Fast List Decoding of Polar Codes: Algorithms and Implementation

A Project Report

submitted by

SRAVAN KUMAR ANKIREDDY

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Fast List Decoding of Polar Codes: Algorithms and Implementation**, submitted by **Sravan Kumar Ankireddy**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Andrew Thangaraj
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Prof. Radha Krishna Ganti
Research Co-Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: IIT Madras, Chennai

Date: 5th May 2019

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my guide, Prof. Andrew Thangaraj, for providing me this exciting and compelling project to work on, which gave me a chance to explore various stages in the development and optimization of algorithms. I am grateful to him for being patient with me while explaining the concepts, which helped me overcome the challenges faced during my work on this project. The ease with which he handles any new topic using the fundamentals was always a massive inspiration and motivated me to work harder. I would also like to thank Prof. Radha Krishna Ganti for his continuous support while working on the embedded implementation, which exposed me to various intricacies of developing an efficient algorithm that can perform well even with limited resources. I can confidently say that this experience made me a better engineer and made me appreciate how designing an algorithm goes beyond the theoretical analysis and the various challenges faced depending on the application.

I would also like to thank all the professors at the Electrical Engineering Department for the knowledge that has helped me transform from a clueless undergraduate student to a clear-minded individual with a purpose. I would like to especially thank Prof. Pradeep Sarvepalli, who introduced me to the beautiful subject of Coding Theory and deepened my interest and appreciation for the field of Communications. Finally, I would like to express my gratitude to my parents for their belief and constant support in all my decisions, and my friends and wingmates, who made my life at the institute memorable.

ABSTRACT

KEYWORDS: Polar Codes; Successive Cancellation Decoding; List Decoding; Belief Propagation; 5G NR

This project aims to develop efficient encoder and decoder algorithms for polar codes. In particular, we studied the two limitations of decoders for polar codes: error correction performance and computational complexity. The first decoding algorithm proposed for polar codes, successive cancellation (SC) decoding, performs very poorly compared to its LDPC counterparts. To improve this, we implemented a list decoding algorithm, a generalization of SC decoder where L most probable decoding paths are considered concurrently at each decoding stage. Additionally, a short length Cyclic Redundancy Check (CRC) is added to identify the correct codeword among L estimates. To reduce the computational complexity, a number of simplifications based on the identification of special nodes have been incorporated, and the Log-Likelihood Ratio (LLR) update functions have been optimized to reduce the amount of data copying, resulting in improved decoding speed. Finally, we explored some novel ideas like concatenating Low-Density Parity Check (LDPC) codes with Polar Codes to achieve superior decoding performance with a trade-off for computational complexity. And using a modified Belief Propagation (BP) decoder to traverse the Tanner graph structure of Polar Codes, which enables the reuse of optimized architecture of Message Passing Algorithm (MPA) with a trade-off for error-correcting performance

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BEC	Binary Erasure Channel
BER	Bit Error Rate
BLER	Block Error Rate
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CRC	Cyclic Redundancy Check
CRC-BP	CRC conacatenated Belief Propagation Decoder
FER	Frame Error Rate
FFT	Fast Fourier Transform
GF	Galio Field
HWF	Hardware Friendly
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratio
PM	Path Metric
Rep	Repetition
RS	Reed-Solomon
SC	Successive Cancellation
SCL	Successive Cancellation List
SNR	Signal to Noise Ratio
SPC	Single Parity Check
SSC	Simplified Successive Cancellation
SSCL	Simplified Successive Cancellation List

NOTATION

N	Length of code
K	Number of information bits
p	Number of parity CRC bits added
$\mathcal{A}$	Set of information bit positions
$\mathcal{A}^{\mathcal{C}}$	Set of frozen bits
u	Transmit vector - before encoding
x	Transmit vector - after encoding
$u_{\mathcal{A}}$	Information bits
$u_{\mathcal{A}^c}$	Frozen bits
G_N	Generator matrix for length N polar code
$G_{\mathcal{A}\mathcal{A}}$	KxK sub-matrix of G containing rows and columns corresponding to information bit p
$G_{\mathcal{A}\mathcal{A}^c}$	KxK sub-matrix of G containing rows and columns corresponding to frozen bit positio
ν	Index of node
α_ν	LLR values/Beliefs vector corresponding to node ν
β_ν	Decisions/estimates vector corresponding to node ν
ε	Erasure probability of Binary Erasure Channel
$\mathcal{W}$	B-DMC channel $\mathcal{W}$
$I(\mathcal{W})$	Symmetric channel capacity of channel $\mathcal{W}$

CHAPTER 1

INTRODUCTION

The design of capacity achieving codes has been pursued by researchers for decades. While Claude Shannon introduced the concept of channel capacity [1] more than 70 years ago, it was not until relatively recently that we came to close to achieving this. While Reed-Solomon (RS) and Bose-Chaudhuri-Hocquenghem (BCH) have good error correction performance and are in widespread use even today, it's not until the discovery of turbo codes [2] in 1993 that error-correcting codes approaching the channel capacity were found. Also the Low Density Paity Check (LDPC) codes discovered by Robert Gallager in 1960s [3] and independently rediscovered by David McKay in 1997.

In 2008 Erdal Arikan introduced polar codes [4], the first codes to asymptotically achieve the symmetric channel capacity of memoryless channels. And the important aspect here is, this is done explicit, non-random construction, using a low complexity successive cancellation decoder. This is made possible by a phenomenon called channel polarisation, in which certain bits can always be estimated reliably while others are completely unreliable. The fraction of reliable bits approaches close to channel capacity as the length of the code increases and thus achieves channel capacity asymptotically.

Despite the simple decoder algorithm and capacity achieving property, polar codes fare worse than LDPC in two aspects. One is the throughput, which can be attributed to the serial nature of successive cancellation decoding. Due to this, the decoder algorithm has very low parallelism and is challenging to improve the speed. The second aspect is that polar codes achieve capacity only asymptotically and fare worse than LDPC codes at moderate lengths in terms of error correcting performance. Hence to tackle these issues advances in decoding algorithms have been made, resulting in more complicated but better error correction performance and lower latency. One can say that these advances played a pivotal role in the selection of polar codes for the control channel in upcoming 5G NR deployment.

CHAPTER 2

BACKGROUND

In this chapter we introduce the concept of channel polarisation and construction of polar codes. We then look at different encoding and decoding methods for polar codes. This chapter proceeds with a description of successive cancellation Decoder, an early low complexity decoder. Then we look at simplified successive cancellation decoder, which provides a significant improvement in speed by utilising some special types of nodes. Then we look at the simplified successive list decoder, which finally made possible the error correction performance of polar codes good enough in comparison with LDPC codes. Finally we also describe the belief propagation decoder and its implementation for decoding polar codes.

2.1 Channel Polarisation

A channel can broadly be described as a medium for transmission of information. The reliability of a channel, which is the probability with which the message transmitted is received correctly, gives us a notion of whether a channel is good or bad. For a good channel, the symmetric capacity approaches to 1. In this report, we are going to consider exclusively B-DMCs, described below.

A generic B-DMC can denoted as $\mathcal{W} : \mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{X} \in \{0, 1\}$ is the input alphabet and $\mathcal{Y}$ is the output alphabet which is arbitrary. We define the transition probability $\mathcal{W}(y|x)$ where $x \in \mathcal{X}, y \in \mathcal{Y}$. Also we will use symmetric channel capacity $I(\mathcal{W})$ as a measure for quality of channel. It is defined as

$$I(\mathcal{W}) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} \frac{1}{2} \mathcal{W}(y|x) \log_2 \frac{\mathcal{W}(y|x)}{\frac{1}{2}\mathcal{W}(y|0) + \frac{1}{2}\mathcal{W}(y|1)}$$

In his seminal paper, Erdal Arikan introduced the concept of channel polarisation, a method in which a set of good channels and bad channels can be created from a given

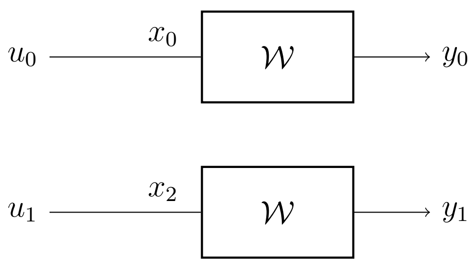


Figure 2.1: B-DMC with no transformation

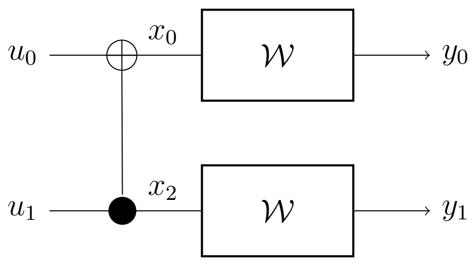


Figure 2.2: B-DMC with two fold Polar transformation

set of N independent and identical copies of a B-DMC $\mathcal{W}$. This can be thought of as aggregating and redistributing the the combined capacity. This operation is done in two stages, channel combining and channel splitting.

In channel combining we begin with N copies of W and combine them in a recursive manner to result in a vector channel $\mathcal{W}_N : \mathcal{X}^{\mathcal{N}} \rightarrow \mathcal{Y}^{\mathcal{N}}$, where N is a power of two, $N = 2^n, n \geq 0$. And in channel splitting, we split the above vector channel into a set of N binary-input coordinate channels denotes as $W_N^{(i)} : \mathcal{X} \rightarrow \mathcal{Y}^{\mathcal{N}} \times \mathcal{X}^{i-1}$, $1 \leq i \leq N$, and the transition probabilities, derivation of which is provided in [4], as:

$$\mathcal{W}_N^{(i)}(y_N^i, u_1^{i-1}|u_i) = \sum_{u_{i+1}^N \in \mathcal{X}^{N-i}} \frac{1}{2^{N-1}} \mathcal{W}_N(y_1^N|u_1^N) \tag{2.1}$$

where $(y_1^N|u_1^N)$ denotes the output of $\mathcal{W}_N^{(i)}$ and u_i its input.

Let's look at the case of two fold polar transformation. When two bits $(u_0, u_1) \in \mathcal{X}^2$ are transmitted using two independent B-DMC W, two values $(y_0, y_1) \in \mathcal{Y}^2$ are received, as shown in Fig 2.1. The mutual information values are:

$$I(Y_0, Y_1; U_0) = I(\mathcal{W}) = I(Y_0, Y_1; U_1)$$

However, if (u_0, u_1) are transformed into (x_0, x_1), as shown in Fig 2.2, the mutual

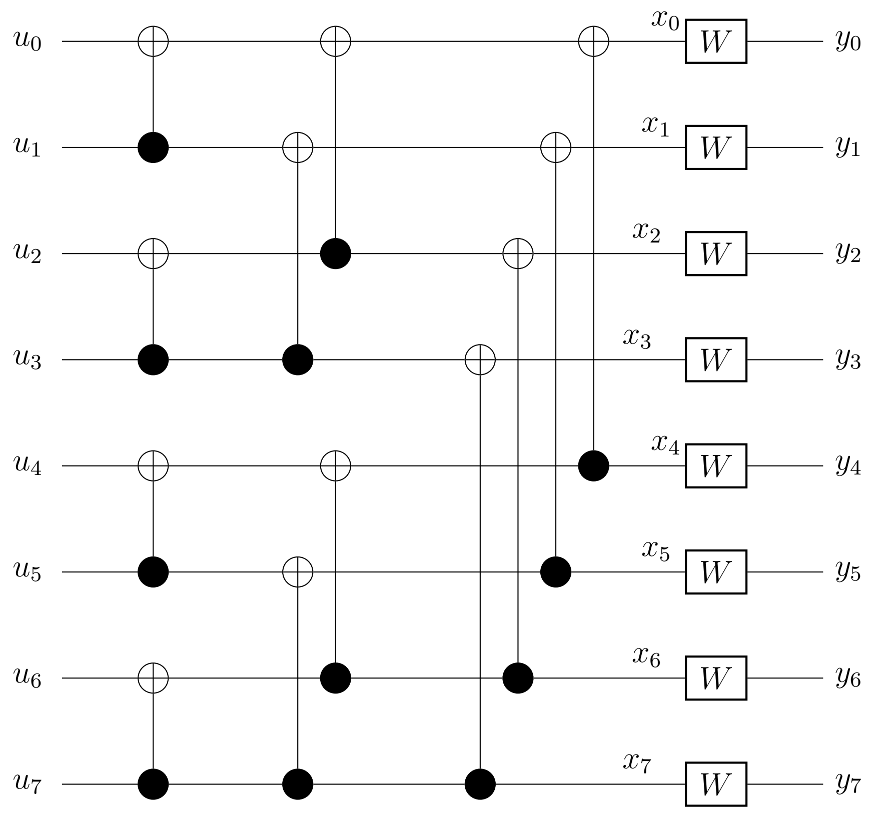


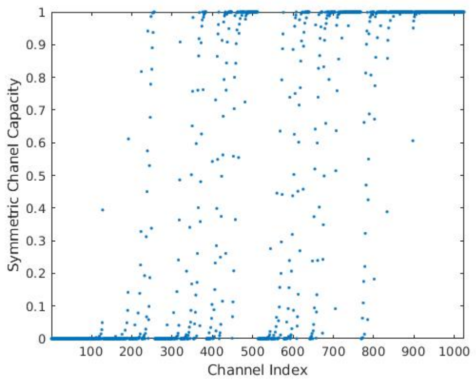
Figure 2.3: B-DMC with eight fold Polar transformation

information becomes

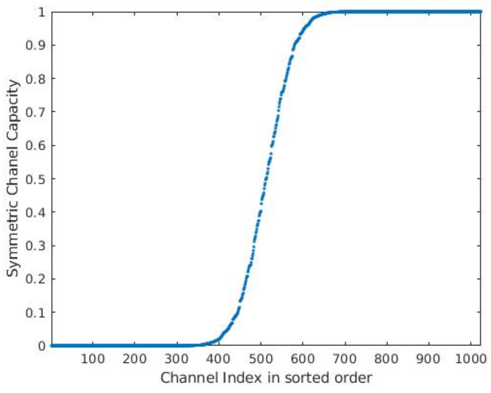
$$I(Y_0, Y_1; U_0) \leq I(\mathcal{W}) \leq I(Y_0, Y_1; U_1)$$

The proof of the above inequality is presented in [4].

We know that $I(\mathcal{W}) = 1$ indicates error free transmission and $I(\mathcal{W}) = 0$ indicates a completely unreliable channel. Hence, we can see that the transformed channel resulted in decreasing the probability of error for u_1 whereas increasing the probability of error for u_0. Fig 2.3 shows the polar transformation for a length 8 polar code. And as the number of channels increases *i.e,* as $N \to \infty$, for any i the probability of correctly estimating u_i approaches either 1 or 0.5 and the fraction of such i approaches the channel capacity *i.e,* rate of the code R approaches the capacity of the channel C. Fig 2.4a shows the effect of polarisation on a BEC with erasure probability $\varepsilon = 0.5$ for a length 1024 polar code. If we sort the channels based on symmetric channel capacity $I(\mathcal{W})$, we can see from Fig 2.4b that the fraction of reliable channels approach the capacity $C = 1 - \varepsilon$ which is 0.5.



(a) Chanel Polarisation effect unsorted



(b) Chanel Polarisation effect sorted

Figure 2.4: Effect of Channel Polarisation on $I(\mathcal{W})$

2.2 Construction of Polar Codes

The construction of a polar code of length N and rate $^K/_N$ is essentially a selection problem, where we need to select the best K channels out of N polarised channels. This is same as forming a information set A_γ of size K. Ideally this should be done by calculating the Bhattacharyya parameters $\{Z(W_N^{(i)}) : 1 \leq i \leq N\}$ and sorting them. But this is computationally inefficient and hence this is reformulated into a decision problem: Given a threshold $\gamma \in [0, 1]$ and an index $i \in \{1, \ldots, N\}$, decide the channel index i corresponds to a good channel or bad channel *i.e,* whether $i \in A_\gamma$. This can be repeated for various settings of γ until we obtain an information set A_γ of size K. One exception to this is a BE and BSC where the parameters $\{Z(W_N^{(i)})\}$ can be exactly calculated recursively was done in [4]. A practical method to determine frozen bit locations for Gaussian Channel (AWGN) was provided in [5].

CHAPTER 3

ENCODING

3.1 Encoding of Polar Codes

There are two ways in which the encoding can be performed. One is non-systematic encoding where we don't preserve the message bits to appear in the codeword and the other is systematic encoding where the codeword includes the message bits unaltered.

In non-systematic encoding, the procedure for encoding is very similar to the process to channel polarisation discussed earlier. Consider the (N, K) where N is the length of codeword and K is the length of information bits. Lets denote the information bit set and frozen bit set as A and A^C respectively, such that $A, A^C \subseteq \{1, \ldots, N\}$ and $\{A + A^C\} = \{1, \ldots, N\}$. The $N - K$ least reliable bits corresponding to set A^C are set to 0 and the K most reliable bits are initialised with message vector of length K. Now the encoding is carried out by recursively performing the polar transformation as shown in

This operation can be described mathematically using a generator matrix G_N of size $N \times N$, which is constructed from the two-bit transformation matrix F represented by

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

To construct a generator matrix G_N from F we use the Kronecker power as:

$$G_N = F^{\otimes log_2^N} = F \otimes F^{\otimes log_2^N - 1}$$

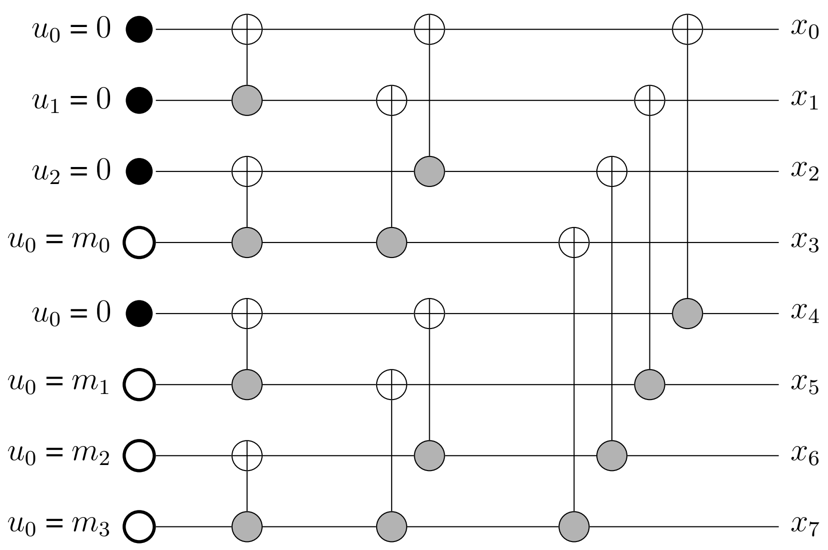


Figure 3.1: Encoding of (8,4) Polar code with frozen nodes set to "0"

where $F^{\otimes 1} = F$. For example, G_8 is given below

$$G_8 = F^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Now the encoding is simply the matrix multiplication of row vector $u_{1\times N}$, where $N-K$ indices are initialised to 0 and K indices are initialised with information bits. And hence the codeword x is given by:

$$x_{1\times N} = u_{1\times N} G_{N\times N} \tag{3.1}$$

Since the above expression is a matrix multiplication, the time complexity comes out to be $O(N^3)$. But by taking advantage of the butterfly like structure, shown in Fig 4.8 , similar to FFT, the complexity of encoding can be brought down to $O(N \log_2 N)$.

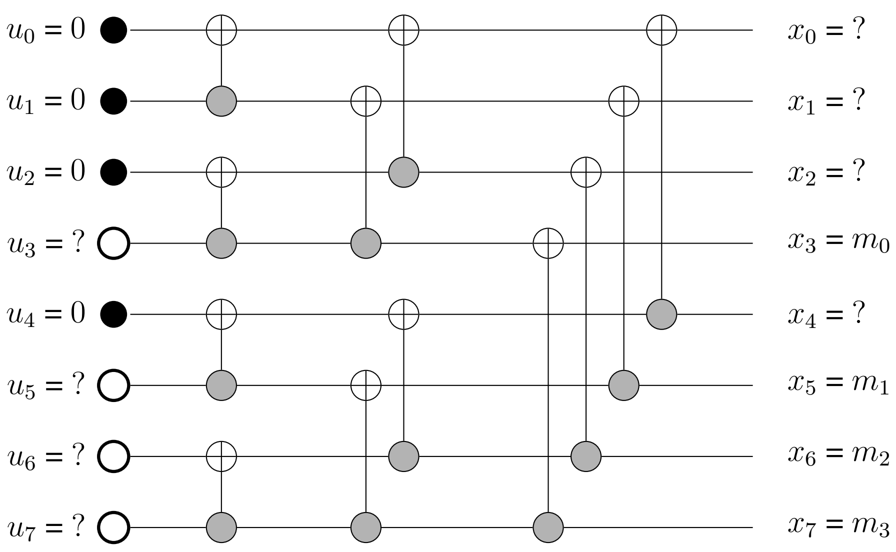


Figure 3.2: Systematic encoding of (8,4) Polar code with information bits preserved

3.2 Systematic Encoding of Polar Codes

Alternatively, one can perform systematic encoding, introduced in [6], where the codeword can be directly separated into K information bits and N parity bits. In systematic encoding, we start by initialising the frozen bits in u to 0 *i.e,* $u_A^C = 0$ and x_A to information bits and then solving for the remaining bits. The equation for calculation parity bits in x is given as follows:

$$x_{A^C} = x_A (G_{AA})^{-1} G_{AA^C} \tag{3.2}$$

where G_{AA} is submatrix of G whose rows and columns corresponds to information bit indices and G_{AA^C}'s rows and columns corresponds to frozen bit indices. There are advantages of systematic encoding over non-systematic encoding such as improved BER performance and faster decoding since extraction of information bits is straight forward.

3.3 Binary tree representation of Polar Codes

If we observe the construction and encoding process of a polar code, it is recursive and a polar code of length N can be represented as concatenation of two polar codes of length $^{N}/_{2}$. Hence a good choice of representation for a polar code is binary tree. The depth breadth of tree would be N and the depth would be $\log_2 N + 1$, with N leaf nodes

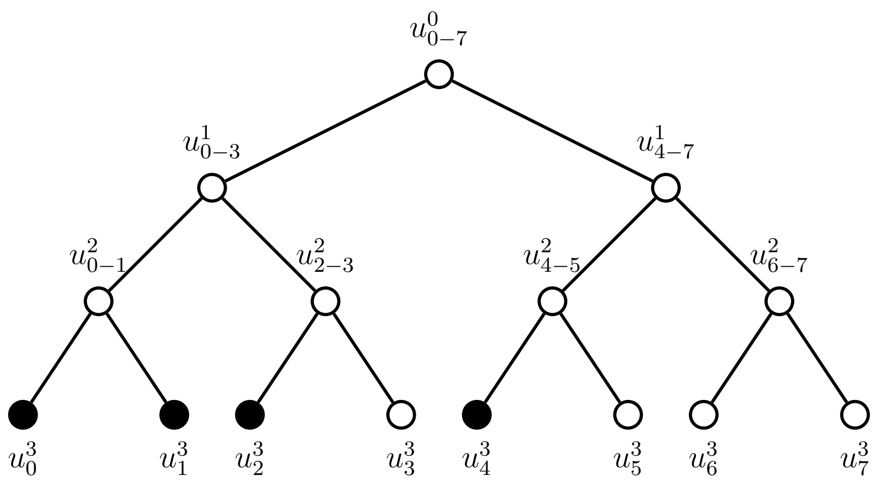


Figure 3.3: Binary tree structure of (8,4) polar code with frozen and information nodes

representing the information bit nodes and the frozen bit nodes. For representation purposes, in the Fig 3.3, we represent the information bit nodes using white and frozen bit nodes using black. And u^l_{i-j} represents vector corresponding to bits i to j at level l.

CHAPTER 4

DECODING

4.1 Successive Cancellation Decoder

One of the interesting properties of polar codes is their ability to achieve capacity with a very low complexity decoding algorithm. Here we look at the Successive Cancellation Decoding algorithm, provided in [4]. As the name suggests, SC decoding algorithm proceeds by decoding the bits sequentially starting with u_0. For a frozen bit node, it is always decoded as 0. And while decoding the u_i, the available bits u_0 to u_0^{i-1} which are represented by the vector u_{i-1} are used according to decode u_i according to the following rule:

$$\hat{u_i} = \begin{cases} 0, & \text{if } i \in A \text{ and } Pr[y, \hat{u_0}^{i-1} | u_i = 0] \geq Pr[y, u_0^{i-1} | \hat{u_i} = 1] \\ 1, & \text{if } i \in A \text{ and } Pr[y, \hat{u_0}^{i-1} | u_i = 0] \leq Pr[y, u_0^{i-1} | \hat{u_i} = 1] \\ 0, & \text{if } i \in A^C \end{cases} \tag{4.1}$$

The probability calculations are computationally easier and lesser prone ot round of errors in log domain. Hence we consider, LLRs instead of probabilities to avoid numerical overflows. LLR for t^{th} bit is defined as:

$$L^{(i)}(y, \hat{u_0}^{i-1}) = log \frac{Pr[y, \hat{u_0}^{i-1} | u_i = 0]}{Pr[y, u_0^{i-1} | \hat{u_i} = 1]}$$

Hence the decision rule changes as:

$$\hat{u_i} = \begin{cases} 0, & \text{if } i \in A \text{ and } L^{(i)}(y, \hat{u_0}^{i-1}) \geq 0 \\ 1, & \text{if } i \in A \text{ and } L^{(i)}(y, \hat{u_0}^{i-1}) \leq 0 \\ 0, & \text{if } i \in A^C \end{cases} \tag{4.2}$$

The binary tree structure of a polar code described earlier can be exploited to simplify the successive cancellation decoding. The binary tree has $n = \log_2 N + 1$ stages

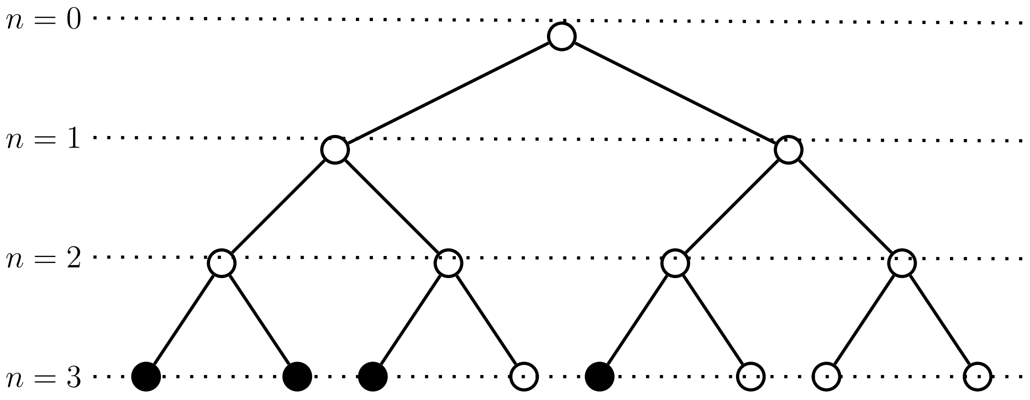


Figure 4.1: Binary tree structure of (8,4) polar code with stages indicated

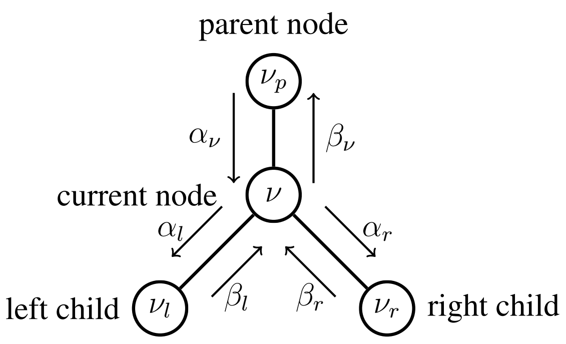


Figure 4.2: SC decoding update rules for each node

with numbering from $s = 0, \ldots, n$. Each stage s contains 2^s nodes with each node corresponding to 2^{n-s} bits.

Inorder traversal of the tree is done to perform the successive cancellation decoding. At each node messages are passed as shown:

Each node passes LLR corresponding LLR values, namely α, to the child nodes and sends the estimated hard bits at the sage, namely β, o the parent node. The left and right messages, α_i^l and α_i^r are calculated as:

$$\alpha_i^l = \ln\left(\frac{1 + e^{\alpha_i + \alpha_{i+2^{n-s-1}}}}{e^{\alpha_i} + e^{\alpha_{i+2^{n-s-1}}}}\right) \tag{4.3}$$

$$\alpha_i^r = \alpha_{i+2^{n-s-1}} + (1 - 2\beta_i^l)\alpha_i \tag{4.4}$$

We define two functions to perform these operations, namely f and g, defined as:

$$\begin{aligned} f(\alpha_1, \alpha_2) &= \ln\left(\frac{1 + e^{\alpha_1+\alpha_2}}{e^{\alpha_1} + e^{\alpha_2}}\right) \\ g(\beta, \alpha_1, \alpha_2) &= (1 - 2\beta)\alpha_1 + \alpha_2 \end{aligned} \tag{4.5}$$

But the f function is computationally expensive and hence we approximate it to a hardware friendly version using min-sum approximation as follows:

$$f_{minsum}(\alpha_1, \alpha_2) = \textit{sign}(\alpha_1)\textit{sign}(\alpha_2)\textit{min}(|\alpha_1|, |\alpha_2|) \tag{4.6}$$

where *sign* gives the sign of input and *min* gives the minimum of the two inputs.

The algorithm starts from the root node of the tree, which is level $n+1$, and traverses till the leaf node which is level 0. For each node, the following set of operations occur.

1. If current node has a left child that was not visited, is calculate α_l and move to left child.
2. If current node has a right child that was not visited, is calculate α_r and move to right child.
3. If both the messages from child nodes are available, calculate β and move to parent node.

Once the leaf node is reached, decisions are made based on the sign of corresponding LLR using the binary quantiser function h as:

$$\beta_\nu = h(\alpha_\nu) \tag{4.7}$$

where h is defined as:

$$h(\alpha) = \begin{cases} 0, & \text{if } \alpha \geq 0 \\ 1, & \text{else} \end{cases}$$

Simulation results for SC Decoder

Fig 4.3 shows BER and BLER performance of SC Decoder for (1024,512) polar code. The simulations were performed on BPSK modulated data and transmitted over AWGN

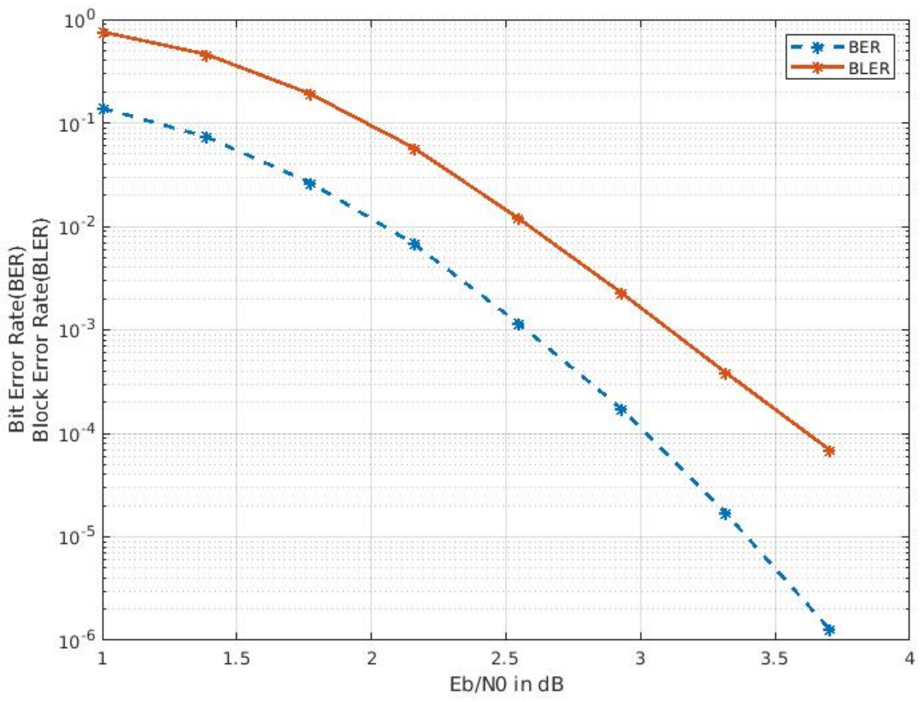


Figure 4.3: BER and BLER of SC Decoder for (1024,512) polar code

channel.

4.2 Simplified Successive Cancellation Decoder

Even though the successive cancellation decoder algorithm is computationally simple, the speed of decoder is limited by the serial nature of the algorithm. To decode u_i, we need to wait till all the bits $\{u_0, u_1, \dots, u_{i-1}\}$ are decoded. Also, in the current SC decoding, only one bit can be estimated at a time. Hence efforts were made to improve the speed of the algorithm to enable estimating more than one bit at a time. One such major breakthrough was simplified successive cancellation decoder, introduced in [7]. In this section, we look at SSC decoding and the different types of special nodes that make the SSC decoder possible. The types of special nodes and the rules for decoding the special nodes are provided below, the proof for which is provided in [7].

We will consider four types of special nodes that enable faster decoding by preventing the need to travel till leaf node to decode the corresponding bits.

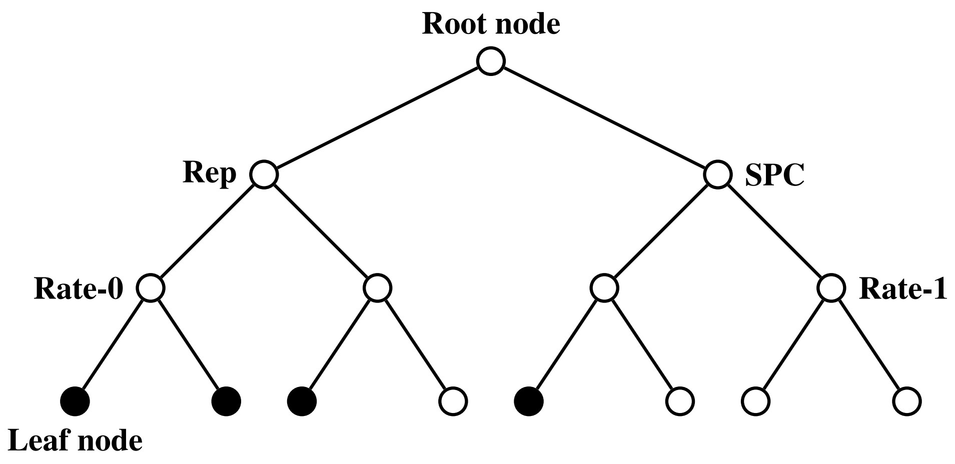


Figure 4.4: Binary tree structure of (8,4) polar code with special nodes labelled

4.2.1 Rate-0 Nodes

Rate-0 node corresponds to root node of a subtree whose leaf nodes are all frozen bit nodes. It is straight forward that for such types of nodes we don't need to travel till leaf node and the corresponding bits can be directly set to 0.

$$\beta_\nu = 0 \tag{4.8}$$

4.2.2 Rate-1 Nodes

Rate-1 node corresponds to root node of a subtree whose leaf nodes are all information bit nodes. We can see that traversing till leaf node and estimating the information bits is same as estimating the β at current node and taking a polar transform of the β. This is because there is no additional information generated by traversing further because there are no frozen bit nodes providing prior information.

$$\beta_\nu = h(\alpha_\nu) \tag{4.9}$$

4.2.3 SPC nodes

SPC node corresponds to root node of a subtree whose leaf nodes are all information bit nodes except for the first node which is frozen bit node. Such nodes can be decoded by estimating the β at current node and flipping the least reliable bit,corresponding to the LLR with the least absolute value. And then a polar transform is performed on β to obtain the estimate at leaf nodes.

$$\begin{aligned}\beta_\nu &= h(\alpha_\nu) \\ \beta_\nu(i_{par}) &= \sum_{i=0;i\neq i_{par}}^{N_\nu} \beta_\nu(i)\end{aligned} \tag{4.10}$$

where i_{par} is the position of least reliable bit and the summation is performed over $GF(2)$ *i.e,* sum *modulo* 2.

4.2.4 Rep Nodes

SPC node corresponds to root node of a subtree whose leaf nodes are all frozen bit nodes except for the first node which is information bit node. Such nodes can be decoded by taking sum of the LLRs at current node and estimating the bit corresponding to repetition node. And then a polar transform is performed on β to obtain the estimate at leaf nodes.

$$\beta_\nu = h\bigg(\sum_{i=0}^{N_\nu} \alpha_\nu(i)\bigg) \tag{4.11}$$

Simulation results for SSC Decoder

Fig 4.5 shows BER and BLER performance of SSC Decoder for (1024,512) polar code. The simulations were performed on BPSK modulated data and transmitted over AWGN channel. We can compare this with Fig 4.3 to see that there is no degradation in BER/BLER performance when compared to SC Decoder.

4.3 Successive Cancellation List Decoder

The SSC decoder algorithm deals with one issue of SC decoder algorithm, which is the latency. But for polar codes to be able to replace the LDPC codes, we need to improve the error-correction performance as well even for moderate lengths. That's where the SCL decoder algorithm comes into picture. Traditionally, list decoding has long been used to improve the error correction performance of block codes [8]. List decoding was

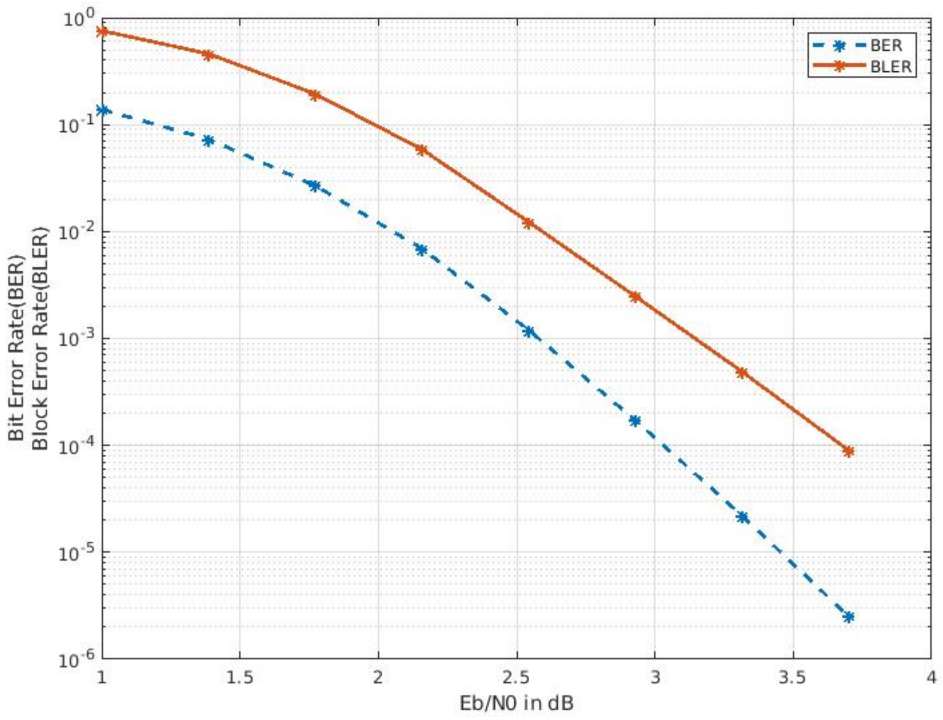


Figure 4.5: BER and BLER of SSC Decoder for (1024,512) polar code

first appllied to polar codes in [9] using likelihoods. Later, this was implemented using LLRs, in [10], reducing the computational complexity but still preserving the error correction performance. The key idea is to make a probabilistic estimate for each possibility rather than making a hard decision *i.e,* for each bit to be estimated we consider both 0 and 1 each with a certain metric specifying the probability. Also at any point we limit the maximum number of possibilities to the list size L, pruning the list each time to consider the best L possibilities so that at the end we end with a list of L candidates instead of one codeword.

For decoding polar codes, the SCL algorithm considers both the possibilities 0 and 1 for the bit to be estimated instead of making a hard decision. The reliability for each path is indicated by using a path metric variable that is updated for each estimate according to the following rule:

$$PM^i = \begin{cases} PM^{i-1}, & \text{if } \hat{u}_i = h(\alpha_v) \\ PM^{i-1} + |\alpha_v|, & \text{otherwise} \end{cases} \tag{4.12}$$

where h is the binary quantizer function. This is essentially penalizing the path by the *belief* every time a decision opposing to that of the binary quantizer rule is made.

Note that only the path metrics corresponding to L existing paths are updated while a frozen node is encountered as no new paths are created, since the estimate for frozen bit node is always 0. While estimating an information bit there are a total of $2L$ possibilities as the list doubles and we prune the list by selecting the best L reliable paths *i.e,* the paths corresponding least L path metrics.

Thus at the end we select the codeword with the least PM from the pool of L candidates as the estimate. While this provided an improvement in error-correction capability of SC decoder, this is still not sufficient to compete with the performance of LDPC codes.

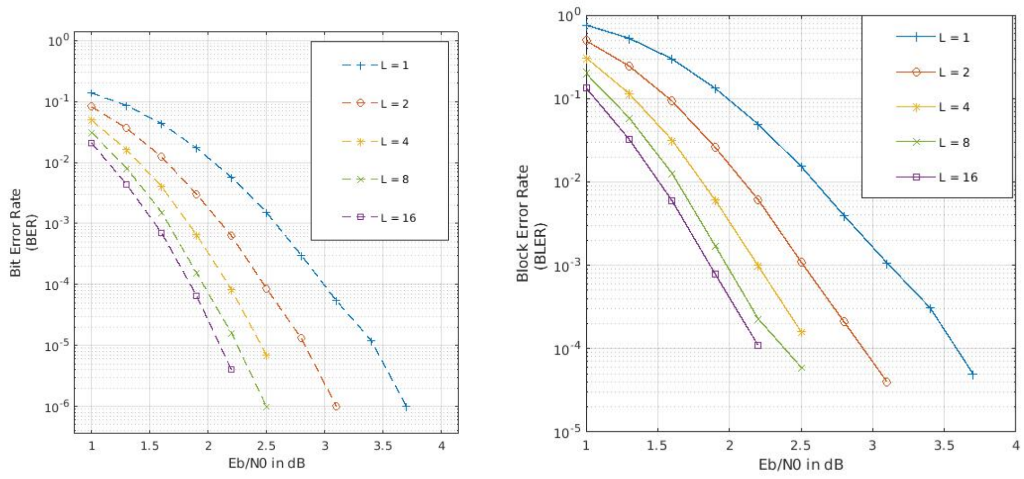
One interesting thing to note is, while the list of L possible candidates contains the correct codeword with high probability it need not always be the candidate with the least PM. Hence we need genie sort of help to identify the correct codeword from the list. A close approximation to this can be achieved by using a CRC aided decoder. We take a closer look at this decoder in the following section.

4.4 CRC aided Successive Cancellation List Decoder

A Cyclic Redundancy Check (CRC) can be thought of as an advanced parity check, that can detected more complicated results than simply checks for odd or even number of bits in the received vector. In CRC, a sequence of redundant bits called check redundancy bits are appended to the data so that the resulting unit becomes exactly divisible by a predetermined binary vector.

In CRC-SCL decoding, we use some of the information bit positions to send the CRC-bits of the message vector rather than the message bits. And at the receiver, after we get the list of L possible candidates, we check for the the candidate with least PM that satisfies the CRC condition. If there is no candidate that satisfies the CRC condition then we directly select the candidate with the least PM [9]. We use the notation (N, K, p) to denote (N, K) polar code with p CRC bits.

Thus the error-correction capability of a polar code can be matched up to that of LDPC code of comparable length by using the needed list size and CRC length. In fact, for moderate lengths CRC-SCL decoders are observed to perform better that that



(a) Bit Error Rate (b) Block Error Rate

Figure 4.6: CRC-SCL Decoding for (1024,512,8) polar code for different list sizes

of LDPC codes of same rate and comparable length.

Fig. LDPC vs polar fig.2.14

But all of this comes at a cost of increased computational complexity and thus increases latency. The latency is observed to be increasing linearly with the list size. Using simplifications for the special nodes provides a considerable improvement. The CRC-SCL algorithm has to be modified in order to take care for updating the PMs correctly, describes in the next section.

Simulation results of CRC-SCL Decoder

Fig 4.6 shows BER and BLER performance of SCL Decoder for (1024,512,8) polar code, where 8 is the number of CRC bits. The simulations were performed on BPSK modulated data, the noise considered is AWGN. We can see the improvement in error correction performance with increase in list size.

4.4.1 CRC aided Simplified Successive Cancellation List Decoder

In CRC-SSCL, we describe the rules to update PMs at the special nodes, described in [11] and [12], that enable us to stop traversing further and decode at the current node. We describe both the exact and HWF versions.

Let the general description of a special node be α_ν for the incoming LLRs, and β_ν for the decisions to be sent to parent node and N_s be the length of bits corresponding to current node.

4.4.1.1 Decoding for Rate-0 Nodes

For Rate-0 nodes, since we always decode the bits as 0 we only need to take care of updating the PMs. For each bit, the PM needs to be updated based on the sign of corresponding LLR for each list. The update rules for PM_l, where l is the list number l, are as follows:

$$PM_l = \begin{cases} PM_l + \ln(1 + e^{-\alpha_{i_l}}), & \text{exact} \\ PM_l + sign(\alpha_{i_l})\alpha_{i_l} - \alpha_{i_l}, & \text{HWF} \end{cases} \tag{4.13}$$

And an all zero vector, which is the β_l, is sent back to to the parent node.

4.4.1.2 Decoding for Rate-1 Nodes

For Rate-1 nodes, we have to estimate the bits one by one updating the PM at each estimate. For each bit the list size doubles to $2L$ and the paths corresponding to least L PMs are retained while the remaining paths are discarded. The update rules for each bit corresponding to path l is as follows:

$$PM_l = \begin{cases} PM_l + \ln(1 + e^{-(1-2\beta_{i_l})\alpha_{i_l}}), & \text{exact} \\ PM_l + sign(\alpha_{i_l})\alpha_{i_l} - (1-2\beta_{i_l})\alpha_{i_l}, & \text{HWF} \end{cases} \tag{4.14}$$

For each path, this is performed N_s times, once for each bit estimated. At the end the β_l corresponding to L paths is sent back to the parent node.

4.4.1.3 Decoding for SPC Nodes

For SPC nodes, the parity check condition should satisfy for the codeword always. Assuming that the bit with least absolute value of LLR as the parity bit, for each path, we first estimate the remaining bits. And in the end, the parity bit is set such that

that parity check constraint is satisfied. The update rules for each bit, except the least reliable bit, corresponding to path l is as follows:

$$PM_l = \begin{cases} PM_l + \ln(1 + e^{-(1-2\beta_{i_l})\alpha_{i_l}}), & \text{exact} \\ PM_l + sign(\alpha_{i_l})\alpha_{i_l} - (1 - 2\beta_{i_l})\alpha_{i_l}, & \text{HWF} \end{cases} \tag{4.15}$$

Note that after the parity correction is made, PM is updated again as:

$$PM_l = \begin{cases} PM_l, & \text{if } \hat{u_i} = h(\alpha_v) \\ PM_l + |\alpha_v|, & \text{otherwise} \end{cases} \tag{4.16}$$

where $\hat{u_i}$ is the parity bit and h is the binary quantizer function.

4.4.1.4 Decoding for Rep Nodes

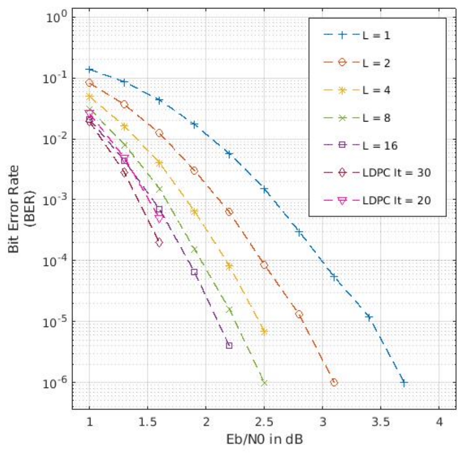
For Rep nodes, we need to estimate only one information bit. Since all the incoming LLRs represent the information about same bit, we consider the sum of LLRs in each path while making a decision. The list size doubles, with one half assuming the Rep node to be all 0s and the other half assuming the Rep node to be all 1s. The update rules for each bit corresponding to path l is as follows:

$$PM_l = \begin{cases} PM_l + \sum_{i=0}^{N_s-1} \ln(1 + e^{-(1-2\beta_{N_s})\alpha_{i_l}}), & \text{exact} \\ PM_l + \frac{1}{2}\sum_{i=0}^{N_s-1} sign(\alpha_{i_l})\alpha_{i_l} - (1 - 2\beta_{N_s})\alpha_{i_l}, & \text{HWF} \end{cases} \tag{4.17}$$

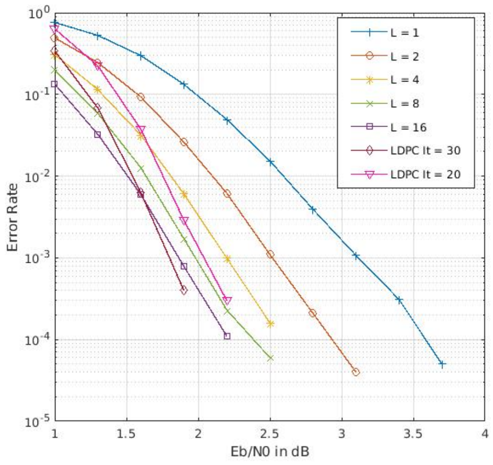
where β_{N_s} corresponds to estimate for the information bit of Rep node. At the end the β_l corresponding to L paths is sent back to the parent node, while discarding the remaining L paths.

Comparison of CRC-SCL with LDPC codes

FRom Fig 4.7, we can see that even though the length of polar code used, $N = 1024$, is less than that of LDPC code, the error correction performance is cloe to that of LDPC code as the list size increases. The parameter L, which is the list size, can be compared



(a) Bit Error Rate



(b) Block Error Rate

Figure 4.7: BER/BLER comparison of (1944,72) for different iterations LDPC with (1024,512,8) polar code for different list sizes

to the maximum number of iterations for an LDPC decoder.

4.5 Fast CRC-SSCL Decoder

Even though SSCL improves the decoding speed by decoding the special nodes without traversing further, we still have to deal with serial nature of decoding algorithm described above when decoding Rate-1 and SPC nodes. Also it is impossible consider all the possible path splitting operations for a special node, since it will result in exponential complexity. Hence to tackle both of these problems, Fast-SSCL algorithm was proposed in [12], where the number of path splitting operations at special node are limited based on the list size with little to no degradation in error correction performance. The rules for updating are provided below, and the proofs for these can be found in [12]

4.5.1 Fast Decoding for for Rate-1 nodes

The following theorem summarises the rules for guaranteed error correction performance for Rate-1 nodes while limiting the path splitting operations.

Theorem 4.5.1 *In Fast-SSCL decoding, the number of path splitting operations re-*

quired for decoding Rate-1 node while preserving the error correction performance is

$$min(L-1, N_\nu)$$

where L is the list size and N_ν is the size of Rate-1 node.

The rule for updating path metric remains the same. The HWF formulation is as follows:

$$PM_{i_l} = \begin{cases} PM_{i_l} + |\alpha_{i_l}|, & \text{if } (1-2\beta_{i_l}) \neq sgn(\alpha_{i_l}) \\ PM_{i_l}, & \text{else} \end{cases} \tag{4.18}$$

It is important to note that the least reliable bits are estimate first. This is because, the bits with high LLR values are less likely to get improvement due to path splitting. Hence, in case of $L-1 \leq N_s$, after estimating the least $L-1$ reliable bits, the remaining bits can simply be estimated through hard decision as follows:

$$\beta_{i_l} = \begin{cases} 0, & \text{if } \alpha_{i_l} \geq 0 \\ 1, & \text{else} \end{cases} \tag{4.19}$$

4.5.2 Fast Decoding for SPC nodes

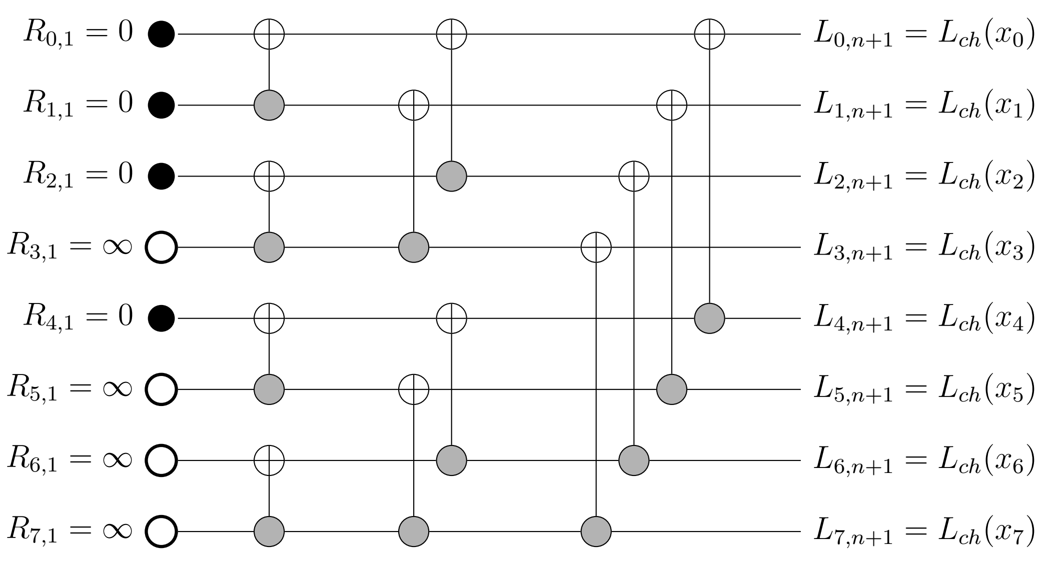
A SPC node can be decoded by ignoring the least reliable bit and decoding the remaining bits as a Rate-1 node. The least reliable bit is then set to satisfy the parity constraint. The following theorem gives the minimum number of path splitting operations.

Theorem 4.5.2 *In Fast-SSCL decoding, the number of path splitting operations required for decoding SPC node while preserving the error correction performance is*

$$min(L, N_\nu)$$

where L is the list size and N_ν is the size of SPC node.

The rules for updating PM and getting hard estimate for remaining bits remains the same as in the case of Rate-1 node. Now, we set the parity bit, *i.e,* the least reliable bit, to satisfy the parity constraint and finally update the PM accordingly.



$R_{0,1} = 0$
$R_{1,1} = 0$
$R_{2,1} = 0$
$R_{3,1} = \infty$
$R_{4,1} = 0$
$R_{5,1} = \infty$
$R_{6,1} = \infty$
$R_{7,1} = \infty$
$L_{0,n+1} = L_{ch}(x_0)$
$L_{1,n+1} = L_{ch}(x_1)$
$L_{2,n+1} = L_{ch}(x_2)$
$L_{3,n+1} = L_{ch}(x_3)$
$L_{4,n+1} = L_{ch}(x_4)$
$L_{5,n+1} = L_{ch}(x_5)$
$L_{6,n+1} = L_{ch}(x_6)$
$L_{7,n+1} = L_{ch}(x_7)$

Figure 4.8: Encoding factor graph of Arikan's BP decoder for a (8,4) polar code with L and R messages initialised

4.6 Belief Propagation Decoder

Belied propagation or message passing algorithm has traditionally been used for performing inference on graphical models, such as Bayesian networks. In particular, BP decoding in LDPC codes has been hugely successful. Over the years highly optimized software and hardware decoders for BP decoding have been developed. Hence it will be highly efficient to come up with a method to use BP decoding to decode polar codes since this will enable us to avoid developing complex new hardware designs specifically for polar codes. And they have since been used in decoding polar codes as well [13]

The main advantage with a BP decoder is the increased parallelism due to the flooding schedule instead of the serial scheduling observed in SC decoding. There have been many approaches at using belief propagation for polar codes. Arikan's original BP decoder was based on BP propagation for Reed-Muller (RM) codes, where the iterative calculations were made on the encoding factor graph. This was different from the usual approach where iterations were performed on the tanner graph, which is the parity check factor graph. First we look at the Arikan's BP decoding, which is different from BP decoder for LDPC codes. Then we look at BP decoding using the parity check matrix that will enable us to directly use the BP decoder for LDPC codes. And we then present a novel parity bit concatenation based BP decoder that gives an improvement over the current BP decoder, using the same decoding methodology.

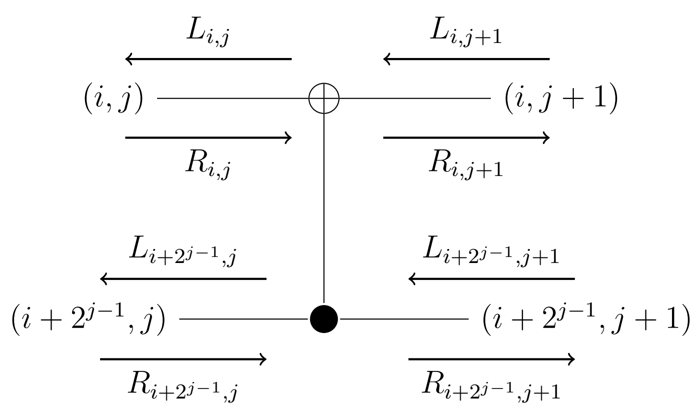


Figure 4.9: Update rules at each node for Left and Right messages

4.6.1 Arikan's BP Decoder

Arikan's BP, provided in [14], is based on describing the encoding factor graph using two types of nodes and designing the update rules accordingly. Starting with the received channel LLRs, decoding is performed by iteratively exchanging left and right messages along the edges of the graph.

The most basic processing unit of the encoding factor graph and the update rules corresponding left and right messages are shown in Fig 4.9. The L-messages Li, j correspond to beliefs propagating from right-to-left and R-messages $R_{i,j}$ correspond to beliefs propagating from left-to-right, at each node, along the encoding factor graph shown in Fig. 4.8 respectively.

At each node in the encoding structure of polar code , the update rules for left and right messages are implemented, shown in Fig 4.9, as:

$$\begin{aligned} L_{i,j} &= f(L_{i,j+1}, (L_{i+2^{j-1},j+1} + R_{i+2^{j-1},j})) \\ L_{i+2^{j-1},j} &= f(R_{i,j}, L_{i,j+1}) + L_{i+2^{j-1},j+1} \\ R_{i,j+1} &= f(R_{i,j}, (L_{i+2^{j-1},j+1} + R_{i+2^{j-1},j})) \\ R_{i+2^{j-1},j+1} &= f(R_{i,j}, L_{i,j+1}) + R_{i+2^{j-1},j} \end{aligned} \tag{4.20}$$

where the function f can be chosen to be exact version or HWF approximation

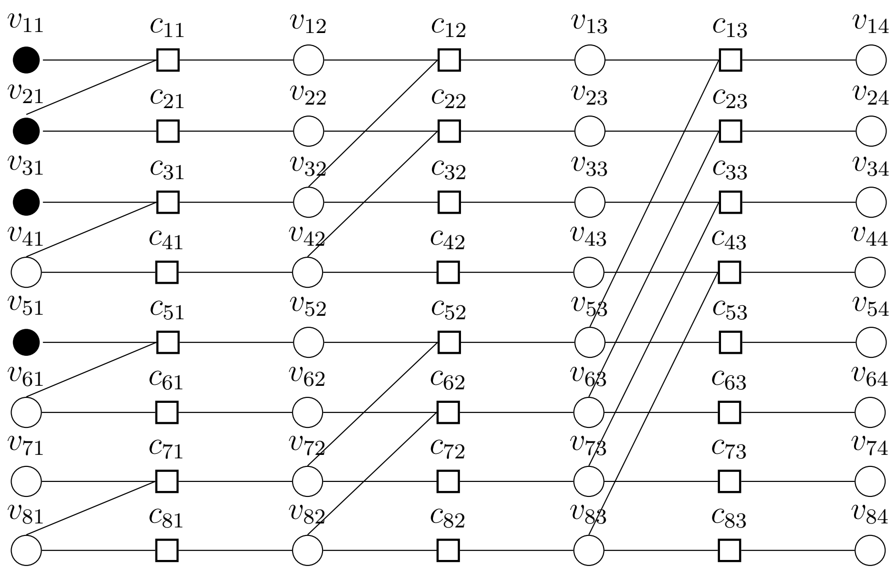


Figure 4.10: Dense factor graph representation for a (8,4) polar code with variable nodes v and check nodes c

accordingly as follows:

$$f(a,b) = \begin{cases} 2artanh(tanh(\frac{a}{2})tanh(\frac{b}{2}))), & \text{exact} \\ sign(a)sign(b)min(|a|,|b|), & \text{HWF} \end{cases} \tag{4.21}$$

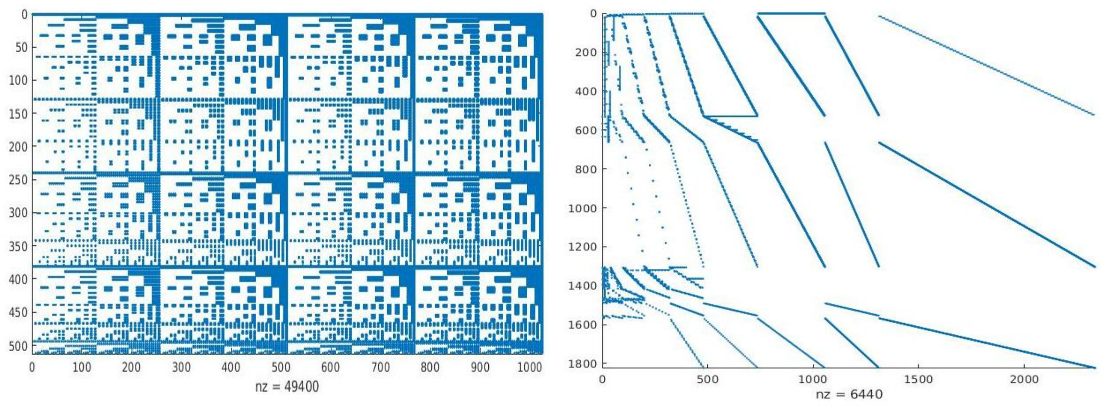
From fig. 4.8, the right most LLRs corresponds to received channel LLRs. Hence we initialise them as:

$$L_{i,n+1} = L_c h(x_i) = \ln \frac{p(x_i = 0|y_i)}{p(x_i = 1|y_i)},\ i \in \{1, 2, \dots, N\} \tag{4.22}$$

And the left most LLRs correspond to information and frozen bit nodes. Hence we fix them to

$$R_{i,1} = \begin{cases} 0, & \text{if } i \in A \\ \infty, & \text{if } i \in A^C \end{cases} \tag{4.23}$$

For decoding, the L-messages $L_{i,j}$ for $j \in \{1, 2, \dots, n\}$and the R-messages $R_{i,j}$ for $i \in \{2, 3, \dots, n\}$ are iteratively updated till the maximum number of iterations are reached or an early stopping criteria, if any, is met. Note that the R-messages $R_{1,j}$ are always fixed. And the end of each iteration, the soft messages corresponding to $\hat{u}$ and



(a) Dense matrix (b) Sparse matrix

Figure 4.11: Visualisation of parity check matrix for (1024,512) polar code

$\hat{x}$ can be obtained respectively as:

$$
\begin{aligned}
L(u_i) &= L_{i,1} \\
L(x_i) &= L_{i,n+1} + R_{i,n+1}
\end{aligned} \tag{4.24}
$$

And after each iteration, hard decisions can be made using the standard binary quantizer function h, defined as:

$$
h(\alpha) = \begin{cases} 0, & \text{if } \alpha \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{4.25}
$$

Using the hard estimates, an early stopping criteria can be designed as to stop further iterations if $\hat{u}G_N = \hat{x}$ where G_N is the generator matrix.

4.6.2 Sparse Graph based BP Decoder

In this section we look at the possibility of using conventional LDPC BP decoder for decoding of polar codes. The LDPC BP decoder requires a parity check matrix or an equivalent tanner graph to start the decoding process. Hence we first need to construct a parity check matrix for the polar code based on the generator matrix G_N. This can be performed by using the columns of G_N with indices in A^C, the frozen bit set. Fig 4.10 shows the factor graph representation of dense parity check matrix and it can be

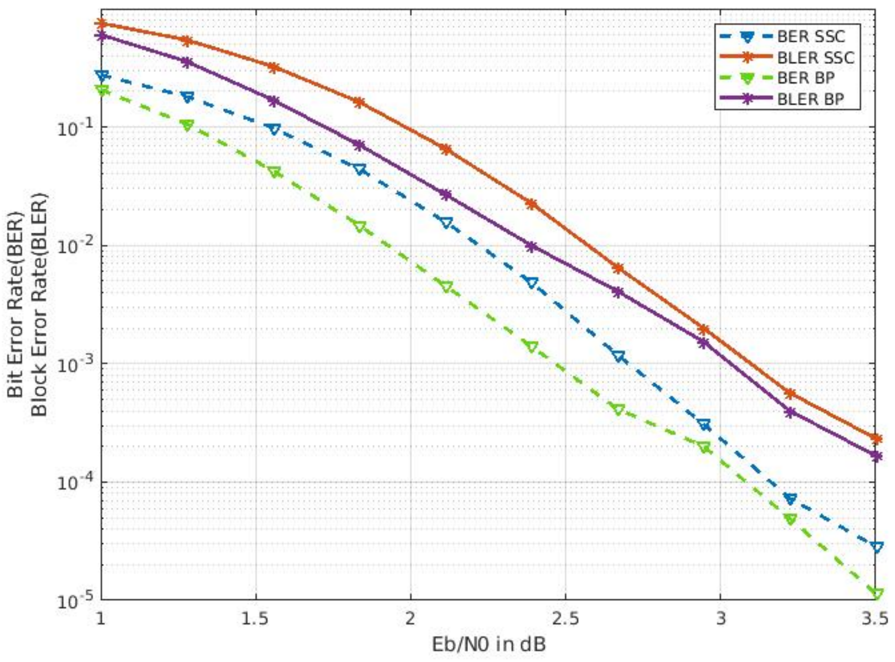


Figure 4.12: Comparison of BP decoder with SC decoder for (1024,512) polar code

seen that this results in a very dense matrix with maximum check node degree as N. It is important to note that the effectiveness of BP decoder depends on the convergence achievable for the beliefs propagated. This is possible only when short cycles are not present in the tanner graph.

Hence we need to prune the parity check matrix by removing the unnecessary variable and check nodes. The pruning procedure is discussed in detail in [15], which removes or replaces redundant nodes in the dense factor graph. This results in a significant reduction in size of parity check matrix, about 80% to 85%, while maintaining sparsity of the matrix.

After the pruning is performed we are left with a parity check matrix of reduced dimensions and density. Fig 4.11 shows the transition of dense parity check matrix to a sparse parity check matrix that results in elimination of short cycles.

Comparison of BP performance of Dense vs Sparse

Fig 4.13, shows a comparison of error correction performance of BP decoding on a dense graph vs sparse graph for polar code. The non-convergence of BP decoder is clearly visible which is due to the large number of short cycles present. The simulations

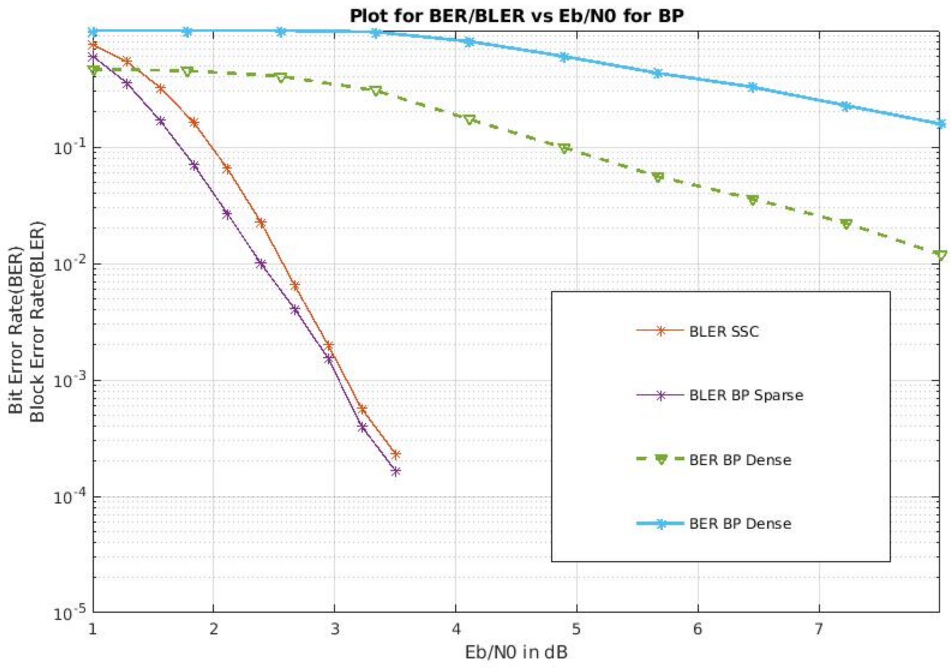


Figure 4.13: Comparison of BP performance of Dense vs Sparse

are performed over BPSK modulated data and over AWGN channel.

Comparison of BP with SCD

Fig 4.12, shows a comparison of error correction performance of BP decoding on a sparse graph vs Successive Cancellation Decoding for polar code. It is seen that with sufficiently sparse graph and sufficient number of iterations, BP decoder can outperform the SC decoder. Here 200 iterations were performed. The simulations are performed over BPSK modulated data and over AWGN channel.

4.6.3 Concatenated BP Decoder

In this section we present a modified BP decoder with parity bits appended to the encoded vector, to improve the BER performance. First we look at the construction of the code and then we look at the decoding process.

The principle is to introduce a overhead with additional parity bits so that BER performance is improved. Let the number of parity bits be p. Hence the final length of codeword vector will be of length $N + p$. After encoding of the message vector is done,

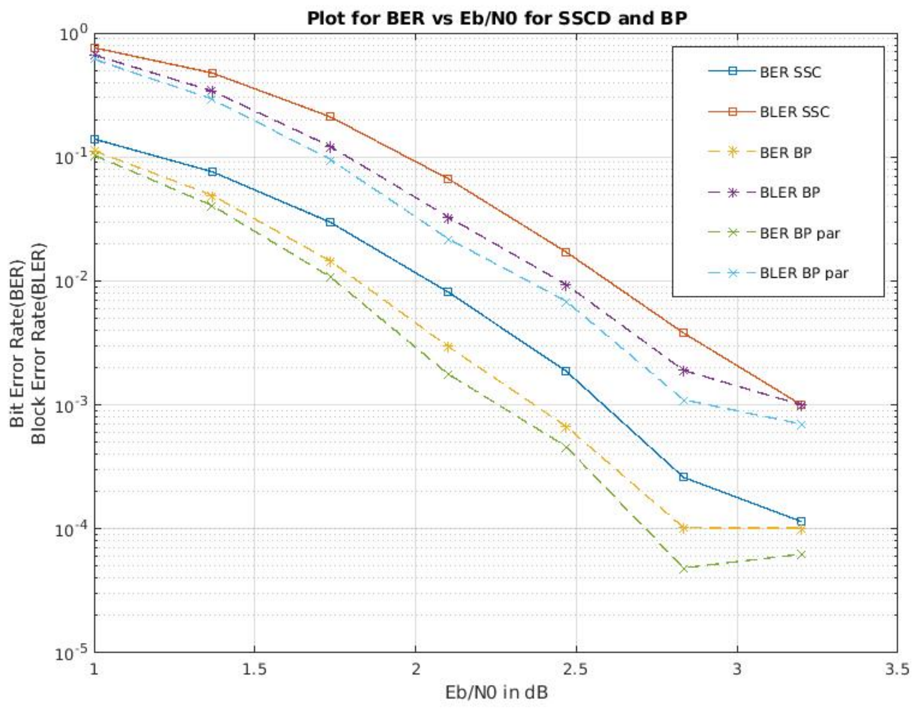


Figure 4.14: Performance of SC vs BP vs BP decoder with parity bits appended

the parity bits are found based on the parity check matrix H_p which is of dimensions $(p, N+p)$.

For decoding the received vector, two LDPC min-sum decoders work alternatively providing an updated estimate of the transmitted codeword. The first decoder corresponds to the polar code and the second decoder corresponds to the parity bits. Let the channel LLR values corresponding to received vector be $L0$. Every iteration consists of two parts in which the first half corresponds to Decoder1 and the second half corresponds to Decoder2. The extrinsic information obtained from the decoder is combined with the channel LLR values and passed to the other decoder and continued till the maximum number of iterations are reached or an early stopping criteria is met. Fig 4.15 shows the iterative decoder for concatenated BP decoding. Starting with received channel LLRs at Decoder1, the decoder operate alternatively combining the extrinsic information with received LLRs till the stopping criteria is met or maximum number of iterations are reached. Fig 4.14 shows that using the concatenated BP decoder, an improvement of about 0.1dB is achieved.

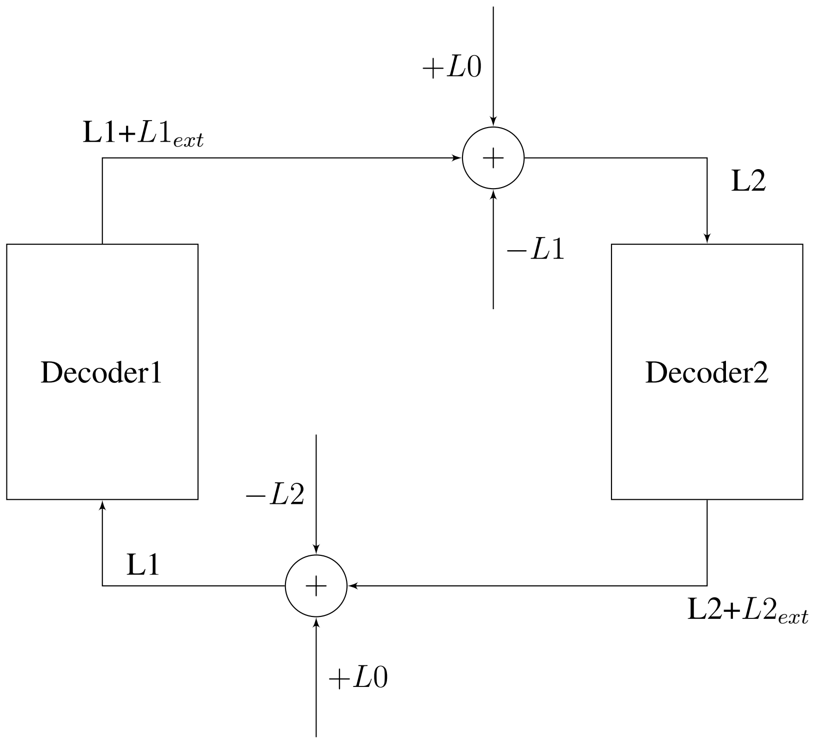
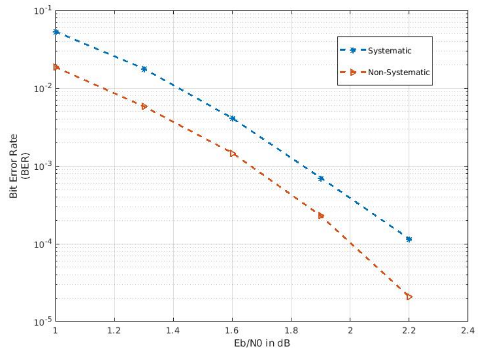


Figure 4.15: Concatenated BP decoder for polar code

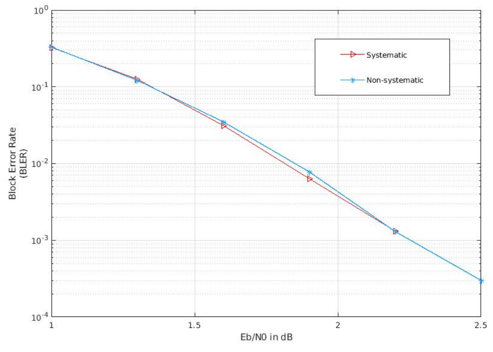
4.7 Systematic Polar Codes

As mentioned earlier, using systematic encoding provides a better BER performance. Even though the systematic encoding of polar codes is not as straightforward as non-systematic encoding, which has a nice butterfly structure, it still has time complexity $O(N \log_2 N)$, as shown in [16].

Intuitively systematic codes are expected to be be more robust to errors because the information bits are preserved as they are and directly extracted from the received vector. And the results consistent with this. The decoding process is very similar to that of non-systematic polar codes. One interesting observation made while decoding polar codes is, estimating the received vector $\hat{x}$ assuming non-systematic encoding and then encoding the estimate to get the transmitted vector $\hat{u}$ doesn't introduce any degradation in error correction performance, which is expected due to amplification of errors happening when the operation $\hat{u} = \hat{x}G_N$ is performed.



(a) BER comparison



(b) BLER comparison

Figure 4.16: BER/BLER performance comparison of (1024,512,8) polar code with list size 4

Simulation results of Systematic Polar Codes

Maintaining all other parameters the same, Fig 4.16a shows that there is an improvement of about 0.18dB in BER performance using systematic polar codes.

CHAPTER 5

IMPLEMENTATION

In this chapter we discuss the approach used in implementing the encoder and decoder algorithms and the throughput results achieved, both x86 and ARM architectures.

Fixed-point Implementation

In the results we present, we consider 8-bit fixed point representation for all the floating point numbers. The received LLRs are quantized suitably and the g function is carried out with saturation addition. It was observed tha the effect of quantization on the BER performance is negligible.

The advantage of using 8-bit quantization will be made more evident when we look at ARM based implementation, where NEON intrinsics can be used to manipulate data in parallel.

The encoder implementation is straight forward with implementation done on the butterfly structure. As for the decoder, we use the binary tree structure to perform in-order tree traversal to decode the received vector.

Memory Layout for α and β

We first look at the SC decoder and extend the same to SCL decoder. Decoder is implemented on the binary tree structure of the polar code. Starting at the root node, at each node we perform one of the three operations:

- Use current LLR to calculate LLR for left child and and propagate left.
- Use current LLR and beta from left child to calculate LLR for right child and and propagate right.
- Combine beta from left child and right child to calculate beta for parent node and propagate up.

From these three steps it is clear that major part of the algorithm involves data fetching and data update, it is important to minimise these operations and also store the data in a cache friendly manner. At each node, we store α_ν and β_ν and corresponding to that node with 2^{n-l} elements each at stage l, starting at root node $l = 0$, where n is the depth of the tree.

For storing the LLR values α, since the decoder is sequential, we operate on one node at a time. Hence at any given point of time, we need to store 2^{n-l} elements corresponding to level l. Also, while operating at level l, we still need to store the LLR values corresponding to previous $l-1$ levels since we need to propagate back to parent node. Hence the total number of α values that need to be stored is $\sum_{l=0}^{n} 2^{n-l} = 2^{n+1}-1$.

For storing the decisions β, we need double this memory because we need to store decisions from both the children nodes till parent node is reached. Hence the total number of β need to be stored is $\sum_{l=0}^{n} 2 * 2^{n-l} = 2^{n+2} - 2$.

For the list decoder of size L, the memory requirement becomes L times since at any point of time there will be L decoders active. Also, for the specific implementation discussed here, there will be additional overhead to keep track of all the surviving paths.

For the list decoder, every time a path splitting operation occurs, the list size doubles and then pruned to retain only the best L paths. This operation results in a new set of α and β for all the L decoding trees. Naively updating the whole tree results in a very large overhead. Hence at each node we perform the following operations:

1. Find the LLRs corresponding left child from L paths using f function and traverse to left child.
2. Perform path splitting and find the indices of surviving paths and corresponding β and traverse to parent node.
3. Calculate the LLRs corresponding right child using the indices, β and the original LLRs and propagate to right child.
4. Perform path splitting and find the indices of surviving paths and corresponding β and traverse to parent node.
5. Combine β from both children and find the indices of final surviving paths and traverse to parent node.

This results in local updating of data and using indices to access the old data rather than creating a new copy, thus minimising the data manipulations.

Table 5.1: Decoder Throughput in Mbps for Successive Cancellation decoder on x86

Decoder	Mbps
SCD	31
SSCD	86

Table 5.2: Decoder Throughput in Mbps for different list size for list decoder on x86

Decoder	L=1	L= 2	L= 4	L = 8	L = 16
SCL	16	9.4	5.1	2.3	1.1
SSCL	27	15.3	8.1	4.6	2.4

5.1 Throughput Results for x86 processors

In this section we present the throughput results for encoder and decoder on x86 architecture supported by Intel processors. All the algorithms were implemented in C language and were run on single core of Intel i7-8700 CPU with base clock frequency of 3.2 GHz. The code was compiled using `gcc` version `7.3.0` and compiler time optimisation flag `-O3` was turned on. The simulations were performed on BPSK modulated data and over AWGN channel.

The encoder throughput achieved was 310Mbps. Table 5.1 and 5.2 provide the decoder throughput results for the successive cancellation and list decoder respectively.

5.2 Throughput Results for ARM processors

In this section we present the throughput results for encoder and decoder on RISC architecture supported by ARM processors. All the algorithms were implemented in C language and were run on single core of ARM A53 processor of Raspberry Pi 3 running linux OS. The decoder ran on single core of ARM A53 processor of Raspberry Pi 3 running linux OS with base clock frequency of 1.2 GHz. The code was compiled using `gcc` version `6.3.0` and compiler time optimisation flag `-O3` and `-march=armv8-a`, `-mtune=cortex-a53`,`-mfpu=neon`, `-ftree-vectorize` were turned on. The simulations were performed on BPSK modulated data and over AWGN channel.

ARM architecture supports SIMD/NEON operations using which data can be loaded and handled in parallel, upto 16 8-bit integers. From Table 5.3 and 5.4 it can be seen

Table 5.3: Decoder Throughput in Mbps for Successive Cancellation decoder on ARM

Decoder	without NEON	with NEON
SCD	4.1	4.7
SSCD	9.9	13.9

Table 5.4: Decoder Throughput in Mbps for different list size for list decoder on ARM

Decoder	L=1	L= 2	L= 4	L = 8	L = 16
SCL	1.7	1.1	0.61	0.28	0.14
SCL with NEON	2.1	1.43	0.78	0.39	0.21
SSCL	3.55	2.1	1.1	0.53	0.24
SSCL with NEON	4.2	2.7	1.41	0.73	0.36

that an improvement of 25% to 30% can be achieved making use of the NEON intrinsics.

The encoder throughput for SC decoder achieved was 35 Mbps when implemented without using NEON intrinsics but improved to about 69 Mbps when NEON intrinsics were used providing an improvement of about 98%. This was possible due to the high level of pluralisation possible in encoding of polar codes. And the encoder throughput for list decoder falls down to 23 Mbps when NEON was not used and 39 Mbps when NEON was used. This was due to the additional CRC encoding introduced in the encoding process when list decoding is performed.

Table 5.3 and 5.4 provide the decoder throughput results for the successive cancellation and list decoder respectively, showing the improvement.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this report we primarily addressed two issues of decoding polar codes, namely latency due to sequential nature and poor BER performance due to error propagation. In this chapter we provide a summary of discussions made in the thesis and possibility for future directions of the work.

6.1 Summary

In Chapter 2, we provided the necessary background on the phenomenon of channel polarisation, the construction of polar codes and the capacity achieving property of polar codes.

In Chapter 3, we discussed two typed of encoding for polar codes, non-systematic and systematic. We discussed the butterfly structure for encoding polar codes that reduced the complexity of non-systematic encoding from $O(N^3)$ to $O(N \log_2 N)$. We also looked at the binary tree structure of polar codes, which is naturally suitable based on the recursive construction of polar codes.

In Chapter 4, we looked at various decoding techniques for polar codes. We started with the successive cancellation decoder that decodes one at a time in a successive manner. Then we looked at simplified successive cancellation decoder that identifies special types of nodes in polar codes that can help in decoding multiple bits without traversing further down the tree. This provided a significant improvement in the decoding speed but we still needed to address the issue of poor error correction performance. Hence we looked at the list decoding of polar codes, which though having a increase in computational complexity provided a significant improvement in error correction performance proving a trade-off choice between latency and error-correction performance. We further looked at incorporating the node simplifications from SSCD into the SCL decoder, providing a significant improvement in decoding speed. Then we looked at the further reducing the latency by limiting the maximum number of path splitting operations per special node, providing a trade-off choice between a minor to no degradation in error correction performance against the latency. This was called the Fast-SSCL. We finally looked at iterative decoding of polar codes using Belief Propagation decoder, from LDPC codes. We looked at forming a parity check matrix for polar codes based on the encoding factor graph and pruning the factor graph to remove short cycles to improve error correction capability and reduce the computational complexity. Further we introduced a novel decoding technique using concatenated BP decoder, that adds additional parity bits to the encoded vector, to obtain a improvement in error correction performance. We also looked at systematic polar codes, that have a better error correction performance compared to non-systematic polar codes.

Finally in chapter 5, we presented the approach followed in implementing the decoders and the throughput results.

6.2 Future Work

The Fast-SSCL algorithm provides a much larger scope for pluralisation than a SSCL algorithm, since all the remaining operations at a special node, after the path splitting operations, can be executed in parallel. Hence this can be taken to our advantage with the help of NEON intrinsics to provide a significant improvement in speed. Also, Fast-SSCL algorithm can be tuned to limit the maximum number of path splitting operations to achieve trade off between speed and error correction performance to perform an adaptive decoding based on SNR.

And the idea of using concatenated codes to improve the performance of polar codes promising. Since BP decoder is always preferred to the list decoder in terms of implementation complexity, this seems to be a promising idea to pursue.

REFERENCES

[1] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

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

[3] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, January 1962.

[4] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Transactions on Information Theory*, vol. 55, pp. 3051–3073, July 2009.

[5] I. Tal and A. Vardy, “How to construct polar codes,” *IEEE Transactions on Information Theory*, vol. 59, pp. 6562–6582, Oct 2013.

[6] E. Arikan, “Systematic polar coding,” *IEEE Communications Letters*, vol. 15, pp. 860–862, August 2011.

[7] A. Alamdar-Yazdi and F. R. Kschischang, “A simplified successive-cancellation decoder for polar codes,” *IEEE Communications Letters*, vol. 15, pp. 1378–1380, December 2011.

[8] P. Elias, “List decoding for noisy channels,” 12 2015.

[9] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Transactions on Information Theory*, vol. 61, pp. 2213–2226, May 2015.

[10] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, “Llr-based successive cancellation list decoding of polar codes,” *CoRR*, vol. abs/1401.3753, 2014.

[11] S. A. Hashemi, C. Condo, and W. J. Gross, “Simplified successive-cancellation list decoding of polar codes,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 815–819, July 2016.

[12] S. A. Hashemi, C. Condo, and W. J. Gross, “Fast and flexible successive-cancellation list decoders for polar codes,” *IEEE Transactions on Signal Processing*, vol. 65, pp. 5756–5769, Nov 2017.

[13] N. Hussami, S. B. Korada, and R. L. Urbanke, “Polar codes for channel and source coding,” *CoRR*, vol. abs/0901.2370, 2009.

[14] E. Arikan, “Polar codes : A pipelined implementation,” 2010.

[15] S. Cammerer, M. Ebada, A. Elkelesh, and S. ten Brink, “Sparse graphs for belief propagation decoding of polar codes,” *CoRR*, vol. abs/1712.08538, 2017.

[16] E. Arikan, “Systematic polar coding,” *IEEE Communications Letters*, vol. 15, pp. 860–862, August 2011.