(W) \triangleq \sum_{y \in \mathcal{Y}} \sqrt{W(y|0)W(y|1)}. \tag{1.4}
\]

The Bhattacharyya parameter is a measure of the reliability of information transmission. If we use $W$ to transmit a bit that is 0 or 1, then $Z(W)$ is an upper bound on the error probability of maximum-likelihood (ML) decision.

These two parameters take values in the range $[0,1]$. It is easy to see this argument for $Z(W)$ and, considering the fact that we are using base-2 logarithms, we can easily see that also $I(W)$ takes values in that range. "Good" channels will have $I(W)$ close to 1 and $Z(W)$ close to 0. In the way those two parameters are defined, one would expect that $I(W)$ goes to 1 iff $Z(W)$ goes to 0 and vice versa. We show this in the following proposition.

**Proposition 1.1.** The following bounds hold for any B-DMC $W$

\[
I(W) \geq \log \frac{2}{1 + Z(W)} \tag{1.5}
\]

\[
I(W) \leq \sqrt{1 - Z(W)^2} \tag{1.6}
\]

Indeed we see that for a transmission rate close to 1 the reliability is high ($Z(W)$ is close to 0). In such a situation we consider the B-DMC to be a good channel. In contrast, for a transmission rate close to 0, the reliability is low ($Z(W)$ is close to 1). In this case, the B-DMC is a bad channel. Channel polarization is a method that moves our channels close to either the extremal good or the extremal bad situation, leaving fewer and fewer channels being in a mediocre position, as we go on applying the method. This theoretically lets us use only the extremal good channels (noiseless channels) and "throw away" the extremal bad ones (useless channels).

A channel $W$ is said to be symmetric if there exists a permutation $\pi$ of the output alphabet such that

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

Well known examples of symmetric channels are the binary symmetric channel (BSC) and the binary erasure channel (BEC). For symmetric channels the symmetric capacity equals the Shannon capacity.
A binary symmetric channel with crossover probability $p$, denoted by BSC($p$), is a B-DMC with binary output alphabet $\mathcal{Y} = \{0, 1\}$, $W(0|0) = W(1|1) = 1 - p$ and $W(1|0) = W(0|1) = p$.

The symmetric capacity and the Bhattacharyya parameter of such a channel are

\[
I(W) = C_{BSC} = 1 - H(p) \tag{1.7}
\]

\[
Z(W) = Z_{BSC} = 2\sqrt{p(1-p)} \tag{1.8}
\]

respectively, where $H(p)$ is the binary entropy function defined as

\[
H(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}. \tag{1.9}
\]

A binary erasure channel (BEC) is a B-DMC with ternary output. For each $y \in \mathcal{Y}$, one of the following is true

- $W(y|0)W(y|1) = 0$ or
- $W(y|0) = W(y|1)$

In the first case, the symbol is transmitted correctly and in the second case, the symbol is called an erasure symbol. The erasure probability is denoted by $\epsilon$ and it is the sum of $W(y|0)$ over all erasure symbols $y$

\[
\epsilon \triangleq \sum_{y \in \mathcal{Y}, W(y|0) = W(y|1)} W(y|0). \tag{1.10}
\]

The symmetric capacity and the Bhattacharyya parameter of a BEC($\epsilon$) channel are

\[
I(W) = C_{BEC} = 1 - \epsilon \tag{1.11}
\]

\[
Z(W) = Z_{BEC} = \epsilon. \tag{1.12}
\]

Figure 1.1 shows that, for any B-DMC, the $(Z(W), I(W))$ pair lies in the shaded region. Considering the above equations, we can see that BEC lies on the lower bound of this region.

### 1.2 Channel Polarization

We start by briefly giving the main notion of channel polarization. Let us have a given B-DMC $W$ with symmetric capacity $I(W)$. Now let us get two independent copies of this channel $W$. If we use these
channels as they are, we have two symmetric capacities of $I(W)$. Channel polarization is an operation that allows us to combine those two channels, creating a vector “super channel”. Afterwards, we split this vector channel back into two new channels with unequal symmetric capacities; the worse channel will have $I(W^-) \leq I(W)$, while the better one will have $I(W^+) \geq I(W)$.

The same procedure can be applied for $N > 2$; as we make $N$ larger and larger, the symmetric capacity terms of the new channels tend more and more towards 0 or 1. Also, for infinitely large $N$, the fraction of extremely good channels, with symmetric capacity arbitrarily close to 1, goes to $I(W)$, while the fraction of extremely bad channels, with symmetric capacity arbitrarily close to 0, goes to $(I(W) - 1)$.

So, out of $N$ independent copies of a given B-DMC $W$, we have created a second set of $N$ polarized “extreme” channels. This polarization phenomenon gives us the opportunity to pass information only through the extreme good channels with $I(W)$ close to 1.

### 1.2.1 Channel Combining

Channel combining is the procedure of using $N$ (where $N = 2^n, n \geq 0$) independent copies of a given B-DMC $W$ in order to recursively create a vector channel $W_N : \mathcal{X}^N \to \mathcal{Y}^N$.

**0th level of recursion** ($n = 0, N = 1$): The 0th level of recursion is simply using the channel $W$ by itself. We define $W_1 \triangleq W$.

**1st level of recursion** ($n = 1, N = 2$): In the 1st level of recursion we combine two independent copies of $W$ in the way that is shown in Figure 1.2. Thus we obtain the vector channel $W_2 : \mathcal{X}^2 \to \mathcal{Y}^2$, with transition probabilities

$$W_2(y_1, y_2 | u_1, u_2) = W(y_1 | x_1) W(y_2 | x_2)$$

$$\triangleq W(y_1 | u_1 \oplus x_2) W(y_2 | u_2).$$

The mapping $u_2 \to x_1^2$ from the input of the vector channel $W_2$ to the input of the raw channels $W^2$ can be written as $x_1^2 = u_2^2 G_2$, where

$$G_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

After analyzing the whole mechanism for $N = 2$, we will generalize the procedure for $N = 2^n$ at the end of this chapter.

### 1.2.2 Channel Splitting

Having combined two independent copies of $W$ (i.e. $W^2$) to create the vector channel $W_2 : \mathcal{X}^2 \to \mathcal{Y}^2$, we continue the channel polarization procedure by splitting back this super channel into two B-DMCs, $W^- : \mathcal{X} \to \mathcal{Y}^2$ and $W^+ : \mathcal{X} \to \mathcal{Y}^2 \times \mathcal{X}$, defined as

$$W_2(y_1, y_2 | u_1) = \sum_{u_2 \in \mathcal{X}} \frac{1}{2} W_2(y_1, y_2 | u_1, u_2)$$

$$W_2(y_1, y_2, u_2 | u_1) = W_2(y_1, y_2 | u_1, u_2).$$
1.2. CHANNEL POLARIZATION

We will see in the next section that the channel seen from \( u_1 \) to \((y_1, y_2)\) is worse than the parent one in terms of symmetric capacity (that is the reason of the \( W^- \) notation). Similarly, the channel seen from \( u_2 \) to \((u_1, y_1, y_2)\) is better than the parent one in terms of symmetric capacity (\( W^+ \) notation).

1.2.3 Transformation of Rate and Reliability

Having described the single-step (local) polar transform \((W, W) \rightarrow (W^-, W^+)\), we will now examine the impact of this transformation on the rate and reliability parameters.

**Proposition 1.2.** Local transformation of rate.

Suppose we apply the single-step polar transform \((W, W) \rightarrow (W^-, W^+)\) on a set of B-DMCs \( W \). Then we have

\[
I(W^-) + I(W^+) = 2I(W) \tag{1.18}
\]

\[
I(W^-) \leq I(W^+) \tag{1.19}
\]

with equality iff \( I(W) = 1 \) or 0 (i.e. the parent channel \( W \) was already an extremal channel itself).

Equation (1.18) tells us that the symmetric capacity is preserved through the single-step channel transformation. Equation (1.19) tells us that polarization exists in the sense that, if the parent channel is not extremal itself, then the symmetric capacities move away from the centre, \( I(W^-) < I(W) < I(W^+) \).

**Proposition 1.3.** Local transformation of reliability.

Suppose we apply the single-step polar transform \((W, W) \rightarrow (W^-, W^+)\) on a set of B-DMCs \( W \). Then we have

\[
Z(W^+) = Z(W)^2 \tag{1.20}
\]

\[
Z(W^-) \leq 2Z(W) - Z(W)^2 \tag{1.21}
\]

\[
Z(W^-) \geq Z(W) \geq Z(W^+) \tag{1.22}
\]

Equality holds in Equation (1.21) iff \( W \) is a BEC. In Equation (1.22), equality holds iff \( Z(W) = 0 \) or 1 (i.e., the parent channel \( W \) was already an extremal channel itself). From Equation (1.20) and Equation (1.21) we can derive

\[
Z(W^-) + Z(W^+) \leq 2Z(W) \tag{1.23}
\]

which tells us that single-step channel transform can only improve reliability. Equality in Equation (1.23) holds iff \( W \) is a BEC.

As we can see, if the B-DMC is a BEC, there exists some special behaviour that is interesting to examine further in the following propositions. We first see that a parent BEC gives BEC children through the single-step transformation.

**Proposition 1.4.** Suppose we apply the single-step polar transform \((W, W) \rightarrow (W^-, W^+)\) and that \( W \) is a BEC with erasure probability \( \epsilon \), then the new channels \( W^- \) and \( W^+ \) are also BECs, with erasure probabilities \( 2\epsilon - \epsilon^2 \) and \( \epsilon^2 \) respectively. Conversely, if \( W^- \) or \( W^+ \) is a BEC, then \( W \) is a BEC.

**Proposition 1.5.** Single-step transformation of rate for a BEC.

Suppose we apply the single-step polar transform \((W, W) \rightarrow (W^-, W^+)\) and that \( W \) is a BEC with erasure probability \( \epsilon \), then

\[
I(W^-) = I(W)^2 \tag{1.24}
\]

\[
I(W^+) = 2I(W) - I(W)^2 \tag{1.25}
\]

where \( I(W) = 1 - \epsilon \).
1.2. CHANNEL POLARIZATION

**Proposition 1.6.** Single-step transformation of reliability for a BEC.

Suppose we apply the single-step polar transform \((W, W) \rightarrow (W^-, W^+)\) and that \(W\) is a BEC with erasure probability \(\epsilon\), then

\[
Z(W^-) = 2Z(W) - Z(W)^2 \tag{1.26}
\]

\[
Z(W^+) = Z(W)^2 \tag{1.27}
\]

where \(Z(W) = \epsilon\).

In the next section, we will show that similar results exist for BECs in the general case of \(N = 2^n\), in a recursive manner.

1.2.4 Blockwise Channel Transformation

Until now we have examined the single-step channel transformation and the 1st level polar transform \((n = 1)\). In this section we are going to generalize the procedure for any \(N = 2^n\). So let us examine first the 2nd level polar transform.

**2nd level of recursion \((n = 2, N = 4)\):** In Figure 1.2 we defined the vector channel \(W_2\). Now we use two independent copies of \(W_2\) to create the vector channel \(W_4 : X^4 \rightarrow Y^4\), as we see in Figure 1.3. The transition probabilities of \(W_4\) are

\[
W_4(y_1^4 | u_1^4) = W_2(y_1^2 | v_1, v_2)W_2(y_3^2 | v_3, v_4) = W_2(y_1^2 | u_1 \oplus u_2, u_3 \oplus u_4)W_2(y_3^2 | u_2, u_4). \tag{1.28}
\]

In Figure 1.3, the input vector \(u_1^4\) to \(W_4\) is first transformed into \(s_1^4\) so that \(s_1 = u_1 \oplus u_2, s_2 = u_2, s_3 = u_3 \oplus u_4\) and \(s_4 = u_4\). \(R_4\) is a permutation operation for \(N = 4\) that maps its input \((s_1, s_2, s_3, s_4)\) to the output \((v_1, v_2, v_3, v_4) = (s_1, s_3, s_2, s_4)\). We will define the general permutation operation \(R_N\) in the sequel.

In Figure 1.3, we can also see the relation of the vector channel \(W_4\) to the 4 independent uses of \(W\), i.e. \(W^4\). We can easily see that the mapping \(u_1^4 \rightarrow x_1^4\) from the input of the vector channel \(W_4\) to the input of the raw channels \(W^4\) can be written as \(x_1^4 = u_1^4G_4\), where

\[
G_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1.30}
\]

![Figure 1.3: Vector channel \(W_4\) (taken from [1]).](image)
We can also have a relationship between the transition probabilities of the vector channel and the raw channels like

$$W_4(y^4_1| u^4_1) = W_4(y^4_1| x^4_1)$$

(1.31)

$$= W_4(y^4_1| u^4_1 G_4).$$

(1.32)

Let us see some structural properties of the matrix $G$, which can help us with some intuitive understanding of the polarization method. Looking at a column of the matrix is the same as standing at the corresponding input $x$ of the raw channel $W$ and looking towards the input of the vector channel $W_N$. For example, for the third column, 1s are located in the second and the fourth positions. We see that standing on $x_3$ and looking backwards to the input of the vector channel $W_N$, we connect to the second and the fourth input of $W_4$, i.e $u_2$ and $u_4$.

Looking at a row of the matrix is the same as standing at the corresponding input $u$ of the vector channel and looking towards the input of the raw channel $W$. For example, for the second row, we have 1s in the first and the third position and we can indeed see that $u_2$ is connected to $x_1$ and $x_3$.

**nth level of recursion:** We are now in the position of describing the general recursive procedure of constructing $W_N$ out of two independent copies of $W_{N/2}$. We can see this generalised procedure in Figure 1.4. The input vector $u^N_1$ of the vector channel $W_N$ is first transformed into vector $s^N_1$ in such a way that $s_{2i-1} = u_{2i-1} \oplus u_{2i}$, and $s_{2i} = u_{2i}$, for $1 \leq i \leq N/2$. $R_N$ is called the reverse shuffle operation and it maps its input $s^N_1 = (s_1, s_2, \ldots, s_N)$ to the output $v^N_1 = (s_1, s_3, \ldots, s_{N-1}, s_2, s_4, \ldots, s_N)$, that becomes the input to the two independent copies of $W_{N/2}$.

As in the previous cases, the overall mapping from the input of the vector channel $W_N$ to the input of the raw channels $W^N$ can be written as $x^N_1 = u^N_1 G_N$, where $G_N$ is called the generator matrix of size $N$. Later, when we examine the encoding procedure more thoroughly, we will show that $G_N = B_N F^n$, for $N = 2^n$ and $n \geq 0$, where $B_N$ is the bit-reversal matrix, and $F$ is defined as

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

(1.33)
The relationship between the transition probabilities of the vector channel and the raw channels can be now written as

\[ W_N(y_1^N|u_1^N) = W_N(y_1^N|u_1^NG_N) . \] (1.34)

Now we are going to generalize the procedure of channel splitting, which we examined before for the case of single-step channel transform. We split the vector channel \( W_N \) into a set of \( N \) B-DMCs \( W_N^{(i)} : X \rightarrow Y^N \times X^{i-1}, 1 \leq i \leq N \), with transition probabilities

\[ W_N^{(i)}(y_1^N, u_1^{i-1}|u_i) = \sum_{u_{i+1}^N \in X^{N-i}} \frac{1}{2^{N-i}} W_N(y_1^N|u_i^N) . \] (1.35)

We did nothing more than to extend the single-step procedure; each pair of the new B-DMC channels is obtained after applying the single-step polar transform to one of the channels of the previous step

\[ (W_N^{(i)}(W_N^{(i)}) \rightarrow (W_N^{(2i-1)}, W_N^{(2i)}). \] (1.36)

The single-step channel transform in its general form, for any \( N = 2^n, n \geq 0, 1 \leq i \leq N \), is written

\[ W_N^{(2i-1)}(y_1^N, u_1^{2i-2}|u_{2i-1}) = \sum_{u_{2i}} \frac{1}{2} W_N(y_1^N, u_1^{2i-2} \oplus u_1^{2i-2}|u_{2i-1} \oplus u_{2i}) \cdot W_N(y_2^N, u_1^{2i-2}|u_{2i}) \] (1.37)

\[ W_N^{(2i)}(y_1^N, u_1^{2i-2}|u_{2i}) = \frac{1}{2} W_N(y_1^N, u_1^{2i-2} \oplus u_1^{2i-2}|u_{2i-1} \oplus u_{2i}) \cdot W_N(y_2^N, u_1^{2i-2}|u_{2i}). \] (1.38)

We have seen how the local single-step polar transform affects the rate and reliability parameters. We next state the same results for the general case.

**Proposition 1.7.** For the transformation \((W_N^{(i)}, W_N^{(i)}) \rightarrow (W_N^{(2i-1)}, W_N^{(2i)}), N = 2^n, n \geq 0, 1 \leq i \leq N\), we have that

\[ I(W_N^{(2i-1)}) + I(W_N^{(2i)}) = 2I(W_N^{(i)}) \] (1.39)

\[ Z(W_N^{(2i-1)}) + Z(W_N^{(2i)}) \leq 2Z(W_N^{(i)}). \] (1.40)

Equation (1.39) tells us that the transformation \((W_N^{(i)}, W_N^{(i)}) \rightarrow (W_N^{(2i-1)}, W_N^{(2i)})\) is rate-preserving and Equation (1.40) tells us that it is also reliability-improving. Equality in Equation (1.40) holds iff \( W \) is a BEC.

**Proposition 1.8.** For the transformation \((W_N^{(i)}, W_N^{(i)}) \rightarrow (W_N^{(2i-1)}, W_N^{(2i)}), N = 2^n, n \geq 0, 1 \leq i \leq N\), the following also hold

\[ I(W_N^{(2i-1)}) \leq I(W_N^{(i)}) \leq I(W_N^{(2i)}) \] (1.41)

\[ Z(W_N^{(2i-1)}) \geq Z(W_N^{(i)}) \geq Z(W_N^{(2i)}). \] (1.42)

Equation (1.41) and Equation (1.42) show that the transformation moves both the rate and the reliability away from the center. Equalities again hold iff \( W \) is a BEC.

Following an inductive method, it can easily be seen from the previous results that the \( n \)-level polar transform will also be rate-preserving and reliability improving in the sense that

\[ \sum_{i=1}^{N} I(W_N^{(i)}) = NJ(W) \] (1.43)

\[ \sum_{i=1}^{N} Z(W_N^{(i)}) \leq NZ(W) \] (1.44)
with equality in (1.44) iff $W$ is a BEC.

Now we are going to restate Propositions 1.5 and 1.6 for the rate and reliability transformations of a BEC in a general recursive way.

**Proposition 1.9.** Recursive transformation of rate for a BEC.

For the recursive channel transformation $(W^{(i)}_{N/2}, W^{(i)}_{N/2}) \rightarrow (W^{(2i-1)}_{N}, W^{(2i)}_{N})$, $N = 2^n$, $n \geq 0$, $1 \leq i \leq N$, and the special case that $W$ is a BEC with erasure probability $\epsilon$, we have that

$$I(W^{(2i-1)}_{N}) = I(W^{(i)}_{N/2})^2$$  \hspace{1cm} (1.45)

$$I(W^{(2i)}_{N}) = 2I(W^{(i)}_{N/2}) - I(W^{(i)}_{N/2})^2$$  \hspace{1cm} (1.46)

where $I(W^{(i)}_{1}) = 1 - \epsilon$.

**Proposition 1.10.** Recursive transformation of reliability for a BEC.

For the recursive channel transformation $(W^{(i)}_{N/2}, W^{(i)}_{N/2}) \rightarrow (W^{(2i-1)}_{N}, W^{(2i)}_{N})$, $N = 2^n$, $n \geq 0$, $1 \leq i \leq N$, and the special case that $W$ is a BEC with erasure probability $\epsilon$, we have that

$$Z(W^{(2i-1)}_{N}) = 2Z(W^{(i)}_{N/2}) - Z(W^{(i)}_{N/2})^2$$  \hspace{1cm} (1.47)

$$Z(W^{(2i)}_{N}) = Z(W^{(i)}_{N/2})^2$$  \hspace{1cm} (1.48)

where $Z(W^{(i)}_{1}) = \epsilon$.

Using Equations (1.45) and (1.46), we can show the polarization effect of the recursive method we are using. In Figure 1.5 we plot the polarization effect for the case of a BEC with erasure probability $\epsilon = 0.5$.

![Figure 1.5: Plot of $I(W^{(i)}_{N})$ versus $i = 1, \cdots, N = 2^{10}$ for a BEC with $\epsilon = 0.5$ (taken from [1]).](image)

**1.3 Polar Coding**

Having analyzed the procedure of channel polarization, we are now going to use it in a channel coding scheme, namely polar coding. Polar coding uses $N$ independent copies of a given B-DMC $W$ to
transform them into $N$ forged polarized channels $W_{N}^{(i)}$. Information is sent just through the best of the new channels, the ones with $Z(W)$ close to 0. The bad polarized channels, with $Z(W)$ close to 1, are kept frozen.

### 1.3.1 Coset Codes

We are first describing a superclass of polar codes, namely $G_N$-coset codes. This is a class of block codes with block lengths $N$ that are powers of 2, $N = 2^n$ for $n \geq 0$. For each $N$, each code in the class is encoded in the same way

$$x_1^N = u_1^N G_N$$

where $G_N$ is the generator matrix of order $N$ that we described before. Let us define an arbitrary subset $\mathcal{A}$ of the set $\{1, \ldots, N\}$ and let $\mathcal{A}^c$ be the complementary set of $\mathcal{A}$ with respect to $\{1, \ldots, N\}$. Then we can rewrite (1.49) as

$$x_1^N = u_A G_N(\mathcal{A}) \oplus u_{A^c} G_N(\mathcal{A}^c)$$

where with $G_N(\mathcal{A})$ we denote the submatrix of $G_N$ formed by the rows of $G$ with indices in $\mathcal{A}$.

A coset code is the mapping from the binary source vectors $u_\mathcal{A}$ to the binary codeword vectors $x_1^N$, with the fact that we fix $\mathcal{A}$ and $u_{A^c}$. It is called a $G_N$-coset code because it is a coset of the block code with generator matrix $G_N(\mathcal{A})$, determined by the frozen vector $u_{A^c} G_N(\mathcal{A}^c)$. If $K = |\mathcal{A}|$, then the code rate is defined as $R = K/N$. A specific $G_N$-coset code is determined by the set of parameters $(N, K, \mathcal{A}, u_{A^c})$. We name $\mathcal{A}$ the information set, $\mathcal{A}^c$ the frozen set, $u_A$ the information vector and $u_{A^c}$ the frozen vector.

Polar codes are a subset of this class of $G_N$-coset codes, specified by a certain rule of choosing the information set $\mathcal{A}$. We will define this specific rule in a while.

### 1.3.2 Idea of SC Decoder

Let us briefly describe the encoding and decoding procedure with the fact that we are using a $G_N$-coset code with parameters $(N, K, \mathcal{A}, u_{A^c})$. A message $u_1^N$, corresponding to the binary source vector $u_\mathcal{A}$, is encoded into the codeword $x_1^N$. This codeword is transmitted through the channel $W_{N}^{(i)}$, possibly corrupted by the noise, and received as $y_1^N$. It is the job of the decoder to find an estimate $\hat{x}_1^N$ of $x_1^N$, or equivalently to find an estimate $\hat{u}_1^N$ of $u_1^N$, because of the one-to-one correspondence between $x_1^N$ and $y_1^N$. Moreover, since $u_{A^c}$ is fixed and therefore known to the decoder, the real job of the decoder is to find an estimate $\hat{u}_\mathcal{A}$ of $u_\mathcal{A}$, given knowledge of $\mathcal{A}$, $u_{A^c}$ and $y_1^N$.

The successive cancellation (SC) decoder for the class of $G_N$-coset codes is described in Algorithm 1.

**Algorithm 1: Successive Cancellation Decoding**

**Input:** Channel output $y_1^N$, $(N, K, \mathcal{A}, u_{A^c})$.

**Output:** $\hat{u}_1^N$ an estimation of $u_1^N$.

**for all** $i \in \{0, 1, \ldots, N - 1\}$ **do**

  **if** $i \in \mathcal{A}^c$ **then**
  
  $\hat{u}_i \leftarrow u_i$

  **else**
  
  **if** $W_{N}^{(i)}(y_1^N, \hat{u}_1^{i-1}|1) \geq 1$ **then**
  
  $\hat{u}_i \leftarrow 0$

  **else**
  
  $\hat{u}_i \leftarrow 1$

  **end if**

**end if**

**end for all**

return $\hat{u}_1^N$
Let us now give the mathematical formalization of the SC decoding procedure. For each \(1 \leq i \leq N\), in the order \(i\) from 1 to \(N\), the decoder computes

\[
\hat{u}_i \triangleq \begin{cases} 
  u_i, & \text{if } i \in A^c \\
  h_i(y_1^N, \hat{u}_{i-1}^i), & \text{if } i \in A
\end{cases}
\]  

(1.51)

where \(h_i : \mathcal{Y}^N \times \mathcal{X}^{i-1} \rightarrow \mathcal{X}, i \in A\) are named decision functions and are defined as

\[
h_i(y_1^N, \hat{u}_{i-1}^i) \triangleq \begin{cases} 
  0, & \text{if } \frac{W_N^{(i)}(y_1^N, \hat{u}_{i-1}^i[0])}{W_N^{(i)}(y_1^N, \hat{u}_{i-1}^i[1])} \geq 1 \\
  1, & \text{otherwise.}
\end{cases}
\]  

(1.52)

The decision functions defined above are very close to the ML decision functions, with one difference: decision functions (1.52) use estimations of the previous decisions. This contains of course the case that all the previous decisions were taken on frozen bits (thus taken correctly). What is really important for the SC decoder though is the fact that we still can reach the symmetric capacity, although we are using a non-ML scheme; the loss in performance is negligibly small, due to the recursive structure of the decoder.

### 1.3.3 Polar Codes Construction Rule

Let \(P_e(N, K, A, u_{A^c})\) denote the block error probability of a specific \(G_N\)-coset code with parameters \((N, K, A, u_{A^c})\), and let \(P_e(N, K, A)\) denote the average block error probability of the class of \(G_N\)-coset codes with parameters \((N, K, A)\), over all choices of \(u_{A^c}\).

**Proposition 1.11.** Assume a B-DMC \(W\) and the class of \(G_N\)-coset codes with parameters \((N, K, A)\), then

\[
P_e(N, K, A) \leq \sum_{i \in A} Z(W_N^{(i)}).
\]  

(1.53)

Hence, for the class of codes with parameters \((N, K, A)\), there exists at least one specific frozen vector \(u_{A^c}\), and thus at least one specific code with parameters \((N, K, A, u_{A^c})\), such that

\[
P_e(N, K, A, u_{A^c}) \leq \sum_{i \in A} Z(W_N^{(i)}).
\]  

(1.54)

Now we are ready to present the criterion that distinguishes polar codes from all other \(G_N\)-coset codes: choose the information set \(A\) so as to minimize the upper bound (right-hand side) of Equation (1.53).

**Proposition 1.12.** For a symmetric B-DMC \(W\), every \(G_N\)-coset code with parameters \((N, K, A, u_{A^c})\) satisfies

\[
P_e(N, K, A, u_{A^c}) \leq \sum_{i \in A} Z(W_N^{(i)}).
\]  

(1.55)

This tells us that if we choose the information set (and thus also the frozen set) appropriately, it does not matter what choice of the frozen bits we make. For any choice of the values of frozen bits we can achieve the desired performance. We used that fact in our Matlab simulations; in order to simplify the simulations we always chose the simple all-zero sequence for the frozen bits, without any performance deterioration.

Now we can give a formal definition of polar codes: for a given B-DMC \(W\), a \(G_N\)-coset code with parameters \((N, K, A, u_{A^c})\) will be called a polar code for the specific channel \(W\) if the information set \(A\) is chosen as a \(K\)-element subset of the set \(\{1, \ldots, N\}\) such that \(Z(W_N^{(i)}) \leq Z(W_N^{(j)})\), for all \(i \in A\) and \(j \in A^c\).
1.3.4 Theorems on Rate of Polarization - Performance of Polar Codes

Now that we have given the polar coding rule, we will introduce the notation $P_e(N, R)$ as the block error probability for polar coding over a B-DMC $W$, under SC decoding. $N$ is the block length, $R \geq 0$ is the rate of the code. This block error probability is averaged over all possible choices for the frozen bits $u_A$. Let us now introduce some basic theorems concerning the performance analysis of polar coding and the rate of polarization.

**Theorem 1.1.** For any given B-DMC $W$ with $I(W) > 0$, and any fixed $R < I(W)$, there exists a sequence of sets $A_N \subset \{1, \ldots, N\}$, $N \in \{1, 2, \ldots, 2^n, \ldots\}$, such that $|A_N| \geq NR$ and

$$Z(W_N^{(i)}) \leq O(N^{-\frac{7}{4}}), \forall i \in A_N.$$  \hspace{1cm} (1.56)

This leads to the following theorem.

**Theorem 1.2.** For any given B-DMC $W$ with $I(W) > 0$, and any fixed $R < I(W)$, block error probability for polar coding under SC decoding satisfies

$$P_e(N, R) = O(N^{-\frac{7}{4}}).$$  \hspace{1cm} (1.57)

This theorem was shown and proved in [1]. A strengthened version of these theorems was shown in [2]. They contain a major improvement on the previous bounds.

**Theorem 1.3.** For any given B-DMC $W$ with $I(W) > 0$, any fixed $R < I(W)$ and any fixed $\beta < \frac{1}{2}$, there exists a sequence of sets $A_N \subset \{1, \ldots, N\}$, $N \in \{1, 2, \ldots, 2^n, \ldots\}$, such that $|A_N| \geq NR$ and

$$\sum_{i \in A_N} Z(W_N^{(i)}) \leq o(2^{-N^\beta}).$$  \hspace{1cm} (1.58)

**Theorem 1.4.** For any given B-DMC $W$ with $I(W) > 0$, any fixed $R < I(W)$ and any fixed $\beta < \frac{1}{2}$, block error probability for polar coding under SC decoding satisfies

$$P_e(N, R) = O(2^{-N^\beta}).$$  \hspace{1cm} (1.59)

1.4 Encoding

In this section, we will give explicit algebraic expressions for the generator matrix. Till now, the only analysis we have made on $G_N$ was that of the Figure 1.4. We will also present $G_N$ in an alternative schematic form, based on the algorithmic analysis. Some important aspects, concerning the encoding complexity for polar codes, will also be mentioned.

1.4.1 Encoding Operation $G_N$

Let us assume that $N = 2^n, n \geq 0$ and let $I_k$, with $k \geq 1$, denote the $k$-dimensional identity matrix. We can make a direct algebraic interpretation of Figure 1.4 in the following way

$$G_N = (I_{N/2} \otimes F)R_N(I_2 \otimes G_{N/2}), \text{ for } N \geq 2$$  \hspace{1cm} (1.60)

corresponding that $G_1 = I_1$.

We can see that Figure 1.6 is an alternative realization of the channel combining operation of Figure 1.4. Its algebraic interpretation is given as

$$G_N = R_N(F \otimes I_{N/2})(I_2 \otimes G_{N/2}), \text{ for } N \geq 2.$$  \hspace{1cm} (1.61)
which is equivalent to Equation (1.60). We can also check that Equations (1.60) and (1.61) are equivalent, in a purely algebraic way, by noticing that \((I_{N/2} \otimes F)R_N = R_N(F \otimes I_{N/2})\). Equation (1.61) can be written as

\[
G_N = R_N(F \otimes G_{N/2})
\] (1.62)

If we use this equation for \(N/2\), we have

\[
G_{N/2} = R_{N/2}(F \otimes G_{N/4}).
\] (1.63)

Substituting Equation (1.63) back into (1.62), we get

\[
G_N = R_N(I_2 \otimes (F \otimes (G_{N/2} \otimes (I_4 \otimes R_{N/4}) \ldots R_{N/2})))
\] (1.64)

Using the identity \((AC) \otimes (BD) = (A \otimes B)(C \otimes D)\), with \(A = I_2, B = R_{N/2}, C = F, D = F \otimes G_{N/4}\), (1.64) becomes

\[
G_N = R_N(I_2 \otimes R_{N/2})(F \otimes (G_{N/4})).
\] (1.65)

Repeating this recursive procedure we get

\[
G_N = B_N F^\otimes n
\] (1.66)

where

\[
B_N \triangleq R_N(I_2 \otimes R_{N/2})(I_4 \otimes R_{N/4}) \ldots (I_{N/2} \otimes R_2)
\] (1.67)

\[
= R_N(I_2 \otimes B_{N/2}).
\] (1.68)

In order to understand the difference between \(R_N\) and \(B_N\), we will introduce the notion of bit-indexing. Given a vector \(a_N^i\), with \(N = 2^n, n \geq 0\), we can denote its \(i\)-th element \(a_i, 1 \leq i \leq N\) alternatively as \(a_{b_1 b_2 \ldots b_n}\), where \(b_1 b_2 \ldots b_n\) is the binary expansion of the integer \(i - 1\). This means, for example, that if we have the vector \(a_8^1\), then \(a_3\) is denoted alternatively as \(a_{010}\) and \(a_6\) is denoted as \(a_{101}\).
1.5. DECODING

$R_N$ is called the reverse shuffle operator and cyclically rotates the bit-indexes of the operand it acts on, by one place to the left; that is, if $v_1^N = u_1^N R_N$, then $v_{b_1...b_n} = u_{b_2...b_n b_1}$. For example, if $v_8^8 = u_8^8 R_8$, then $v_{100} = u_{001}$, which means that $v_5 = u_2$. So, we go from $u_8^8 = (u_{000}, u_{001}, u_{010}, u_{011}, u_{100}, u_{101}$, $u_{110}, u_{111})$ to $v_8^8 = (v_{000}, v_{010}, v_{100}, v_{011}, v_{010}, v_{101}, v_{111})$. In typical indexing, this means that we go from $u_8^8 = (u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8)$ to $v_8^8 = (v_1, v_3, v_5, v_7, v_2, v_4, v_6, v_8)$. We clearly see that the reverse shuffle operator rearranges the elements of its operand $u_1^8$ with respect to odd-even parity of their indices.

$B_N$ is called the bit-reversal operator: if $v_1^N = u_1^N B_N$, then $v_{b_1b_2...b_n} = u_{b_nb_{n-1}...b_1}$, for all $b_1...b_n \in \{0,1\}$. For example, if $v_8^8 = u_8^8 R_8$, then $v_{110} = u_{011}$, which means that $v_7 = u_4$. We should mention here that $B_8$ happens to have the same effect on $u_1^8$ as $R_8$ does, but this does not happen for bigger $N$.

1.4.2 Encoding Complexity

We will try to find the encoding complexity, computational and memory, of an arbitrary $G_N$-coset code, with parameters $(N, K, \mathcal{A}, u_{\mathcal{A}})$, in a schematic way, in order to obtain a solid intuitive notion. Let $x_{C,E}(N)$ denote the worst case computational complexity and $x_{M,E}(N)$ denote the worst case memory complexity over the class of all $(N, K, \mathcal{A}, u_{\mathcal{A}})$, with a specific $N$. Let us also assume that the computational complexity of a mod-2 addition process is unitary (1 unit), and that the computational complexity of the $R_N$ operation is $N$ units. Encoding using matrix-vector multiplication leads to encoding complexity of $O(N^2)$. We can obtain a vast improvement over computational complexity if we use the recursive butterfly structure, which resembles the FFT structure. Observing Figure 1.4, we have

$$x_{C,E}(N) \leq N/2 + N + 2x_{C,E}(N/2)$$

where we consider $x_{C,E}(2) = 3$, as we can see in Figure 1.2. Computing Equation (1.69) recursively, we have that

$$x_{C,E}(N) \leq \frac{3}{2} N \log N, \text{ for all } N = 2^n, n \geq 1.$$  

(1.70)

So we have proven that the computational complexity of the encoding operation is $O(N \log N)$.

In order to compute $G_N$, we need to store $2(N/2)$ quantities from the previous recursive level, and this applies to all levels. Thus, the memory complexity of the encoding procedure is $O(N \log N)$.

1.5 Decoding

In this section we are first going to analyze some aspects on decoding complexity and, afterwards, we will give a thorough explanation of the decoding procedure.

1.5.1 Decoding Complexity

As we mentioned before, the main job of the decoder is to produce an estimate $\hat{u}_1^N$ of the source vector $u_1^N$. Recall that the source vector $u_1^N$ consists of a message part $u_{\mathcal{A}}$ and a frozen part $u_{\mathcal{A}'}$. Let us assume that the decoder consists of $N$ decision elements (DEs), one for each source element $u_i, 1 \leq i \leq N$. We first activate DE 1, then DE 2 and so on, until DE $N$. If $i \in \mathcal{A}$, then DE knows the value (it is a frozen position), so it just sets $\hat{u}_i = u_i$. If $i \in \mathcal{A}'$, then DE $i$ has to wait until it has received all the decisions from the previous DEs and then computes the likelihood ratio

$$L_N^{(i)}(y_1^N, \hat{u}_i^{i-1}) \triangleq \frac{W_N^{(i)}(y_1^N, \hat{u}_i^{i-1} | 0)}{W_N^{(i)}(y_1^N, \hat{u}_i^{i-1} | 1)}.$$  

(1.71)

The DE makes its decision by the following rule

$$\hat{u}_i = \begin{cases} 0, & \text{if } L_N^{(i)}(y_1^N, \hat{u}_i^{i-1}) \geq 1 \\ 1, & \text{otherwise} \end{cases}.$$  

(1.72)
This decision is then available to all succeeding DEs.

In this algorithm, at the decision level, there is only need for the computation of the LRs of the non-frozen positions. These computations can be done recursively. If we consider the DEs’ level to be the level with length $N$, then each calculation of an LR at this length $N$ requires the calculation of two LRs at length $N/2$. In the same manner, each calculation of an LR at length $N/2$ requires the calculation of two LRs at length $N/4$ and so on, until we reach length 1, where we can compute the LRs directly by

$$L^{(1)} = \frac{W(y_i | 0)}{W(y_i | 1)}.$$  

(1.73)

The recursive computing of the LRs is done with correctly adapting Equations (1.7) and (1.8), as

$$L^{(2i-1)}(y^N_1, \hat{u}^{2i-2}) = \frac{L^{(i)}(y^{N/2}_1, \hat{u}^{2i-2}) L^{(i)}(y^{N}_{N/2+1}, \hat{u}^{2i-2}) + 1}{L^{(i)}(y^{N/2}_1, \hat{u}^{2i-2}) + L^{(i)}(y^{N}_{N/2+1}, \hat{u}^{2i-2})},$$  

(1.74)

$$L^{(2i)}(y^N_1, \hat{u}^{2i-1}) = [L^{(i)}(y^{N/2}_1, \hat{u}^{2i-2})]^{1-2\hat{u}^{2i-1}}, L^{(i)}(y^{N}_{N/2+1}, \hat{u}^{2i-2}).$$  

(1.75)

This recursion has computational and memory complexity $O(N)$ for each $u_i, i \in \mathcal{A}$, since we have to compute and store $(2N - 1)$ LRs for each $u_i, i \in \mathcal{A}$. This procedure needs to be done $K = |\mathcal{A}|$ times, leading to overall complexity $O(N^2)$. Computational and memory complexities of this order of magnitude are clearly impractical from implementation perspective, leading us to adopt a more efficient algorithm, with much lower complexities.

The main idea of the efficient algorithm lays on the fact that the decoder graph consists of butterflies, as we can see in Figure 1.7. Butterflies are 2-by-2 bipartite graphs that tie together adjacent levels of the graph. There are four nodes in each butterfly pattern; the upper-left and the lower-left nodes of the butterfly are assembled from the same pair of LR, which are the upper-right and the lower-right nodes of the butterfly. So when the right nodes of a butterfly are computed once, in favour of the upper-left node, they do not need to be computed again in favour of the lower-left node; they are kept in memory and they are ready for use. Let us generalize the above in a formalized way. Each LR value in the pair of left nodes in a butterfly

$$(L^{(2i-1)}(y^N_1, \hat{u}^{2i-2}), L^{(2i)}(y^N_1, \hat{u}^{2i-1}))$$  

is assembled from the same pair of LRs of the right nodes

$$(L^{(i)}(y^{N/2}_1, \hat{u}^{2i-2}), L^{(i)}(y^{N}_{N/2+1}, \hat{u}^{2i-2})).$$  

(1.76)

(1.77)

Thus, the calculation of all $N$ LRs at length $N$ requires the calculation of $N$ LRs at length $N/2$ and not $2N$ calculations as in the previous algorithm. This is true for all lengths up to 1, i.e., $\{N, N/2, \ldots, 1\}$. The total number of distinct LRs needed to be calculated and stored with this algorithm is exactly $N(1 + \log N)$. Thus, both the computational and memory complexity of the SC decoder is $O(N \log N)$.

### 1.5.2 Recursive SC Decoding Algorithm

Let us now pay more attention on the exact procedure of the recursive SC-decoder, through an example with $N = 8$. Let us have a code with parameters $(N, K, \mathcal{A}, u_A) = (8, 4, (4, 6, 7, 8), (0, 0, 0, 0))$. The whole analysis will be based on the implementation of the decoder described in Figure 1.7. This analysis will provide us with a profound understanding not only on the backbone of the decoding procedure, but also on the order of the computations.

In Figure 1.7, there are $N(1 + \log N) = 32$ nodes; the leftmost level gives its result to $N = 8$ decision elements. Each node corresponds to an LR request, needed to be computed for the algorithm. The columns form the level of recursion; i.e., the first column corresponds to length $N = 8$, the second to $N/2 = 4$, and the last column corresponds to length $N/N = 1$. The first column is the decision
level and it is connected to the decision elements. The last column is the channel level and it is the level where the LRs are computed directly.

Each node in the graph has two labels; for example, the fourth node from the bottom in the second column has labels \((y_4^1, \hat{u}_1^4, e \oplus \hat{u}_4^{1, o})\) and 22. The first label signs that it is the node corresponding to \(L_4^{(3)}(y_4^1, \hat{u}_1^4, e \oplus \hat{u}_4^{1, o})\). The second label denotes that it is the 22nd node to be activated in the algorithm.

The decoding procedure starts with DE 1 activating node 1, requesting \(L_8^{(1)}(y_8^1)\). In order for this LR to be calculated, the two values of the right nodes of the butterfly are needed. Since they are not ready yet, node 1 activates node 2 for the calculation of \(L_4^{(1)}(y_4^1)\). Its right butterfly nodes are also not ready, so node 2 activates node 3 for the \(L_2^{(1)}(y_2^1)\). In the same manner, node 3 activates node 4 for \(L_1^{(1)}(y_1^1)\). This LR is on the channel level and it can be computed directly. When it is computed, its value is available to both the left nodes of the butterfly, namely nodes 3 and 23 (when time comes). Program control is passed from node 4 back to where it was received, namely node 3.

Node 3 asks again the question if it has everything needed from its right side; the answer is no, so it activates node 5 for \(L_1^{(1)}(y_5^1)\). This LR is computed directly and control is passed back again to node 3. Now node 3 has everything it needs in order to calculate its LR value, so it calculates it and passes control back to node 2. Node 2 still needs some information from the right side, so it activates node 6 for the calculation of \(L_2^{(1)}(y_2^1)\). Node 6 activates node 7, which computes \(L_1^{(1)}(y_3^1)\) and, afterwards, it activates node 8 for \(L_1^{(1)}(y_4^1)\). Now node 6 assembles its LR value and passes it to node 2, which in turn can now assemble its value and pass it to node 1.

Node 1 still needs information from the right side, so it activates node 9 for the calculation of \(L_4^{(1)}(y_9^1)\). Node 9 activates node 10, which activates nodes 11 and 12. Node 10 assembles its LR value, passes it to node 9, which activates node 13. Node 13 activates node 14, receives from it the value \(L_1^{(1)}(y_7^1)\), activates node 15 and receives from it \(L_1^{(1)}(y_8^1)\). Node 13 assembles \(L_2^{(1)}(y_6^1)\) and passes it back to node 9. Now node 9 assembles \(L_3^{(4)}(y_6^1)\) and passes control to node 1, which assembles \(L_8^{(1)}(y_1^1)\). This value was the initial value requested by DE 1, which now receives this LR value. Since \(u_1^1\) is a
frozen bit, DE 1 ignores this value and declares $u_1 = 0$, no matter what the LR said. Control is passed to DE 2.

DE 2 activates node 16 for the calculation of $L_S^{(2)}(y_1^8, \hat{u}_1)$. Node 16 is ready to assemble this LR from the already received LRs from nodes 2 and 9, namely $L_2^{(1)}(y_1^4)$ and $L_4^{(1)}(y_6^2)$. Once computed, $L_S^{(2)}(y_1^8, \hat{u}_1)$ is passed to DE 2, which ignores the value, since $u_1$ is a frozen bit, and declares $u_2 = 0$. Control is passed to DE 3.

DE 3 activates node 17 for the calculation of $L_S^{(3)}(y_1^8, \hat{u}_2^3)$. Information from the right is needed, so node 18 is activated. Node 18 can instantly assemble its value from the already computed $L_2^{(1)}(y_1^4)$ and $L_4^{(1)}(y_6^2)$. Next, node 17 activates node 19 for $L_4^{(2)}(y_6^2, \hat{u}_2)$. After receiving this value, node 17 assembles $L_S^{(3)}(y_1^8, \hat{u}_2^3)$, passes it to DE 3, which ignores this value, since $u_1$ is a frozen bit. DE 3 declares $u_2 = 0$ and passes control to DE 4.

DE 4 activates node 20 for the calculation of $L_S^{(4)}(y_1^8, \hat{u}_3^3)$, which can be computed directly from $L_4^{(2)}(y_4^1, \hat{u}_1 \oplus \hat{u}_2)$ and $L_4^{(2)}(y_6^2, \hat{u}_2)$. The value $L_S^{(4)}(y_1^8, \hat{u}_3^3)$ is passed to DE 4. $u_4$ is not a frozen bit, so the decision of the value of $u_4$, is made in accordance with $L_4^{(4)}(y_1^8, \hat{u}_3^3)$. Therefore, either $u_4 = 0$ if $L_4^{(4)}(y_1^8, \hat{u}_3^3) \geq 1$, or $u_4 = 1$ if $L_4^{(4)}(y_1^8, \hat{u}_3^3) < 1$. Program control is then passed to DE 5, which activated node 21 and so forth. The algorithm continues the same way until DE 8 receives $L_S^{(8)}(y_1^8, \hat{u}_7^7)$ and, depending on its value, it makes a decision on $u_8$.

Now that we have examined the exact way the decoder takes its decisions, it is time to make some important observations. First of all, we are going to examine the tree structure of the recursive procedure. When node 1 is activated for $L_8^{(1)}(y_1^8)$, the whole process in order to get this value can be seen as a tree rooted at node 1, with paths going from left to right; the tree is spanning all the $N = 8$ nodes at the rightmost (channel) level. This tree can be seen as splitting into two subtrees, one rooted at node 2 and the other rooting at node 9. Those two subtrees are disjoint; the first one is used for the calculation of $L_4^{(1)}(y_1^4)$ and the other one is used for the calculation of $L_4^{(1)}(y_6^2)$. The fact that the subtrees are disjoint can be used in order to compute each subtree independently and in parallel. Moreover, splitting the subtree into more subtrees can be done for all levels, except for the channel level, which is the last one, e.g. subtree rooted at node 2 can be seen as splitting into 2 subtrees, one rooting at node 3 and the other rooting at node 6, and so forth. As we see, a high degree of parallelism can be achieved.

Secondly, we will examine the butterfly structure units of the decoder. As we have mentioned before, butterflies are 2-by-2 bipartite graphs, formed by nodes of adjacent levels. The butterflies tie together two adjacent levels. For example, nodes 2, 18, 3 and 6 form a butterfly that ties together the levels with length 4 and length 2. The important fact is that the subtrees rooted at nodes 2 and 18 (the left nodes of the butterfly), split at the same subtrees rooted at nodes 3 and 6 (the right nodes of the butterfly). We have given the algorithm in such a way that the upper-left node is always activated first, the upper-right node is always activated second and the lower-right node is activated third. The lower-left node is always activated last and has the information it needs ready to be used. The fact that the lower-left node has always to wait for the result of the upper-left node and thus, for the result of the two right nodes, puts a constraint on the time of calculations. The upper-left node assembles its LR value using Equation (1.74) and the lower-left node assembles its LR value from Equation (1.75).

1.6 Construction of Polar Codes

We have already mentioned that the construction rule for polar codes is based on the minimization of the right-hand side of Equation (1.53). This means that a polar code construction algorithm will ideally compute the Bhattacharyya parameters $Z(W_N^{(i)}), \forall i \in A$ of all the new forged channels and sort them. Thus, it should create the information set $A$ by the indices of the $K$ out of $N$ most reliable channels.

For the BEC case, the Bhattacharyya parameters $Z(W_N^{(i)})$ can be calculated by the recursive formulas (1.47) and (1.48). Moreover, these parameters can be calculated with computational complexity
1.6. CONSTRUCTION OF POLAR CODES

\( O(N) \). Unfortunately, the BEC case is the only case where the code construction can be done easily and in an exact way at the same time. For every other case, we use approximate solutions for computing and sorting estimations of the Bhattacharyya parameters \( Z(W_N^{(i)}) \).

What makes things easier for an approximate solution is the fact that, as Arikan have shown in [1], the Bhattacharyya parameter \( Z(W_N^{(i)}) \) is the expectation of the RV:

\[
\sqrt{\frac{W_N^{(i)}(Y_1^i, U_1^{i-1}|U_i \oplus 1)}{W_N^{(i)}(Y_1^i, U_1^{i-1}|U_i)}}
\]

(1.78)

where \((U_1^N, Y_1^N)\) is sampled from the joint probability assignment

\[
P_{U_1^N, Y_1^N}(u_1^N, y_1^N) \triangleq 2^{-N} W_N(y_1^N|u_1^N).
\]

In a Monte Carlo approach, we generate, from the given distribution, samples of \((U_1^N, Y_1^N)\) and we calculate the empirical means \(\hat{Z}(W_N^{(i)})\). As we can see, the sample values of (1.78) are the square roots of the LRs, which are computed by the DEs in an SC decoder. So, interestingly, we can use an SC decoder in order to obtain estimations of the Bhattacharyya parameters \( Z(W_N^{(i)}) \). The information set \( A \) for such a decoder should be taken as the null set.
Chapter 2

Improved SC Decoders for Polar Codes

Polar codes, as we mentioned in the first chapter, are capacity achieving. Nevertheless, this exceptional behaviour only comes as an asymptotic property. What is important from an implementation perspective is the performance of polar codes at practical block lengths. As Arıkan noted himself, the performance of polar codes at short to moderate block lengths, under SC decoding, is mediocre. There are two possible explanations for this poor performance. The first one is that SC decoding has much worse performance than the performance of the Maximum Likelihood (ML) decoding. The second one is that polar codes are inherently weak at these lengths.

In this chapter we examine the improvement of the decoding performance. In the first section, we present a space-efficient implementation that improves the memory complexity of the simple SC decoder. This is going to prove useful in improving the computational complexities of the algorithms presented in the next sections of this chapter. Afterwards, in Section 2.2, SC decoding is represented as a path searching process on a code tree. In section 2.3, list SC decoder is presented and its structure and performance are examined thoroughly, while in section 2.4, another improved SC decoder, namely the stack SC decoder, is examined. Section 2.5 deals with the semi-parallel approach to the hardware architecture of the SC decoder.

2.1 Space-Efficient Implementation of SC Decoding

In the previous chapter, we mentioned that both the computational and memory complexities of the SC decoder are $O(N \log N)$. In this section, we are going to examine an implementation of the SC decoder that reduces the memory complexity to $O(N)$ [7]. The reason why, at first sight, the memory complexity seems to be $O(N \log N)$ is that in a naive implementation we store an LR (or equivalently seen, a pair of likelihoods) for each node in the decoding graph. In other words, exactly $N(\log N + 1)$ likelihood pairs need to be stored.

The space-efficient idea will be easily understood through a simple example. Let us consider the decoding graph of Figure 2.1, for $N = 4$. In order to decode $\hat{u}_1$, the likelihood pair $W_{4}^{(1)}(y_{1}^{4}|u_{1})$, $u_{1} \in \{0, 1\}$, needs to be computed and kept in a memory position. The key concept is that, after the decoding of $\hat{u}_1$, the pair of likelihoods $W_{4}^{(1)}(y_{1}^{4}|u_{1})$, $u_{1} \in \{0, 1\}$, will never be used again, therefore, the memory position they occupy can be freed. This free memory position can be reused to keep the next pair of likelihoods we are going to compute for the decoding of $\hat{u}_2$; this pair of likelihoods is $W_{4}^{(2)}(y_{1}^{4}, \hat{u}_1|u_{2})$, $u_{2} \in \{0, 1\}$. After the decoding of $\hat{u}_2$, this likelihood pair will never be used again, so the memory position they occupy can be freed. What is more, after the decoding of $\hat{u}_2$, the two likelihood pairs of the right stage, $W_{2}^{(1)}(y_{2}^{2}|u_{1} \oplus u_{2})$, $u_{1} \oplus u_{2} \in \{0, 1\}$, and $W_{2}^{(2)}(y_{2}^{2}|u_{2})$, $u_{2} \in \{0, 1\}$, are never going to be used again. This means that the memory positions that they are kept in, can be used for storing the next two likelihood pairs of this stage, i.e., $W_{2}^{(1)}(y_{2}^{2}, \hat{u}_1 \oplus \hat{u}_2|u_{3} \oplus u_{4})$, $u_{3} \oplus u_{4} \in \{0, 1\}$, and $W_{2}^{(2)}(y_{2}^{2}, \hat{u}_2|u_{4})$, $u_{4} \in \{0, 1\}$.

The modulo-2 sums of previously decoded codeword bits that are used in the recursive Equa-
2.2 SC as a path searching process on a code tree

A polar code with length $N$ can be represented by the code tree $T$, which is a full binary tree [6]. If we denote the set of nodes with $V$, and the set of edges with $E$, then $|V| = 2^{N+1} - 1$ and $|E| = 2^{N+1} - 2$. We call depth $d$ of a node the length of the path that begins in the root and ends at this particular node. The set of all nodes at a given depth is denoted by $V_d$, $d = 0, 1, \ldots, N$. The nodes $v \in V_N$ are called leaf nodes. The depth of the root node is always 0 and the depth of the leaf nodes is always $N$.
The edges are divided in $N$ levels $E_l$, $i = 1, 2, \ldots, N$. Each edge in $E_l$ declares either a 0 or 1, and is adjacent to a node belonging in depth $l - 1$ and a node belonging in depth $l$. Each node $v \in V_d$ has two descendant nodes that belong in $V_{d+1}$, except for the leaf nodes, that have no descendant. One descendant corresponds to the edge 0 and the other corresponds to the edge 1. We can see all these details in the example tree of Figure 2.2, with $N = 4$, $|V| = 31$, and $|E| = 30$.

A decoding path, as the one in the example of Figure 2.2, consists of subsequent edges in the code tree. A decoding path of length $i$ consists of $i$ edges and it is represented by the vector $v^i$, where $v_i$ corresponds to the binary label, (0 or 1), of the corresponding edge. The reliability of a decoding path is given by the channel transition probabilities $W_N^{(i)}(y^N, u^i|u_i)$. It is better, in terms of numerical stability and from a hardware perspective, to work with an equivalent, normalized version of the above reliability parameter; this is going to be also helpful in understanding the decoding path searching process more easily. The normalized version of the channel transition probabilities is the a posteriori probabilities (APPs)

$$P^{(i)}_N(v^i_1|y^N_1) = \frac{W^{(i)}_N(y^N_1, u^i_1|u_i)}{2P(y^N_1)}. \tag{2.2}$$

The normalization is accomplished by dividing the transition probabilities with the factor $2P(y^N_1)$; this results in a very interesting property: the sum of the APPs for all the decoding paths of a certain length $i$ is always 1, i.e.,

$$\sum_{v^i_1 \in \{0,1\}^i} P^{(i)}_N(v^i_1|y^N_1) = 1. \tag{2.3}$$

We can recognize this property in the example of Figure 2.2.

In the same way that we recursively compute the transition probabilities through Equations (1.37) and (1.38), we can also recursively compute the APPs

$$P^{(2i-1)}_N(v^{2i-1}_1|y^N_1) = \sum_{v^{2i}_1 \in \{0,1\}} P^{(i)}_N(v^{2i}_1 \oplus v^{2i}_1|y^N_1) \cdot P^{(i)}_N(v^{2i}_1|y^N_{i+1}) \tag{2.4}$$

$$P^{(2i)}_N(v^{2i}_1|y^N_1) = P^{(i)}_N(v^{2i}_1 \oplus v^{2i}_1|y^N_1) \cdot P^{(i)}_N(v^{2i}_1|y^N_{i+1}). \tag{2.5}$$

SC decoding is a depth-first search on the code tree. Standing on a node in depth $i$, there are two possible decoding choices, corresponding to the two outgoing edges of this node. These two edges are labeled with the two possible choices for $u_i$, i.e., $u_i = 0$ and $u_i = 1$. If $u_i$ is a frozen bit, then the decoder follows the edge that corresponds to the frozen value, no matter what the values of the two APPs are. If $u_i$ is an information bit, the decoder uses the APPs as metrics, and it chooses the edge with the largest metric. Clearly, this is a greedy search algorithm, where, in each level, only the edge with the largest metric out of the two, is selected for being part of the decoding path.

We can easily understand that with such a greedy approach, we cannot revisit any past decisions. This means that, in case of a decoding error in a certain bit, there is no possibility of correcting this error; it is kept as a decoding choice and may cause even more bit errors in this specific codeword, because of error propagation. For this reason, error propagation is an inherent problem of SC decoding. We will provide a deeper analysis on the characteristics of error propagation in SC decoding, in the following chapters of this thesis.

In the example of Figure 2.2, the blue line shows the decoding search path, while the red line shows the final estimated decoded codeword; in simple SC, the two lines always coincide. The enumerated nodes are the ones visited and chosen in the decoding procedure. The black nodes correspond to the nodes in each level whose metrics (APPs) are calculated, but finally not taken as a path choice. The metrics are also calculated for the frozen positions, although these metrics are not used for taking the decision in those positions. The grey nodes are the nodes whose metrics are not calculated at all. As we see, if we sum the metrics of the two descendant nodes that every node is split into, we get the
2.3 List SC Decoding of Polar Codes

As we saw, the performance of the simple SC decoder is quite worse than the bound of the ML performance. What would happen though if we could give the SC decoder some more opportunities? More specifically, instead of traversing one path at a time (simple SC) or all the paths at the same time (ML), let us traverse \( L \) paths at the same time, where \( 1 \leq L \leq 2^N \). For \( L = 1 \), we have the case of a simple SC decoder, while for \( L = 2^N \), we have the case of the ML decoder. In the beginning, we will describe and formalize the list SC decoding algorithm. After that we will examine the complexity of the list SC decoder. Finally, we will examine the error correcting performance of this algorithm, making also some notes on its advantages and drawbacks. The analysis given here was first introduced in [14] and [6].
2.3. LIST SC DECODING OF POLAR CODES

2.3.1 List SC Decoding Algorithm

As mentioned before, the SC decoder follows a depth-first approach, meaning that it explores only one path from the root node to the leaf nodes of the tree. The list SC decoder on the other hand, uses a breadth-first approach by searching more than one paths in each level of the tree, under a complexity constraint. The constraint is that it follows no more than \( L \) paths at the same time. This constraint \( L \) is thus named searching width.

At each level, the list SC decoder doubles the existing paths by appending on them either a 0 or a 1. The new APP for the new paths are computed, and if the candidate paths are more than \( L \), the excess paths with the lowest metrics are discarded. This way, at most \( L \) paths with the largest metrics at each level are kept in a list structure, so that they are again extended and used in the same manner on the next level.

At the last level of the tree, the leaf level, there are exactly \( L \) paths that survived, hopefully the ones having the \( L \) best metrics of all the possible ones on the entire tree. The decoder chooses the one that has the largest metric out of the final \( L \) paths, in the hope that it is also the one with the largest metric out of all the possible paths on the tree. Thus, the estimated codeword is declared by the binary labels \( u_1^N \) that correspond to the edges \( \{e_1, e_1, ..., e_N\} \) of this chosen path.

In the following, we will describe the steps of the list SC algorithm. We denote the set of candidate paths that correspond to the level-\( i \)-as \( L(i) \). They are kept in a list structure and updated at every level.

The list SC decoding algorithm operates as follows:

1. **Initialize the paths, level-0.** The initial list contains a null path with a probability metric set to 1.
   \[
   L(0) = \emptyset, \quad APP(\emptyset) = 1.
   \]

2. **Appending next bit.** All the existing paths are tail-expanded with one bit, once with a 0 once with a 1. Thus, the total number of the new candidate paths is doubled.
   \[
   L(i) = \left\{ (u_{i-1}^i, u_i) \mid u_{i-1}^i \in L(i-1), \, u_i \in \{0, 1\} \right\}, \quad \text{for each } u_i^i \in L(i). \tag{2.6}
   \]
   Each corresponding path metric is updated following the rules of (2.4) and (2.5).

3. **Delete half of the candidate paths.** In this step half of the candidate paths, the ones with the lowest probability metric are deleted, so that exactly \( L \) paths are preserved. If the candidate paths happened to be no more than \( L \), then no path is deleted and the algorithm proceeds to the next step.

4. **Repeat.** Repeat steps 2 and 3, until leaf-level (level \( N \)) is reached.

5. **Output the estimated vector.** When level \( N \) is reached, the path with the largest probability metric is decided by the decoder as the estimated codeword.

2.3.2 Complexity of List SC Decoder

After expanding the paths and choosing \( L \) of them, namely after steps 2 and 3 of the list SC algorithm, there are three possible scenarios for the parent paths. The first is that the parent path is forked into two paths. This means that both of the two children paths are kept in the candidate list. The second scenario is that the parent path is extended with a single choice, either 0 or 1. In this scenario only one of the two emerging paths is kept. The third scenario is that the parent path is discarded completely because both its children belong to the set of candidates with the lowest metrics.

The worst scenario concerning complexity is the first one. Each time a path is forked into two, in a naive implementation all of the data structures of the parent path would be duplicated and a copy would be given to each of the two children paths. The size of the data structures of the parent path are \( \Omega(N) \). The maximum possible number of splits is \( N \) times for each of the \( L \) paths. The above leads to a complexity of \( \Omega(LN^2) \) just for the copying.
2.3. LIST SC DECODING OF POLAR CODES

Fortunately, Tal and Vardy suggested a so-called “lazy copy” technique in [14], reducing the computational complexity to $O(LN \log N)$. This technique makes use of the fact that, most of the times, only a small fraction of the likelihoods that are contained in the data structure are needed for the decoding of the next bit. Thus, instead of copying the whole data structure of the parent path, it is sufficient to just copy those exact likelihoods that are needed. As we saw in the space-efficient implementation of the SC decoder, the total number of calculated likelihoods is $O(N \log N)$, thus, for each path, the total number of likelihoods that need to be potentially copied is $O(N \log N)$ per path. Since the number of paths that are being processed at the same time is $L$, we have a total computational complexity of $O(LN \log N)$ for copying. The remainder of the decoding process consists of $L$ parallel SC decoders, whose complexity is also $O(LN \log N)$.

2.3.3 Performance of List SC Decoder and CRC-Aided Decoder

In Figure 2.4 we can see the word error rate (WER) for a polar code of length $N = 2^{11}$ for an SNR range of 1 to 3 dB. Various list sizes up to $L = 32$ are considered, as well as the ML bound case. List SC decoding with $L = 1$ is equivalent to the simple SC decoder. As we clearly observe, the performance of the list SC decoder improves as the list size increases. At the same time though, a diminishing returns effect on the improvement can be observed, especially for the higher SNRs. For SNRs higher than 2 dB there is almost no gain by using list sizes more than $L = 4$ and for SNRs higher than 2.5 dB the same holds for using list sizes more than $L = 2$.

One interesting observation that Tal and Vardy made is the following. Many of the times that a decoding error occurred, it also happened that the choice error was actually done on the leaf level; this means that the correct codeword was actually part of the final set of $L$ candidate paths, but it was not selected because at the last stage another candidate had a higher metric. In order to overcome this problem they suggested the use of a CRC-aided list SC decoder. The decoding procedure follows exactly the same order as the list SC decoder with a modification only at the leaf level. At that level, the path that is finally chosen is the one with the correct CRC, or as we could more informally say, the one which “passes” the CRC. If more than one final paths have a correct CRC then the one among them with the highest probability metric is chosen.

Recall that the rate of a polar code is given by $R = \frac{K}{N}$, where $K = |A|$. The CRC-aided list SC decoder replaces $r$ of the $K$ information bits with the CRC of the remaining $K - r$ information bits, where $r > 0$. Thus, the rate of the resulting code is $R' = \frac{K-r}{N} < R$. This means that there is a rate penalty which depends on $r$ and should be taken into account in the simulations, especially when CRC-aided decoders are compared with non-CRC ones.

In Figure 2.5, the bit error rate (BER) for a polar code of length $N = 2^{11}$ is depicted, again for an SNR range of 1 to 3 dB. The list size for the three different list SC decoders is $L = 32$. The CRC used
2.4 Stack SC Decoding of Polar Codes

Chen, Liu and Lin presented another improved decoding scheme for polar codes, namely stack SC decoder in [6]. The observation that was made is that any decoding path on any level of the tree will always have a larger, or at least equal, metric than any of its children paths. Thus, if a metric of a path at level \( N \) is larger than the metric of another path at level \( i \), with \( i < N \), then it should also be larger than the metric of all the children paths of the latter.

This gives rise to the idea that we may be able to avoid processing \( L \) paths on a level before moving to the next level; a single candidate path can extend alone deep into the tree, until its metric is no longer the larger of all the candidate paths being kept on a stack structure. As soon as it no longer has the larger metric, the next candidate path with the largest metric begins extending down on the tree, again until no longer it has the highest metric, and so on. Once a path reaches the leaf level, it means that it still has the largest metric among the candidate paths, and is thus declared as the estimated codeword. This concept is similar to sphere decoding for multiple-input multiple-output (MIMO) systems.

2.4.1 Stack SC Decoding Algorithm

The candidate paths of the stack SC decoder are kept in an ordered stack structure \( S \). The top path in the stack has the largest metric, so whenever the leaf level is reached, this top stack is chosen and is popped out as the estimated codeword. A major difference with the list SC decoder, where all the paths have the same length in each level of the tree, is that in stack SC the candidate paths have different lengths throughout the entire decoding procedure.

We denote with \( D \) the depth of the stack structure, which we call the maximal stack depth, and with \( L \) the maximum number of paths with a certain length that are allowed to exist at the same time in the stack, which we call the searching width. Let also \( c_i \) denote the number of paths with length \( i \) during the decoding procedure. Thus, the vector \( c_i^N = (c_1, c_2, \ldots, c_N) \) helps in keeping track of the number of popping paths with all the possible lengths. A stack SC decoder with maximal stack depth \( D \) and searching width \( L \) is denoted as stackSC\((D,L)\). The steps of the decoding algorithm are described as follows:
2.4. STACK SC DECODING OF POLAR CODES

Figure 2.6: Code tree for stack SC with N = 4. Stack SC also manages to find the correct path with two fewer metric computations than list SC \( \hat{u}_1 = 1000 \) (taken from [6]).

1. **Initialize the path, level-0.** The null path is introduced in the stack structure and its metric is set to zero. The counting vector \( c_N \) is initialized also to zeros in all its positions. The current stack depth is set to 1, \( |S| = 1 \).

2. **Pop out.** Pop out the path \( u^i_1 \) that lies on the top of the stack and increase the corresponding counting position if the path is not null, \( c_{i-1} = c_{i-1} + 1 \).

3. **Expand.** If next bit, \( i \), is a frozen bit, then expand the vector \( u^{i-1}_1 \) with the frozen value. If it is an information bit, then expand the path both the possible ways, namely \( u^i_1 = (u^{i-1}_1, 0) \) and \( u^i_1 = (u^{i-1}_1, 1) \). Compute the corresponding probability path metrics appropriately following the update rules of (2.4) and (2.5).

4. **Delete and push.** Delete the path that lies on the bottom of the stack, if \( |S| > D - 2 \). Then, if on the previous step we expanded with a frozen bit, push the one extended path into the stack. Otherwise, for the non-frozen bit case, push the two new extended paths of the previous step into the stack.

5. **Compete and delete.** If \( c_{i-1} = L \), delete from the stack all the paths with length less than or equal to \( i - 1 \).

6. **Sort.** Sort the paths in the stack from bottom to top with ascending order.

7. **Repeat.** Repeat the algorithm from step 2 if the leaf level has not been reached yet.

8. **Stop and output.** When level \( N \) is reached by the top path of the stack, pop it out and declare its binary contents as the estimated codeword.

In the toy example illustrated in Figure 2.6 we notice that the stack SC algorithm finds the correct codeword with two fewer metric computations than the list SC algorithm.

2.4.2 Complexity of Stack SC Decoder

The ideas of the space-efficient implementation of SC decoding and the “lazy copy” technique can be also applied to the stack SC algorithm. Applying these techniques, the stack SC algorithm has a computational complexity of \( O(LN \log N) \), similar to the list SC. However, for the same \( L \), the average number of computations for the stack SC decoding algorithm is be lower. The reduction in the computational complexity is especially seen in the high SNR regime, where it is often clear which codeword is the correct one.
The memory complexity of stack SC decoder is $O(DN)$. In order to prevent performance deterioration, the stack depth needs to be as high as $LN$, leading to a memory complexity for the stack that is $O(LN^2)$. A hybrid version of the two algorithms was proposed in [6], which allows trade-offs between the computational and the memory complexity between the list SC algorithm and the stack SC algorithm.

Because of the high memory requirements and the non-deterministic decoding latency of stack SC decoding, list SC decoding is more attractive from an implementation perspective.
Chapter 3

Flip SC Decoder

Under successive cancellation (SC) decoding, polar codes are inferior to other codes of similar block-length in terms of frame error rate. While more sophisticated decoding algorithms such as list SC or stack SC decoding partially mitigate this performance loss, they suffer from an increase in complexity. In this chapter, we describe a new flavor of the SC decoder, called the SC flip decoder. Our algorithm preserves the low memory requirements of the basic SC decoder and adjusts the required decoding effort to the signal quality. In the waterfall region, its average computational complexity is almost as low as that of the SC decoder.

In the first section we recognize and analyze the effect of error propagation in SC decoding and we consider its impact in the performance of the SC decoder. We overcome this impact by creating a decoder that uses an “oracle” each time an error occurs, thus not allowing error propagation at all. This way we measure the total number of errors that were actual errors, introduced by the noise. After that we employ a second oracle-assisted decoder that can only intervene once and correct the first actual error caused by the channel. This oracle-assisted decoder depicts the performance bound of our idea. In section 3.2 we introduce our actual algorithm, namely SC flip decoder. Complexity and performance analysis is also analytically presented.

3.1 Error Propagation in SC Decoding

In SC decoding, erroneous bit decisions can be caused by channel noise or by error propagation due to previous erroneous bit decisions. The first erroneous decision is always caused by the channel noise since there are no previous errors, so error propagation does not affect the frame error rate of polar codes, but only the bit error rate.

3.1.1 Effect of Error Propagation

The erroneous decisions due to error propagation are caused by erroneous decision feedback, which in turns leads to erroneous partial sums. Erroneous partial sums can corrupt the output LLR values at all stages, including, most importantly, the decision LLRs at level $n$.

For example, assume that, for the polar code in Figure 3.1, the frozen set is $\mathcal{A}^c = \{1, 2, 5, 6\}$ and the information set is $\mathcal{A} = \{3, 4, 7, 8\}$. Moreover, assume that the all-zero codeword was transmitted and that $\hat{u}_3$ was erroneously decoded as $\hat{u}_3 = 1$ due to channel noise. Now suppose that the two LLRs that are used to calculate the next decision LLR (i.e., $L_3^{(4)}$), namely, $L_2^{(2)}$ and $L_2^{(6)}$, are both positive and $L_2^{(2)} > L_2^{(6)}$. By applying the $g$ node update rule with $u = u_3^{(3)} = \hat{u}_3 = 1$, the resulting decision LLR $L_3^{(4)} = L_2^{(6)} - L_2^{(2)}$ has a negative value which leads to a second erroneous decision, while with the correct partial sum $u = 0$ the decision would have been correct.
### 3.1. ERROR PROPAGATION IN SC DECODING

Figure 3.1: The computation graph of the SC decoder for \( N = 8 \). The \( f \) nodes are green and \( g \) nodes are blue and in the parentheses are the partial sums that are used by each \( g \) node.

![Computation Graph](image)

Figure 3.2: Histogram showing the relative frequency of the number of errors caused by the channel for a polar code with \( N = 1024 \) and \( R = 0.5 \) for three different SNR values.

![Histogram](image)
3.1. ERROR PROPAGATION IN SC DECODING

Figure 3.3: Histogram showing the relative frequency of the number of errors actually caused by the channel for Eb/N0 = 2.00 and three different code lengths, N = 1024, 2048, 4096.

3.1.2 Significance of Error Propagation

The foregoing analysis of the effects of error propagation insinuates the following question: Given that we had an erroneously decoded codeword with many erroneous bits, how many of these bits were actually wrong because of channel noise rather than due to previous erroneous decisions? In order to answer to this question, we employ an oracle-assisted SC decoder. Each time an error occurs at the decision level, the oracle corrects it instantaneously without allowing it to affect any future bit decisions. Moreover, the oracle-assisted SC decoder counts the number of times it had to correct an erroneous decision.

In Figure 3.2 we plot a histogram of the number of errors caused by channel noise (given that there was at least one error) for three different Eb/N0 values for a polar code with N = 1024 and R = 0.5 over an AWGN channel. We observe that most frequently the channel introduces only one error and that this behavior becomes even more prominent for increasing Eb/N0 values. In Figure 3.3 we plot a histogram of the number of errors caused by channel noise for polar codes with three different block lengths and R = 0.5 over an AWGN channel at Eb/N0 = 2 dB. We observe that, the relative frequency of the single error event increases with increasing block lengths. This happens because, as N gets larger, the synthetic channels $W_n^{(i)}(y_1^N, u_1^{i-1} | u_i)$ become more polarized, meaning that all information channels in $\mathcal{A}$ become better.

3.1.3 Oracle-Assisted SC Decoder

From the discussion in the previous section, it is clear that, by identifying the position of the first erroneous bit decision and inverting that decision, the performance of the SC decoder could be improved significantly. In order to examine the potential benefits of correcting a single error we employ a second oracle-assisted SC decoder, which is only allowed to intervene once in the decoding process in order to correct the first erroneous bit decision.

In Figure 3.4 we compare the performance of the SC decoder with that of the oracle-assisted SC decoder for a polar code of three block lengths and R = 0.5 over an AWGN channel. We observe that correcting a single erroneous bit decision significantly improves the performance of the SC decoder.
3.2 SCFlipDecoder

The goal of SC flip decoding is to identify the first error that occurs during SC decoding without the aid of an oracle.

3.2.1 SC Flip Decoding Algorithm

Assume that we are given a polar code of rate $\tilde{R} = \frac{k}{N}$ with a set of information bits $\tilde{A}$. We use an $r$-bit CRC that tells us, with high probability, whether the codeword estimate $\tilde{u}^N_1$ given by the SC decoder is a valid codeword or not. In order to incorporate the CRC, the rate of the polar code is increased to $R = \tilde{R} + \frac{r}{N} = \frac{k + r}{N}$, so that the effective information rate remains unaltered. Equivalently, the set of information bits $\tilde{A}$ is extended with the $r$ most reliable channel indices in $\tilde{A}^c$, denoted by $\tilde{A}^r_{r_{\text{max}}}$. Thus, $\tilde{A} = \tilde{A} \cup \tilde{A}^r_{r_{\text{max}}}$.

The SC flip decoder starts by performing standard SC decoding in order to produce a first estimated codeword $\tilde{u}^N_1$. If $\tilde{u}^N_1$ passes the CRC, then decoding is completed. If the CRC fails, the SC flip algorithm is given $T$ additional attempts to identify the first error that occurred in the codeword. To this end, let $\mathcal{U}$ denote the set of the $T$ least reliable decisions, i.e., the set containing the indices $i \in \tilde{A}$ corresponding to the $T$ smallest $|P_n^{(i)}(y^N_1, \tilde{u}^{i-1}_1 | u_i)|$ values. After the set $\mathcal{U}$ has been constructed, SC decoding is restarted for a total of no more than $T$ additional attempts. In each attempt, a single $\tilde{u}_k, k \in \mathcal{U}$, is flipped with respect to the initial decision of the SC algorithm. The algorithm terminates when a valid codeword has been found or when all $T$ additional attempts have failed. Note that, for $T = 0$, SC flip decoding is equivalent to SC decoding.

The SC flip algorithm is formalized in the SCFlip($y^N_1, \mathcal{A}, k$) function in Fig. 3.5. The SC($y^N_1, \mathcal{A}, k$) function performs SC decoding based on the channel output $y^N_1$ and the set of non-frozen bits $\mathcal{A}$ with a slight twist: when $k > 0$, the codeword bit $u_k$ is decoded by flipping the value obtained from the decoding rule.

Note that SC flip decoding is similar to chase decoding for polar codes [12]. The main differences are that SC flip decoding only considers error patterns containing a single error and that these error patterns are not generated offline using the a-priori reliabilities of the synthetic channels $W_n^{(i)}(y^N_1, u^{i-1}_1 | u_i)$, but online using the decision LLRs $L_n^{(i)}$, which reflect the actual reliabilities of the bit decisions for each transmitted codeword and channel noise realization.
3.2. SCFLIPDECODER

1: function SCFLIPDECODER($y_i^N, A, T$)
2: \( (\hat{u}_i^N, L(y_i^N, \hat{u}_i^{1−1}|u_i)) \leftarrow \text{SC}(y_i^N, A, 0); \)
3: if $T > 1$ and $\text{CRC}(\hat{u}_i^N) = \text{failure}$ then
4: \( U \leftarrow i \in A \text{ of } T \text{ smallest } |L(y_i^N, \hat{u}_i^{1−1}|u_i)|; \)
5: for $j \leftarrow 1$ to $T$ do
6: \( k \leftarrow U(j); \)
7: \( \hat{u}_i^N \leftarrow \text{SC}(y_i^N, A, k); \)
8: if $\text{CRC}(\hat{u}_i^N) = \text{success}$ then
9: break;
10: end if
11: end for
12: end if
13: return $\hat{u}_i^N$;

Figure 3.5: SC flip decoding with maximum trials $T$.

3.2.2 Complexity of SC Flip Decoding

In this section, we derive the worst-case and average-case computational complexities of the SC flip algorithm, as well as its memory complexity.

Proposition 3.1. The worst-case computational complexity of the SCFLIP algorithm defined in Fig. 3.5 is $O(TN \log N)$.

Proof. SC decoding in line 2 has complexity $O(N \log N)$ and the computation of the CRC in line 3 has complexity $O(N)$. Moreover, the sorting step in line 4 can be implemented with complexity $O(N \log N)$ (e.g., using merge sort). Finally, the operations in the loop (lines 5–11) have complexity $O(N \log N)$ and the loop runs $T$ times in the worst case. Thus, the overall worst-case complexity is $O(TN \log N)$. \hfill \square

Proposition 3.1 shows that, in the worst case, the complexity of our algorithm increases linearly with the parameter $T$, meaning that its complexity scaling is no better than that of SC list decoding. However, if we consider the average complexity, then the situation is much more favorable, as the following result shows.

Proposition 3.2. Let $P_e(R, \text{SNR})$ denote the frame error rate of a polar code of rate $R$ at the given SNR point. Then, the average-case computational complexity of the SCFLIP algorithm defined in Fig. 3.5 is $O(N \log N (1 + T \cdot P_e(R, \text{SNR})))$, where $R = \frac{k+r}{N}$.

Proof. It suffices to observe that the loop in lines 5–11 runs only if SC decoding fails and the CRC detects the failure, which happens with probability at most $P_e(R, \text{SNR})$. \hfill \square

As the SNR increases, the FER drops asymptotically to zero. Thus, for high SNR the average computational complexity of SC flip decoding converges to the computational complexity of SC decoding. In other words, SC flip exhibits an energy-proportional behavior where more energy is spent when the problem is difficult (i.e., at low SNR) and less energy is spent when the problem is easy (i.e., at high SNR).

Proposition 3.3. The SCFLIP algorithm defined in Fig. 3.5 requires $O(N)$ memory positions.

Proof. SC decoding in line 2 requires $O(N)$ memory positions. The storage of the CRC calculated in lines 3 and 8 requires exactly $C$ memory positions, where $C \leq N$. The sorting step in line 4 can be implemented with $O(N)$ memory positions (e.g., using merge sort), while storing the $T$ smallest values requires exactly $T$ memory positions, where $T \leq N$. Moreover, the SC decoding performed in line 7 can re-use the memory positions of the SC decoding in line 2, so no additional memory is required. Thus, the overall memory scaling behavior is $O(N)$. \hfill \square
3.2. SCFLIPDECODER

In Figure 3.6 we compare the performance of the SC flip decoder with $T = 4, 16, 32$, and a 16-bit CRC with the SC decoder and the oracle-assisted SC decoder described in Section 3.1.3. Note that the oracle-assisted decoder characterizes a performance bound for the SC flip decoder. We observe that SC flip decoding with $T = 4$ already leads to a gain of one order of magnitude in terms of FER at Eb/N0 = 3.5 dB. With $T = 32$, we can reap all the benefits of the oracle-assisted SC decoder, since the $T = 32$ curve is shifted to the right with respect to the oracle-assisted curve by an amount that corresponds exactly to the rate loss incurred by the 16-bit CRC.

In Figure 3.7 we depict the same curves for a code length $N = 4096$, while keeping the ratio $T/N$ constant. We observe that it seems to become more difficult to reach the bound performance of the oracle-assisted SC decoder. As $N$ increases, the channels get more polarized, which would suggest the opposite behavior. However, at the same time, the absolute number of the possible positions for the first error increases as well. Our results suggest that the aforementioned negative effect negates the positive effect of channel polarization.

In Fig. 3.8, we compare the performance of standard SC decoding, SC flip decoding, and SC list decoding. We observe that the performance of the SC flip decoder with $T = 32$ is almost identical to that of the SC list decoder with $L = 2$, but with half the computational complexity at high Eb/N0 values and half the memory complexity at all Eb/N0 values. For higher list sizes, such as $L = 4$, SC list decoding outperforms SC flip decoding, at the cost of significantly higher complexity, since the performance of SC flip decoding is limited by the fact that it can only correct a single error.

3.2.4 Average Computational Complexity of SC Flip Decoding

In Figure 3.9, we compare the average computational complexity of standard SC decoding, SC list decoding, and SC flip decoding. We observe that, as predicted by Proposition 3.2, at low SNR the average computational complexity of SC flip decoding is $(T + 1)$ times larger than that of SC decoding but at higher SNR the computational complexity is practically identical to that of SC decoding. Moreover, the energy-proportional behavior of SC flip decoding is evident since, contrary to SC list decoding, the computational complexity decreases rapidly with decreasing difficulty of the decoding problem (i.e., increasing SNR). We also emphasize that SC flip decoding is not a viable option for the low SNR region, but this a not a region of interest for practical systems because the FER is very high.
Figure 3.7: Frame error rate of SC decoding, SC flip decoding with $T = 4, 16, 32$ and the oracle-assisted SC decoder for a polar code of length $N = 4096$ and $R = 0.5$.

Figure 3.8: Frame error rate of SC decoding, SC flip decoding with $T = 32$ and SC list decoding with $L = 2, 4$ for a polar code of length $N = 1024$ and $R = 0.5$. 
3.2 SCFLIPDECODER

Figure 3.9: Average complexity of SC flip decoding normalized with respect to the complexity of SC decoding for a polar code of length $N = 1024$ and $R = 0.5$.

### 3.2.5 Implementation Considerations

In order to derive the computational complexity results in Section 3.2.2, we assumed that SC decoding is restarted from scratch for each trial of SC flip decoding. However, in a real implementation this is not necessary as SC decoding can just be continued from the index $k$. Moreover, we assumed that the sorting of the decision LLRs is performed after the first round of SC decoding is completed. However, the sorting can also be performed online by using insertion sort. The worst-case complexity of insertion sort is $O(T^2)$. If $T \ll N$, this may be acceptable (or even lower than merge sort) in practice. Furthermore, since the bit decisions $\hat{u}_i$ are produced serially by the SC decoder, in a hardware implementation the calculation of the CRC can be done in parallel to SC decoding, thus completely masking its delay.
Chapter 4

Conclusion

In this diploma thesis our major concern have been to improve the finite length performance of polar codes under SC decoding, while at the same time keeping computation and memory complexity as low as possible. We have explored the impact of error propagation on SC decoding and have shown the significant performance improvement that can be gained by correcting the first error that occurs in the decoder. The bound in the performance improvement for this idea was explored.

We have introduced *successive cancellation flip* decoding for polar codes. This algorithm improves the frame error rate performance by opportunistically retrying alternative decisions for bits that turned out to be unreliable in a failing initial decoding iteration. By exploring alternative passes in the decoding tree one after another until a correct codeword is found, the average complexity and memory requirements are kept low, while approaching the performance of more complex tree-search based decoders.
Chapter 5

Bibliography


