EFFICIENT ALGORITHMS FOR POLAR CODES

By

Harish Vangala

A Thesis Submitted for the Degree of
Doctor of Philosophy
Department of Electrical and Computer Systems Engineering
Faculty of Engineering, Monash University, Australia

MAY 2017

Copyright Notice

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

Abstract

Polar codes, invented by Arıkan, are one of the state-of-the-art channel codes that provably achieve Shannon capacity on any binary-input discrete memoryless channel (BI-DMC). They are selected as a control channel coding scheme in the 5^{th} generation wireless communication systems (5G). They are suitable for practical applications mainly due to their low complexity and recursive structure of both encoder and decoder. They are shown to outperform the state-of-the art LDPC codes when decoded with advanced list successive cancellation decoder with cyclic redundancy check (LSCD with CRC), which is a well-known improvement of the original successive cancellation decoder (SCD) proposed by Arıkan.

In this thesis we focus on how to improve polar codes in a way it aids their adoption into future generation communication systems. We study the fundamental aspects of polar coding, namely, encoding, decoding, and code construction, and obtain new efficient algorithms for each of them. We also provide efficient pseudocodes wherever possible.

Our first contribution is to present systematic polar codes (SPC) as a better alternative to the non-systematic polar codes, originally proposed by Arıkan, while both have same block error rate (BLER), former has lower bit error rate (BER). We propose three new encoding algorithms that bring down the encoding complexity of SPC to that of non-systematic polar codes. Next, we focus on the decoder and propose three new decoders that achieve a natural decoding order in SCD, lower the latency of an SCD, reduce the hardware resources of an SCD, and parallelize the LSCD. Later we perform an exhaustive survey of all polar code constructions (PCC) in literature with the objective of finding the best PCC to produce polar codes with the least BLER and BER. We propose a novel heuristic algorithm to address the problem of non-universality of polar codes and find the best design-SNR to use with any PCC. We find by exhaustive simulations that the choice of a PCC does not play any role when we know the choice of design-SNR which varies with the PCC. We propose a simple heuristic for the choice of design-SNR based on Monte-Carlo (MC) simulations. This heuristic is slow and has a high complexity because of the MC simulations. So, we later extend it and propose a more advanced algorithm that finds the best design-SNR by simply running one or more PCCs instead of the high complexity MC simulations. This new algorithm makes use of an accurate estimation of BLER that we propose based on running multiple PCCs.

We finally demonstrate an FPGA implementation project of polar codes that we undertook as a collaboration. We also present a software package provided online for free to simulate polar codes and is currently being used actively by many researchers across the globe. A new version of this software package has recently been developed for MathWorks, Inc. for inclusion into the Communications System Toolbox™ of MATLAB®.

Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma in any university or equivalent institution, and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Harish Vangala

Acknowledgements

This thesis is a result of the enormous encouragement and support from my supervisors and well-wishers Prof. Emanuele Viterbo and Dr. Yi Hong. They have been the constant source of motivation, inspiration and are the prime reason towards the completion this thesis. I will be indebted to them throughout my research life.

I would like to sincerely thank Monash University and Faculty of Engineering for providing me an opportunity and supporting me academically and financially to study under the esteemed guidance of Emanuele Viterbo & Yi Hong at the Department of Electrical and Computer Systems Engineering. My doctoral studies and the stay in Melbourne were a life-changing experience that brought me closer to where my heart lies, namely, the research. I will cherish the moments that I spent here throughout my life.

I would like to thank the staff mainly Maria Scalzo, Emily Simic, Roslyn Remington, and Geoff Binns for making all the administrative tasks easy. Thanks to all my friends for their emotional support and the productive technical discussions throughout my candidature.

I sincerely thank my parents and my loving sister for their endless support.

Contents

Abstract v

Contents x

List of Figures xii

List of Tables xv

List of Algorithms xvii

List of Abbreviations xix

1 Introduction **1**

1.1 Literature Survey 2

1.2 A Quick Summary and a List of Our Main Contributions 4

1.3 Organization of the Thesis 5

2 Polar Codes **7**

2.1 The Notation 7

2.2 The Polar Encoding 9

2.3 The Decoding of Polar Codes 10

2.4 The Code Construction 17

3 Efficient Encoders for Polar Codes **21**

3.1 Introduction to Systematic Polar Codes 21

3.2 New Systematic Polar Encoders 24

3.3 Non-Recursive Systematic Polar Encoder - EncoderA 24

3.4 Recursive Systematic Polar Encoders - EncoderB and EncoderC 25

3.5 An Efficient Non-Recursive NSPE 28

3.6 Summary 29

4 Efficient Decoders for Polar Codes 33

4.1 Efficient Successive Cancellation Decoders . . . 33

4.2 Simplification of SCD on a Binary Erasure Channel . . . 37

4.3 The Permuted Successive Cancellation Decoder (PSCD) . . . 41

4.4 Improved Multiple Folded Successive Cancellation Decoding . . . 48

4.5 A New Parallelized LSCD . . . 52

4.6 Summary . . . 61

5 Efficient Construction of Polar Codes 63

5.1 Introduction . . . 63

5.4 Description of the Main PCC Algorithms . . . 66

5.5 Selecting the Best Design-SNR . . . 73

5.6 The Comparison of PCCs at Finite Length . . . 73

5.7 Pseudocodes for Implementing the PCCs . . . 77

5.8 A New Quantizer of BI-DMC with Optimal Mutual Information . . . 82

5.9 Summary . . . 90

6 Performance Estimation of Polar Codes 91

6.1 Bit Channel Metrics and BLER estimation from PCC . . . 92

6.2 New Comparison Strategies for PCCs . . . 96

6.3 The Efficient Choice of a Reliable Design-SNR . . . 99

6.4 Summary . . . 104

7 An FPGA Implementation and a Software Package for Polar Codes 105

7.1 Introduction to the FPGA Implementation . . . 105

7.2 The IMFSCD on FPGA . . . 105

7.3 The Software Package and Tutorials . . . 111

8 Conclusions and Future Directions 113

8.1 Conclusions . . . 113

8.2 Future Directions . . . 114

Bibliography 117

Appendix: Documentation for “Polar Codes in MATLAB” 127

List of Figures

2.1 The fundamental block representing the Arıkan's kernel (also the encoder for $N = 2$ and is used multiple times in recursion to implement the actual encoding for arbitrary N) 10

2.2 Arıkan's $O(N \log_2 N)$ complexity encoder graph to implement (2.1) for $N = 16$ formed by a smart and recursive application of the fundamental block in Fig. 2.1 . 11

2.3 Summary of basic operations of an SCD 14

2.4 The encoding graph with nodes identified for use with SCD for $N = 8$ 17

2.5 The polar code construction in (2.18) illustrated for $N = 4$ 18

3.1 The non-systematic polar encoder (NSPE) with an exact complexity of $\left(\frac{N}{2} \log_2 N\right)$ XORs, when $(N, K, \mathcal{F}) = (8, 5, \{0, 2, 4\})$ (see (3.1)) 23

3.2 The flow-of-calculations required for the SPE of EncoderC on the encoder graph . . 25

3.3 The tree of recursive computations 27

3.4 Traversal of nodes in recursion, at $N = 8$ 28

4.1 A binary erasure channel (BEC) with erasure probability ϵ 38

4.2 Illustration of the permutations in the encoder graph 42

4.3 Performance of SCD under original polar code construction, PSCD with same code, and PSCD under new code construction; all constructions initialized with $z_{0,0} = 0.5$ (-1.592dB) for $N = 1024$ and $R = 0.7$ in BI-AWGN channel. All permutations are chosen randomly, except for the specific permutation of reversal. 46

4.4 Performance of PSCD used for the new codes constructed using Algorithm-10, optimized at 5dB, at $N = 8192$ and $R = 0.9$ in BI-AWGN channel 46

4.5 Illustrating folding for $N = 8$ 49

4.6 FER comparison of IMFSCD($\kappa = 1, 2, 3$) vs. SCD, at $N = 8192, R = 0.8$ and code construction @ 0dB [126] 52

4.7 BER comparison of IMFSCD($\kappa = 1, 2, 3$) vs. SCD, and code construction @ 0dB [126] 53

4.8 FER of a single parity check code of length 8 concatenated with a polar code of length 1024, at effective rate 0.5 compared with a polar codes with $N = 8192, R = 0.5$ and $N = 1024, R = 0.5$ 53

4.9 BER of a single parity check code of length 8 concatenated with a polar code of length 1024, at effective rate 0.5 compared with a polar codes with $N = 8192, R = 0.5$ and $N = 1024, R = 0.5$ 54

5.1 The first 2 iterations of the bit channel estimation under PCC-0 in an AWGN channel using Eqs. (5.5) and (5.6) 67

5.2 The first 2 iterations of the bit channel estimation under PCC-2 in a BI-AWGN channel 71

5.3 The first 2 iterations of the bit channel estimation under PCC-3 in a BI-AWGN channel using Eqs. (5.15) and (5.16) 72

5.4 The BER performance of polar codes under PCC-0 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$ 74

5.5 The BER performance of polar codes under PCC-1 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$ 74

5.6 The BER performance of polar codes under PCC-2 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$ 75

5.7 The BER performance of polar codes under PCC-3 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$ 75

5.8 Comparison of all the constructions when design-SNRs are chosen according to the search algorithm ($N = 2048$ and $R = 0.5$) 76

5.9 Quantization of a BI-DMC to reduce channel's output alphabet size and corresponding finite probability transition matrices as tables 83

5.10 System model for channel quantization 84

5.11 The directed acyclic graph representing the adjacency matrix definition in (5.20) . 86

6.1 An accurate BLER estimation obtained using all PCCs $N = 256$ and $R = 0.5$ 96

6.2 *Approach 1*: the acccuracy of FER or BLER estimates from various PCCs at $N = 1024, R = 0.9$ 97

6.3 *Approach 1*: the acccuracy of FER or BLER estimates from various PCCs at $N = 2048, R = 0.9$ 97

6.4 *Approach 2*: BLER estimates after choosing design-SNRs at $N = 2048$ and $R = 0.5$ 99

7.1 Illustrating folding for $N = 8$ and $N = 16$ 106

7.2 Hardware architecture of an SCD of $N = 16$ 108

7.3 Hardware architecture of the IMFSCD with square folding a reminiscent of $N = 4$ 109

7.4 Poster presented at *Semi-annual Conference of Intel Collaborative Research Institutes on Mobile Networking and Computing*, held at Beijing, 2015. 110

7.5 Visitor traffic to our website `www.polarcodes.com` between Oct 2016 – Mar 2017 (The peak visits were during the time when polar codes were accepted into 5G) . . 111

A.1 The overall conceptual structure of the MATLAB package (parts in gray are unavailable, may be available in future versions) 129
A.2 Operational division of the package contents 130

List of Tables

3.1 Summary of Systematic Polar Encoders 24

4.1 Simple look-up tables for likelihood transformations of SCD in a BEC 40
4.2 Look-up table with an alternative simpler notation 40
4.3 Summary of changes in SCD algorithm for PSCD 44

5.1 Optimal design-SNR and operating-SNR values for various combinations of N and R and PCC algorithm. (The target-BER=10^{-4}) 76

7.1 Comparison of hardware resources required for IMFSCD and SCD 107

A.1 The core utilities of the MATLAB package 131
A.2 Quick descriptions of all the arguments the earlier table 132
A.3 Table of micro-scale functions constituting the basic modules in Appendix A.3.3 . . 138

List of Algorithms

1 The Polar Encoding (Recursive) 12
2 Main Module of SCD Algorithm 18
3 **UpdateL**$(\mathbf{i},\mathbf{j})$: Recursive L.R. computation 19
4 **UpdateB**$(\mathbf{i},\mathbf{j})$: Broadcasting of decisions 19

5 The Non-Recursive Polar Encoding 30
6 EncoderA$(\mathbf{y},\mathbf{x})$ 30
7 $\mathbf{v} =$ EncoderB$(i,j,\mathbf{y},\mathbf{x},\mathbf{r})$ 31
8 EncoderC$(i,j,\mathbf{y},\mathbf{x},\mathbf{r})$ 31

9 **nextindex**(π,p) : Decoding Order 45
10 Polar Code Construction for PSCD 47
11 **EP_LSCD**: The Eff. Parallelized LSCD 58
12 **bestSCDindex**(l) 58
13 **advanceLIST**(l,s): Advance & prune list 59
14 [I, decisionbits] = HowToCopy(M, s) 60

15 **PCC-0**: The Bhattacharyya bounds 77
16 **indices_of_greatest_elements**$(\mathbf{v},\ l)$ 77
17 **PCC-1**: The Monte-Carlo estimation 78
18 **UpdateL**$(\mathbf{i},\mathbf{j})$: Recursive LR comp. of SCD 78
19 **PCC-2**: Tal & Vardy's $\Big($See Fig. 5.2$\Big)$ 79
20 **discretizeAWGN()**: The block Q_1 80
21 **ConvolveA()**: The block C_A in Fig. 5.2 80
22 **ConvolveB()**: The block C_B in Fig. 5.2 81
23 **Quantize_to_size()**: The block Q_2 in Fig. 5.2 81
24 **PCC-3**: The Gaussian approximation 82
25 **indices_of_least_elements**$(\mathbf{v},\ l)$ 82
26 **OQA**: K-hop SSSPP solution based on Bellman-Ford algorithm (Guerin'02) . 88
27 Modifications to Algorithm 26 (replaces lines 4-14 in Algorithm 26) 89

List of Abbreviations

3GPP	3rd Generation Partnership Project
AFER	Approximate Frame Error Rate
APP	Aposteriori Probabilities
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BI-AWGN	Binary-Input Additive White Gaussian Noise
BI-DMS	Binary-Input Discrete Memoryless and Symmetric Channel
BLER	Block Error Rate
BP	Belief Propagation
BPE	Blockwise polar Encoding
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CA-LSCD	Cyclic Redundancy Check Aided List Successive Cancellation Decoder
CRC	Cyclic Redundancy Check
DAG	Directed Acyclic Graph
dB	Decibel
DMC	Discrete Memoryless Channel
DMS	Discrete Memoryless and Symmetric Channel
eMBB	Enhanced Mobile Broadband
FER	Frame Error Rate
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
IMFSCD	Improved Multiple Folded Successive Cancellation Decoder
IPE	Interleaved Polar Encoding
LDPC	Low-Density Party-Check
LLR	Log-Likelihood Ratio
LP	Linear Program
LR	Likelihood Ratio
LSCD	List Successive Cancellation Decoder
MAP	Maximum Aposteriori Probability
MI	Mutual Information
MLD	Maximum Likelihood Decoder
ML	Maximum Likelihood
NSPC	Non-Systematic Polar Codes

NSPE	Non-Systematic Polar Encoding
OQA	Optimal Quantizer Algorithm
PCC	Polar Code Construction
PE	Processing Element
PMI	Partial Mutual Information
PSCC	Permuted Successive Cancellation Decoder based Polar Code Construction
PSCD	Permuted Successive Cancellation Decoder
SCD	Successive Cancellation Decoder
SNR	Signal-to-Noise Ratio
SPE	Systematic Polar Encoding
SSSPP	Single Source Shortest Path Problem
TPM	Transition Probability Matrix
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Chapter 1

Introduction

Shannon's rigorous foundation of telecommunications as "*a mathematical theory of communication*" [1] left open a number of elusive goals to the research community ever since. One among them is to design practical coding schemes that asymptotically approach *channel capacity* or the *Shannon limit*. Arguably, it is simultaneously the oldest and the most intriguing problem ever since within the community. The problem remains still open for a wide range of channel models. To this end, a significant amount of research was dedicated and numerous coding schemes were developed. A leap of advancement in binary codes was finally made in 1990s with the invention of turbo codes and the low-density parity-check (LDPC) codes. These codes, with different forms of low complexity iterative decoding, can *approach* capacity of an additive white Gaussian noise (AWGN) channel up to a fraction of a decibel (dB) of signal-to-noise ratio (SNR) . This satisfied many, since the problem was solved in several practical scenarios. However, neither these codes are proven to be asymptotically *capacity achieving* nor they are the best possible codes at finite length [2, Fig. 5.2], [3, Fig. 15]. Furthermore, these codes also have several problems such as error floors [4–7], high encoding complexity [8], non-deterministic iterative decoding complexity and latency. Additionally, they have other implementation challenges such as routing congestion, memory-access problems, etc., mainly due to their random structure [9, 10]. Thus, the central problem of developing practical and capacity achieving channel codes was still open on any communication channel of general interest.

This six-decade-old problem in information theory was finally solved by Arıkan by inventing *polar codes* using a novel concept of *channel polarization* [11]. Polar codes became the first ever family of low complexity codes that are proven to be asymptotically capacity achieving over a wide range of channels known as binary-input discrete memoryless channels (BI-DMC). Though it was initially proven to achieve capacity over the class of binary-input discrete memoryless symmetric (BI-DMS) channels only, the results were later extended to a general DMC. Accordingly, a much general theory of polarization was formulated [12–16]. Polar codes garnered a significant attention in the literature over the past decade as a remarkable result in information theory. They also served as an inspiration to find other possible classes of capacity achieving codes. Two such recent notable results are the spatially coupled LDPC code ensembles with iterative decoding over BI-DMS channels [17–19] and Reed-Muller codes with maximum-aposteriori-probability (MAP) decoding over binary erasure channels [20, 21]. Several practical extensions are also in progress [22–25]. Recently an interesting equivalence of polar codes and *optimised codes for bitwise multistage decoding*, introduced by Stolte [26], was also established [26, 27].

In addition to contributing to a significant theoretical advancement, polar codes naturally possess several practically attractive features such as deterministic structure, simple implementations, fixed and low computational complexity at encoder and decoder, excellent performance, lack of error floors, and high area-efficiency. Indeed, numerous implementations have been reported in literature demonstrating these advantages [28–35].

Due to these advantages, polar codes have been introduced into 5G after the recent No.87 RAN1 meeting by 3rd Generation Partnership Project (3GPP) held in Nevada, US, on 17 November 2016. Polar codes were selected as the control channel coding scheme for the application of 5G in the enhanced mobile broadband (eMBB) scenario [36–38]. However, there still exist some important practical issues that need to be addressed as they are incorporated into future systems. This includes improving the performance over state-of-the-art codes, proposing efficient systematic encoders, reducing the latency, etc. The main objective of this thesis is to study each of these issues and propose efficient algorithms to address them. In this thesis, we broadly divide our key contributions into three areas, namely, encoding, decoding, and code construction. A concise summary of all the contributions is given in Section 1.2. We will see some of the important literature on polar codes in the next section.

1.1 Literature Survey

The literature on polar codes since Arıkan's seminal paper [11] has spanned across multiple disciplines including information theory, coding theory, cryptography, storage, and wireless communications in general. The range of potential applications of polar codes is thus significantly large and indeed, equally large amount of literature is available. Therefore we select the fundamental binary polar codes on an AWGN channel, binary erasure channel (BEC), binary symmetric channel (BSC). However, we consider an AWGN channel to illustrate the concepts whenever they are easily extensible to the other channels. In this section, we will briefly survey the literature with an emphasis on implementation challenges related to binary polar codes.

In spite of having an asymptotically capacity achieving property over discrete memoryless channels, the foremost challenge faced by polar codes was their inferior performance both in error correction and delay at finite length (non-asymptotic regime) when compared with the state-of-the-art coding techniques. The main reasons for such a performance are identified to be use of a suboptimal *successive cancellation decoder* (SCD), sequential processing at the decoder, and a lower minimum distance structure of the code itself. This understanding inspired a significant amount of new research on polar codes, which can be broadly divided into two categories as we discuss below.

First approach for improving the performance of polar codes is to improve the original SCD so that it produces a close to optimal performance. It may be noted that coding theory has a rich collection of decoders applicable for different kinds of linear codes. Motivated by this observation, many attempts have already been made to extend the well-known decoders in literature to polar codes and obtain a near optimal performance. Note that the optimal performance is obtained only by using the exponentially complex maximum-likelihood (ML) decoding. One of the early attempts in this direction was to apply a belief propagation (BP) decoder for polar codes [39–41]. Later, several variants of BP decoding were also proposed [42–52]. A significant advancement was made by a list successive cancellation decoding (LSCD) [53] which improved the decoding performance much closer to ML decoding. It was found that the polar codes with LSCD can perform better than the state-of-the-art LDPC codes used in WiMAX standard [53]. This encouraged the community to also propose several efficient variants of LSCD [54–68]. It should be noted that the performance of polar codes is still not that of the best possible codes at any finite code length, as was found by using finite block length regime analysis [53, Fig. 3], [2], [69]. In this context, a significant number of novel decoding techniques have also been proposed to achieve near-ML performance with practical decoding complexity. This includes a linear program (LP) decoding [70, 71], multi-dimensional decoding [72], stack decoding [73, 74], sequential decoding [75, 76], ordered-statistic decoding [69, 77], soft cancellation (SCAN) decoder capable of producing soft-output [78, 79], fast Hadamard transform based decoding [80], and efficient ML decoding using folding [81, 82]. In addition, a number of variants improving different aspects of the original SCD are also available [83–93]. A number of hardware implementations have already demonstrated the efficiency of several of these decoding techniques especially the variants of SCD and LSCD [28–35, 38, 94–100]. From the decoder perspective, we make contributions on improving the performance further in addition to addressing several issues that were not addressed earlier, namely, delay reduction, reduced complexity implementations, and the natural decoding order.

The second approach for improving the performance of polar codes is to alter the code itself to have better performance while lending itself to efficient decoding. This can include several techniques such as improving the polar code construction, using generalized Arıkan kernels, code concatenation, and using the systematic code form for a better bit error rate (BER). Improving the construction is one of the first problems addressed in the literature of polar codes. Accordingly, a number of constructions were proposed [39, 42, 101–106]. Another interesting attempt to improve polar codes was to use alternatives to Arıkan's original 2×2 kernel. It was found that it is possible to have alternative capacity achieving kernels with larger dimensions but have higher error exponent [107–110]. Code concatenation was another practically attractive alternative and was addressed in [41, 111–118]. An interesting perspective of polar codes as an instance of generalized concatenated codes is presented in [113]. Systematic codes are also being preferred for their improved BER in various settings including that of concatenation and generalized kernels [33, 100, 118–123]. In this aspect we make some important contributions on the code construction and systematic polar codes. We propose efficient systematic encoding algorithms, perform an extensive survey of all notable constructions, and propose methods to optimize the construction algorithms.

1.2 A Quick Summary and a List of Our Main Contributions

Our main contributions span all three fundamental aspects of polar codes, namely, encoding, decoding, and code construction. In addition, we have frame error estimation, hardware demonstrations, and software packages in MATLAB and C++.

Amidst the exciting and wide range of multidisciplinary applications of polar codes, our contributions to polar codes have also received citations in the 3GPP TSG RAN WG1 meetings on 5G [124].

Our software package has received good acknowledgements from the community and was receiving regular reads a week as of today [125]. After a proposal from MathWorks, Inc., we have also finished a project in August 2017, specializing our package so as to be included into the Communication System Toolbox™ of MATLAB®.

The chapters of this thesis are organized in such a way that the table of contents becomes an illustrative guide to our contributions. We will give an informative description of our contributions below. The list of all our publications given at the end of the thesis will supplement this.

1. *New encoders*: Chapter 3 discusses our contributions on the encoder side. We start with an efficient implementation of original polar encoder in recursive and non-recursive forms, and we solve the problem of systematically encoding polar codes formed with arbitrary frozen indices. We propose three systematic polar encoders that perform systematic polar encoding as efficiently as the original (non-systematic) polar codes.
2. *New decoders*: Chapter 4 discusses our contributions on the decoder side. We provide a few efficient implementations of the original SCD, and also the LSCD, and consequently the CRC-aided LSCD. We then propose several novel decoders, namely, the permuted successive cancellation decoder (PSCD), and the improved multiple folded successive cancellation decoder (IMFSCD), and the efficient nearly parallel LSCD.
3. *New constructions*: Chapter 5 discusses our contributions to polar code construction (PCC). We first provide an efficient implementation of each of the four well-known construction algorithms and compare them in terms of the performance of the polar codes they produce. We use extensive simulations to list the good design-SNR values as a reference, until a code-length 2^{16} (65,536). We find that the choice of a PCC does not effect if we choose design-SNR appropriately depending on the PCC. We therefore propose a heuristic for the choice of a design-SNR. We also provide a construction algorithm that is based on a new optimal quantizer proposed using some graph theoretic concepts.

4. *A sample hardware implementation*: We undertake a collaborative effort towards an FPGA implementation of SCD, which shows the efficiency of our novel multiple folded successive cancellation decoder. In this exercise, we establish a trade-off between the hardware requirements and the latency, and find an exciting further generalization that also extends the trade-off to a higher granularity.
5. *New utilities*: We have found two useful utilities as an extension of our earlier contributions on polar code constructions. They are the efficient estimation of the frame error rate of a given polar code with SCD, and an extension of the earlier heuristic to find the design-SNR. The latter is an efficient algorithm to find a good design-SNR avoiding the use of any Monte-Carlo simulations of high complexity.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we revise some of the fundamentals of polar codes and then explain the state of the art literature on polar codes. In Chapter 3, we illustrate efficient algorithms for systematic and non-systematic encoding, give a detailed characterization of their computational complexity, and discuss the comparisons. In Chapter 4, we discuss several novel variants of the original SCD, followed by an efficient implementation of the LSCD to address latency and performance problems in SCD. In Chapter 5, we introduce a range of PCCs and provide a detailed performance comparison. We address the problem of non-universality by proposing a simple heuristic based on Monte-Carlo simulations. Later in Chapter 6, we extend this study to find new applications of the PCCs such as accurate BLER estimation and estimating the optimal design-SNR. The latter algorithm forms an advanced heuristic to address the well-known problem of non-universality of polar codes. In Chapter 7, we provide an FPGA implementation of IMFSCD and our MATLAB software package which is currently being used widely. We conclude the thesis in Chapter 8 by providing open problems for future studies.

Chapter 2

Polar Codes

In this chapter, we revise the basic definition of polar codes along with its encoding, decoding, and construction algorithms. We also provide a basic pseudocode implementation for each of them. We aim to provide this chapter as a standalone, concise, and quick reference guide to the fundamentals of polar codes. The definitions here will be used throughout the thesis, and the concepts will be extended towards more advanced topics in later chapters.

The following is a quick summary of the main contributions from this chapter:

1. A quick and self-contained overview of polar codes and its fundamentals, namely, encoding, decoding, and code construction.
2. Standalone, efficient, recursive pseudocode implementations of encoder, decoder, and the code construction.
3. Complexity analysis of the basic encoding, decoding, and code construction algorithms.

2.1 The Notation

This section describes the common terminology, and notation used throughout the thesis to describe various concepts of polar codes.

2.1.1 The Common Mathematical Objects

Vectors and Matrices:

The vectors are denoted with lower case bold letters such as $\mathbf{x}, \mathbf{y}, \mathbf{p}, \mathbf{q}$, and the matrices are denoted with capital and bold letters such as $\mathbf{A}, \mathbf{F}, \mathbb{F}$. The individual elements of the same may be denoted using letters and subscripts, for example, $\mathbf{x} = [x_0\ x_1\ \ldots\ x_{10}]$, $\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ y_3 \end{bmatrix}$, and $\mathbf{A} = [a_{ij}]$.

One important binary matrix in this thesis and for polar codes in general is the Arıkan's kernel of dimensions 2×2. It is represented by either $\mathbf{F} \triangleq \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$ or $\mathbb{F} \triangleq \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, depending on

---

Parts of this chapter are based on our published work [126] whether the vectors are represented as columns or rows respectively. Such a choice is only for a notational convenience on different occasions.

Sets and Operations:

Sets are denoted with calligraphic letters such as $\mathcal{N}, \mathcal{F}, \mathcal{I}$, and these three letters have reserved meanings: the set of first N integers, the set of frozen bit indices, and the set of information bit indices respectively (see next section for details). $\subset$ represents the subset relation of two sets, $\cup$ represents the union, and $\backslash$ represents the set difference operation on two sets, for example, $\mathcal{F} \subset \mathcal{N}$ $\mathcal{N} = \mathcal{F} \cup \mathcal{I}$, and $\mathcal{I} = \mathcal{N} \backslash \mathcal{F}$.

The set of first N whole numbers $\mathcal{N} \triangleq \{0, 1, \ldots, N-1\}$, is often used to describe the set of all indices of the elements of vectors and matrices. Throughout the thesis we will almost always use the *zero-based numbering* [127] having the *index origin* or the first index equal to 0.

The $+$ operator can represent different operations depending on the operands. For example, it denotes a binary XOR when the operands are binary, and an integer addition when the operands are integers.

Subvectors and Submatrices:

Given any subset of indices $\mathcal{I}$ from a vector $\mathbf{x}$ (row or column), we denote the corresponding subvector as $\mathbf{x}_{\mathcal{I}}$. Given any subset of column indices $\mathcal{I}$ of a matrix $\mathbf{A}$, we denote the corresponding submatrix with $\mathbf{A}_{\mathcal{I}}$. Similarly if $\mathcal{I}$ denotes a subset of row indices of the matrix $\mathbf{A}$, the corresponding submatrix is denoted $\mathbf{A}_{(\mathcal{I})}$.

Operations on Matrices and Vectors:

Let $\mathbf{A}^T$ denote the transpose of the matrix. An n-fold Kronecker product of a given matrix is denoted as $\mathbf{A}^{\otimes n} = \mathbf{A} \otimes \cdots \otimes \mathbf{A}$ (n instances). More specifically, the n-fold Kronecker products of $\mathbf{F}$ and $\mathbb{F}$ are denoted with $\mathbf{F}^{\otimes n}$ and $\mathbb{F}^{\otimes n}$ respectively.

2.1.2 Code Parameters: $(N, K, \mathcal{F}, \mathbf{v}_{\mathcal{F}})$ or $(N, K, \mathcal{F})$

A linear block code is generally specified along with three important parameters, namely, the code length (N), information length (K), and the minimum distance (d_{min}). The triple (N, K, d_{min}) is usually given to specify them, under a general description. A polar code also follows this tradition closely as below.

A *polar code* is completely specified by the four-tuple $(N, K, \mathcal{F}, \mathbf{v}_{\mathcal{F}})$, where N is the code length in bits (or *block-length*), K is the number of information bits encoded per codeword (or *code dimension*), $\mathbf{v}_{\mathcal{F}}$ is the binary vector of length $N-K$ known as *frozen bits*, and $\mathcal{F} \subset \mathcal{N}$ is a subset of $N-K$ indices known as *frozen bit indices.* The complement of the set $\mathcal{F}$ is denoted as $\mathcal{I} = \mathcal{F}^c = \mathcal{N} \backslash \mathcal{F}$, and is called *information bit indices.* N is always a power of 2 and is denoted with $n \triangleq \log_2 N$ or $N = 2^n$.

When the *frozen bits* are zeros ($\mathbf{v}_{\mathcal{F}} = \mathbf{0}$, default), we specify it more compactly as $(N, K, \mathcal{F})$.

2.2 The Polar Encoding

For an $(N, K, \mathcal{F}, \mathbf{v}_{\mathcal{F}})$ polar code, we describe below the encoding operation of a vector $\mathbf{u}$ of K information bits. We denote the codeword corresponding to $\mathbf{u}$ by $\mathbf{x}$ of N code bits, which is generated as:

$$\mathbf{x} = \mathbf{F}^{\otimes n} \cdot \mathbf{d} \tag{2.1}$$

where $\mathbf{d} \in \{0,1\}^N$ such that $\mathbf{d}_{\mathcal{F}} = \mathbf{v}_{\mathcal{F}}$ and $\mathbf{d}_{\mathcal{F}^c} = \mathbf{u}$. In other words, $\mathbf{d}$ is nothing but $\mathbf{u}$ padded with constant frozen bits. More elaborately, the above may be expanded as follows.

$$\mathbf{x} = \left(\mathbf{F}^{\otimes n}\right)_{\mathcal{I}} \mathbf{u} \;+\; \left(\mathbf{F}^{\otimes n}\right)_{\mathcal{F}} \mathbf{v}_{\mathcal{F}}, \tag{2.2}$$

$$= \mathbf{G}\,\mathbf{u} \;+\; \mathbf{c}, \tag{2.3}$$

where $\mathbf{G}$ is the generator matrix and $\mathbf{c} = (\mathbf{F}^{\otimes n})_{\mathcal{F}} \cdot \mathbf{v}_{\mathcal{F}}$ is a constant binary offset vector which is zero when the frozen bits are zeros, i.e., $\mathbf{v}_{\mathcal{F}} = \mathbf{0}$.

The Choice of $\mathbf{v}_{\mathcal{F}}$*:*

It is known that performance of each polar code is identical for any choice of $\mathbf{v}_{\mathcal{F}}$ when the channel is symmetric [11]. But when $\mathbf{v}_{\mathcal{F}} = \mathbf{0}$ the polar code becomes a *linear code*, and when $\mathbf{v}_{\mathcal{F}} \neq \mathbf{0}$, the polar code becomes a *coset code* [11], which is simply a linear code with the constant offset $\mathbf{c}$. The latter is a significantly different code in the sense that the codeword $\mathbf{0}$, that is present in all linear codes, may not be present in the coset code. However, $\mathbf{v}_{\mathcal{F}} = \mathbf{0}$ is a common choice that allows the $(N, K, \mathcal{F})$ polar code to be a standard linear code in its non-systematic form, with its generator matrix $\mathbf{G} = (\mathbf{F}^{\otimes n})_{\mathcal{I}}$.

Avoiding the Use of the Bit-Reversal Permutation $\mathbf{B}_N$ *from [11]:*

Due to the design and also due to the choice of the orientation of the SCD circuit used by Arıkan, the SCD decodes the message bits sequentially in a bit-reversed order rather than a natural order. To avoid this Arıkan proposed a permutation at the encoding stage itself. This may be written by cascading another matrix, namely, the bit-reversal permutation matrix $\mathbf{B}_N$, as matrix multiplication prior to the above encoding operation. This technique ultimately leads to a natural decoding order of bits at the SCD, thereby simplifying the notation that Arıkan used to prove his coding theorems. Today the usage of $\mathbf{B}_N$ to describe encoding is equally popular as the alternatives that avoid it, since Arıkan clearly mentions in [11] that it is completely optional with no loss whatsoever, as far as the implementations take care of the decoding order. From a pedagogical perspective, we believe that such a permutation adds unnecessary difficulty in understanding a much simpler encoding operation, and therefore choose to avoid it throughout our thesis.

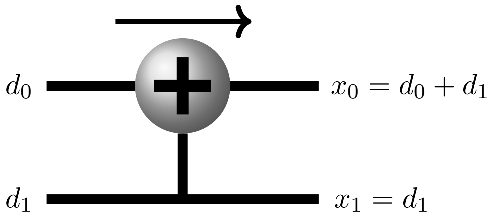


Figure 2.1: The fundamental block representing the Arıkan's kernel (also the encoder for $N = 2$ and is used multiple times in recursion to implement the actual encoding for arbitrary N)

The Implementation:

An efficient low complexity implementation of the (2.1) was suggested by Arıkan, and is shown in Fig. 2.2. We note that this implementation is a binary arithmetic circuit of $n = \log_2 N$ stages in general. The advantage of such an implementation is that it has $\mathcal{O}(N \log N)$ complexity only, against the straight forward matrix multiplication from (2.1) which has $\mathcal{O}(N^2)$ complexity. This is reminiscent of other popular algorithms in computer science and signal processing literature such as the fast Fourier transform (FFT) [128], the fast Hadamard transform [129], the fast Möbius transform [130], the fast integer multiplication [131, 132], and the fast polynomial multiplication [133].

In short, the encoding algorithm in any form is a smart and recursive application of the Fig. 2.1, which is simply a circuit representation of the kernel $\mathbf{F}$. The simplest way to implement such an algorithm is using a recursion as shown in Algorithm 1 and is due to Arıkan [34]. A more efficient implementation uses a non-recursive approach and is described in the next chapter.

Alternative Representations:

Finally, we note that the encoding equations (2.1) and (2.2) can be modified when the vectors are represented as rows instead of columns using the equivalent kernel $\mathbb{F} = \mathbf{F}^T$ as defined earlier. This form is useful when we discuss systematic polar codes later.

2.3 The Decoding of Polar Codes

Polar codes are originally decoded using an improved low complexity sequential algorithm called as *successive cancellation decoding* (SCD). Undoubtedly, the SCD for its simplicity became the prime reason for all the glory of polar codes after having been established as capacity achieving by Arıkan. Even today, most of the practical decoders such as the list decoder are variants of SCD. In the following, we will first formulate the problem formally, and then give its full algorithm. A more detailed analysis of the algorithms and their alternatives is given later in Chapter 4.

2.3.1 The Decoding Problem - Formulation

Consider a message vector $\mathbf{u}_{K\times 1}$ encoded into a codeword $\mathbf{x}_{N\times 1}$ and sent through a noisy channel, for example, a binary input discrete memoryless symmetric (DMS) channel (such as BSC, BEC),

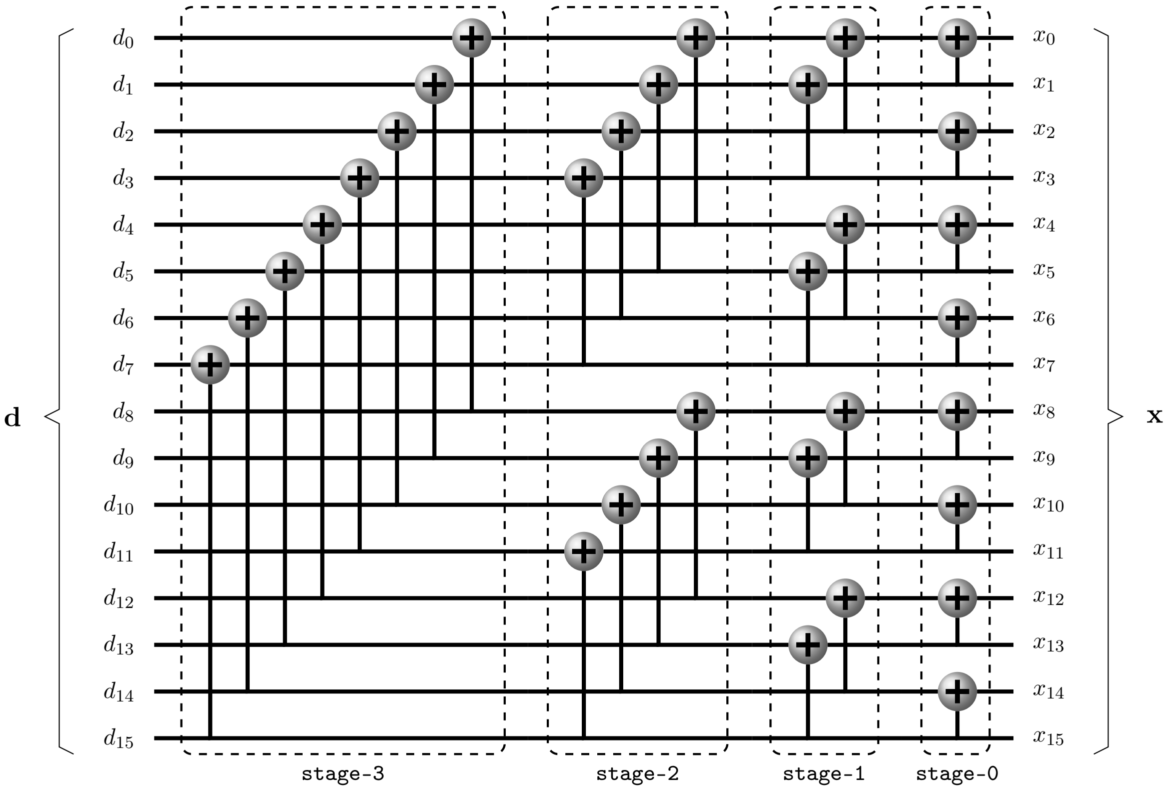
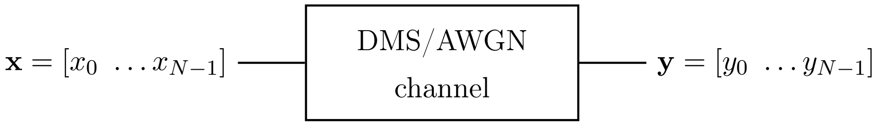


Figure 2.2: Arıkan's $O(N \log_2 N)$ complexity encoder graph to implement (2.1) for $N = 16$ formed by a smart and recursive application of the fundamental block in Fig. 2.1

and AWGN channel. Let the noisy version of $\mathbf{x}$, denoted by $\mathbf{y}$, be received by the receiver present at the output of the channel. Note that each element x_i in $\mathbf{x}$ is independently perturbed by channel noise to give the corresponding noisy observation y_i in $\mathbf{y}$.


$\mathbf{x} = [x_0 \ \ldots x_{N-1}]$
DMS/AWGN
channel
$\mathbf{y} = [y_0 \ \ldots y_{N-1}]$

In general, each y_i could have been resulted by all the two possible values of the channel input, namely, x_i, but with different probabilities which can be computed explicitly as below. This establishes a Bernoulli distribution, known as the aposteriori probabilities (APP) of x_i given each observation y_i.

$$\begin{aligned}
\mathbf{p}^{(i)} &= [p_0^{(i)}, p_1^{(i)}] && (2.4)\\
&= [\Pr\{x_i = 0|y_i\}, \Pr\{x_i = 1|y_i\}] \quad (\text{Note: } \Pr\{x_i = 0\} = \Pr\{x_i = 1\} = \tfrac{1}{2}) && (2.5)\\
&\propto \left[\frac{\Pr\{x_i = 0|y_i\}}{\Pr\{x_i = 1|y_i\}} \text{ or } \frac{\Pr\{y_i|x_i = 0\}}{\Pr\{y_i|x_i = 1\}},\ 1\right] = [L(y_i),\ 1] && (2.6)
\end{aligned}$$

Note that there is no loss of information from (2.5) to (2.6) because, one can be used to obtain the other simply by using appropriate normalization. However as we shall see, the likelihood ratios

Algorithm 1 The Polar Encoding (Recursive)

```
Input   : $(N, K, \mathcal{F}, \mathbf{v}_{\mathcal{F}} = 0)$ and $K$ message bits $\mathbf{u}$.
Output  : $\mathbf{x}$, the codeword as the encoded message $\mathbf{u}$.
Allocate $N$ bits as vector $\mathbf{x}$
Set $\mathbf{x}_{\mathcal{I}} = \mathbf{u}$ and $\mathbf{x}_{\mathcal{F}} = \mathbf{v}_{\mathcal{F}}$        ▷ Preparation of d in (2.1)
Encode($\mathbf{x}$, $N$)        ▷ Pass by reference (in-place results)
Return $\mathbf{x}$.
--------------------------------------------------------------
Function Encode($\mathbf{d}$, $N$)
Let $\mathbf{d} = \left[d_0, d_1, \ldots, d_{N-1}\right]$
if $N = 2$ then
    $d_0 = d_0 + d_1$
    Return
else
    Let $\mathcal{A} = \{0, 1, \ldots, \frac{N}{2} - 1\}$ and $\mathcal{B} = \{\frac{N}{2}, \ldots, N - 1\}$
    $\mathbf{d}_{\mathcal{A}} = \mathbf{d}_{\mathcal{A}} + \mathbf{d}_{\mathcal{B}}$
    Encode($\mathbf{d}_{\mathcal{A}}$, $\frac{N}{2}$)
    Encode($\mathbf{d}_{\mathcal{B}}$, $\frac{N}{2}$)
end
Return
end function
```

have some significant computational and storage advantages as a single real number, especially in the logarithmic domain. We formally define the *likelihood ratio* and the *log-likelihood ratio* as below. Here $\ln(\cdot)$ represents the natural logarithm with base e.

$$L(y_i) \text{ or } L(\mathbf{p}^{(i)}) \triangleq \frac{\Pr(y_i | x_i = 0)}{\Pr(y_i | x_i = 1)} \qquad \text{(The \textit{Likelihood Ratio})} \tag{2.7}$$

$$l(y_i) \text{ or } l(\mathbf{p}^{(i)}) \triangleq \ln\left(\frac{\Pr(y_i | x_i = 0)}{\Pr(y_i | x_i = 1)}\right) \qquad \text{(The \textit{log-likelihood ratio})} \tag{2.8}$$

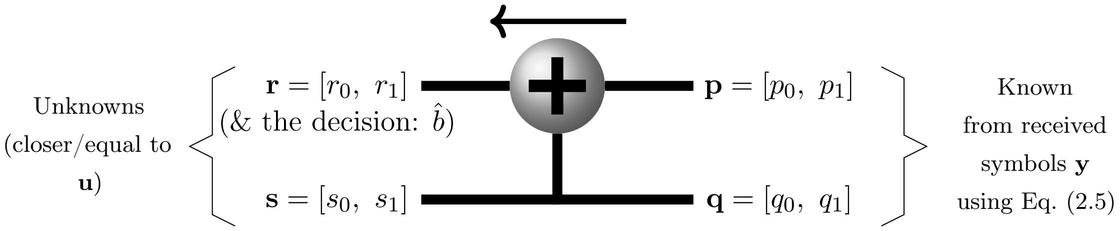
In summary, the N received symbols in $\mathbf{y}$ are statistically completely described by any one of the N APPs, or N likelihood ratios, or N log-likelihood ratios. Now, the successive cancellation decoder is simply an efficient approximate solution to the following general problem of bitwise maximum aposteriori probability (bitwise-MAP) computation.

The Decoding Problem

Given N likelihoods of bits in codeword $\mathbf{x}$, how can we efficiently compute the likelihoods of the original K message bits in $\mathbf{d}_{\mathcal{I}}$?

2.3.2 Fundamental Computational Block of SCD

Recall the encoding algorithm where the recursive application of a smaller operation produced an efficient encoding algorithm. Similarly, the SCD undertakes a basic block of similar size and proceeds as follows. Consider the below circuit with various APPs indicated.


Unknowns
(closer/equal to
$\mathbf{u}$)
$\mathbf{r} = [r_0,\ r_1]$
(& the decision: $\hat{b}$)
$\mathbf{s} = [s_0,\ s_1]$
$\mathbf{p} = [p_0,\ p_1]$
$\mathbf{q} = [q_0,\ q_1]$
Known
from received
symbols $\mathbf{y}$
using Eq. (2.5)

Step 1: Computation of $\mathbf{r}$: In the above case, the computation of the unknown APPs/likelihoods $\mathbf{r}$ is easily verified as follows.

$$\mathbf{r} = \big[p_0q_0 + p_1q_1,\ p_0q_1 + p_1q_0\big] \tag{2.9}$$

$$L(\mathbf{r}) = \frac{L(\mathbf{p})L(\mathbf{q}) + 1}{L(\mathbf{p}) + L(\mathbf{q})}$$

$$l(\mathbf{r}) = \ln\left(\frac{e^{l(\mathbf{p})+l(\mathbf{q})} + 1}{e^{l(\mathbf{p})} + e^{l(\mathbf{q})}}\right)$$

(this computation requires *log-sum-exp* technique [134]) (2.10)

$$\approx \operatorname{sign}(l(\mathbf{p})) \cdot \operatorname{sign}(l(\mathbf{q})) \cdot \min\big(l(\mathbf{p}), l(\mathbf{q})\big), \text{ where } \operatorname{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & else. \end{cases}$$

(It is a lossy, but efficient approximation)

However, the APP/likelihood computed above should not be used for making the corresponding bit decisions yet, except when the computed APP/likelihood values correspond to the message bits $\mathbf{u}$. In the other cases, the bit is rather computed using the message bits that are already decoded. In any case, a bit ($\hat{b}$) is made available alongside of the above APP/likelihood, and is essential for the computation of $\mathbf{s}$.

In case $\mathbf{r}$ represents one of the message bits $\mathbf{d}$, the ML decision is made as follows.

$$\mathbf{1}_{\{r_0 < r_1\}} = \begin{cases} 0, & \text{if } r_0 \geq r_1 \text{ or } L(\mathbf{r}) \geq 1 \text{ or } l(\mathbf{r}) \geq 0 \\ 1, & \text{otherwise.} \end{cases} \tag{2.11}$$

Step 2: Computation of $\mathbf{s}$: Given $\mathbf{p}$ and $\mathbf{q}$ alone, the optimal $\mathbf{s}$ is equal to $\mathbf{q}$. In other words, the knowledge of $\mathbf{p}$ does not help!

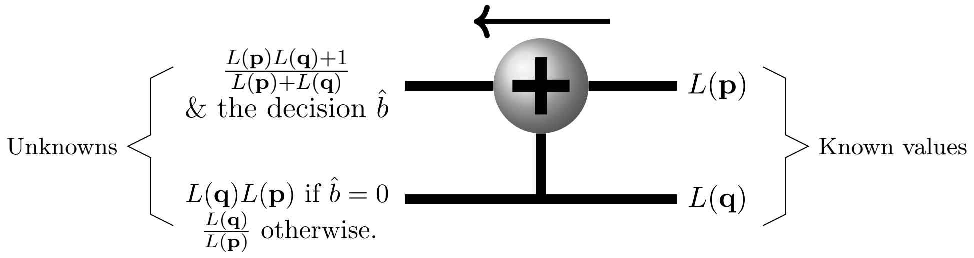

$\frac{L(\mathbf{p})L(\mathbf{q})+1}{L(\mathbf{p})+L(\mathbf{q})}$
& the decision $\hat{b}$
Unknowns
$L(\mathbf{q})L(\mathbf{p})$ if $\hat{b}=0$
$\frac{L(\mathbf{q})}{L(\mathbf{p})}$ otherwise.
$L(\mathbf{p})$
$L(\mathbf{q})$
Known values

Figure 2.3: Summary of basic operations of an SCD

For this case, Arıkan proposes the following, which is strikingly simple but effective. If the bit corresponding to the APP **r** (denoted $\hat{b}$) is made available accurately, either as a frozen bit or as a consequence of the earlier decisions, the knowledge of **p** can improve the estimation of the APP **s**. The corresponding improved computation of **s** is as follows. Note that in the degenerate case of frozen bits indicating a zero probability event i.e., ($\hat{b}=0$ and $p_0q_0+p_1q_1=0$), or ($\hat{b}=1$ and $p_0q_1+p_1q_0=0$), we choose to consider the result as equally likely.

$$\mathbf{s} = \begin{cases} \begin{cases} \left[\frac{p_0q_0}{p_0q_0+p_1q_1}, \frac{p_1q_1}{p_0q_0+p_1q_1}\right] & \text{if } \hat{b}=0 \text{ and } p_0q_0+p_1q_1 \neq 0 \\ [0.5, 0.5] \text{ if } \hat{b}=0 \text{ and } p_0q_0+p_1q_1=0, \end{cases} \\ \begin{cases} \left[\frac{p_1q_0}{p_1q_0+p_0q_1}, \frac{p_0q_1}{p_0q_1+p_1q_0}\right] & \text{if } \hat{b}=1 \text{ and } p_0q_1+p_1q_0 \neq 0 \\ [0.5, 0.5] \text{ if } \hat{b}=1 \text{ and } p_0q_1+p_1q_0=0, \end{cases} \end{cases} \tag{2.12}$$

$$\begin{gathered} L(\mathbf{s}) = \begin{cases} L(\mathbf{q}) \cdot L(\mathbf{p}), & \text{if } \hat{b}=0 \\ L(\mathbf{q})/L(\mathbf{p}), & \text{otherwise} \end{cases} \\ l(\mathbf{s}) = \begin{cases} l(\mathbf{q}) + l(\mathbf{p}) & \text{if } \hat{b}=0 \\ l(\mathbf{q}) - l(\mathbf{p}) & \text{otherwise} \end{cases} \end{gathered} \tag{2.13}$$

The following Fig. 2.3 will summarize the fundamental operations in an SCD using likelihood ratios.

A note on the use of log-domain computations

Due to the effect of polarization the dynamic range of all likelihood ratios tends to be quite large, and can cause an overflow even with the double-precision floating-point arithmetic. In addition, it is usually desirable for hardware platforms to be able to perform the computations in a fixed point arithmetic, which has much lesser dynamic range and precision.

Fortunately, by representing the likelihoods in logarithmic domain (i.e. storing $\log(x)$ instead of x) one can reduce the dynamic range considerably, without any significant loss of precision.

Therefore the log-domain computations become essential for both hardware and software platforms, especially when we work with polar codes of larger length $N \geq 512$.

2.3.3 The Main SCD Algorithm as a Recursion

In short, the SCD is simply a recursive application of a pair of simple operations (2.10) and (2.13) while propagating the decisions made on the message bits (2.11). These operations are to be performed in a specific and unique sequence greedily as described in [11]. This greedy order of computations is such that an operation is performed immediately whenever the necessary inputs are available. Once the computation of the likelihood of a message bit is reached, new decisions are made and propagated in the opposite direction thereby opening the opportunities for more number of operations.

In summary, the SCD is represented by the following basic pseudocode that combines Algorithm 2 with recursive functions **UpdateL** and **UpdateB**. It is also possible to unfold these routines into a non-recursive form, and reduce the memory and computational requirements of the decoder. For the reasons of better readability, we restrict ourselves to the recursive form alone. We remark that a special *decoding order* of message bits, namely, the *bit-reversal* order, naturally arises as there is a successive and greedy application of (2.10),(2.13), and (2.11). This ordering is unique for this decoder structure.

In the above implementation a majority of the memory is filled with the matrices $\mathbf{B}$ and $\mathbf{L}$ denoting the matrices of bits and likelihood ratios respectively. These are accessible to all three modules and make up for the $O(N \log N)$ space complexity of the decoder. The function **bitreversal**(i) represents the new index obtained by the reversal permutation of the bits in the n-bit binary representation of index i.

Note that the SCD algorithm essentially follows the same encoder diagram in Fig. 2.2, but in the reverse direction of encoding. The likelihood ratio values evolve in the reverse direction towards message bits $\mathbf{u}$, and can be mapped to a N binary-trees of computations with message bits at their root nodes.

With the objective of providing a simple description of the SCD, we have created a library and also a video tutorial [125].

2.3.4 Complexity of the SCD

It was estimated by Arıkan [11] that the SCD has a complexity of $\mathcal{O}(N \log N)$ operations. We find that we can actually count the exact number of various operations as follows. As a general practice, the complexity computation excludes the intermediate computations such as indices, and varies depending upon the implementation and the platform.

For this computation, assume $N(1 + \log_2 N)$ nodes in an encoding circuit as shown in Fig. 2.4 for $N = 8$. This will have N nodes at input and same number of output nodes after each of

$n = \log_2(N)$ stages, cascaded. Each node carries a likelihood ratio (LR) (or a log-likelihood ratio (LLR)) and a bit. We will first see how the likelihoods are computed and then the bits.

The likelihoods: We initialize the circuit with N likelihoods stored in nodes at the right extreme of the circuit, computed from the N received symbols. This requires N computations with equations varying based on the channel model. The likelihoods at the remaining the nodes are unknown. In short, only N out of $(N+1)\log_2 N$ likelihoods are known.

Depending on the position, the likelihood computation at any node requires either the calculation of (2.10) or (2.13). Moreover, we may easily verify that both the operations are used in an equal number. Hence we will need exactly $\frac{N}{2}\log_2 N$ number of (2.10) operations, and the same number of (2.10) operations by the end of an SCD.

The bits: When it comes to the bit portion of the nodes, we start with none of the nodes filled in. As the bit decisions are made one-by-one at the left extreme of the circuit (message bits) and propagated/broadcasted to the rest of the nodes in the circuit by performing computations following the encoder circuit. As an interesting consequence, the operations by the end of the SCD are no different than one encoding operation because N message bit decisions (including frozen bits) are precisely transformed to the coded bits at the right extreme with no additional computation. Hence we have $\frac{N}{2}\log_2 N$ XORs.

In summary, we have the following set of calculations essential for one SCD operation. This satisfies the Arıkan's prediction of $O(N\log N)$ complexity, but illustrates it more accurately.

$$\text{Initial likelihood computations} = N \tag{2.14}$$

$$\text{The number of uses of (2.10)} = \tfrac{N}{2}\log_2 N \tag{2.15}$$

$$\text{The number of uses of (2.13)} = \tfrac{N}{2}\log_2 N \tag{2.16}$$

$$\text{The number of binary XORs} = \tfrac{N}{2}\log_2 N \tag{2.17}$$

The N initial likelihood computations may be considered as a part of the necessary initialization of the algorithm and can often be ignored when complexity numbers are specified.

However, as noticed earlier the problem with SCD is the complex ordering but not the operations themselves. Achieving the above numbers assumes that one has either computed by-hand, the sequence of these operations or that he has a smart program that computes this sequence of operations. The latter is a slightly difficult problem and is essential for larger values of N at which polar codes have impressive performance. This is also essential for a software program or a hardware implementation of SCD. This has warranted several patented implementations targetting various platforms.

A number of SCD variants are proposed with the objective of further reducing the complexity and latency of SCD. However, implementing the original SCD with least amount of overheads is itself a difficult task. We will discuss one such efficient implementation of the SCD in Section 4.1.

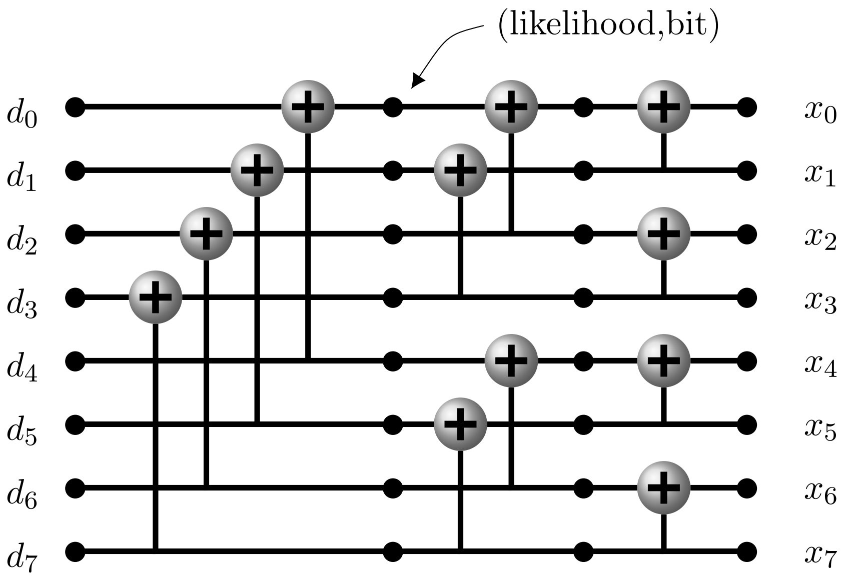


Figure 2.4: The encoding graph with nodes identified for use with SCD for $N = 8$

2.4 The Code Construction

The choice of the set $\mathcal{F}$ is an important step in polar coding often referred to as *polar code construction.* A significant amount of literature is devoted to this operation [11, 39, 101, 104, 105]. The original algorithm, proposed in [39] and improved later ([105] etc.), is based on the Bhattacharyya bound approximation. Others improve this approximation at the cost of higher complexity. For simplicity, we use the original polar code construction algorithm, which is given by the following recursion for $j = 0, \ldots, n-1$. For example, on an AWGN channel, its initial is chosen as the Bhattacharyya parameter of AWGN channel at a given E_b/N_0, namely, $z_{0,0} = \exp(-E_b/N_0)$ [39, 105], and is expected to optimize the code performance at that E_b/N_0.

$$z_{j+1,i} = \begin{cases} 2z_{j,i} - z_{j,i}^2 \ , & \text{if } 0 \leq i < 2^j \\ \\ z_{j,i-2^j}^2 \ , & \text{if } 2^j \leq i < 2^{j+1} \end{cases} \tag{2.18}$$

The indices of the highest $N - K$ values in the set of N final stage values $\{z_{n,i} : i = 0, 1, \ldots, N-1\}$, form the set $\mathcal{F}$. This algorithm is an evolution of the Bhattacharyya parameters of channels from right to left (see Fig. 2.2), preferable to be applied in log-domain to avoid underflow. The overall complexity of the construction is $\mathcal{O}(N)$ or more accurately $(N-2)$ uses of the equation (2.18). The two possible types of computations in (2.18) are used in equal number of times.

An illustrative description of the above algorithm is given as follows using Fig. 2.5. The recursion is actually similar to that of an SCD except for the lack of any decoding order. But the objective here is estimating the long-term performance of various bit channels, after many instances of decoding. So we start from the N identical channel instances represented with their

Algorithm 2 Main Module of SCD Algorithm

```
Input  : $(N, K, \mathcal{F}, \mathbf{v}_{\mathcal{F}} = 0)$ and received $\mathbf{y}$.
Output : $\hat{\mathbf{u}}$, the estimation of transmitted $\mathbf{u}$.
Allocate and make visible to the other functions $N \times (n+1)$ matrices $\mathbf{B}$, $\mathbf{L}$ and set them to NaN.
$n = \log_2 N$ and $l =$NaN
Initialize last column $\mathbf{L}[:][n] =$Pr$(\mathbf{y}|0)/$Pr$(\mathbf{y}|1)$ with likelihoods from channel observations $\mathbf{y}$
for $i = 0, 1, \dots, N-1$ do
    $l =$ bitreversal$(i)$
    UpdateL$(l, 0)$                                ▷ Update $\mathbf{L}[l][0]$ and $\mathbf{L}$
    if $l \in \mathcal{F}$ then
        $\mathbf{B}[l][0] = 0$
    else
        $\mathbf{B}[l][0] = \begin{cases} 0, & \text{if } \mathbf{L}[l][0] \geq 1 \\ 1, & \text{else} \end{cases}$
    end
    UpdateB$(l, 0)$                                ▷ Broadcast bit-$l$ & update $\mathbf{B}$
end
$\mathbf{d} = \mathbf{B}[:][0]$                    ▷ First column of matrix $\mathbf{B}$
$\hat{\mathbf{u}} = \mathbf{d}_{\mathcal{F}^c}$    ▷ Information bits
```

approximate probability of error (the Bhattacharyya parameter) as experienced by each bit of the codeword independently. We then proceed to estimate the performance of the message bit channels recursively, which exhibit a clear polarization. Upon finding the performance of all N possible bit channels in $\mathbf{d}$, we select those K channels with the least probability of error for transmitting the information and the rest of the channels for storing frozen bits.

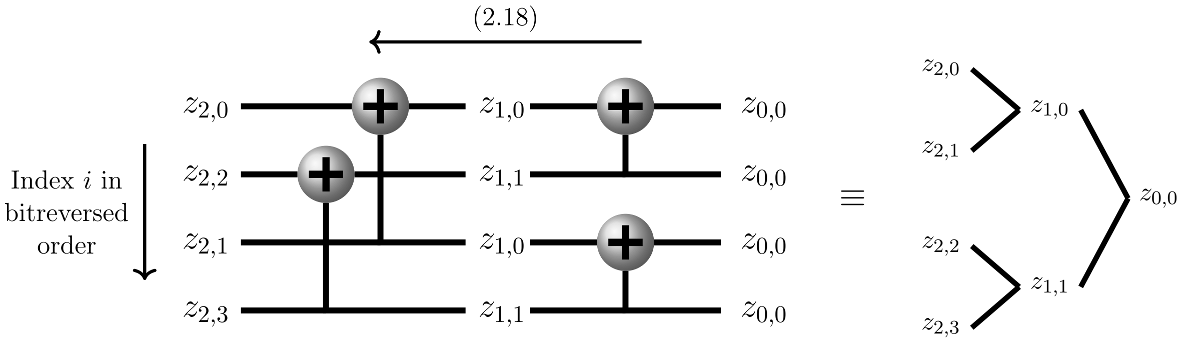


Figure 2.5: The polar code construction in (2.18) illustrated for $N = 4$

Function 3 UpdateL(i, j) : Recursive L.R. computation

```
Input  : Element indices $i, j$
Output : Recursively updated matrix $\mathbf{L}$
s = $2^{n-j}$
$l = (i \bmod s)$
if $l < s/2$ then                                        ▷ Upper branch
    if ($\mathbf{L}[i][j+1] =$ NaN ) then
        UpdateL($i, j+1$); end;
    if ($\mathbf{L}[i+s/2][j+1] =$ NaN ) then
        UpdateL($i+s/2, j+1$); end;
    $\mathbf{L}[i][j] = \frac{\mathbf{L}[i][j+1]\mathbf{L}[i+s/2][j+1]+1}{\mathbf{L}[i][j+1]+\mathbf{L}[i+s/2][j+1]}$
else                                                     ▷ Lower branch
    if $\mathbf{B}[i-s/2][j] = 0$ then
        $\mathbf{L}[i][j] = \mathbf{L}[i][j+1]\mathbf{L}[i-s/2][j+1]$
    else
        $\mathbf{L}[i][j] = \frac{\mathbf{L}[i][j+1]}{\mathbf{L}[i-s/2][j+1]}$
    end
end
```

Function 4 UpdateB(i, j) : Broadcasting of decisions

```
Input  : Element indices $i, j$
Output : Recursively updated matrix $\mathbf{B}$
$s = 2^{n-j}$
$l = (i \bmod s)$
if $l < s/2$ or $j \geq n$ then                          ▷ Upper branch
    return
else                                                     ▷ Lower branch
    $\mathbf{B}[i-s/2][j+1] = \mathbf{B}[i][j] \oplus \mathbf{B}[i-s/2][j]$
    $\mathbf{B}[i][j+1] = \mathbf{B}[i][j]$
    UpdateB($i, j+1$)
    UpdateB($i-s/2, j+1$)
end
```

Chapter 3

Efficient Encoders for Polar Codes

In this chapter, we see how to encode polar codes systematically. We briefly revisit the encoding of the original non-systematic polar codes and proceed to the discussion of systematic encoding. Our main objective will be to construct systematic encoders that are as efficient as the non-systematic encoding.

The following is a quick summary of the main contributions from this chapter:

1. We discuss the motivations and importance of systematic polar codes and formulate the problem of systematic encoding as a system of linear equations.
2. We show a non-recursive, non-systematic polar encoding implementation, which we find to be quite efficient both in memory and computations. This is used as a benchmark for our subsequent encoders.
3. We propose three efficient systematic encoders of the same complexity order as that of a non-systematic polar encoding, namely, EncoderA, EncoderB, EncoderC. Specifically, EncoderC uses exactly the same number of XORs as that of NSPE.

3.1 Introduction to Systematic Polar Codes

It is well known that every linear block code has an equivalent systematic form, where information bits appear in the codeword explicitly at some designated indices. Though polar codes are no exception to this general theory, efficient encoders are desirable, especially at longer block lengths. The current chapter precisely addresses the problem of designing efficient algorithms for systematically encoding the polar codes. We will start with a listing of advantages and then a brief review of existing literature on systematic polar codes, before we move to the next section.

In general, there are several advantages to using systematic linear codes against their non-systematic counterparts. The explicit visibility of the information bits within codewords allows the decoder to skip the high complexity decoding algorithms and directly decode the message bits using a very fast hard decision decoding algorithm whenever it desires to. There are several

---

Parts of this chapter are based on our published work [121] scenarios where such a need arises. For example, it might help the receiver address the demand for lower latency, or the demand for lower power consumption, or the event of a decoding failure. Also in the low SNR regime, the decoder performance might in fact be worse than decoding the uncoded bits and therefore it is better to skip the decoding algorithm entirely. And in high SNR regime, the system could be efficient enough that the receiver may choose to skip the high complexity, high latency decoding algorithm. Finally in the event of hardware failures, it may not be possible to perform full decoding. Systematic codes are also friendly to designing concatenated codes and the relevant variants of turbo codes. When it comes to systematic polar codes alone, we have the additional advantage that they have a much better BER performance (but same frame error rate) than the original non-systematic polar codes (NSPC).

Several other attempts were also made in literature towards proposing efficient algorithms for systematic polar encoding. In [119], a systematic polar encoder (SPE) was proposed based on an SCD over a BEC. In [33], another encoder was proposed using a cascade of two non-systematic polar encoder (NSPE) circuits. It was recently proved in [135] that such encoder works only under certain conditions. They also proved that if a given polar code does not satisfy these conditions they can transform it into a new polar code that performs at least as well as the original one. In [136, Section 3], following Arıkan's recursion, a systematic encoder was proposed for general $\ell \times \ell$ triangular kernels with a complexity $\mathcal{O}(N \log N)$ and a memory of $\mathcal{O}(N)$ bits. We note that all the above SPEs have complexity, memory, and latency higher than an NSPE.

In this chapter, the efficiency of an encoder is measured by the exact number of XOR operations and the exact memory required in number of bits for various computations (excluding the input/output).

3.1.1 A Revisit to Non-Systematic Polar Encoder (NSPE)

For convenience, throughout this chapter we consider vectors represented as rows instead of columns. This leads to the below equivalent definition of polar codes against what we saw earlier in Chapter 2. We will continue to use the column-based notation in the rest of the thesis.

As we saw earlier in Chapter 2, for an $(N, K, \mathcal{F})$ polar code, the generator matrix is $\mathbf{G} = (\mathbb{F}^{\otimes n})_{(\mathcal{I})}$. Therefore, given a message vector $\mathbf{u}$ of K information bits, a codeword $\mathbf{x}$ is generated as:

$$\mathbf{x} = \mathbf{u} \cdot \mathbf{G} = \mathbf{d} \cdot \mathbb{F}^{\otimes n} \tag{3.1}$$

where $\mathbf{d}$ is a vector of N bits with $\mathbf{d}_{\mathcal{I}} = \mathbf{u}$, and $\mathbf{d}_{\mathcal{F}} = \mathbf{0}$, where $\mathcal{I} \triangleq \mathcal{N} \backslash \mathcal{F}$, and $\mathcal{N} \triangleq \{0, 1, \ldots, N-1\}$ as defined earlier. The bits $\mathbf{d}_{\mathcal{F}}$ are called the *frozen bits* and are set to zero, and $\mathbf{d}_{\mathcal{I}}$ are information bits and are set to the message vector $\mathbf{u}$.

Due to the recursive construction of the matrix $\mathbb{F}^{\otimes n}$, we can perform this matrix-vector multiplication in $\Theta(N \log N)$ only, as explained earlier in Algorithm 1. This NSPE is illustrated

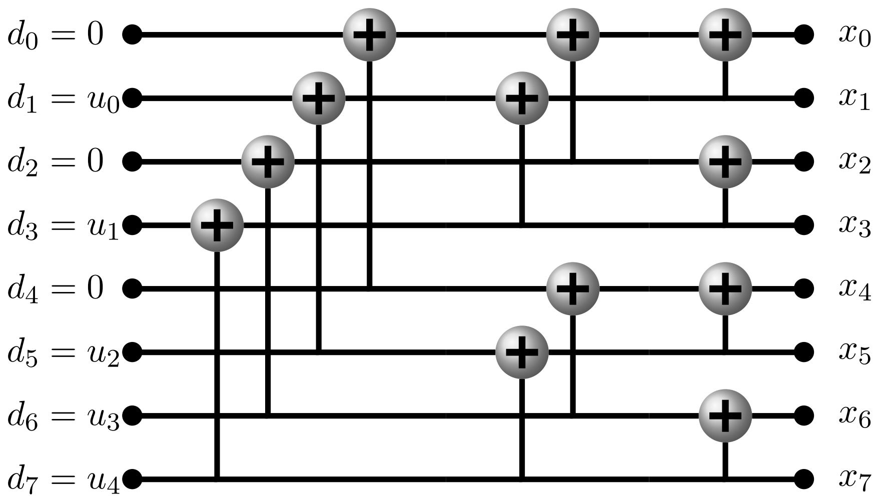


Figure 3.1: The non-systematic polar encoder (NSPE) with an exact complexity of $\left(\frac{N}{2}\log_2 N\right)$ XORs, when $(N, K, \mathcal{F}) = (8, 5, \{0, 2, 4\})$ (see (3.1))

in Fig. 3.1 and requires exactly $\left(\frac{N}{2}\log_2 N\right)$ XORs and $2N$ bits. This forms a benchmark for our new systematic encoders.

3.1.2 The Systematic Polar Encoding

A systematic polar code may be described as equivalent to the original polar code in (3.1), except that the message vectors are mapped to codewords, such that the message-bits are explicitly visible [119]. Consider the indices of K bits in a codeword $\mathbf{x}$, where the message bits appear explicitly. It was shown in [119] that this set can be chosen equal to the set of *information bit indices* $\mathcal{I}$. Note that, it slightly differs from what is usually considered as a systematic linear block code, where message bits appear as *first* K bits in a codeword. This motivates our interpretation of the SPE below.

The SPE of an information vector $\mathbf{u}$ of K bits, is the solution of the set of linear equations:

$$\mathbf{x} = \mathbf{y} \cdot \mathbb{F}^{\otimes n}, \text{ where } \mathbf{x}_{\mathcal{I}} = \mathbf{u}, \text{ and } \mathbf{y}_{\mathcal{I}^c} = 0 \tag{3.2}$$

where, $\mathbf{y}_{\mathcal{I}}$ and $\mathbf{x}_{\mathcal{I}^c}$ are the unknowns. Within the systematic codeword $\mathbf{x}$, $\mathbf{x}_{\mathcal{I}} = \mathbf{u}$ are information bits and $\mathbf{x}_{\mathcal{I}^c}$ are parity bits. Frozen bits are given by $\mathbf{y}_{\mathcal{I}^c} = 0$. For example, if $(N, K, \mathcal{I}) = (4, 2, \{0, 2\})$ and $\mathbf{u} = [1\ 0]$, then (3.2) becomes:

$$\begin{bmatrix}1, x_1, 0, x_3\end{bmatrix} = \begin{bmatrix}y_0, 0, y_2, 0\end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

where y_1, y_3 and x_2, x_4 are the unknowns. Note that there are exactly N unknowns, shared between $\mathbf{x}$ and $\mathbf{y}$. It is easy to see that (3.2) is a system of linear equations, up to a rearrangement.

Algorithm	**Recursiveness**	**No. of bits (excl. I/O)**	**No. of XORs**
EncoderA	No	$N(1+\log_2 N)$	$\frac{N}{2}\log_2 N$
EncoderB	Yes	$2N-1$	$N(1+\log_2 N)$
EncoderC	Yes	N	$N(1+2\log_2 N)$
NSPE (Fig. 3.1) (benchmark)	Yes/No	$2N$	$\frac{N}{2}\log_2 N$

Table 3.1: Summary of Systematic Polar Encoders

Since $\mathbb{F}^{\otimes n}$ is a triangular matrix, a straightforward *Gaussian elimination* (GE) could be used to solve the equations. Then the number of XORs required to perform GE would be approximately equal to the number of ones in the matrix: $3^{\log_2 N} = N^{\log_2 3} \approx N^{1.585}$. This is much higher than the achievable complexity of $\Theta(N \log N)$ XORs, as indicated by Arikan. In fact, devising efficient encoders achieving this complexity bound forms our core objective here.

3.2 New Systematic Polar Encoders

In this section, we propose three novel systematic polar encoders based on efficient methods of solving (3.2). Our first SPE requires the least number of XORs, exactly equal to that of an NSPE. Later, using the recursion in [119] we show two explicit SPEs, which recursively split the set of encoder equations in (3.2). For each of our encoders, we compute the exact number of XORs and memory required in bits (see Table 3.1).

The pseudocodes of all three encoders are provided at the end as **EncoderA**$(\mathbf{y}, \mathbf{x})$, **EncoderB**$(i, j, \mathbf{y}, \mathbf{x}, \mathbf{r})$, and **EncoderC**$(i, j, \mathbf{y}, \mathbf{x}, \mathbf{r})$. To perform the encoding of message vector $\mathbf{u}$, we need to initialize the arguments as: $i = 1$, $j = N$, $\mathbf{x}_{\mathcal{I}} = \mathbf{u}$, $\mathbf{x}_{\mathcal{I}^c} =$ undef, $\mathbf{y}_{\mathcal{I}} =$ undef, $\mathbf{y}_{\mathcal{I}^c} = \mathbf{0}$ and $\mathbf{r} = \mathbf{0}_{N\times 1}$. The values of the triple $(N, K, \mathcal{I})$ are implicitly available to all the routines. The solutions are reflected in the values of $\mathbf{x}$ and $\mathbf{y}$ after each function call, according to (3.2). Note that the MATLAB indexing convention (indices starting with one) is used in the pseudocode, unlike the *zero based indexing* (indices starting with zero [127]) used throughout this thesis.

3.3 Non-Recursive Systematic Polar Encoder - EncoderA

In the circuit implementation of NSPE in Fig. 3.1, bits evolve from left to right in n stages. Our **EncoderA** uses exactly the same circuit, except for a different order of computations and some intermediate memory elements.

We first see that in an SPE based on (3.2), all the N known bits and N unknown bits of (3.2) are distributed on the two extremes of the circuit shown in Fig. 3.2. The circuit has

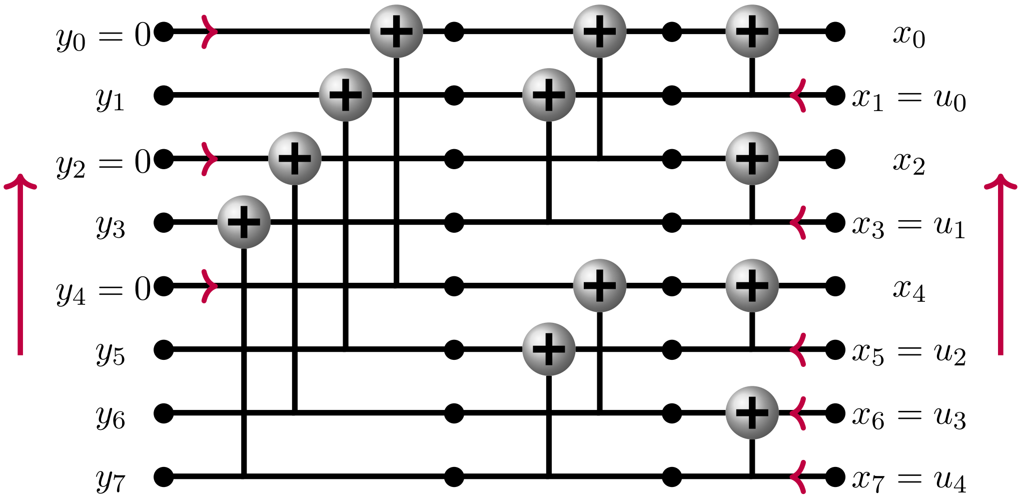


Figure 3.2: The flow-of-calculations required for the SPE of EncoderC on the encoder graph

$N(1 + \log_2 N)$ nodes, each storing a bit. Each horizontal connection holds $n + 1$ nodes. Further, on any horizontal connection, only 1 out of $n + 1$ nodes is known at one of the two extremes. Therefore we start calculations at the known node and move gradually towards the other extreme, calculating the node values one by one. An illustration of such order of computations is shown in Fig. 3.2. The key observation that justifies this computation is as follows. The computation of any of the $(n + 1)$ nodes on any given horizontal connection involves only the nodes from the same connection and the nodes below it. Also note that, the order of evaluation of the horizontal connections must be bottom-up. This is reminiscent of a backward substitution.

This procedure requires exactly the same number $\frac{N}{2} \log_2 N$ of XOR computations as an NSPE, and $N(1 + \log_2 N)$ bits (see the pseudocode of **EncoderA( )** and Table 3.1).

3.4 Recursive Systematic Polar Encoders - EncoderB and EncoderC

We revisit Arikan's recursive idea based on a divide-and-conquer iteration, which splits the set of equations (3.2) into two at each step. This enables us to design efficient encoders that trade the number of XORs with memory.

Consider a vector $\mathbf{z} = [z_0, \ldots, z_{N-1}]$ of N elements. We define a *level-l partition* of $\mathbf{z}$ as,

$$\mathbf{z} = [\mathbf{z}_{2^l}^{(l)}, \ldots, \mathbf{z}_2^{(l)}, \mathbf{z}_1^{(l)}], \quad 0 \le l \le n, \tag{3.3}$$

where, $\mathbf{z}_i^{(l)}, 1 \le i \le 2^l$ denotes a subvector of $\mathbf{z}$ of length 2^{n-l}, described as follows. Let $a_i = (2^l - i) \cdot 2^{n-l}$, $1 \le i \le 2^l$. Then, $\mathbf{z}_i^{(l)} \triangleq [z_{a_i}, \ldots, z_{a_i + 2^{n-l} - 1}]$. In other words, $\mathbf{z}_i^{(l)}$ is simply a pointer to the above sub-array of $\mathbf{z}$ (as in a programming language like C). The partition indices are given in reverse order so that they follow the order of evaluation in our encoders.

Such indices also simplify the traversal of a tree structure that we use later. Finally, we observe that $\mathbf{z}_i^{(n)} = z_{N-i}, 1 \leq i \leq N$ and $\mathbf{z}_1^{(0)} = \mathbf{z}$.

Let $\{\mathbf{r}_{l,i} : 0 \leq l \leq n, 1 \leq i \leq 2^l\}$ denote a collection of $2N-1$ different length binary vectors corresponding to the $2N-1$ nodes of a binary tree shown in Fig. 3.4. Let l be the level in the tree (or recursion index) and i be the subvector index, for the nodes at the same level in the tree. All the 2^l vectors at a given level l have the same length 2^{n-l} and can be concatenated as a single, length N vector. Although all these vectors add up to a total of $N(1+\log_2 N)$ bits, we will later show how to significantly reduce such memory requirement.

Recursion at level $l = 1$: Consider the level-1 partitioning: $\mathbf{x} = [\mathbf{x}_2^{(1)}, \mathbf{x}_1^{(1)}]$ and $\mathbf{y} = [\mathbf{y}_2^{(1)}, \mathbf{y}_1^{(1)}]$. Using the block decomposition $\mathbb{F}^{\otimes n}$, (3.2) can be rewritten as:

$$\mathbf{x}_1^{(1)} = \mathbf{y}_1^{(1)} \mathbb{F}^{\otimes n-1} \quad \text{and} \tag{3.4}$$

$$\mathbf{x}_2^{(1)} = \mathbf{y}_2^{(1)} \mathbb{F}^{\otimes n-1} + \mathbf{x}_1^{(1)} \tag{3.5}$$

Clearly, the set of equations in (3.4) can be independently solved (to obtain $\mathbf{x}_1^{(1)}$) and substituted in (3.5) to save one submatrix multiplication. Note that solving (3.5) is different from solving (3.4) due to the presence of a *binary offset* vector $\mathbf{x}_1^{(1)}$.

We will store these offset vectors as: $\mathbf{r}_{1,1} = \mathbf{0}$ and $\mathbf{r}_{1,2} = \mathbf{x}_1^{(1)}$.

Recursion at levels $2 \leq l \leq n$: In general, at level $(l-1)$, we find 2^{l-1} equations, relating the level $l-1$ partition of vectors of $\mathbf{x}$ and $\mathbf{y}$ as

$$\mathbf{x}_i^{(l-1)} = \mathbf{y}_i^{(l-1)} \mathbb{F}^{\otimes n-l+1} + \mathbf{r}_{l-1,i} \quad 1 \leq i \leq 2^{l-1}. \tag{3.6}$$

Then, at level $2 \leq l \leq n$, each of the above equations split similarly to (3.4) and (3.5) as follows

$$\mathbf{x}_{2i-1}^{(l)} = \mathbf{y}_{2i-1}^{(l)} \mathbb{F}^{\otimes n-l} + \mathbf{r}_{l,2i-1} \quad \text{and} \tag{3.7}$$

$$\mathbf{x}_{2i}^{(l)} = \mathbf{y}_{2i}^{(l)} \mathbb{F}^{\otimes n-l} + \mathbf{r}_{l,2i}, \tag{3.8}$$

where the new binary offsets $\{\mathbf{r}_{l,2i-1}, \mathbf{r}_{l,2i}\}$ at level l are computed as

$$\mathbf{r}_{l,2i-1} = (\mathbf{r}_{l-1,i})_1^{(1)} \tag{3.9}$$

$$\mathbf{r}_{l,2i} = (\mathbf{r}_{l-1,i})_1^{(1)} + (\mathbf{r}_{l-1,i})_2^{(1)} + \mathbf{x}_{2i-1}^{(l)}, \tag{3.10}$$

where the two halves of $\mathbf{r}_{l-1,i}$ are denoted by $(\mathbf{r}_{l-1,i})_1^{(1)}$ and $(\mathbf{r}_{l-1,i})_2^{(1)}$, using a similar notation to (3.3). Note that (3.10) can be applied only after solving (3.7), since the solution $\mathbf{x}_{2i-1}^{(l)}$ is required. We will next see a more convenient visualization of these recursive computations.

Binary tree representation — Consider a perfect binary tree with $n+1$ levels $l = 0, \ldots, n$, with root node at level 0. The 2^{l-1} nodes at a level $l-1$ are associated with the 2^{l-1} set of 2^{n-l+1} equations from (3.6). Each node at level $l-1$ splits into two at level l as shown in Fig. 3.3. The solution of (3.2) is obtained by an *inorder* traversal of the tree [137].

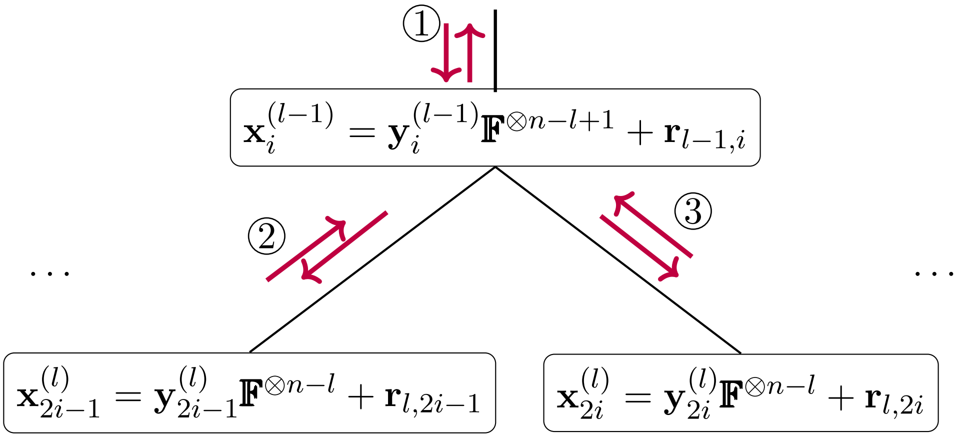

①
$\mathbf{x}_i^{(l-1)} = \mathbf{y}_i^{(l-1)}\mathbb{F}^{\otimes n-l+1} + \mathbf{r}_{l-1,i}$
②
③
. . .
. . .
$\mathbf{x}_{2i-1}^{(l)} = \mathbf{y}_{2i-1}^{(l)}\mathbb{F}^{\otimes n-l} + \mathbf{r}_{l,2i-1}$
$\mathbf{x}_{2i}^{(l)} = \mathbf{y}_{2i}^{(l)}\mathbb{F}^{\otimes n-l} + \mathbf{r}_{l,2i}$

Figure 3.3: The tree of recursive computations

Each time we visit a leaf node (i.e., at $l = n$), we obtain an equation of the form: $\mathbf{x}_i^{(n)} = \mathbf{y}_i^{(n)} + \mathbf{r}_{n,i}$, where either $\mathbf{x}_i^{(n)}$ or $\mathbf{y}_i^{(n)}$ is the unknown to solve for. Computing $\mathbf{r}_{n,i}$ using the successive updates of (3.9) and (3.10), is therefore the objective of this traversal. The binary offsets $\mathbf{r}_{l,i}$ are stored in a tree structure in Fig. 3.4 as explained before.

As a part of the in-order traversal of the tree in Fig. 3.3 (or Fig. 3.4), we visit each node three times, except for the leaf nodes which are visited only once.

Let us consider the sequence of visits and respective operations for a node at level $l - 1$ in the tree as shown in Figs. 3.3 and 3.4.

1. *First Visit* ① The required offsets $\mathbf{r}_{l,2i-1}$ are simply copied using (3.9). Then a recursive call down the left branch is made which, upon return, gives the solution of (3.7).
2. *Second Visit* ② Returning from the left subtree we have knowledge of $\mathbf{x}_{2i-1}^{(l)}$, so that $\mathbf{r}_{l,2i}$ can be updated by using (3.10). Then a recursive call down the right branch is made which, upon return, gives the solution of (3.8).
3. *Third Visit* ③ By now we have visited the entire subtree and we have the all solutions of (3.6). Then we return to the parent node.

The exact XOR count: Let $f(N)$ denote the number of XORs required to solve (3.2). From the recursive equations (3.7)–(3.10) we have:

$$f(N) = 2 \cdot f(\tfrac{N}{2}) + \beta\tfrac{N}{2} \text{ and } f(1) = 1. \tag{3.11}$$

The first term on the right hand side accounts for solving recursively (3.7) and (3.8), while the second term $\beta\frac{N}{2}$ accounts for the computations required by (3.9) and (3.10). In the case where we have allocated memory for each node of the tree in Fig. 3.4 (namely, **EncoderB**), we have $\beta = 2$. A different implementation (namely, **EncoderC**), saving on the memory to store the intermediate values of the $\mathbf{r}_{l,i}$ offset vectors will have $\beta = 4$. The closed form solution of (3.11) yields $f(N) = N + \frac{\beta N}{2}\log_2 N$.

Note that we ignore all the index computations to address the subvectors and any operation

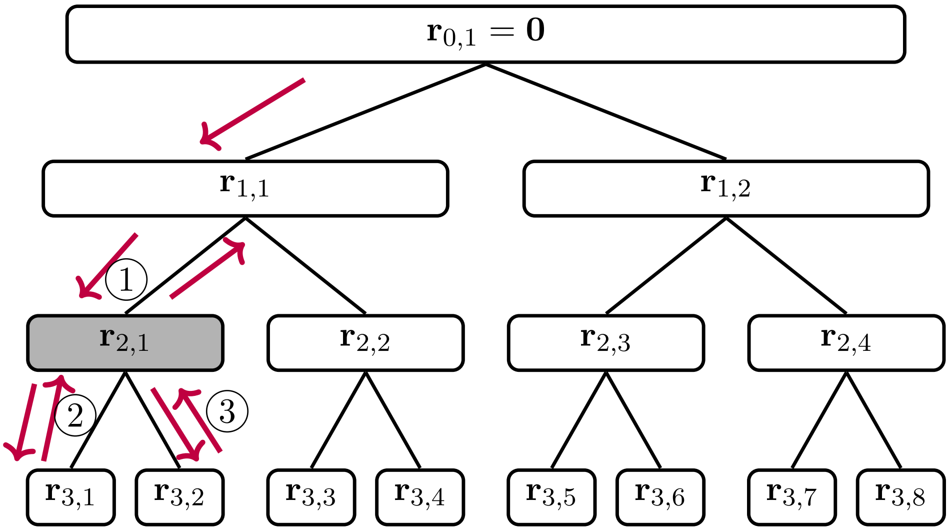


Figure 3.4: Traversal of nodes in recursion, at $N = 8$

to identify frozen bits that are implementation dependent and can be avoided by hardwiring.

EncoderB requires exactly $N(1 + \log_2 N)$ XORs, which is approximately the same number of XORs required by the encoder proposed in [33]. However, our **EncoderB** works without any restrictions on code rate and frozen bit indices. Our pseudocode **EncoderB( )** implements the storage of offset vectors as a local variable $\mathbf{v}$ allocated upon each recursive call and released on return. At any point in the tree traversal, we have a chain of at most $n + 1$ recursive function calls being active (a path from the root node to a leaf node in Fig. 3.4). Therefore, the maximum size of allocated memory by these function calls is exactly $N + \frac{N}{2} + \ldots + 1 = 2N - 1$ bits. Note that the same memory allocation/deallocation process can be implemented by simply addressing an array of $2N - 1$ bits.

EncoderC provides a memory-efficient implementation by using a single N-bit vector, since at any level l of Fig. 3.4, we need to store N bits of binary offsets $\mathbf{r}_{l,i}$. However, this slightly increases the number of XORs due to the following reason. We update the offsets every time we visit a non-leaf node for the second time. Therefore when we visit it for the third time, we should restore the original offset to enable computations at the parent node (see Fig. 3.4). This requires using twice equation (3.10). Hence doubling the number of XORs yields $\beta = 4$, and leads to an overall $N(1 + 2\log_2 N)$ XOR count, with only N bits memory (excl. I/O). **EncoderC**( ) provides its pseudocode.

3.5 An Efficient Non-Recursive NSPE

We have seen in Chapter 2 that the polar encoding can be performed using a simple recursive algorithm, given by Arıkan [34]. However, it is usually preferable to use a non-recursive algorithm whenever possible, and is often faster than its recursive counterparts. The non-recursive algorithm can simply be an unrolled version of the recursive algorithm, an optimized version of the unrolled version, or a completely new algorithm.

In the context of standard NSPE, we propose the following non-recursive algorithm independently as Algorithm 5. We find a significant speed improvement in our implementations with our non-recursive encoding algorithm. Note that such an improvement does not refer to the any change in the complexity order. In fact both require exactly the same number of XOR operations. The improvement indicates inefficiency of the recursive algorithms in general, on most platforms. Most probably this is due to their additional overheads such as the need to use a stack for implementing a recursion. On the other hand a flat non-recursive implementation enjoys having no such overheads.

3.6 Summary

We have designed three efficient, low complexity algorithms to perform systematic polar encoding, at the same complexity order of $\Theta(N \log N)$. All the three algorithms work for any arbitrary choice of frozen bit indices (not necessarily polar). They further illustrate a trade-off between the required number of XORs and the memory required for an SPE. We benchmark our encoders with a standard NSPE as detailed in Table 3.1. Our first encoder (**EncoderA**) is a non-recursive encoder that requires the least number of XORs, exactly equal to that of an NSPE. The remaining two encoders (**EncoderB** and **EncoderC**) are recursive. The first recursive encoder **EncoderB** requires approx. four times the XOR operations as an NSPE. Our last, recursive encoder **EncoderC** requires approx. twice as many XORs as an NSPE. Further improvements such as lower memory and a higher parallelization are interesting directions for our future work.

Finally we have also proposed a non-recursive, and efficient implementation of the NSPE (Algorithm 5) which has exactly the same complexity as the recursive algorithm presented in Algorithm 1. It is found to be faster than the recursive implementation (which is commonly used in the literature) mainly due to the lack of recursive overheads in general.

Algorithm 5 The Non-Recursive Polar Encoding

```
Input  : $(N, K, \mathcal{F}, \mathbf{v}_{\mathcal{F}} = 0)$ and $K$ message bits $\mathbf{u}$.
Output : $\mathbf{x}$, the codeword as the encoded message $\mathbf{u}$.
Allocate $N$ bits as vector $\mathbf{x}$
Set $\mathbf{x}_{\mathcal{I}} = \mathbf{u}$ and $\mathbf{x}_{\mathcal{F}} = \mathbf{v}_{\mathcal{F}}$    △ Preparation of the $\mathbf{d}$ in (2.1)
$n \triangleq \log_2(N)$
for $s = 0, 1, \ldots, n-1$ do
    $T = 2^s$, and $B = 2^{n-s}$    △ #sub-blocks and sub-block size
    for $b = 0, 1, \ldots, T-1$ do
        for $c = 0, 1, \ldots, \frac{B}{2} - 1$ do
            $\mathbf{x}[b * B + c] = \mathbf{x}[b * B + c] + \mathbf{x}[b * B + \frac{B}{2} + c]$
        end
    end
end
Return $\mathbf{x}$.
```

Function 6 EncoderA$(\mathbf{y}, \mathbf{x})$

```
Input  : $(N, K, \mathcal{F})$, $\mathbf{y}$, $\mathbf{x}$ with unfilled bits (variables of (3.2)).
Output : Updated vectors $\mathbf{y}$ and $\mathbf{x}$, with solutions of (3.2) filled in.
$n \triangleq \log_2(N)$ and $\mathbf{X}$ is an $N \times (n+1)$ matrix ▷ $\Theta(N \log N)$ bits
Set: $\mathbf{X}[:][1] = \mathbf{y}$ and $\mathbf{X}[:][n+1] = \mathbf{x}$    ▷ First & last columns
for $i = N, N-1 \ldots, 1$ do
    if $i \in \mathcal{I}$ then $s = n+1$; $\delta = -1$;
    else $s = 1$; $\delta = 1$; end
    Let the binary representation: $i = b_1 b_2 \ldots b_n$, with $b_1$ as MSB
    for $j = 1, 2, \ldots, n$ do
        $t = s + j\delta$; $\kappa = 2^{n-t+\delta}$
        if $b_t = 1$ then
            $\mathbf{X}[i][t] = \mathbf{X}[i][t-\delta] \oplus \mathbf{X}[i+\kappa][t-\delta]$
        else
            $\mathbf{X}[i][t] = \mathbf{X}[i][t-\delta]$
        end
    end
end
$\mathbf{y} = \mathbf{X}[:][1]$ & $\mathbf{x} = \mathbf{X}[:][n+1]$  ▷ Solutions in first & last columns
```

Function 7 $\mathbf{v} =$ EncoderB$(i, j, \mathbf{y}, \mathbf{x}, \mathbf{r})$

```
Input  : $(N, K, \mathcal{F})$ subvectors of $\mathbf{y}, \mathbf{x}$ formed by bits from index $i$ to $j$ and offset
         bits from a vector $\mathbf{r}$ of $L = j - i + 1$ elements;
Output : Updated $\mathbf{y}$, $\mathbf{x}$ and a returned vector $\mathbf{v}$ of same size as $\mathbf{r}$ represents a
         useful intermediate computation

$m \triangleq \lfloor \frac{i+j}{2} \rfloor$ and $L \triangleq (j - i + 1)$
Initialize: $\mathbf{v}$ — an $L \times 1$ vector                $\triangleright$ $2N - 1$ bits memory
if $L = 1$ then
    if $i \in \mathcal{I}$ then: $\mathbf{y}[i] = \mathbf{x}[i] \oplus \mathbf{r}[1]$ and $\mathbf{v}[1] = \mathbf{y}[i]$
    else:               $\mathbf{x}[i] = \mathbf{y}[i] \oplus \mathbf{r}[1]$ and $\mathbf{v}[1] = \mathbf{y}[i]$
else
    Denote the half-partitions: $\mathbf{v} \triangleq \left[\begin{smallmatrix}\mathbf{v_2}\\ \mathbf{v_1}\end{smallmatrix}\right]$ and $\mathbf{r} \triangleq \left[\begin{smallmatrix}\mathbf{r_2}\\ \mathbf{r_1}\end{smallmatrix}\right]$
    $\mathbf{v_1} = \text{EncoderB}\big(m+1, j, \mathbf{y}, \mathbf{x}, \mathbf{r_1}\big)$
            $\triangleright$ $\mathbf{v_1} \leftarrow \mathbf{x}^{(l)}_{2t-1} + (\mathbf{r}_{l-1,t})^{(1)}_1$ in (3.6), for some $t$ and $l$
    $\mathbf{v_2} = \mathbf{r_2} \oplus \mathbf{v_1}$          $\triangleright$ $\mathbf{v_2} \leftarrow \mathbf{r}_{l,2t}$ in (3.8) using (3.10)     $\boxed{\frac{L}{2}\ \text{XORs}}$
    $\mathbf{v_2} = \text{EncoderB}\big(i, m, \mathbf{y}, \mathbf{x}, \mathbf{v_2}\big)$    $\triangleright$ $\mathbf{v_2} \leftarrow (\mathbf{x}^{(l)}_{2t} + \mathbf{r}_{l,2t})$ in (3.8)
    $\mathbf{v_2} = \mathbf{v_2} \oplus \mathbf{v_1}$                                                 $\boxed{\frac{L}{2}\ \text{XORs}}$
            $\triangleright$ $\mathbf{v_2} \leftarrow \mathbf{x}^{(l)}_{2t} + (\mathbf{r}_{l-1,t})^{(1)}_2$ in (3.6)
end
Return $\mathbf{v}$        $\triangleright$ returns $(\mathbf{x}[i:j] + \left[\begin{smallmatrix}\mathbf{r_1}\\ \mathbf{r_2}\end{smallmatrix}\right]) = \mathbf{x}^{(l-1)}_t + \mathbf{r}_{l-1,t}$ in (3.6)
```

Function 8 EncoderC$(i, j, \mathbf{y}, \mathbf{x}, \mathbf{r})$

```
Input  : $(N, K, \mathcal{F})$, Subvectors of $N \times 1$ size vectors $\mathbf{y}, \mathbf{x}$ and $\mathbf{r}$ formed by the
         consecutive $L = j - i + 1$ bits from $i$ to $j$;
Output : Updated $\mathbf{y}, \mathbf{x}$ and $\mathbf{r}$

if $i = j$ then
    if $i \in \mathcal{I}$ then $\mathbf{y}[i] = \mathbf{x}[i] \oplus \mathbf{r}[i]$
    else: $\mathbf{x}[i] = \mathbf{y}[i] \oplus \mathbf{r}[i]$
else
    $m \triangleq \lfloor \frac{i+j}{2} \rfloor$
    EncoderC$(m+1, j, \mathbf{y}, \mathbf{x}, \mathbf{r})$
    $\mathbf{r}[i:m] = \mathbf{r}[i:m] \oplus \mathbf{r}[m+1:j] \oplus \mathbf{x}[m+1:j]$          $\boxed{L\ \text{XORs}}$
    EncoderC$(i, m, \mathbf{y}, \mathbf{x}, \mathbf{r})$
    $\mathbf{r}[i:m] = \mathbf{r}[i:m] \oplus \mathbf{r}[m+1:j] \oplus \mathbf{x}[m+1:j]$          $\boxed{L\ \text{XORs}}$
end
```

Chapter 4

Efficient Decoders for Polar Codes

In this chapter, we see several interesting variants of the classic SCD, in order to improve their performance in terms of their latency or hardware requirements.

The following is a quick summary of the main contributions from this chapter:

1. We first provide a non-recursive SCD implementation and its pseudocode, which we find to be quite efficient both in memory and computations. This is useful as a benchmark for all our subsequent decoders. We also provide a significantly faster specialization to SCD on binary erasure channels, by proposing three 3×3 lookup tables instead of the usual likelihood arithmetic of relatively higher complexity.
2. We propose a new *permuted successive cancellation decoder* (PSCD) which has a natural decoding order, and has exactly the same performance of the original SCD whenever the code construction is tailored to the PSCD.
3. We propose a new *improved multiple folded successive cancellation decoder* (IMFSCD), which reduces the latency significantly based the folding level. This also establishes a trade-off between the latency and the hardware resources by reuse.
4. We propose a new *parallelized list successive cancellation decoder* that has its latency brought close to that of an SCD, by using a parallelization.

4.1 Efficient Successive Cancellation Decoders

Recall the complexity discussion of SCD in Section 2.3.4, where we have seen that the SCD has a low complexity of $\mathcal{O}(N \log N)$ operations. Finding more efficient implementations is still an active topic of research. Note also that an SCD forms the basis for many of the advanced decoders such as LSCD.

The basic idea of the most of the currently available efficient implementations is the path-breaking idea simultaneously introduced by several researchers [53, 94, 141] that the essential space requirement for an SCD is $2N-1$ nodes only (much less than $N(1+\log_2 N)$ nodes that

---

Parts of this chapter are based on [138], and our published works [126, 139, 140] we found earlier). These $2N-1$ nodes form a binary tree of depth equal to $\log_2 N$ levels, with the root node containing a message/frozen bit.

The complete set of iterations of an SCD at $N = 8$ are illustrated in the next subsection. The below observations will follow, describing the overall operation of the SCD itself. Recall from Chapter 2 that the objective of the SCD is precisely to fill-in all the unknown likelihoods and bits in the circuit, by the end of the decoding, including that of message/frozen bits.

1. There are N iterations that compute the likelihoods of various message bits and frozen bits using N distinct computational trees that share $N(1+\log_2 N)$ nodes, $2N-1$ nodes at a time. No two of N trees are the same, though some of the levels might be common.
2. As iterations progress, the indices of the desired message/frozen bit likelihoods follow bit-reversed ordering.
3. Each of the N computational trees aims to compute the likelihood at the root node, by a sequence of computations using a combination of (2.10) and (2.13).
4. Each of the N computational trees has same $\log_2 N$ level depth, however the active computations might start at any level above to the leaves. This is because the remaining lower levels have already been completed in earlier iterations. This is key in reducing the complexity of the SCD from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\log_2 N)$ (See [11]). The *active-depth* simply denotes the number of active levels in the tree.
5. As illustrated in Sections 2.3.2 and 2.3.3, a bit is computed in every iteration by using an ML decision (2.11) on the likelihood computed at the root node. This bit combined with the similar bits from earlier iterations are greedily propagated to the other end as deep as possible using XOR operations on the available bits thus far. This is often referred to as the *broadcasting* of bit decisions.
6. A bit at any node is available only after its likelihood available. (The nodes in Section 4.1.1 are shown in gray color immediately after its bit is available)
7. At any iteration, we require at most $(2N-1)$ nodes that exist on the computational tree. By storing these nodes alone, we can significantly reduce the space requirements from $N(1+\log_2 N)$ nodes to just $2N-1$ nodes (plus N output bits).
8. The broadcasting of the bit decision in the last iteration will fill a full binary tree of $2N-1$ nodes, however this can be skipped due to no subsequent iterations to make use of this update.
9. *Uniform structure in any active tree:* Consider any computational tree of the given active-depth t. Such a portion of the tree of fewer levels will always have (i) all likelihood operations the same in any level and (ii) all but the first level of computations will be using (2.10), while the first level at the bottom of the active tree uses (2.13) for all its computations.
10. By virtue of the chosen order of operations in an SCD, the required bits for using (2.13) at

the first level of computations is always available from the earlier iterations.

11. The nature of computations in various levels of any full computational tree (not just the active portion) can be inferred directly from the n-bit binary representation of the index of the root node. Associate the MSB with the level involving the root node, and so on until the LSB being associated with the level that involves leaf nodes. Now a 0 represents the use of (2.10) and 1 represents the use of (2.13) in the respective levels. This observation also leads to the active-depth computation that we will see in Section 4.1.2.

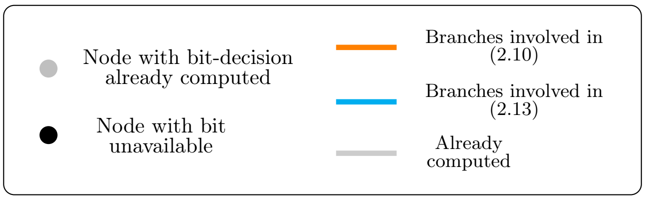
By having the above computational tree based implementation, it will also be easy to identify the right order of computations unlike the earlier recursive algorithm. This finally leads to a more efficient non-recursive algorithm which more accurately achieves the optimal complexity numbers.

With SCD, it is always much easier to explain these computations by showing appropriate diagrams. Therefore in Section 4.1.1, we will consider a diagram at each iteration which will explicitly illustrate the computational details. These iterations are independent of the rate of the code.

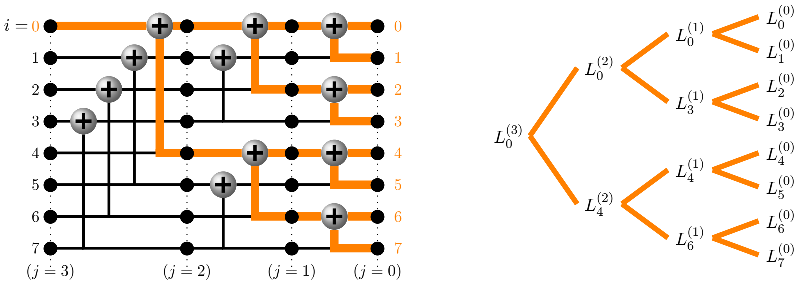
4.1.1 All SCD Iterations Illustrated for the Case $N = 8$

Consider the following legend to read the figures illustrating the 8 iterations of the SCD at $N = 8$ in exact sequence. All the figures essentially discuss the computation of nodes carrying an unknown likelihoods $L_i^{(j)}$ and a bit-decision. Here, i is the row index and j is the column index in the matrix of $N(\log_2 N + 1)$ nodes.

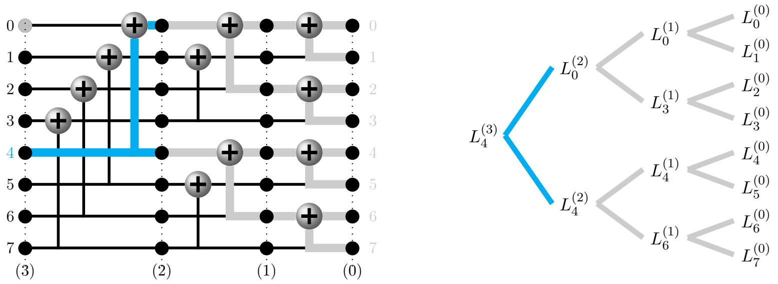
Legend for the next 8 figures


Node with bit-decision already computed
Node with bit unavailable
Branches involved in (2.10)
Branches involved in (2.13)
Already computed

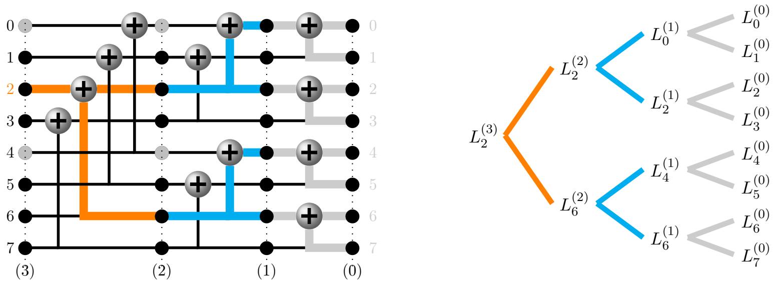
Iteration-0: Root node at index=0 (bit-reversed order)



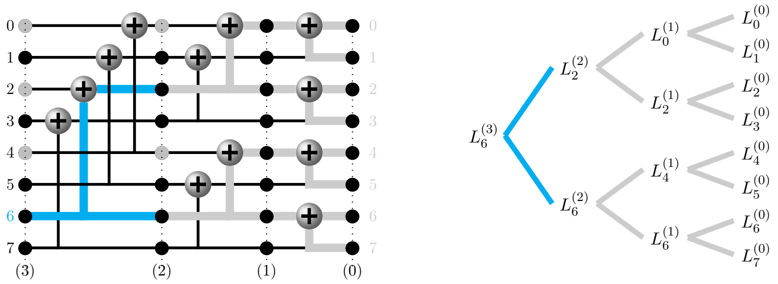
Iteration-1: Root node at index=4 (bit-reversed order)



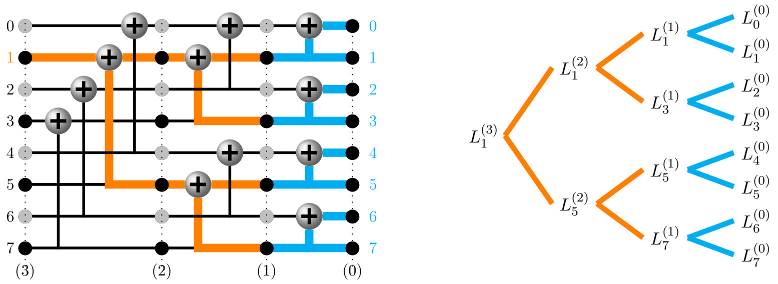
Iteration-2: Root node at index=2 (bit-reversed order)



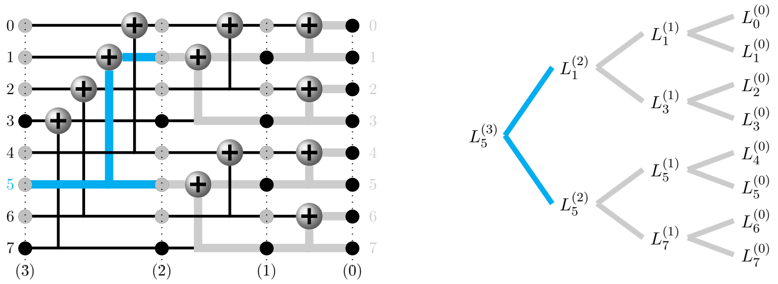
Iteration-3: Root node at index=6 (bit-reversed order)



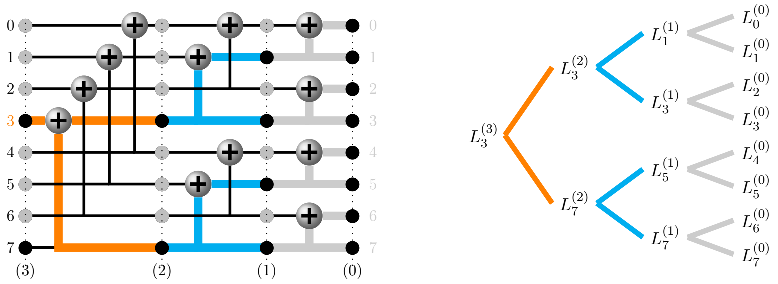
Iteration-4: Root node at index=1 (bit-reversed order)



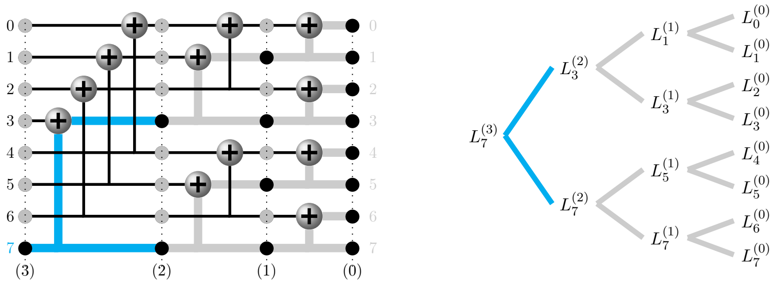
Iteration-5: Root node at index=5 (bit-reversed order)



Iteration-6: Root at index=3 (bit-reversed order)



Iteration-7: Root at index=7 (bit-reversed order)



4.1.2 Computing the Active-Depth of the Computational Tree

One may quickly identify that a critical parameter to start following the respective operations on any computational tree is the *active-depth* at which the computations should start. A naive way is to always start from the leaf nodes but it is prohibitive for its quadratic complexity order. Therefore knowing the level of the tree at which the computations should should start is essential.

Fortunately, this has a very simple closed-form expression as follows. No analogous formula was reported earlier in literature to the best of our knowledge. Let i be the index of the root-node.

$$\text{active-depth}(i) = \begin{cases} \log_2 N - \min\left\{j \geq 0 : 2^j > i\right\} + 1, & \text{if } i \neq 0 \\ \log_2 N, & \text{else.} \end{cases} \tag{4.1}$$

Alternatively it has the following simpler description. Let i have a binary representation $b_1 b_2 \ldots b_n$ with b_1 as MSB. The depth of the active tree is simply the index of the first non-zero bit read from MSB. When all bits are zero ($i = 0$), the depth is set to the maximum possible depth that is $\log_2 N$ levels.

4.2 Simplification of SCD on a Binary Erasure Channel

4.2.1 Motivation

The SCD is simply a sequential application of two real valued operators on N initial likelihoods, sequentially producing N new likelihoods. Any quest for further reducing the complexity should either reduce the number of uses or the speed of each operation or both.

Relatively fewer research is available for simplifying and increasing the speed of the two operations. A justifying argument could be that any high complexity operation can be reduced to probably the fastest possible operation of a simple *lookup* in a table of function values up to desired accuracy. However, lookup tables may not always be feasible and simplifications are always preferred whenever they are available. One particularly popular simplification that we already found is in the two last equations of (2.10) and (2.13), which eliminated the use of any logarithms or exponentials and used only the addition/subtraction and the minimum/maximum operations. Such simplifications are especially critical for high accuracy software/hardware implementations because higher accuracy requires unreasonably high space for lookup tables.

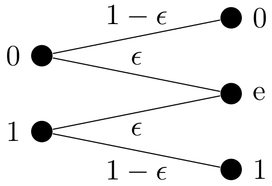


Figure 4.1: A binary erasure channel (BEC) with erasure probability ϵ

4.2.2 The Binary Erasure Channel

BEC is a channel with three possible outputs as shown in Fig. 4.1. Such a BEC has always been special for polar codes ever since Arıkan's invention for its special channel polarization characteristics such as follows [11, Sec.III]. Recall that the Bhattacharyya parameter (or the Z-parameter) of a BEC is equal to the erasure probability ϵ.

1. BEC has an extremal behavior of polarization (after using of an Arıkan's kernel), producing the channels with reliability among all BI-DMC channels.
2. The bounds on the Z-parameters (or the Bhattacharyya parameters) and the values of the mutual information of the bit channels after each kernel satisfy with *equality*. This is equivalent to the above statement.
3. After each polarization step, both the new channels remain to be BEC with the erasure probabilities equal to the above Z-parameters.
4. All the bit channels as defined by Arıkan (with exact side information of earlier decoded bits) are binary erasure channels. This is a consequence of the above plus the fact that Arıkan's bit channels are a result of n successive polarization steps.

In addition to the above specialities, BEC also enjoys an extensive and exclusive use in applications such as storage. Any specialized improvements to SCD in a BEC will therefore be significantly important. We will now present a novel simplification to the likelihood operations, similar to the LLR approximations (2.10) and (2.13) in an AWGN channel. As we will see, the proposed simplifications require only three 3×3 look-up tables altogether for likelihood computations.

4.2.3 Simplifications

Our simplifications are based on the below mentioned observations. Especially the second observation may be viewed as a consequence of the last two special characteristics of a BEC listed above.

1. The likelihood of any output symbol can be any of the three values, namely $\big\{0, 1, \infty\big\}$. The possibility of ∞ invalidates the likelihood based equations in (2.10) and (2.13) and therefore Eqs. (2.9) and (2.12) involving the aposteriori probabilities (APP) must be used instead.
2. The APP of the initial BEC can be one of $\big[1, 0\big]$, $\big[0.5, 0.5\big]$, $\big[0, 1\big]$. More interestingly, the APP of the polarized channels after any polarization transformation is again one of the above three.
3. As a result of the above observation, the likelihood computation operation can be reduced to simply mapping one APP to the other, which always will be one of the above three values. This is unlike any other BI-DMC where the APP can have infinitely many possibilities, hence it is a drastic reduction.

Finally the simplifications may be summarized as follows. Consider the three possible likelihood APP $\big[1, 0\big]$, $\big[0, 1\big]$, $\big[0.5, 0.5\big]$. Now the likelihood transformations will be applied on any two APP using Eqs. (2.9) and (2.12) and results may be listed as a table shown in Table 4.1. In an alternative notation, we may use the same original received symbols in the tables instead of their respective APP. This will make the computations look like a simple ternary operation. In this case, one may view the overall SCD as two algebraic field operations going in parallel in opposite directions, namely, the likelihood updates in ternary alphabet $\{0, 1, e\}$, and the broadcasting of binary decisions.

4.2.4 Summary

We have operations (2.10) and (2.13) converted into three small 3×3 look-up tables, for use with SCD in a BEC. We noticed about 50% speed up per decoding in our simulations when compared to the use of (2.10) and (2.13) or Eqs. (2.9) and (2.12).

4.2.5 A Note on the Indeterminate Cases of Likelihood Computation

The first likelihood operation (2.9) faces no problem for any combination of likelihoods. But the second likelihood operation (2.12) can face indeterminate situations (it is already addressed in (2.12)), which are exhaustively listed and explained as follows.

When a frozen bit opposes the decision propagated to the node, the frozen bit clearly overrides the decision. However, a more difficult problem will immediately arise in the next iteration. The earlier decision becomes an event with zero probability according to the likelihoods, and cannot be taken as a conditional event which is necessary for SCD's likelihood computation. Therefore the bit's APP becomes indeterminate and not computable. In such a case, the likelihood is

Eq. (2.9)	$\mathbf{q} = [1,0]$	$\mathbf{q} = [0,1]$	$\mathbf{q} = [0.5,0.5]$
$\mathbf{p} = [1,0]$	$[1,0]$	$[0,1]$	$[0.5,0.5]$
$\mathbf{p} = [0,1]$	$[0,1]$	$[1,0]$	$[0.5,0.5]$
$\mathbf{p} = [0.5,0.5]$	$[0.5,0.5]$	$[0.5,0.5]$	$[0.5,0.5]$

Eq. (2.12) & $\hat{b} = 0$	$\mathbf{q} = [1,0]$	$\mathbf{q} = [0,1]$	$\mathbf{q} = [0.5,0.5]$
$\mathbf{p} = [1,0]$	$[1,0]$	$[0.5,0.5]$	$[1,0]$
$\mathbf{p} = [0,1]$	$[0.5,0.5]$	$[0,1]$	$[0,1]$
$\mathbf{p} = [0.5,0.5]$	$[1,0]$	$[0,1]$	$[0.5,0.5]$

Eq. (2.12) & $\hat{b} = 1$	$\mathbf{q} = [1,0]$	$\mathbf{q} = [0,1]$	$\mathbf{q} = [0.5,0.5]$
$\mathbf{p} = [1,0]$	$[0.5,0.5]$	$[0,1]$	$[0,1]$
$\mathbf{p} = [0,1]$	$[1,0]$	$[0.5,0.5]$	$[1,0]$
$\mathbf{p} = [0.5,0.5]$	$[0,1]$	$[1,0]$	$[0.5,0.5]$

Table 4.1: Simple look-up tables for likelihood transformations of SCD in a BEC

Eq. (2.9)	0	1	e
0	0	0	e
1	1	0	e
e	e	e	e

Eq. (2.12) & $\hat{b} = 0$	0	1	e
0	0	e	0
1	e	1	1
e	0	1	e

Eq. (2.12) & $\hat{b} = 1$	0	1	e
0	e	1	1
1	0	e	0
e	1	0	e

Table 4.2: Look-up table with an alternative simpler notation

considered in favor of an erasure, namely, $\mathbf{s} = [0.5, 0.5]$, and if this corresponds to a message bit and a decision is to be made, it may be chosen randomly (or fixed to 0 or 1) as suggested by Arıkan for the conflicting case of likelihoods (LR=1 or LLR=0).

Indeterminate case	Chosen solution for (2.12)
$\hat{b} = 0,\ \mathbf{p} = [0,1],\ \mathbf{q} = [1,0]$	$\mathbf{s} = [0.5, 0.5]$
$\hat{b} = 0,\ \mathbf{p} = [1,0],\ \mathbf{q} = [0,1]$	$\mathbf{s} = [0.5, 0.5]$
$\hat{b} = 1,\ \mathbf{p} = [0,1],\ \mathbf{q} = [0,1]$	$\mathbf{s} = [0.5, 0.5]$
$\hat{b} = 0,\ \mathbf{p} = [1,0],\ \mathbf{q} = [1,0]$	$\mathbf{s} = [0.5, 0.5]$

4.3 The Permuted Successive Cancellation Decoder (PSCD)

We will now discuss an interesting variant of the SCD which is called a *permuted successive cancellation decoder* (PSCD). There were several motivations in attempting this variant. First is to look for any better performance or latency. Second is to look for insights into the SCD that will enable our better understanding for further developments. Third is to verify whether the symmetries that are found in the encoding side extend to the decoder side, which usually has non-linear operations. Finally, we seek if we can have any alternative decoding orders that are relatively easier.

The PSCD satisfies a majority of the above expectations, except that its performance is found to be identical to the original SCD. In particular, the PSCD makes the below contributions to the polar codes.

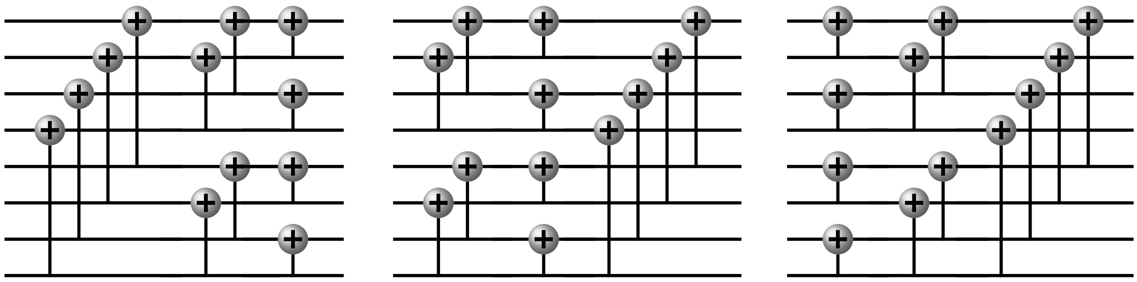
1. We find $n!$ distinct variants of the PSCD that have identical performance as well as complexity to that of Arıkan's original SCD. But as we will see, it is necessary to modify the code construction algorithm resulting in a different code in each case.
2. We find that using one of the variants formed by placing all the n stages of the decoder in reverse, leads to a natural decoding order which has a significant potential to simplify the implementations. This is mainly because any order other than the natural order, even if it is fixed, requires additional storage as well as computations. Further this essential overhead goes unnoticed often from the complexity calculations due to the significant dependence on the implementation.
3. Though it was conjectured with examples earlier, we formally prove that there exists a permutation symmetry within the stages of the encoding circuit.
4. Further, we will show that the above symmetry on the encoding side translates to the decoding side by producing performance equivalent to that of original code, even though the code construction algorithm needs to be adjusted.

4.3.1 Definition of the Permutation of the Encoding/Decoding Structure

In the context of a PSCD, we define a *permutation* as the rearrangement of the stages $\{0, 1, \ldots, n-1\}$ in Fig. 2.2. The new decoder is called a *permuted successive cancellation decoder* (PSCD).

A PSCD is completely specified by the permutation vector $\underline{\pi} \triangleq [\pi_0, \pi_1, \ldots, \pi_{n-1}]$, where decoding stages $\pi_0, \pi_1 \ldots$ appear from left to right. For example, $\underline{\pi} = [n-1, \ldots, 1, 0]$ denotes Arikan's classic SCD (Fig. 2.2) and $\underline{\pi} = [0, 1, \ldots, n-1]$ denotes the PSCD with fully reversed order of stages. Below are the three instances of PSCD for $N = 8$.

In the next two subsections, we see the effect of a $\underline{\pi}$ - permuted graph on the encoding and decoding of polar codes. Note that to know whether the permutation has any effect on the encoding, it is necessary and sufficient to see whether swapping two stages of the encoding (commutation of encoding stages) has any effect at all.



$\underline{\pi} = [2, 1, 0]$ $\underline{\pi} = [1, 0, 2]$ $\underline{\pi} = [0, 1, 2]$

Figure 4.2: Illustration of the permutations in the encoder graph

4.3.2 Effect of Permutation on Encoder

Theorem:

1. The encoding in (2.1) can be written as:

$$\mathbf{F}^{\otimes n} = \prod_{i=0}^{n-1} (\mathbf{I}_{2^{n-1-i}} \otimes \mathbf{F} \otimes \mathbf{I}_{2^i}) \tag{4.2}$$

2. The n stages of the encoder in Fig. 2.2, are respectively represented by different product terms in (4.2).

3. The terms in above matrix product, representing various stages of encoding as shown in Fig. 2.2, *commute* with each other, i.e. $\forall i, j$,

$$(\mathbf{I}_{2^{n-1-i}} \otimes \mathbf{F} \otimes \mathbf{I}_{2^i}) \cdot (\mathbf{I}_{2^{n-1-j}} \otimes \mathbf{F} \otimes \mathbf{I}_{2^j}) = (\mathbf{I}_{2^{n-1-j}} \otimes \mathbf{F} \otimes \mathbf{I}_{2^j}) \cdot (\mathbf{I}_{2^{n-1-i}} \otimes \mathbf{F} \otimes \mathbf{I}_{2^i}) \tag{4.3}$$

We call this a *permutation-invariant property* of the encoder.

Proof:

1. Consider the following identity for the Kronecker product operation of m of matrices $\{\mathbf{A}_1, \ldots, \mathbf{A}_m\}$ and $\{\mathbf{B}_1, \ldots, \mathbf{B}_m\}$, each set containing square matrices of the same dimensions.

$$(\mathbf{A}_1 \cdots \mathbf{A}_m) \otimes (\mathbf{B}_1 \cdots \mathbf{B}_m) = (\mathbf{A}_1 \otimes \mathbf{B}_1) \cdots (\mathbf{A}_m \otimes \mathbf{B}_m) \tag{4.4}$$

Further, let $\left\{\mathbf{A}_{ij}\right\}_{i=1,j=1}^{i=m,j=p}$ be a set of mp matrices. The following will hold true whenever the matrix products are well defined.

$$\begin{gathered} \big(\mathbf{A}_{11} \cdots \mathbf{A}_{1n}\big) \otimes \big(\mathbf{A}_{21} \cdots \mathbf{A}_{2p}\big) \otimes \ldots \otimes \big(\mathbf{A}_{m1} \cdots \mathbf{A}_{mp}\big) = \\ \big(\mathbf{A}_{11} \otimes \cdots \otimes \mathbf{A}_{m1}\big)\big(\mathbf{A}_{12} \otimes \cdots \otimes \mathbf{A}_{m2}\big) \cdots \big(\mathbf{A}_{1p} \otimes \cdots \otimes \mathbf{A}_{mp}\big) \end{gathered} \tag{4.5}$$

Using the above two properties, we can rewrite the Kronecker products $\mathbf{F}^{\otimes 2}, \mathbf{F}^{\otimes 3}, \mathbf{F}^{\otimes 4}, \ldots$ as follows:

$$\begin{aligned} \mathbf{F} \otimes \mathbf{F} &= (\mathbf{I}_2 \cdot \mathbf{F}) \otimes (\mathbf{F} \cdot \mathbf{I}_2) \\ &= (\mathbf{I}_2 \otimes \mathbf{F}) \cdot (\mathbf{F} \otimes \mathbf{I}_2) && \text{(using (4.4))} \end{aligned}$$

$$\begin{aligned}\mathbf{F}\otimes\mathbf{F}\otimes\mathbf{F} &= \big(\mathbf{I}_2\cdot\mathbf{I}_2\cdot\mathbf{F}\big)\otimes\big(\mathbf{I}_2\cdot\mathbf{F}\cdot\mathbf{I}_2\big)\otimes\big(\mathbf{F}\cdot\mathbf{I}_2\cdot\mathbf{I}_2\big) && \text{(pad 2 identity matrices to each term)}\\ &= (\mathbf{I}_4\otimes\mathbf{F})\cdot(\mathbf{I}_2\otimes\mathbf{F}\otimes\mathbf{I}_2)\cdot(\mathbf{F}\otimes\mathbf{I}_4) && \text{(using (4.5))}\end{aligned}$$

$$\begin{aligned}\mathbf{F}\otimes\mathbf{F}\otimes\mathbf{F}\otimes\mathbf{F} &= \big(\mathbf{I}_2\cdot\mathbf{I}_2\cdot\mathbf{I}_2\cdot\mathbf{F}\big)\otimes\big(\mathbf{I}_2\cdot\mathbf{I}_2\cdot\mathbf{F}\cdot\mathbf{I}_2\big)\otimes\big(\mathbf{I}_2\cdot\mathbf{F}\cdot\mathbf{I}_2\cdot\mathbf{I}_2\big)\otimes\big(\mathbf{F}\cdot\mathbf{I}_2\cdot\mathbf{I}_2\cdot\mathbf{I}_2\big) && \text{(pad 3 identity matrices to each term)}\\ &= (\mathbf{I}_8\otimes\mathbf{F})\cdot(\mathbf{I}_4\otimes\mathbf{F}\otimes\mathbf{I}_2)\cdot(\mathbf{I}_2\otimes\mathbf{F}\otimes\mathbf{I}_4)\cdot(\mathbf{F}\otimes\mathbf{I}_8) && \text{(using (4.5))}\end{aligned}$$

On generalizing above procedure for arbitrary $\mathbf{F}^{\otimes n}$, we readily have (4.2), i.e.,

$$\mathbf{F}^{\otimes n} = \prod_{i=0}^{n-1}\big(\mathbf{I}_{2^{n-1-i}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^i}\big),\ \text{where } \mathbf{I}_1 \triangleq 1.$$

2. As the circuit is a visual representation of the equations, it is not difficult to verify for $N = 16$ that each stage in Fig. 2.2 is an implementation of the product terms above. By appropriate extension (as a form of induction) we can recursively *construct* the same encoding circuit for arbitrary N with n stages representing n product terms above.

3. Now, without any loss of generality, consider any two elements of the product (4.2) with distinct indices j, i $(j > i)$.

$$\begin{aligned}&\Big(\mathbf{I}_{2^{n-1-i}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^i}\Big)\cdot\Big(\mathbf{I}_{2^{n-1-j}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^j}\Big)\\ &= \Big\{\mathbf{I}_{2^{n-j}}\otimes\big(\mathbf{I}_{2^{j-1-i}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^i}\big)\Big\}\cdot\Big\{\big(\mathbf{I}_{2^{n-1-j}}\otimes\mathbf{F}\big)\otimes\mathbf{I}_{2^j}\Big\}\\ &= \Big\{\mathbf{I}_{2^{n-j}}\cdot\big(\mathbf{I}_{2^{n-1-j}}\otimes\mathbf{F}\big)\Big\}\otimes\Big\{\big(\mathbf{I}_{2^{j-1-i}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^i}\big)\cdot\mathbf{I}_{2^j}\Big\} && \text{(using (4.4))}\\ &= \Big\{\big(\mathbf{I}_{2^{n-1-j}}\otimes\mathbf{F}\big)\cdot\mathbf{I}_{2^{n-j}}\Big\}\otimes\Big\{\mathbf{I}_{2^j}\cdot\big(\mathbf{I}_{2^{j-1-i}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^i}\big)\Big\}\\ &= \Big(\mathbf{I}_{2^{n-1-j}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^j}\Big)\cdot\Big(\mathbf{I}_{2^{n-1-i}}\otimes\mathbf{F}\otimes\mathbf{I}_{2^i}\Big) && \text{(using (4.4))}\end{aligned}$$

The above proves that any two terms of (4.2) commute with each other. This further implies that any arbitrary permutation in the encoder graph is equivalent to the original graph. ∎

4.3.3 Effect of Permutation on Decoder

Since the encoder is permutation-invariant, it is interesting to know how a decoder would behave with a permutation $\underline{\pi}$. It can be easily observed that the likelihood evolution in the graph from right to left [11] is a non-linear operation and hence the order of the stages may affect the overall decoder performance. A study of such effects is our main contribution in this section.

Given a PSCD specified by $\underline{\pi}$, Algorithm 2 must be modified for: (i) the appropriate new decoding order replacing bit-reversal decoding order (step-5, Algorithm 2) (ii) the evolution of likelihoods (step-6, Algorithm 2), and (iii) broadcasting of decisions (step-12, Algorithm 2). Table 4.3 below summarizes these modifications.

Module	**Line No.**	**Replace With**
Algorithm 2 (SCD)	5	$l = \textbf{nextindex}(\underline{\pi}, l)$
UpdateL(i, j)	1	$s = 2^{1+\pi_{n-1-j}}$
UpdateB(i, j)	1	$s = 2^{1+\pi_{n-1-j}}$

Table 4.3: Summary of changes in SCD algorithm for PSCD

Note that, the first modification requires a new function rather than bit-reversal function. A naive solution, such as an exhaustive search to find which next channel is possible, but will have a prohibitive $O(N)$ additional complexity for each bit. This is avoided by using the function **nextindex**$(\underline{\pi}, p)$. The new function **nextindex**$(\underline{\pi}, p)$ results in the same $\mathcal{O}(\log N)$ complexity as that of a bit-reversal algorithm.

In simulations (Sec. 4.3.4) we observe that the performance of such a decoder can become poorer. However, we will show below how we can address this issue.

A polar code is constructed to optimize its performance assuming that the decoder uses Arikan's standard SCD structure. Therefore it may not be *fair* to expect the same performance on a new suboptimal decoder, for which the code is not matched. This motivates us to design a matching code construction for PSCD. Hence we propose Algorithm 10, an efficient implementation of (2.18) extended for PSCD, subsuming the original polar code construction algorithm (2.18) as a special case, when $\underline{\pi} = [n-1, \ldots, 1, 0]$. Similar to (2.18) we initialize the algorithm to optimize the code at a given E_b/N_0 and it is suggested to use log-domain calculations to avoid underflow.

The extended code construction (Algorithm 10) yields the same upper bounds as the Bhattacharyya bounds of the bit channels, but in a different order. By similarly freezing the least reliable bits in such new order, we should obtain a similar FER performance, whenever these bounds are tight. More importantly, based on this invariance of bounds to permutation, we can conclude that the PSCD with the matched code construction continues to be capacity achieving.

We also show next, by simulations that the BER of PSCD for matched code (using Algorithm 10) is also the same, irrespective of the new decoding order induced by PSCD.

The Reversal Permutation — Note that there are $n!$ alternative permutations for use under PSCD and matched code, having performance as good as SCD and original polar code. Among them, one particular case of interest is the *reversal* permutation i.e., $\underline{\pi} = [0, 1, \ldots, n-1]$, since it has

```
Function 9 nextindex($\pi$,$p$) : Decoding Order
    Input  : Current index $p$ and permutation $\underline{\pi} = [\pi_0, \ldots, \pi_{n-1}]$
    Output : Next decodable bit index $q$, in the new decoding order
    if ($p ==$ NaN ) then $q = 0$ and return.                      ▷ First index
    Let $p \equiv \sum_{t=0}^{n-1} p_t 2^t$ be the binary-representation of $p$
    $x = \left(\sum_{t=0}^{n-1} p_{\pi_t} 2^t\right)$              ▷ Leftmost stage in $\underline{\pi}$ is LSB
    $x = x + 1$
    Let $x \equiv \sum_{t=0}^{n-1} x_t 2^t$
    $q = \left(\sum_{t=0}^{n-1} x_t 2^{\pi_t}\right)$
```

an interesting property that the decoding order becomes equal to the natural order of bits at encoding. This may be of great practical interest, for example, to avoid the additional complexity of bit-reversal permutation (or **nextindex**($\underline{\pi}, p$) in general), thereby reducing the overall latency of the decoder at no loss in performance (see Fig. 4.3).

4.3.4 Simulations

In this section we consider block-lengths $N = 256, 1024$, and 8192 with rates $R = 0.5, 0.7$, and 0.9, respectively. In Figs. 4.3 and 4.4, we use abbreviations SCC and PSCC to denote the code constructions for SCD and PSCD, respectively. We consider Binary Input Additive White Gaussian Noise (BI-AWGN) channel in all simulations. Following our discussion in the earlier sections, we observed in simulations that FER of PSCD under PSCC is indeed the same as that of a standard polar code under SCD. We therefore omit the plots of FER and show only the plots of BER.

The Fig. 4.3 shows the performance comparison of SCD and PSCD used with Arikan's polar code construction, and the performance of PSCD with the newly proposed code construction for PSCD at $N = 1024$ and $R = 0.7$. The code is constructed using (2.18) with an initialization $z_{0,0} = 0.5$ as in [39], equivalent to optimizing the code at -1.592dB[105]. We observe that typically there is a performance degradation whenever PSCD is used instead of SCD for the polar code matched to the construction of SCD, but, the performance of the new code constructed to match the PSCD is as good as the performance of the classic SCD.

Fig. 4.4 illustrates the performance of the new code construction with a PSCD and Arikan's original polar code with an SCD and also a PSCD at $N = 8192$ and a high rate of $R = 0.9$. The constructions in this case are performed at 5dB. It shows that a similar performance can be obtained for all permutations and at all rates, when a PSCD along with a matched code construction is used. Hence we claim that the permuted polar codes designed for PSCD are a perfect alternative to the original polar codes at the same complexity.

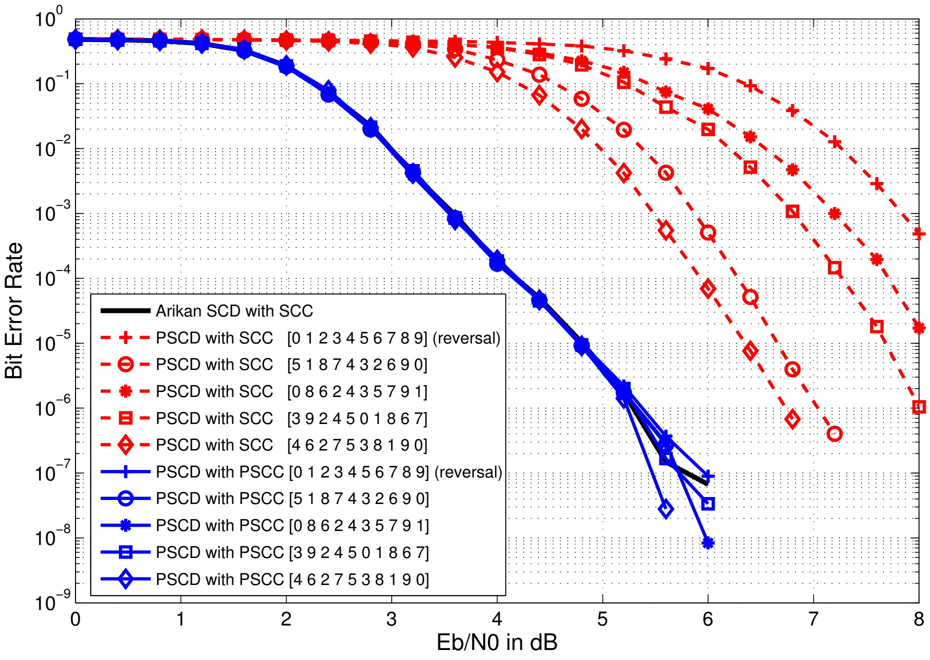


Figure 4.3: Performance of SCD under original polar code construction, PSCD with same code, and PSCD under new code construction; all constructions initialized with $z_{0,0} = 0.5$ (-1.592dB) for $N = 1024$ and $R = 0.7$ in BI-AWGN channel. All permutations are chosen randomly, except for the specific permutation of reversal.

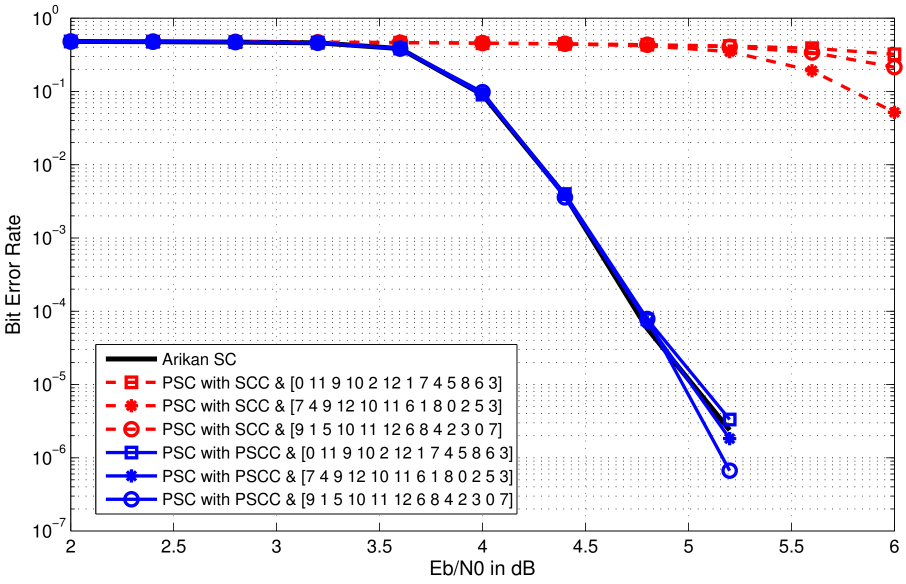


Figure 4.4: Performance of PSCD used for the new codes constructed using Algorithm-10, optimized at 5dB, at $N = 8192$ and $R = 0.9$ in BI-AWGN channel

Algorithm 10 Polar Code Construction for PSCD

```
Input  : $N$, $K$, initialization $z_{0,0} = \exp(-E_b/N_0)$ and the permutation
         $\underline{\pi} = [\pi_0, \pi_1, \ldots, \pi_{n-1}]$
Output : $\mathcal{F} \subset \{0, 1, \ldots, N-1\}$ with $|\mathcal{F}| = N - K$
$z \in \mathbb{R}^N$ and $z[i] = z_{0,0}$ $\forall i = 0, 1, \ldots, N-1$; also $n \triangleq \log_2 N$
for $j = 0 : n-1$ do                                  ▷ for each stage (right-to-left)
    $s = 2^{1+\pi_{n-1-j}}$                              ▷ the sub-stage size
    for $l = 0 : \frac{N}{s} - 1$ do                      ▷ For each sub-stage
        for $t = 0 : \frac{s}{2} - 1$ do                  ▷ For each connection
            $T = z[ls + t]$
            $z[ls + t] = 2T - T^2$                        ▷ Upper channel
            $z[ls + s/2 + t] = T^2$                       ▷ Lower channel
        end
    end
end
$[z, idx] = $ Sort$(z,$ 'descending'$)$ ▷ obtain in $idx$, the indices of $z$ vector when sorted in descending order
$\mathcal{F} = idx[0 : N - K - 1]$        ▷ return first $N - K$ indices
```

4.4 Improved Multiple Folded Successive Cancellation Decoding

The folded successive cancellation decoding is based on the ideas introduced in [81, 89, 92] exploiting the recursive structure in the encoding equation and its implementation graph. A formal definition of the technique is provided below. The basic idea is a decomposition of larger graphs into smaller graphs, where all graphs correspond to SCDs. A different perspective of this technique leading to a significant savings in hardware resources, will be discussed later in Section 7.1.

4.4.1 Folding Operation at the Encoder

Consider below the encoding equation rewritten using the well known Kronecker product property $\mathbf{AB} \otimes \mathbf{CD} = (\mathbf{A} \otimes \mathbf{C}) \cdot (\mathbf{B} \otimes \mathbf{D})$,

$$
\begin{aligned}
\mathbf{x} &= \mathbf{F}^{\otimes n}\mathbf{d} && (4.6)\\
&= \left(\mathbf{F}^{\otimes \kappa} \otimes \mathbf{F}^{\otimes n-\kappa}\right)\mathbf{d} && (4.7)\\
&= \left(\mathbf{I}_{2^\kappa} \cdot \mathbf{F}^{\otimes \kappa} \otimes \mathbf{F}^{\otimes n-\kappa} \cdot \mathbf{I}_{2^{n-\kappa}}\right)\mathbf{d} && (4.8)\\
&= \left(\mathbf{I}_{2^\kappa} \otimes \mathbf{F}^{\otimes n-\kappa}\right) \cdot \underbrace{\left(\mathbf{F}^{\otimes \kappa} \otimes \mathbf{I}_{2^{n-\kappa}}\right)\mathbf{d}}_{\mathbf{v}} && (4.9)\\
&= \begin{bmatrix} \mathbf{F}^{\otimes n-\kappa} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{F}^{\otimes n-\kappa} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{2^\kappa} \end{bmatrix} = \begin{bmatrix} \mathbf{F}^{\otimes n-\kappa} \cdot \mathbf{v}_1 \\ \vdots \\ \mathbf{F}^{\otimes n-\kappa} \cdot \mathbf{v}_{2^\kappa} \end{bmatrix} && (4.10)\\
\text{where } \mathbf{v} &= \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{2^\kappa} \end{bmatrix} \triangleq \left(\mathbf{F}^{\otimes \kappa} \otimes \mathbf{I}_{2^{n-\kappa}}\right)\mathbf{d}, && (4.11)
\end{aligned}
$$

and $\mathbf{v}_1, \cdots, \mathbf{v}_{2^\kappa}$, are the subvectors of $\mathbf{v}$ of length $2^{n-\kappa}$ or $N/2^\kappa$.

We may infer from (4.10) that given any $\kappa \leq n-1$, the encoding equation for codelength N can be visualized as a collection of several independent polar encodings of shorter codelength $N/2^\kappa$ on the partitions of vector $\mathbf{v}$. Therefore we denote this encoding stage as *block-wise polar encoding* (BPE) stage. However, we need a pre-processing step (4.11) on the information vector $\mathbf{d}$ to obtain $\mathbf{v}$. Interestingly, the pre-processing operation (4.11) can also be implemented by another set of independent polar encoders of codelength 2^κ as shown below. We call this pre-processing stage as *interleaved polar encoding* (IPE) stage. Clearly, the two stages give rise to the original polar encoding of codelength N.

Recall the following property of the Kronecker product of square matrices $\mathbf{A}$ and $\mathbf{B}$, with possibly different dimensions:

$$\mathbf{P}^T(\mathbf{A}\otimes\mathbf{B})\mathbf{P} = (\mathbf{B}\otimes\mathbf{A}) \tag{4.12}$$

where $\mathbf{P}$ is a block-interleaver permutation matrix of rows, with depth equal to the dimension of square matrix $\mathbf{A}$. This can be easily verified by writing the expansions of Kronecker products $(\mathbf{A}\otimes\mathbf{B})$ and $(\mathbf{B}\otimes\mathbf{A})$ and then identifying the permutation involved to match them.

Using (4.12), we can rewrite (4.11) as follows.

$$\begin{aligned}\mathbf{v} &= (\mathbf{F}^{\otimes\kappa}\otimes\mathbf{I}_{2^{n-\kappa}})\,\mathbf{d} &(4.13)\\ &= \mathbf{P}^T(\mathbf{I}_{2^{n-\kappa}}\otimes\mathbf{F}^{\otimes\kappa})\,(\mathbf{P}\,\mathbf{d}) &(4.14)\\ &= \mathbf{P}^T\begin{bmatrix}\mathbf{F}^{\otimes\kappa} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{F}^{\otimes\kappa}\end{bmatrix}\begin{bmatrix}\mathbf{d}_1\\ \vdots\\ \mathbf{d}_{2^{n-\kappa}}\end{bmatrix} &(4.15)\end{aligned}$$

where $\mathbf{P}$ is the permutation matrix corresponding to a block-interleaver of depth $N/2^\kappa$. From (4.15), it is evident that we can implement the IPE stage by applying $2^{n-\kappa}$ parallel polar encoders to the interleaved subvectors of $\mathbf{d}$, namely $\mathbf{d}_1,\mathbf{d}_2,\ldots,\mathbf{d}_{2^{n-\kappa}}$, each of length 2^κ and then de-interleaving their output before feeding the BPE stage. Fig. 4.5 illustrates the folding process for $N = 8, \kappa = 1, 2$. Note that all the three implementations in Fig. 4.5 are equivalent polar encoders. A general implementation of the new encoder is elaborated below. Note also that, the frozen bits will be scattered to the component codes in a way that each one of them is not necessarily a polar code.

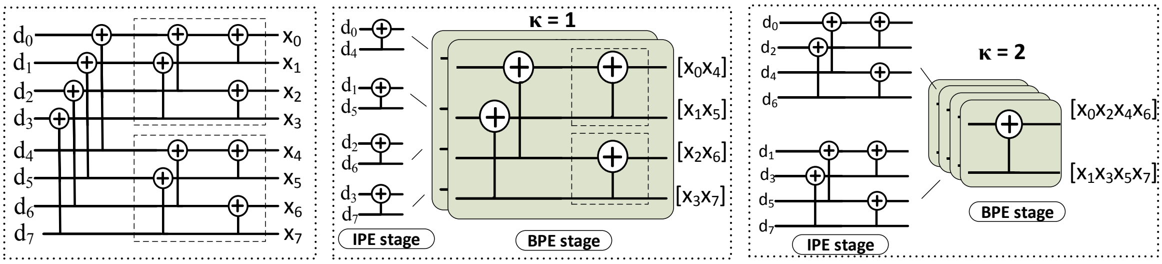


Figure 4.5: Illustrating folding for $N = 8$

By observing Fig. 4.5, the BPE stage may be vectorized by grouping every 2^κ bits output from an IPE stage encoder. The BPE stage encoding can also be viewed as operating over $GF(q)$, with $q = 2^{2^\kappa}$, where the addition operator is the element-wise XOR on bit components. As a result, each layer in the BPE stage processes the subvectors from $\mathbf{v}$, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{2^\kappa}$, and hence implements (4.10).

A simple interpretation of the multiple-folded polar encoder with folding-length κ is given as follows. Consider a rectangular interleaver with 2^κ columns holding N bits. First, we fill the interleaver memory columnwise with the elements of $\mathbf{d}$. Then, in IPE stage we encode in-place, all rows in parallel, using polar encoders of codelength 2^κ. Then, the BPE stage encodes in-place, all columns in parallel, using polar encoders of codelength $N/2^\kappa$. Finally, the memory is read columnwise to obtain the polar codeword $\mathbf{x}$. A similar interpretation will be used for the decoder, based on a rectangular memory of N real values representing the bit likelihoods.

4.4.2 Folding Operation at the Decoder

A key element in decoding a polar code is the use of the encoder graph in reverse direction. After folding, we split the encoder graph in *two* stages, which in turn are formed by different polar encoders of shorter codelengths 2^κ and $N/2^\kappa$. With this new interpretation, we may decode *independently* each of the component codes in each stage, for example, BPE stage can be decoded with a standard successive cancellation decoder and the IPE stage can be decoded by using a MLD or a list decoder.

Following the q-ary interpretation of the BPE stage encoding, an extension of SCD was proposed for BPE stage decoding in [89, 92], on the q-ary symbols from $GF(q)$ with $q = 2^{2^\kappa}$, formed by the groups of 2^κ bits in Fig. 4.5. As a result, the *probability mass function* (pmf) of each q-ary symbol gets transformed (replacing likelihood transformation of binary symbols) at different component 2×2 blocks of the decoder circuit. Such a pmf transformation takes $O(2^{2^\kappa} \cdot 2^\kappa)$ complexity, in their best implementation. In this section, we observe that this q-ary interpretation is not necessary using our new interpretation because the bit likelihoods within each q-ary symbol, corresponding to different layers in the BPE stage decoding, are independent of each other. In other words, layers of the BPE stage are independent, and hence the bits within each q-ary symbol can be processed independently and parallelly at a complexity of $O(2^\kappa)$ only. This is a significant complexity reduction from the earlier work [89, 92] which has $O(2^{2^\kappa} \cdot 2^\kappa)$ complexity.

Choice of decoders: We propose to use a standard SCD algorithm in BPE stage parallelly at each layer, and MLD at the IPE stage. Note that, for small values of κ the IPE stage MLD deals with codes of length 2^κ and hence has negligible complexity. We name this new decoder an *improved multiple folded successive cancellation decoder* (IMFSCD) with folding length κ. Note that had we used SCD at both the stages, the result would have been exactly equal to the standard SCD algorithm of codelength N. By replacing an SCD with an MLD, we expect IMFSCD to perform at least as well as or better than the standard SCD of codelength N. This is confirmed by our simulations for various test cases.

Similar to the encoder, a simple interpretation of the IMFSCD can be given. Consider a rectangular interleaver with 2^κ columns and $2^{n-\kappa}$ rows, holding N *real* values. The rectangular interleaver memory is filled columnwise in *natural order*, with the likelihoods computed from the channel observations. Each column is then decoded in parallel SCDs for codes of length $N/2^\kappa$. This results in a rowwise evolution of the likelihoods stored separately, in the bit-reversed order of rows. Once we have a row of bit likelihoods, we pass it on to an MLD for the code of length 2^κ, which gives a row of 2^κ decisions including frozen bits. The appropriate polar encoded decisions are then broadcasted parallelly just as in a standard SCD. The information bit decisions are stored separately in a similar rectangular interleaver of bits, in the same locations and finally read columnwise in the *natural order* to output the decisions.

Figs. 4.6 and 4.7 show our simulations of polar codes performance under IMFSCD assuming BPSK modulation and AWGN channel for a codelength $N = 8192$ and rate $R = 0.8$. The code construction is taken from [126] and is performed at 0dB. The plots show the comparison of an IMFSCD with $\kappa = 1, 2, 3$ versus a standard SCD. We will see in Figs. 4.6 and 4.7 that the IMFSCD can give exactly the same performance as that of a standard SCD.

Latency gain and complexity of IMFSCD: The proposed MLD replacing the SCD for the IPE stage, will have higher complexity compared to a standard SCD, but this increase is not significant so far as the value of κ is small ($\kappa \leq 3$). Note that the previously proposed decoders [89, 92] have an much higher increase in complexity by a factor of 2^{2^κ} in their best implementation, due to the q-ary symbol processing.

The latency of the IMFSCD of length N is equal to a standard SCD of length $N/2^\kappa$, plus the latency of the MLD in the IPE stage. However, unlike SCD, MLD can be highly parallelized. Therefore, the difference in latency can be minimized, which is the key in reducing the latency $(2N - 1)$ of Arikan's SCD (see [11]) under maximally parallelized implementation.

Then the latency of IMFSCD can be reduced to a value up to $(N/2^{\kappa-1} - 1)$, which is a significant reduction by a factor of 2^κ, when compared to the standard SCD.

4.4.3 A Comparison with Concatenated Codes

As we noted earlier, a range of new decoders can be proposed using the above interpretation of *multiple folding*. This model closely resembles the architecture of a concatenated code with an inner code and an outer code. Interestingly, we may note that neither the outer codes nor the inner codes in multiple folding architecture are polar codes (i.e., the frozen bits are not selected according to the standard construction). In the IPE stage, this is due to frozen bits being spread randomly across the bits seen by IPE stage encoders. In the BPE stage, the locations of frozen bits are not compatible with a polar code construction.

In the light of this observation, an interesting comparison is to verify how a concatenated code performs compared to the IMFSCD polar code with the same rate and codelength. As a simple example, we use a single parity check code at the IPE stage and the same polar code for all component codes of the BPE stage. This choice is motivated by the fact that single parity

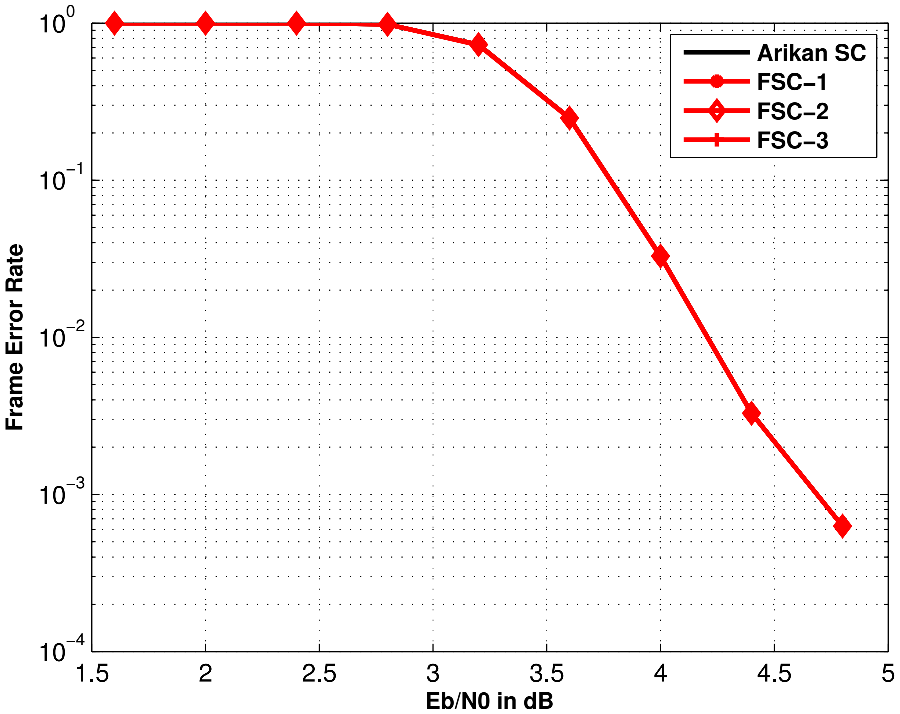


Figure 4.6: FER comparison of IMFSCD($\kappa = 1, 2, 3$) vs. SCD, at $N = 8192, R = 0.8$ and code construction @ 0dB [126]

check codes are a classic example of low complexity ML decodable codes, matching the choice of an IPE stage decoding in an IMFSCD. The rate of the inner polar code is adjusted so as to match the overall rate of the concatenated code.

The simulation of a single parity check code of length 8, concatenated with polar code of length 1024 at overall rate $R = 0.5$ is considered in Figs. 4.8 and 4.9. We compare the performance of this concatenated code with polar codes with $N = 1024, R = 0.5$ and $N = 8192, R = 0.5$, being decoded using the standard SCD. We see that the concatenated code performs worse when compared to a classic polar code of an equal length and rate. This implies that the joint optimization performed by polar code construction is more powerful than a simple single parity check code concatenation. Further comparisons are deferred to our future work.

4.5 A New Parallelized LSCD

4.5.1 Introduction

A LSCD is an effient low complexity algorithm for decoding polar codes[53]. The algorithm essentially uses a list of L copies of the SCD. Using an efficient implementation of the list, the complexity of LSCD is shown to be $\mathcal{O}(L \cdot N \log N)$ and the delay of the algorithm is $\mathcal{O}(N \cdot L)$.

It was shown in [53] that a list decoder with a sufficiently large *list size* L, can achieve an error performance arbitrarily close to that of an optimal MLD. A *CRC aided LSCD* (CA-LSCD) was also proposed in [53] as a concatenation of a polar code and a cyclic-redundancy check (CRC)

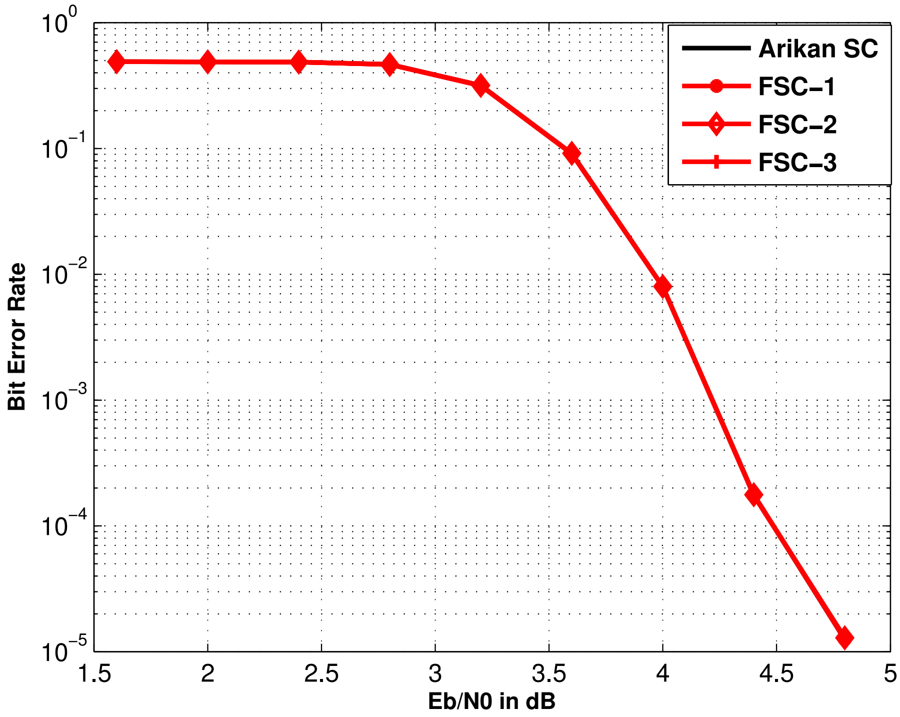


Figure 4.7: BER comparison of IMFSCD($\kappa = 1, 2, 3$) vs. SCD, and code construction @ 0dB [126]

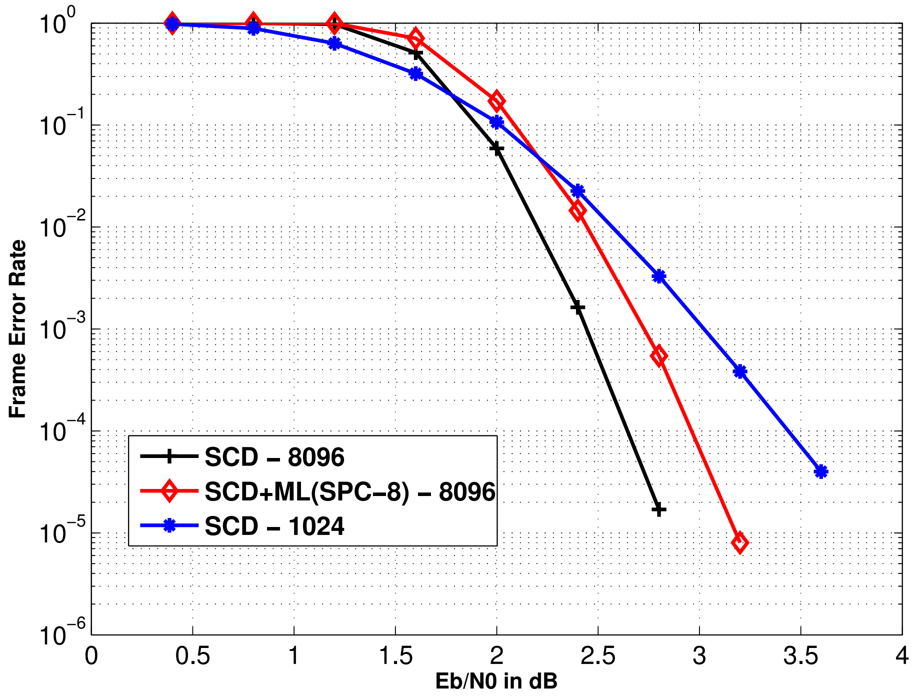


Figure 4.8: FER of a single parity check code of length 8 concatenated with a polar code of length 1024, at effective rate 0.5 compared with a polar codes with $N = 8192, R = 0.5$ and $N = 1024, R = 0.5$

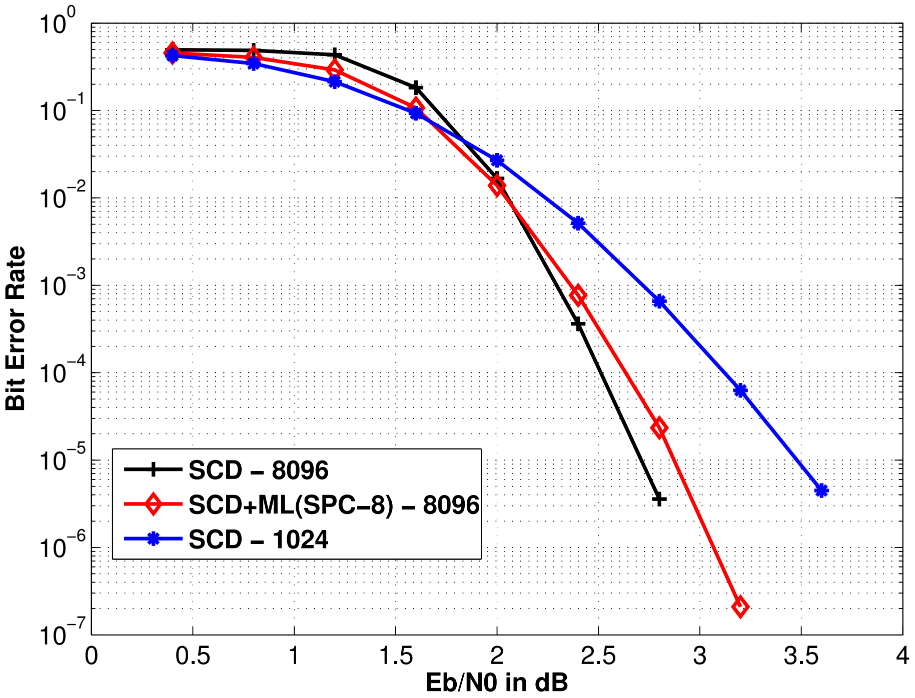


Figure 4.9: BER of a single parity check code of length 8 concatenated with a polar code of length 1024, at effective rate 0.5 compared with a polar codes with $N = 8192, R = 0.5$ and $N = 1024, R = 0.5$

code. It was also shown that a CA-LSCD can give better performance than the state-of-the-art LDPC codes. This finding has created a significant interest in research community towards more efficient list decoding algorithms [54–63, 142–147].

The SCD was the original decoder proposed by Arikan for decoding polar codes. An LSCD gives a significantly improved error performance over SCD. Unfortunately, its decoding delay and complexity both increase by a factor of L when compared to an SCD. Further, larger values of L (e.g. $L \geq 16$) are essential to elicit the best possible performance from the decoder. Recently, there have been several proposals to further improve its performance, reduce the storage requirements, delay and complexity. A brief review of these recent advances in literature is given below.

4.5.2 Recent Advances in LSCD

In [56] an *adaptive CA-LSCD* is proposed. The idea is to simply re-attempt decoding with the list size L doubled (upto a maximum, $L \leq L_{max}$), until a code word with a valid CRC is found. This technique was found to have a capacity-approaching performance at very large values of L_{max}, as illustrated with an example of $L_{max} = 262144$ at $N = 2048$ and $R = 0.5$. However, the adaptive CA-LSCD has a very high complexity as well as delay for its use in practice. Another interesting variant of CA-LSCD in the context of a special concatenated polar code was proposed in [54].

The works [142–146] have illustrated several efficient hardware implementations of the LSCD and its variants, with varying hardware efficiencies and high throughputs upto 181 Mb/s [145]. These papers have demonstrated that polar codes under LSCD (or its variants) can provide a competitive alternative to the state-of-the-art LDPC and Turbo codes.

In [59] and [57], interesting ideas were proposed to reduce the overall complexity of the LSCD algorithm. In [59] Bhattacharyya parameters and the likelihoods of bits are used to distinguish reliably received information bits from the received codeword. Then an LSCD is performed only for the unreliable bits, while much simpler SCD is performed for the bits which are received reliably. This reduces the complexity of LSCD significantly while error performance remains almost same. In [57], the LSCD was first presented as a simple tree-pruning path search algorithm in the code-tree representation. Then a more intelligent tree-pruning was proposed to avoid some unnecessary path searching operations. This resulted in a significant reduction of the overall complexity. It was shown that the complexity of their decoder is close to that of an SCD at higher SNR, while having no observable performance loss. However, both these proposals do not seem to reduce significantly the overall high delay of LSCD.

In [58] and [62], LLR based LSCD is proposed. The basic version of the LSCD requires that for each bit computed in the list of L SCDs, two likelihood probabilities are stored. This is different from the standard SCD where a single LLR or a LR is used for each bit. As a result, the memory requirement of each SCD in the list is doubled and the computational complexity also increases. Instead, [58, 62] illustrate that one LLR per bit is indeed sufficient to implement an LSCD. This reduces the memory requirements and the complexity of LSCD significantly. Further the LLR-based calculations allow several useful approximations in the log-domain, resulting in a much simpler hardware implementations. However, there is no significant impact on the overall decoding delay of LSCD.

Finally, in [60, 61, 63] the high latency of the LSCD is addressed. In [61] a symbol-based LSCD was proposed based on the idea of *parallel decoders* proposed earlier in [87, 90] to reduce the overall delay. The basic idea was also independently developed as a *multiple folded SCD* in [89, 139]. The core idea of all these works [87, 89, 90, 139] was to combine and treat several bits of the polar code together as a q-ary symbol. These ideas are extended to LSCD in [61] and complexity is reduced by avoiding the MLD involved in the earlier proposals. In [60], a Chase-like decoding process was used to achieve six times improvement in speed of LSCD. In [63], a reformulation of LSCD was made to decode several bits simultaneously. It was demonstrated that the hardware implementations can achieve very high throughputs upto 501 Mb/s.

4.5.3 Our Contributions on LSCD

Now, we propose a latency reduction technique which provides significant additional reduction in the overall delay. Specifically, we reduce the latency of LSCD by first dividing each iteration of LSCD into four stages and then by improving each stage separately. Three out of four stages are fully parallelized while the remaining stage is a simple selection algorithm to select L best metric values out of $2L$ metrics. Further, the number of memory copy operations required is minimized. We also combine several other improvements in recent LSCD literature (mainly [58, 62]) that significantly reduce the storage requirements. A list of main contributions of this section is given below.

1. We provide a simple pseudocode implementation of LSCD, that includes several algorithmic implementations such as [53], as special cases.
2. We identify the critical algorithmic component of LSCD and provide a simple four-stage description to it with the objective of high parallelization.
3. We show that three out of these four stages are highly parallelizable, while the remaining stage is a simple selection algorithm.
4. We show that an efficient parallelized implementation of our algorithm can ensure a significant delay reduction in LSCD upto a factor of L. This makes the latency of the LSCD comparable to an SCD.

4.5.4 The New Efficiently Parallelized LSCD

List decoding is a classic technique that dates back to 1950's [148] and enhances the performance of many existing decoders. In the context of polar codes, an efficient implementation of the list decoding technique was proposed in [53], using the original SCD algorithm. Specifically, by using a technique called *lazy copy*, the authors reduced the complexity significantly compared to the naive implementations of list decoders. This version therefore became the basis for all the later advances in the list decoding algorithm for polar codes, reviewed in Section 4.5.1. We refer the reader to [53] for the details. We combine the recent modifications in [58, 62] using one LLR value instead of two likelihood probabilities for each bit.

Let an SCD implementation with $\mathcal{O}(N)$ space requirements (Section 4.1), be abstractly captured by a data structure which provides the following resources. These resources are independently allocated for each instantiation of the data structure.

- **LLR** and **B** denote the central memory components of the SCD that store likelihoods and bits respectively (their dimensions depend on the space efficiency).
- **initialize(y)** : To initialize the data-structure for decoding the received codeword **y**. It stores the initial likelihoods into **LLR**.
- **updateLLR()** : To update the log-likelihood-ratio of the subsequent bit in SCD, by updating several intermediate values in **LLR** according to SCD's butterfly-structure[11].
- **decisionmetric(x)** : Returns the metric associated with the decision x on the current bit. This metric indicates how likely is the partial codeword (decoded thus far) after taking decision x on the current bit. This calculation is taken from [58, 62]. This function can be called only after making a call to **updateLLR()**, which obtains the LLR of the current bit and aids the metric calculation.

- **updateB(x)** : To take a decoding bit-decision $x \in \{0, 1\}$ on the current bit and update memory **B** by broadcasting the latest bit decision within SCD's butterfly-structure[11].
- **latestbit()** : To return the decision x taken on the latest bit based on the latest bit-likelihood computed earlier. This can be called only after **updateB(x)**.
- **extractmsg()** : To extract K message bits at the end of the SCD algorithm.

A list of L SCD data-structures like the one above, is maintained as an array "**LIST**". Finally, our latest improvement to LSCD is discussed below, which we denote by algorithm **EP_LSCD**. The algorithm utilizes three important routines, namely, **bestSCDindex( )**, **advanceLIST( )**, **HowToCopy( )**.

The main component of **EP_LSCD** is the function **advanceLIST( )**, whose primary objective is to complete one iteration of each SCD in the list. However, the critical difference here is that we consider all possible decisions instead of just the ML decisions. The algorithm is then to select L best decisions that give best L metric values.

advanceLIST( ) is also responsible for the overall delay and complexity of the overall decoder. In fact, the original LSCD [53] becomes a special case of the same algorithm **EP_LSCD**, only with a difference in the implemention of **advanceLIST( )**. Our primary focus here therefore, is to improve this particular routine. Our proposal is described as follows.

Consider the following novel interpretation of advanceLIST() in four stages, with the main objective of parallelizability. A description of these stages is given below, along with a discussion of the parallelizations opportunities.

1. **Stage 1:** This stage updates the LLR's of each SCD in the list. We identify that this stage is fully parallelizable by making simultaneous calls to **updateLLR**( ).
2. **Stage 2:** There are $2l$ bit decisions possible, which depict $2l$ branches emanating from existing l branches in the code-tree. We now calculate the relative metrics of these $2l$ branches denoting how likely they are. Now the objective is to find at most L best branches in terms of these metric values. The algorithm **HowToCopy( )** finds the best L branches. Now, the storage memory **LLR** and **B** can be common to a pair of branches that are selected in this stage. Therefore a memory copy operation is essential. But, this memory copy is identified as an algorithmically cumbersome operation within a hardware implementation of LSCD[145]. Therefore we also identify in this stage, a way to minimize the number of memory copy operations. The copy operation is performed in the next stage.
3. **Stage 3:** We simply perform the memory copy operations that are required. Due to the optimization of memory copy operations at the earlier stage 2, this memory copy can be fully parallelized. We can therefore perform all the copy operations simultaneously.
4. **Stage 4:** Finally, the best decisions identified in step 2 are considered for broadcasting. We identify the full parallelizability of this stage, broadcasting the decisions simultaneously at all the SCDs in the list.

As we also see in the pseudocode of **advanceLIST( )**, the stages 1,3,4 contain for-loops that depict at most L operations which can completely be parallelized. Such a parallelization results in a significant reduction in the overall delay upto a factor of L. The remaining stage 2 involves a simple selection algorithm, described as **HowToCopy( )**. In an efficient implementation, the delay from such a small algorithm at stage 2 can be negligible. Finally, by fully utilizing this parallelization, the overall delay can be reduced by a factor of L, which makes our **EP_LSCD**, almost as fast as an SCD.

Algorithm 11 EP_LSCD: The Eff. Parallelized LSCD

```
INPUT  : $(N, K, \mathcal{F})$, list size $L$ and received vector $\mathbf{y}$
OUTPUT : Decoded message of $K$ bits
LIST[$i$], $\forall i = 0, 1, \ldots, L-1$ are SCD data structures, accessible across all routines
LIST[0].initialize($\mathbf{y}$)
$l = 1$
for $i = 0 : N-1$ do
    $j$ = bitreversed($i$)
    if $j \in \mathcal{F}$ then
        for $t = 0 : l-1$ do
            LIST[t].updateLLR( )
            LIST[t].updateB('0')
        end
    else
        if $2l < L$ then
            advanceLIST($l$, $2l$)          ▷ main algorithm
            $l = 2l$
        else
            advanceLIST($l$, $L$)           ▷ main algorithm
            $l = L$
        end
    end
end
$t$ =bestSCDindex($L$)
    ▷ index of the SCD with best metric among SCDs: $0, 1, \ldots L-1$
return (LIST[$t$].extractmsg( ))
```

4.5.5 Performance of the New LSCD

Our main algorithm **EP_LSCD** is a parallelized version of the LSCD. Such a parallelization is critical for hardware implementations in terms of a significant reduction of latency. We have therefore provided a new algorithm that allows high parallelization of the LSCD and brings the latency close to that of an SCD.

The performance is therefore exactly same as that of an LSCD. Further, the later proposed improvements such as systematic polar codes and CRC-concatenation are also not affected with

Function 12 bestSCDindex(l)

INPUT : **LIST**, an array of (at least) l SCDs

OUTPUT : Integer index of the SCD with best metric, among SCDs $0, 1, \dots l-1$

```
$b$=LIST[0].latestbit( )
$mbest$ = LIST[0].decisionmetric($b$)
$ibest = 0$
for $i = 1 : l-1$ do
    $b$=LIST[$i$].latestbit( )
    $m$ = LIST[$i$].decisionmetric($b$)
    if $mbest < m$ then
        $ibest = i$;
    end
end
return $ibest$
```

Function 13 advanceLIST(l, s): Advance & prune list

INPUT : **LIST**, an array of SCDs in which l are active

OUTPUT : **LIST** of size $s \geq l$ (Equivalently, an array grown from l to $2l$ SCDs corresponding to all possible $2l$ decisions and then pruned to size s).

```
Stage-1: “Parallelizable” LLR update
for $t = 0 : l-1$ do
    LIST[$t$].updateLLR()
end
Stage-2: Decide the final result of list-pruning
Initialize $2 \times l$ real vector “M”
for $t = 0 : l-1$ do
    M[0][$t$] = LIST[$t$].decisionmetric(‘0’)
    M[1][$t$] = LIST[$t$].decisionmetric(‘1’)
end
[I, deciisonbits] = HowToCopy(M,s)
  ▷ I — A vector of ‘$s$’ LIST-indices, which informs how to copy
  ▷ a few SCDs in LIST, growing its size from $l$ to $s$ ($l \leq s \leq 2l$)
  ▷ decisionbits — A vector of ‘$s$’ bit-decisions towards the final list
Stage-3: “Parallelizable” SCD copying (optimized)
for $t = 0 : s-1$ do
    if $t \neq$ I($t$) then
        LIST[$t$] = LIST[I($t$)]
    end
end
Stage-4: “Parallelizable” broadcasting of decision bits
for $t = 0 : s-1$ do
    LIST[$t$].updateB(decisionbits(t))
end
```

Function 14 [I, decisionbits] = HowToCopy(M, s)

```
INPUT  : M — an $2 \times l$ array of metrics
         $s$ — number of best metrics to choose from M
OUTPUT : I — $s$ column indices of M that contain the $s$ best elements of M. (A $1 \times s$ array)
         decisionbits — $s$ row-indices of the $s$ best elements in M, where all are single bit
         numbers. (A $1 \times s$ array)
$mm$ = reshape-as-row($\mathbf{M}$)
  ▷ reshape as a single ($1 \times 2l$) row vector, taken row after row
[$\sim$, I] =Sort($mm$,'descending')
  ▷ $I$ denotes reordered element indices when sorted in descending
$\mathbf{I} = \mathbf{I}[0 : s-1]$ ▷ truncate to $s$ indices (of $s$ highest elements)
Optimizing the number of copy operations:
counts[$j$] = 0, $\forall j = 0, \dots, s-1$
for $j = 0 : s-1$ do
    if $\mathbf{I}[j] < l$ then
        decisionbits[$\mathbf{I}[j]$] = 0
        counts$\big[I[j]\big]$ = counts$\big[I[j]\big] + 1$
    else
        decisionbits[$\mathbf{I}[j] - l$] = 1
        counts$\big[I[j] - l\big]$ = counts$\big[I[j] - l\big] + 1$
    end
end
zeroindices = $\{i \mid \text{counts}[i] = 0,\ i = 0, \dots, s-1\}$
oneindices = $\{i \mid \text{counts}[i] = 1,\ i = 0, \dots, s-1\}$
twoindices = $\{i \mid \text{counts}[i] = 2,\ i = 0, \dots, s-1\}$
I[oneindices] = oneindices
I[zeroindices] = I[twoindices] = twoindices
decisionbits[zeroindices] = 1
decisionbits[twoindices] = 0
Return (I, decisionbits)
```

the latest implementation. We therefore claim that our algorithm is a fine replacement of the basic LSCD algorithm in [53] and usable with almost all later improvements, at no loss in performance.

4.5.6 Summary of the New LSCD

We have proposed an L-fold parallelized architecture for the LSCD, to bring its overall delay closer to that of an SCD. We first divided each iteration of the LSCD into two stages and then propose a fully parallelized implementation of the first stage dividing it into L independent SCD components. A full parallelization is made for three out of four intermediate stages of each LSCD iteration. The remaining stage is a small selection algorithm to find L best metric values out of $2L$ metrics such that the number of memory copy operations is minimized. As a result, delay is reduced significantly for each iteration. The final decoder with full parallelization, has a decoding latency much closer to that of an SCD at no loss in performance. We also combined into our algorithm, some of the other improvements from the recent LSCD literature.

We have thus provided a fresh treatment to the LSCD algorithm and also provided a pseudocode that subsumes several of the earlier LSCD algorithms such as [53]. We identified a critical component of our **EP_LSCD** algorithm and provided a simple four-stage description with rich parallelization opportunity. By utilizing this opportunity, we have reduced the delay upto a factor of L and made the decoder as fast as simple SCD.

4.6 Summary

We have proposed a new variant of Arikan's SCD algorithm, which involves reordering of the decoding stages within SCD. We have found that the performance is exactly the same as the original decoder once we match the code construction to the new PSCD, while the performance is typically degraded when they are not matched. We also saw how a particular *reversal* permutation can avoid the bit-reversal order of decoding and outputs bits in natural order.

We have proposed a new formulation of the folding technique, by using which we have introduced a two stage concatenated interpretation of the standard polar encoder. Following this interpretation of the encoder, we have proposed a new low complexity decoder IMFSCD, with much lower latency over the standard SCD. The latency gain can be up to a factor of 2^{κ} depending upon the ML stage implementation, with significantly lower complexity than the earlier decoders based on folding. The performance with IMFSCD is exactly the same as that of an SCD. We have also supervised an FPGA implementation of our IMFSCD on a software defined radio platform to see the other benefits of it, namely, a significant reduction in the hardware resource measured by the number of gates.

We have also proposed a novel variant of the LSCD whose latency is significantly reduced and brought closer to that of an SCD, without any loss of performance. This stands out as a new implementation of the LSCD with parallelization of the L stages built in.

Chapter 5

Efficient Construction of Polar Codes

The construction of polar codes is simply the selection of the parameter $\mathcal{F}$. Due to the lack of any comparison of the large number of construction algorithms in literature, we start with a survey. We then identify an important problem with the polar code construction, namely the non-universality. We address this problem by using a novel heuristic. We finally propose an optimal quantization algorithm which can be instrumental tool for new construction algorithms. This quantizer uses a graph theory formulation to quantize in such a way that it maximizes the resultant mutual information over BI-DMC channels.

The following is a quick summary of the main contributions from this chapter:

1. We provide a detailed survey of important polar code construction (PCC) algorithms in literature and compare them with each other in complexity and performance.
2. We propose a Monte-Carlo based heuristic algorithm to find a reliable design-SNR of any PCC in order to obtain the polar codes of the best possible performance. This enables a fair comparison, and finds that all PCCs can produce equally good polar codes so far as a reliable design-SNR is known. (In the next chapter we extend the heuristic algorithm to avoid the high complexity Monte-Carlo simulations)
3. We provide a new channel quantizer algorithm which has optimal mutual information based on graph theory. We may use this to devise novel PCC algorithms of Tal & Vardy's model.

5.1 Introduction

An (N,K,$\mathcal{F}$) PCC comprises the selection of the best K out of N possible bit channels, according to their Bhattacharyya parameter values. Two problems arise in this process: (i) the PCC yields no universal codes, i.e. the structure of the polar code depends on the SNR at which the PCC algorithm is used; (ii) the *exact* error probability of the bit channels cannot be computed, except for the BEC. In this chapter, we focus on these two problems in the design of polar codes, for the BI-AWGN channel.

Parts of this chapter are based on [149–151]

Many practical applications require the use of the same code across a range of *operating-SNR*s. Due to the non-universality of polar codes [11, 101], the code designed at a particular SNR denoted as the *design-SNR* may not perform well across the range of operating-SNRs. In this chapter, we will first illustrate by simulations, how critical the choice of the design-SNR is for the performance of the resulting polar code. We will then choose the best design-SNR for each PCC. Some work towards designing *universal polar codes* can be found in [14, 152–154], however, the universality is typically achieved at the cost of additional complexity at encoder/decoder. We will therefore focus only on Arıkan's original polar codes.

Although a number of algorithms have been proposed in literature to estimate the bit channel reliabilities, it is not known which PCC algorithm can construct the best polar codes for a target BER performance. Filling this gap in literature forms the objective of the following study, while also addressing the problem of non-universality.

5.2 Overview of Literature on Polar Code Constructions

In this chapter, we focus on the four major PCCs denoted **PCC-0**, **PCC-1**, **PCC-2**, **PCC-3**. All the other heuristic constructions such as [155–157] are extensions of these methods.

- **PCC-0** [11, 39]: The bit channel reliabilities are recursively estimated through the Bhattacharyya parameters of various bit channels.
- **PCC-1** [11]: An alternative method is to use a Monte-Carlo simulation to estimate the Bhattacharyya parameters (see [11, Sec. IX]).
- **PCC-2** [101]: A more accurate PCC was proposed by Tal and Vardy [101], approximating the BI-AWGN bit channels as binary-input symmetric-output channels with finite output alphabet of size 2μ.
- **PCC-3** [104]: Another method based on Gaussian approximation (applicable to AWGN channels) is proposed in [104] and rediscovered in [105, 106].

We first provide a brief description of these four PCCs and then show a detailed comparison. We report the best design-SNRs for all the four PCCs. We find by extensive simulations that all four PCCs can result in equally performing polar codes at a target BER, provided that their design-SNR is appropriately chosen.

The earliest construction of polar codes is based on the evolution of simple bounds on the Bhattacharyya parameters of bit channels [39]. Though these bounds are proved in [11] only for BI-DMS channels, they may also be extended for infinite alphabet channels such as BI-AWGN. Although the bounds are loose for many of the bit channels and more accurate methods are devised later, the codes designed using such bounds exhibit good performance. This construction enjoys the least $\mathcal{O}(N)$ complexity among all (excluding the selection of K best among N metrics obtained). This includes $2N-1$ transformation operations corresponding to the transformation of upper bounds by polarizing kernel. This well-known method was also studied in a recent paper

[42]. We denote this construction algorithm by **PCC-0**.

Another earlier construction is proposed in [11], based on a Monte-Carlo simulation of the bit channels. This can be applied to a wide range of channels including finite and infinite alphabet channels such as AWGN. However, the algorithm has the greatest complexity $\mathcal{O}(MN \log N)$ among all, where M is the number of iterations of the Monte-Carlo simulation. We denote this construction algorithm by **PCC-1**.

A more recent construction algorithm was proposed by Tal and Vardy [101] based an earlier proposal from Mori and Tanaka [102, 103]. At first it was proposed to find the bit channels by evaluating of their full finite alphabet distributions. The algorithm becomes intractable due to the explosion of the alphabet size to a power of N by the end of n channel transformations. This problem is specifically addressed in [101] by employing a novel low complexity near-optimal quantizer. In addition, they provide theoretical guarantees for the loss of performance due to the quantization. Note that, some channels are better estimated by simple bounds on Bhattacharyya parameters [11, 39]. Hence, in [101] the authors conclude a final algorithm by improving the BER estimates with the Bhattacharyya parameters whenever they are better. The final algorithm is considered by far the most accurate construction algorithm available with theoretical guarantees. We denote this construction algorithm by **PCC-2**.

The algorithm is extendable to infinite output channels by using a quantization algorithm. In [101], the authors propose a quantization algorithm for AWGN channels. When the channel output has μ symbols, the final complexity of the algorithm is $\mathcal{O}(N \cdot \mu^2 \log \mu)$ (excluding the selection of K best among N metrics obtained). This contains $2N-1$ number of $\mathcal{O}(\mu^2)$ complexity bit channel convolutions plus $2N-1$ number of $\mathcal{O}(\mu^2 \log \mu)$ complexity quantizer operations. The quantizer uses an intelligent data-structure that combines a heap and a list. The initialization of the algorithm involves the AWGN channel quantization to μ symbols, which takes an additional $\mathcal{O}(\mu)$ complexity. Overall, this algorithm has the second largest complexity, only next to the earlier Monte-Carlo based construction algorithm by Arikan.

For AWGN channels, the estimation of bit channels based on Gaussian approximation is proposed in [104]. This enables to use the Gaussian distribution approximations on the intermediate likelihoods. This was found to approximate well the actual bit channels of polar codes [105, 106]. A similar algorithm after the original proposal in [104] was studied in [105, 106].

The Gaussian approximation algorithm has a complexity of $\mathcal{O}(N)$ function computations (excluding the selection of K best among N metrics obtained) similar to the Bhattacharyya bounds based algorithm, but involves relatively higher complexity function computations. Overall, this construction algorithm enjoys the second least complexity. We denote this construction algorithm by **PCC-3**.

There also exist several other heuristic constructions, extended or inspired from the above constructions, e.g. [155–157]. However, these methods are less interesting due to poorer performance, higher complexity, and having no theoretical guarantees. Finally, other constructions are also available for broadcast channels, Rayleigh-fading channels, deletion channels [158–160], for universal polar codes [152], for non-binary input alphabets [161], different kernels [162–164], and concatenated codes [113, 114], which fall out of the scope of this thesis.

5.3 The Channel Model

We consider a BI-AWGN channel for the illustration of various PCCs. The SNR is defined as E_b/N_0, where E_b denotes the energy per information bit and N_0 is the noise power spectral density. Using binary phase shift keying (BPSK) signaling, the coded bit $x \in \mathbf{x}$ is modulated as $\tilde{x}$, i.e., $0 \to -\sqrt{RE_b}$ and $1 \to +\sqrt{RE_b}$, where $RE_b = \frac{K}{N}E_b$ is the energy per coded bit, and the output of the channel is given by

$$y = \tilde{x} + n, \quad n \sim \mathcal{N}\left(0, \tfrac{N_0}{2}\right). \tag{5.1}$$

where $\mathcal{N}\left(0, \frac{N_0}{2}\right)$ denotes a Gaussian distribution with zero mean and variance $\frac{N_0}{2}$. The LLR of a received symbol is defined as:

$$L(y) \triangleq \log \frac{f_{Y|X}(y|x=0)}{f_{Y|X}(y|x=1)} = \frac{-4y\sqrt{RE_b}}{N_0} \tag{5.2}$$

where $f_{Y|X}(.)$ represents the pair of likelihood probability density functions in the above AWGN channel:

$$f_{Y|X}(y|x=0) \sim \mathcal{N}\left(-\sqrt{RE_b}, \tfrac{N_0}{2}\right) \text{ and} \tag{5.3}$$

$$f_{Y|X}(y|x=1) \sim \mathcal{N}\left(+\sqrt{RE_b}, \tfrac{N_0}{2}\right) \tag{5.4}$$

5.4 Description of the Main PCC Algorithms

In this section we review the four main PCCs (**PCC-0**, **PCC-1**, **PCC-2**, **PCC-3**) and provide their pseudocode implementations in the end of the section.

We noticed that a numerical underflow frequently occurs when we use double-precision floating-point numbers in the below algorithms, especially when $N \geq 256$. Therefore, it may be preferable to perform all the calculations in log-domain.

5.4.1 PCC-0: Bhattacharyya Bounds of Bit Channels

Arıkan proposed the earliest polar code construction (**PCC-0**) for symmetric binary-input discrete memoryless channels in [11, 39]. For $N = 2^n$, $n \geq 1$, and $1 \leq K \leq N$, an $(N, K, \mathcal{F})$ polar code is a block code with the generator matrix $\mathbf{G} = (\mathbf{F}^{\otimes n})_{\mathcal{F}^c}$, where the frozen bit indices $\mathcal{F}$ can be obtained according to the bit channel reliabilities, estimated recursively through the Bhattacharyya parameters. Specifically, after the j-th polarizing transform, $j = 1, \ldots, n$, there are 2^j bit channels with Bhattacharyya parameters $\mathbf{z}^{(j)} = \{z_p^{(j)}\}_{p=0}^{2^j-1}$, where the elements evolve recursively as,

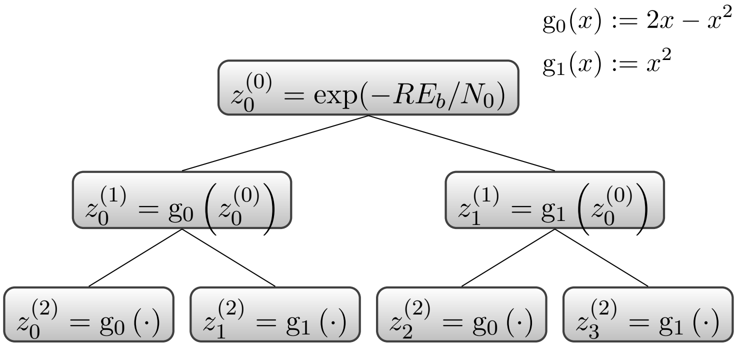


Figure 5.1: The first 2 iterations of the bit channel estimation under PCC-0 in an AWGN channel using Eqs. (5.5) and (5.6)

$$z_{2k}^{(j)} = g_0\left(z_k^{(j-1)}\right) \triangleq 2 \cdot z_k^{(j-1)} - \left(z_k^{(j-1)}\right)^2 \tag{5.5}$$

and

$$z_{2k+1}^{(j)} = g_1\left(z_k^{(j-1)}\right) \triangleq \left(z_k^{(j-1)}\right)^2 \tag{5.6}$$

for $k = 0, \dots, 2^{j-1} - 1$, with initial value $z_0^{(0)}$ equal to the Bhattacharyya parameter of the underlying channel. When $j = n$, the above recursion generates a N-dimensional vector $\mathbf{z}^{(n)} = [z_0^{(n)}, \dots, z_{N-1}^{(n)}]$, which is filled in according to the above recursion. These N values correspond to various bit channels, but in a bit-reversed order. We will now perform the following three simple operations, to complete the polar code construction.

1. $\mathbf{z}^{(n)} :=$ bitreversed $\left(\mathbf{z}^{(n)}\right)$, where bitreversed() represents the bit-reversal permutation of the elements of a vector.
2. Sort $\mathbf{z}^{(n)}$ in ascending order.
3. Find the indices of the last $N - K$ values (after step 1) and output as $\mathcal{F}$, which is the final output, representing the desired set of frozen bit indices.

Note that for a BI-AWGN channel, its Bhattacharyya parameter can be computed as $\exp(-RE_b/N_0)$ [105]. Setting the initial to be 0.5 yields $\frac{RE_b}{N_0} = -1.5917$dB, or equivalently $\frac{E_b}{N_0} = (-1.5917 - 10\log_{10} R)$ dB, which matches the worst-case initial from [39].

Also, it is known in general that the Bhattacharyya parameter is an upper bound on the error probability of bit channels and may not be accurate as an estimate of error probability. However, it provides a reasonable ranking of the bit channel quality. Our simulations in Section 5.6 support this claim as an affirmative.

The **PCC-0** algorithm is illustrated in Fig. 5.1, which represents a computational tree that extends until we have $N = 2^n$ leaf nodes, which are then subjected to the three simple operations above to complete the PCC-0. A complete pseudocode of PCC-0 is given as **Algorithm PCC-0**.

5.4.2 PCC-1: Monte-Carlo Estimation of Bit Channels

An alternative method, presented in [11], is to estimate the bit channel reliabilities using Monte-Carlo simulations. The basic idea is to simulate M decoding instances and use the expectation of the random variables in [11, Eq. (80)], i.e., the square root of the likelihood ratios, to estimate the Bhattacharyya parameters. In this method, when estimating i^{th} bit channel, one should force all earlier bits to be decoded correctly (genie-aided decoding). Note that we calculate BER of bit channels instead of their Bhattacharyya parameters using [11, Eq. (54) and (80)] and we simulate all-zero codeword transmission only. Using such a BER estimate, the overall BLER ([106, Eq. (3)]) is improved.

On the other hand, the disadvantage of the **PCC-1** is that its accuracy is severely limited by the number of Monte-Carlo iterations M. Specifically, some good bit channels with extremely low BER will have zero estimate using **PCC-1**. In particular, if there are more than K bit channels with a zero estimate, we need to increase the number of iterations M to improve the accuracy of the estimation. The corresponding pseudocode is also given.

5.4.3 PCC-2: Tal & Vardy's estimation of bit channel TPMs

In Arıkan's channel polarization, the bit channels can be constructed recursively using Arıkan's channel transformations $\mathcal{W} \boxplus \mathcal{W}$ and $\mathcal{W} \boxtimes \mathcal{W}$, where $\mathcal{W} : \mathcal{X} \to \mathcal{Y}$ is a BMS channel, and $\mathcal{X} = \{0, 1\}$. The transformations $\mathcal{W} \boxplus \mathcal{W}$ and $\mathcal{W} \boxtimes \mathcal{W}$ are given as follows [101].

$$\begin{aligned} \mathcal{W} \boxplus \mathcal{W}(y_0, y_1|d_0) &\triangleq \frac{1}{2}\Big\{\mathcal{W}(y_0|d_0)\mathcal{W}(y_1|0) + \mathcal{W}(y_0|d_0 \oplus 1)\mathcal{W}(y_1|1)\Big\} \\ &\qquad \forall y_0, y_1 \in \mathcal{Y} \text{ and } d_0 \in \mathcal{X} \end{aligned} \tag{5.7}$$

$$\begin{aligned} \mathcal{W} \boxtimes \mathcal{W}(y_0, y_1, d_0|d_1) &\triangleq \frac{1}{2}\mathcal{W}(y_0|d_0 \oplus d_1)\mathcal{W}(y_1|d_1) \\ &\qquad \forall y_0, y_1 \in \mathcal{Y} \text{ and } d_0, d_1 \in \mathcal{X} \end{aligned} \tag{5.8}$$

where the output alphabet of $\mathcal{W} \boxplus \mathcal{W}$ is $\mathcal{Y}^2$ and the output alphabet of $\mathcal{W} \boxtimes \mathcal{W}$ is $\mathcal{Y}^2 \times \mathcal{X}$. We note that each transform application squares the output alphabet size, and thus, applying many channel transformations incurs an explosion in the output alphabet size. The full pseudocode implementation of such channel polarizations are given as **ConvolveA()** and **ConvolveB()** (representing the blocks C_A and C_B in Fig. 5.2 respectively). To solve this problem, Tal & Vardy in [101] proposed a new PCC that approximates the bit channel with an intractably large alphabet by another channel with a manageable alphabet size.

The basic procedure of Tal & Vardy's PCC [101] for a BI-AWGN channel may be explained in three steps as follows.

Step 1. For a BI-AWGN channel, a quantization algorithm is used to quantize the infinite output alphabet to size 2μ to initialize the PCC. Specifically, the quantization aims at finding μ

consecutive semi-open intervals of the positive real line $\mathbb{R}^+$ (sufficient due to symmetry) and the intervals are the pre-images of μ uniform partitions of $[0,1]$, which is the *range* of instantaneous capacity function, given by

$$\mathbf{C}[\lambda(y)] = 1 - \log_2(1+\lambda) + \frac{\lambda}{\lambda+1}\log_2\lambda, \tag{5.9}$$

where y is the output of the channel and

$$\lambda(y) \triangleq \frac{f_Y(y|0)}{f_Y(y|1)}. \tag{5.10}$$

is the likelihood ratio of y with the conditional probability densities $f_Y(y|0)$ and $f_Y(y|1)$. For a BI-AWGN channel, after normalizing the noise variance in (5.1), the conditional probability densities are given by

$$f_Y(\cdot|0) \sim \mathcal{N}\left(-\sqrt{\tfrac{2RE_b}{N_0}}, 1\right), \quad f_Y(\cdot|1) \sim \mathcal{N}\left(\sqrt{\tfrac{2RE_b}{N_0}}, 1\right)$$

with the corresponding likelihood ratio

$$\lambda(y) = \exp\left(-2y\sqrt{\tfrac{2RE_b}{N_0}}\right). \tag{5.11}$$

We then obtain $\mu+1$ consecutive points $(a_0 = 0) < a_1 < \cdots < a_{\mu-1} < (a_\mu = \infty)$ by solving

$$\mathbf{C}[\lambda(a_j)] = \tfrac{j}{\mu}, \ \forall j = 0, 1, \ldots, \mu-1, \tag{5.12}$$

where $\{a_j\}$ are the points that partition the entire positive real line into μ intervals, corresponding to μ quantized symbols.

Then the entries of the $2 \times \mu$ dimensional transition probability matrix (TPM) $\mathbf{P} = [p_{ji}]$ of the quantized channel (with an input alphabet $\{0,1\}$ and an output alphabet $\{0,1,\ldots,\mu-1\}$) can be obtained as

$$\begin{aligned}
p_{ji} &\triangleq \Pr(j \text{ is received } \mid i \text{ is transmitted}) \\
&= \int_{a_j}^{a_{j+1}} f_Y(y|i)\, dy \\
&= \begin{cases} Q\left(\sqrt{\tfrac{2RE_b}{N_0}} + a_j\right) - Q\left(\sqrt{\tfrac{2RE_b}{N_0}} + a_{j+1}\right) \text{ if } i = 0 \\ Q\left(-\sqrt{\tfrac{2RE_b}{N_0}} + a_j\right) - Q\left(-\sqrt{\tfrac{2RE_b}{N_0}} + a_{j+1}\right) \text{ if } i = 1 \end{cases}
\end{aligned} \tag{5.13}$$

and

$$Q(x) \triangleq \tfrac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-x^2/2) dx.$$

A full pseudocode implementation of such a quantization algorithm is given as **discretizeAWGN()**, representing the block Q_1 in Fig. 5.2.

Given a BI-AWGN channel W, the function $\mathbf{C}[\lambda(y)]$ is used to compute channel capacity $I(W) = \int_0^\infty (f_Y(y|0) + f_Y(y|1))\mathbf{C}[\lambda(y)]\, \mathrm{d}y$.

Step 2. Once TPM is generated, channel polarization of (5.7) and (5.8) can be performed (see the corresponding pseudocode **ConvolveA()** and **ConvolveB()** representing the blocks C_A and C_B in Fig. 5.2, respectively).

Step 3. As discussed above, given a BI-DMS with output alphabet size 2μ, each channel polarization results in the output alphabet size of $2\mu^2$ or $4\mu^2$, respectively. Then a BI-DMS quantization (merging) algorithm is used to bring its output-alphabet size back to 2μ, while preserving the bit channel quality and minimizing the loss of the capacity incurred by this operation. This quantization algorithm requires a special data structure which is a combination of a *heap* and a *linked list*, i.e., *heaplist.* Its usage is explained as follows.

Assuming a BMS channel with output size $L > 2\mu$, a heaplist allows us to efficiently perform the quantization to reduce its alphabet size to 2μ. It holds L symbol probabilities (columns) from the TPM as a list and $L-1$ loss-of-capacity values as a heap. The loss-of-capacity values result from the symbol merger operation, which performs only on the two consecutive symbols in the list. Given that each column in the list represents a symbol, the symbol merger operation can be interpreted as replacing the two columns by a single column equal to their sum, corresponding to one new symbol. Thus each merger reduces the size of the heaplist by one *element.*

The pseudocode implementation of such an algorithm is given as **Quantize_to_size()**, representing the block Q_2 in Fig. 5.2, with the following three main operations:

- *initialize_heaplist*(): Given a $2 \times L$ TPM, initialize the list part of the heaplist with its columns in the increasing order of the associated likelihoods, all ≥ 1. This sorting operation has $\mathcal{O}(L \log L)$ complexity. The heap part of the data-structure is stored with the $L-1$ values of loss in capacity when two consecutive symbols in the list are merged.
- *minloss_index*(): Find the value of minimum loss-of-capacity, and return the index of the column in the list attached to the minimum loss. The index indicates the optimal symbol merger.
- *merge_at_index*(): Perform the corresponding symbol merger operation and update the heaplist by this new element, while at the same time reduce the size of the heap and the list by one element.
- Miscellaneous:
  size() – to know the number of columns of the TPM currently maintained within the heaplist.
  $TPM()$ – to extract the TPM from heap, in a two-row matrix format.
  has_duplicates() – true or false based on whether there are columns in list with same likelihood ratio.

In summary, this heaplist enables a low complexity $\mathcal{O}(\log L)$ operation for extracting the minimum of the loss-of-capacity values in heap, while simultaneously maintaining the list of L symbols in a sorted order of likelihoods. It thus aids a very efficient implementation of the corresponding quantization operation at each iteration.

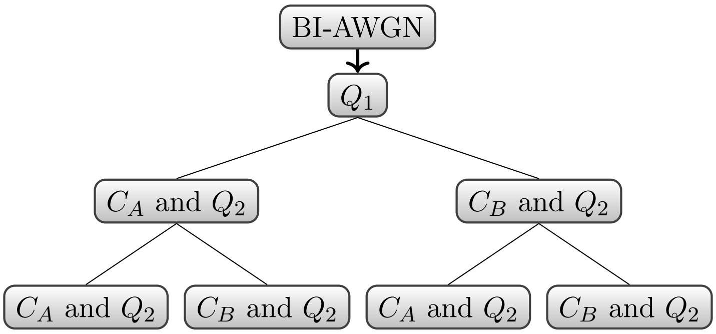


Figure 5.2: The first 2 iterations of the bit channel estimation under PCC-2 in a BI-AWGN channel

The overall algorithm including the earlier three steps may be summarized as a simple tree diagram in Fig. 5.2, where steps 2 and 3 are repeated multiple times till we have estimated all the bit channels. Similar to PCC-0, the algorithm terminates after completing n levels (iterations) of the tree where we have N leaves in the tree, representing various bit channels in bit-reversed order. A complete pseudocode implementation of the same is given as **Algorithm PCC-2**.

5.4.4 PCC-3: Trifonov's Gaussian Approximation of Bit Channels

Trifonov proposed the Gaussian approximation for the PCC in [104], and later similar PCCs were independently proposed in [105, 106].

The core idea is to estimate the log-likelihood ratios (LLRs) of various bit channels as Gaussian random variables (conditioned on input) with different mean values, similar to the case of an uncoded transmission in a BI-AWGN under BPSK modulation and optimal decoding. We can then simply compare the BERs and choose the set $\mathcal{F}$ corresponding to the worst $N-K$ bit channels.

Recall that, in a BI-AWGN channel $\mathcal{W}$, if μ is the mean-LLR (using (5.2)), conditioned on input $'0'$ under BPSK signalling, the bit channel's BER under ML decoding is given by:

$$P_e(\mathcal{W}) = Q\left(\sqrt{\frac{\mu}{2}}\right) \tag{5.14}$$

where, $Q(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\eta^2/2}\,\mathrm{d}\eta$. Note that the probability of error is inversely proportional to the mean μ and therefore μ is a sufficient statistic to compare two distinct bit channels.

Starting with the original BI-AWGN channel, we will now discuss how to recursively calculate the mean LLRs after each Arıkan's polarizing transformation.

Let us denote the mean values $\mu_k^{(j)} \triangleq \mathbb{E}\left[L_k^{(j)}\right]$, where $\left\{L_k^{(j)} : 0 \leq k \leq 2^j - 1, j = 1, \ldots, n\right\}$, are the LLRs after j likelihood transformations. The initial $\mu_0^{(0)}$ corresponds to the original BI-AWGN, therefore $\mu_0^{(0)} = \frac{4RE_b}{N_0}$. It was shown in [106, 165] that after each Arıkan's polarizing

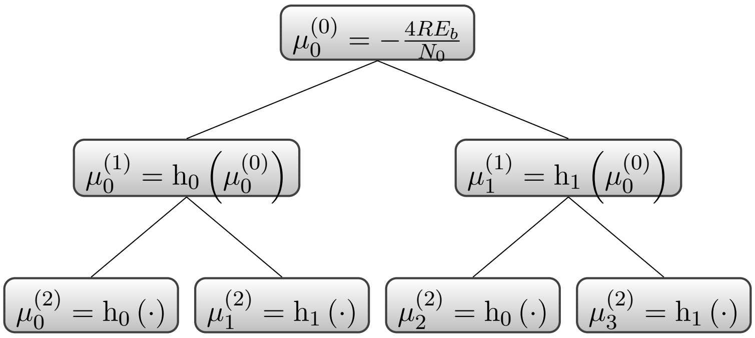

$\mu_0^{(0)} = -\frac{4RE_b}{N_0}$
$\mu_0^{(1)} = h_0\left(\mu_0^{(0)}\right)$
$\mu_1^{(1)} = h_1\left(\mu_0^{(0)}\right)$
$\mu_0^{(2)} = h_0(\cdot)$
$\mu_1^{(2)} = h_1(\cdot)$
$\mu_2^{(2)} = h_0(\cdot)$
$\mu_3^{(2)} = h_1(\cdot)$

Figure 5.3: The first 2 iterations of the bit channel estimation under PCC-3 in a BI-AWGN channel using Eqs. (5.15) and (5.16)

transformation, a bit channel LLR $\mu_k^{(j-1)}$ transforms to two distinct values as:

$$\mu_{2k}^{(j)} = \mathrm{h}_0(\mu_k^{(j-1)}) \triangleq \phi^{-1}\left(1 - \left(1 - \phi\left(\mu_k^{(j-1)}\right)\right)^2\right) \tag{5.15}$$

$$\mu_{2k+1}^{(j)} = \mathrm{h}_1(\mu_k^{(j-1)}) \triangleq 2\mu_k^{(j-1)} \tag{5.16}$$

where,

$$\phi(x) \triangleq 1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{\infty} \tanh(\tau/2)\exp\left(-\frac{(\tau - x)^2}{4x}\right) d\tau$$

which in turn can be computed efficiently using the following approximation

$$\phi(x) \approx \begin{cases} \exp(-0.4527x^{0.86} + 0.0218), \text{ if } 0 < x \leq 10 \\ \sqrt{\frac{\pi}{x}}(1 - \frac{10}{7x}) \exp(-x/4), \text{ if } x > 10 \end{cases} \tag{5.17}$$

Due to the unavailability of a closed-form expression in the range $x > 10$, a bisection method may be employed in this range to efficiently compute the values of ϕ^{-1}.

The overall algorithm PCC-3 can be represented as a simple binary tree in Fig. 5.3. We will stop after n iterations, where we observe N mean values corresponding to N bit channels but in bit-reversed order. Denote $\mathbf{z} \triangleq [\mu_0^{(n)}, \mu_1^{(n)}, \ldots, \mu_{N-1}^{(n)}]$. Recall that due to underlying Gaussian approximation and the corresponding (5.14), a greater value of mean represents a lower BER. Therefore, we may perform the following three simple operations to complete the polar code construction.

Step 1 $\mathbf{z} :=$ bitreversed $\left(\mathbf{z}^{(n)}\right)$, where bitreversed() represents the bit-reversal permutation of the elements of a vector.

Step 2 Sort $\mathbf{z}^{(n)}$ in descending order remembering the original order.

Step 3 Find the original indices of the last $N - K$ values and output as $\mathcal{F}$, which is the final output, representing the desired set of frozen bit indices.

A complete pseudocode implementation of PCC-3 is given as **Algorithm PCC-3**. The computation of ϕ and its inverse should be performed in log-domain to avoid an underflow.

5.5 Selecting the Best Design-SNR

The basic procedure of selecting the appropriate design-SNR is summarized below.

1. Fix a PCC algorithm and a target-BER value.
2. Consider a set of design-SNRs $\{S_1, S_2, \ldots, S_m\}$.
3. Design m polar codes at design-SNRs of S_i, $i = 1, \ldots, m$ (i.e. find an $\mathcal{F}$ at each S_i), using the given construction method.
4. For each of the m polar codes, find the operating-SNR at which the target-BER is achieved.
5. Select the polar code that achieves the target-BER at the minimum operating-SNR. Output the corresponding design-SNR as the best design-SNR.

5.6 The Comparison of PCCs at Finite Length

In this section, we first illustrate the non-universal behaviour of the polar codes generated by the four PCCs over BI-AWGN channels. We then report the best design-SNRs and operating-SNRs of the four PCCs with various codeword lengths N and code rates R. Finally, we compare the BER performance of the four PCCs, given their design-SNRs are appropriately chosen. We consider SCD in all simulations.

We illustrate from Figs. 5.4 to 5.7 the non-universal behaviour of the polar codes generated by the four PCCs at $R = 0.5$ with $N = 2048$. We observe that, at a given operating-SNR, the BERs of the polar codes, generated by the same PCC algorithm but at different design-SNRs, vary significantly, while at a given design-SNR, these PCCs produces different polar codes with various error performance. We also observe that the sensitivity of the BER performance against the choice of design-SNR, varies among different PCCs. In particular, in Fig. 5.7, we notice that the BER of the polar codes generated by PCC-3 varies the most with the design-SNRs. All these observations motivate us to find the best design-SNRs for the four PCCs.

According to the BER curves in Figs. 5.4 to 5.7, we follow the procedure in Section 5.5 to select the appropriate design-SNRs. Using the four PCCs, given the target-BER of 10^{-4}, we reported the best design-SNRs and the corresponding operating-SNRs in Table 5.1, for the polar codes of rates 0.5 and 0.9 with codeword lengths $N = 2^{10}, 2^{11}, 2^{13}, 2^{16}$.

After choosing the best design-SNRs for the four PCCs, we compare the BER performance of the corresponding polar codes in Fig. 5.8 at $R = 0.5$ with $N = 2048$. We find that the four PCC algorithms are equally good. This demonstrate that the ranking of the qualities of various bit channels remains intact in all the PCC algorithms, when an appropriate design-SNR is chosen. Note that, in PCC-0, although Bhattacharyya parameter is only an upper bound on the error performance of the bit channel, the ranking of bit channel quality is good enough to generate the equally efficient polar codes, when compared with other PCCs.

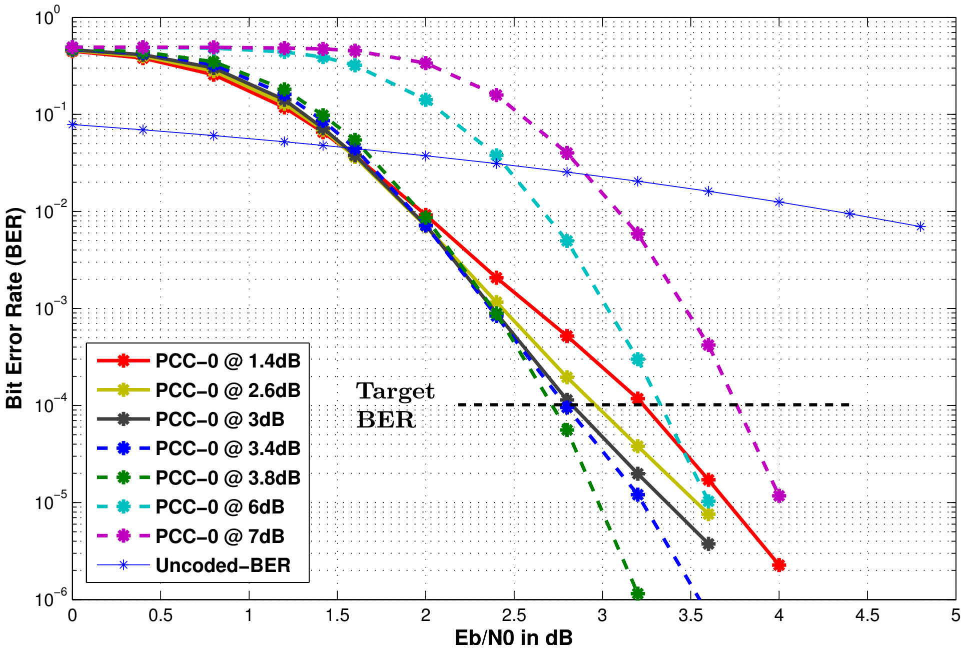


Figure 5.4: The BER performance of polar codes under PCC-0 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$

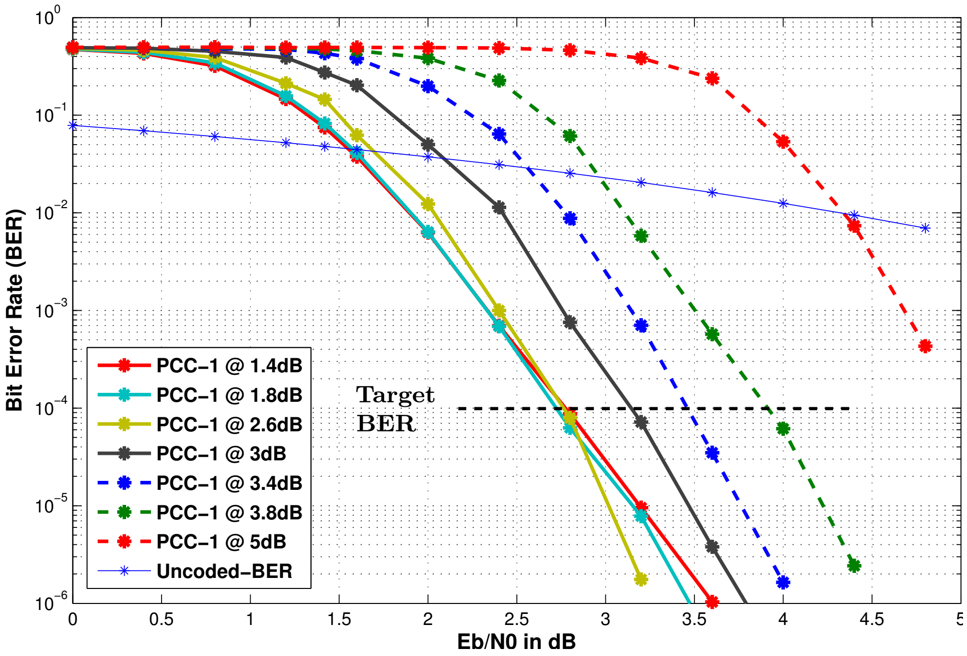


Figure 5.5: The BER performance of polar codes under PCC-1 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$

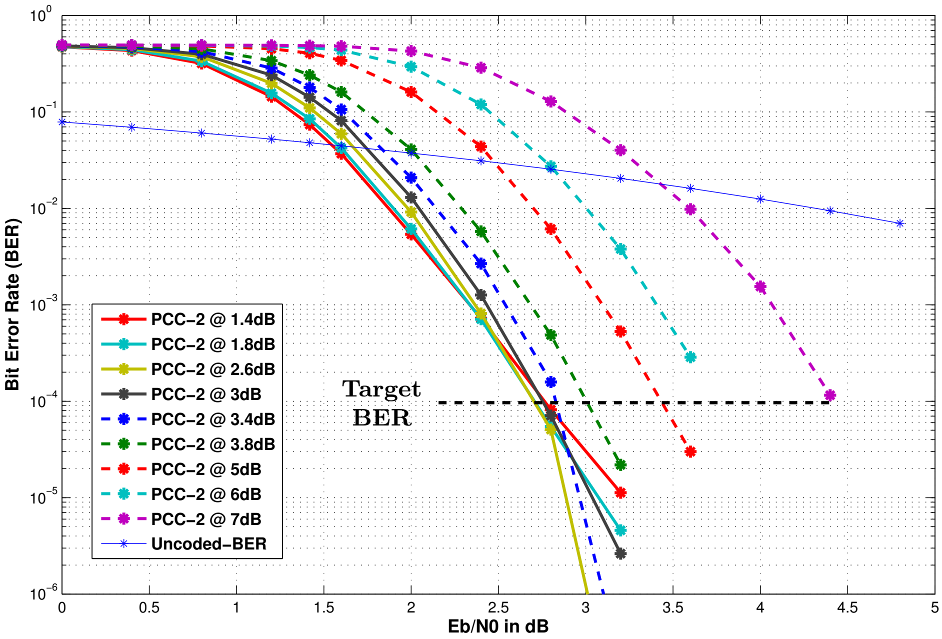


Figure 5.6: The BER performance of polar codes under PCC-2 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$

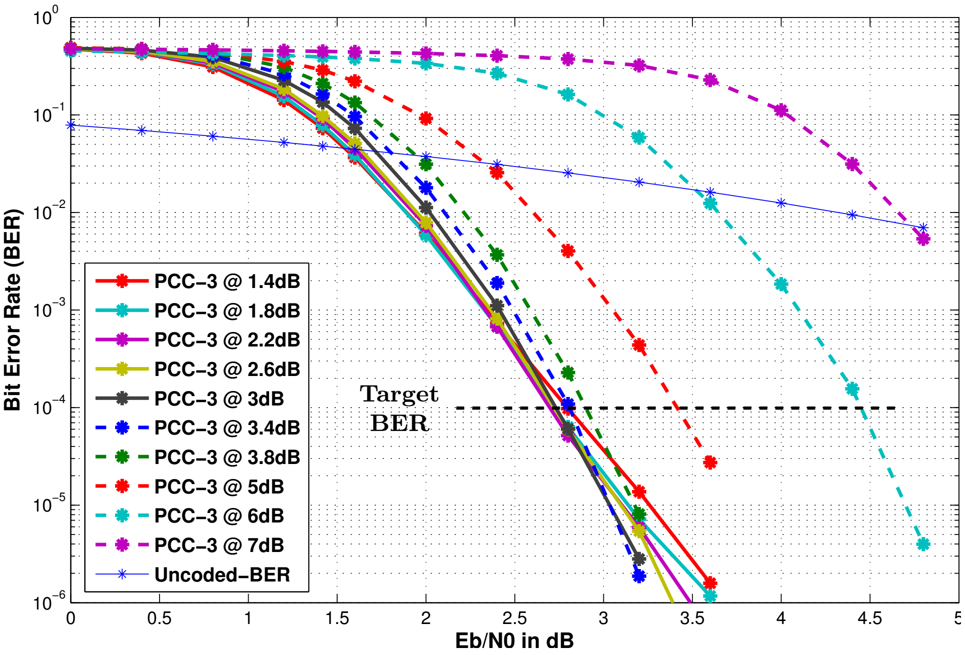


Figure 5.7: The BER performance of polar codes under PCC-3 algorithms and design-SNR values. $N = 2048$ and $R = 0.5$

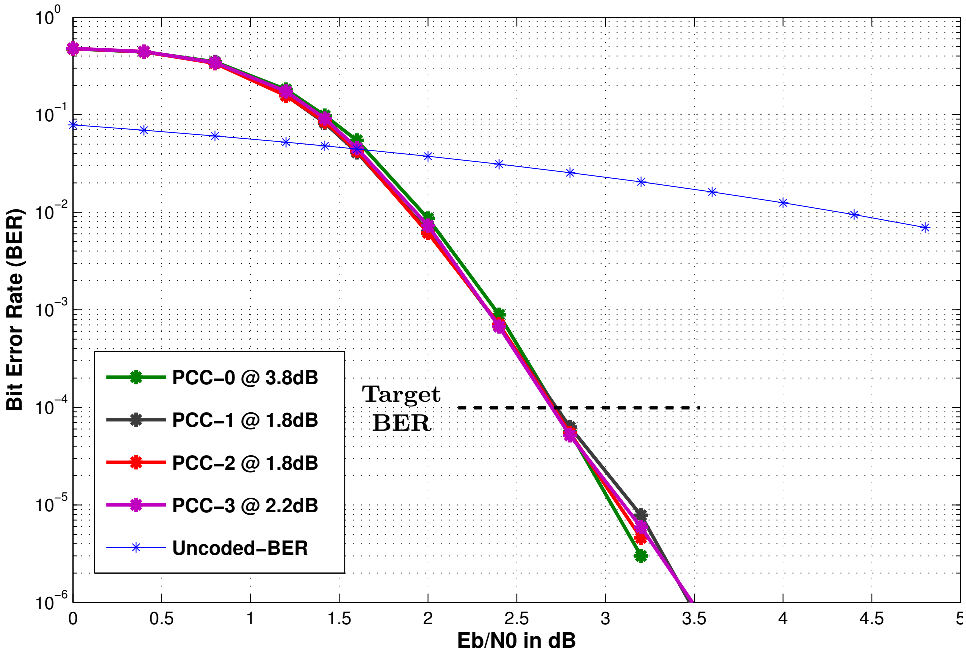


Figure 5.8: Comparison of all the constructions when design-SNRs are chosen according to the search algorithm ($N = 2048$ and $R = 0.5$)

		$\mathbf{N = 2^{10}}$		$\mathbf{N = 2^{11}}$		$\mathbf{N = 2^{13}}$		$\mathbf{N = 2^{16}}$	
		d-SNR	**op-SNR**	**d-SNR**	**op-SNR**	**d-SNR**	**op-SNR**	**d-SNR**	**op-SNR**
R = 0.5	**PCC-0**	3.8 dB	3.1 dB	3.8 dB	2.7 dB	3.0 dB	2.2 dB	2.6 dB	1.6 dB
	PCC-1	2.2 dB	3.1 dB	1.8 dB	2.7 dB	1.8 dB	2.2 dB	1.0 dB	1.5 dB
	PCC-2	2.2 dB	3.1 dB	1.8 dB	2.7 dB	1.4 dB	2.2 dB	1.4 dB	1.5 dB
	PCC-3	1.8 dB	3.1 dB	2.2 dB	2.7 dB	2.2 dB	2.2 dB	2.2 dB	1.8 dB
R = 0.9	**PCC-0**	5.2 dB	5.5 dB	5.2 dB	5.2 dB	4.8 dB	4.8 dB	4.4 dB	4.3 dB
	PCC-1	10 dB	5.5 dB	10 dB	5.2 dB	9.0 dB	4.8 dB	7.6 dB	4.3 dB
	PCC-2	4.4 dB	5.5 dB	4.0 dB	5.2 dB	4.0 dB	4.8 dB	4.0 dB	4.3 dB
	PCC-3	4.4 dB	5.5 dB	4.8 dB	5.2 dB	4.4 dB	4.8 dB	4.0 dB	4.2 dB

Table 5.1: Optimal design-SNR and operating-SNR values for various combinations of N and R and PCC algorithm. (The target-BER=10^{-4})

5.7 Pseudocodes for Implementing the PCCs

Below are the four algorithms and their associative functions for implementing the four PCC algorithms that we discussed so far. Our pseudocodes present the logic in such a way that it is implementation friendly. The grouping of the algorithms is as follows.

1. Algorithm **PCC-0**, and function **indices_of_greatest_elements()**
2. Algorithm **PCC-1**, and function **UpdateL()**
3. Algorithm **PCC-2**, and functions **discretizeAWGN()**, **ConvolveA()**, **ConvolveB()**, **Quantize_to_size()**
4. Algorithm: **PCC-3**, and function **indices_of_least_elements()**

Algorithm 15 PCC-0: The Bhattacharyya bounds

```
INPUT   : $N$, $K$, and $EdB$ = the design-SNR ($E_b/N_0$) in dB
OUTPUT : $\mathcal{F} \subset \{0, 1, \ldots, N-1\}$ with $|\mathcal{F}| = N - K$
$n = \log_2 N$, $R = K/N$, and $S = R \cdot 10^{EdB/10}$
$\mathbf{z} \in \mathbb{R}^N$, initialize $\mathbf{z}[0] = \exp(-S)$
for $j = 1 : n$ do                         ▷ For each of $n$ stages in Fig. 2.2, right-to-left
    $u = 2^j$
    for $k = 0 : \frac{u}{2} - 1$ do            ▷ For each vertical connection
        $T = \mathbf{z}[k]$
        $\mathbf{z}[k] = 2T - T^2$                         ▷ (5.5)
        $\mathbf{z}[u/2 + k] = T^2$                        ▷ (5.6)
    end
end
$\mathcal{F}$ = indices_of_greatest_elements$\Big(\mathbf{z}, N - K\Big)$
    // Find indices of the greatest $N - K$ elements
Return $\mathcal{F}$
```

Function 16 indices_of_greatest_elements($\mathbf{v}$, l)

```
INPUT   : $\mathbf{v}$ – a $|\mathbf{v}| \times 1$ vector; $l$ – an integer, $l \leq |\mathbf{v}|$
OUTPUT : $\mathcal{I}$ – an $l \times 1$ integer vector containing $l$ indices of $\{0, 1, \ldots, |\mathbf{v}| - 1\}$
$[\mathbf{v}, idx]$ = Sort($\mathbf{v}$, 'descending')
    //obtain in $idx$, the $|\mathbf{v}|$ indices of vector $\mathbf{v}$ when sorted in descending order
$\mathcal{I} = idx[0 : l - 1]$        ▷ Store the first $l$ indices
Return $\mathcal{I}$
    // Note: This is a simple implementation is sufficient for
    // $|\mathbf{v}|$ up to a few thousands. For optimal performance, one should
    // use more advanced selection algorithms[166, 167] that are $\mathcal{O}(N)$.
```

Algorithm 17 PCC-1: The Monte-Carlo estimation

```
INPUT  : $N$, $K$, $EdB$ = the design-SNR ($E_b/N_0$) in dB,
         $M$ = Monte-Carlo size, $randn()$ — a standard
         Gaussian pseudo random number generator
OUTPUT : $\mathcal{F} \subset \{0, 1, \ldots, N-1\}$ with $|\mathcal{F}| = N - K$
Allocate and make visible to the other functions $N \times (n+1)$ matrices $\mathbf{B}$, $\mathbf{L}$ and set $\mathbf{B} = 0$
Bit-channel metrics $\mathbf{c} \in \mathbb{R}^N$, initialize $\mathbf{c} = 0$
$n = \log_2 N$, $R = K/N$, and $S = R \cdot 10^{EdB/10}$
for $t = 1 : M$ do
    $\mathbf{y} = -\sqrt{2S} + randn(N, 1)$                    ▷ Normalized AWGN, all-zero
    $\mathbf{L} =$ NaN                                          ▷ Reset $\mathbf{L}$
    Initialize the last column of $\mathbf{L}$:
        $\mathbf{L}[j][n] = \Pr(y_j|0)/\Pr(y_j|1) = \exp\left(-2y_j\sqrt{2S}\right) \ \forall j$
    for $i = 0, 1, \ldots, N-1$ do                              ▷ The SCD with all frozen
        $l = \mathbf{bitreversal}(i)$
        $\mathbf{UpdateL}(l, 0)$                                ▷ Update $\mathbf{L}$, esp. $\mathbf{L}[l][0]$
        $\hat{\mathbf{d}}[l] = \begin{cases} 0, & \text{if } \mathbf{L}[l][0] \geq 1 \\ 1, & \text{else} \end{cases}$
    end
    $\mathbf{c} = \mathbf{c} + \hat{\mathbf{d}}$                ▷ Real addition
end
$\mathbf{z} = \mathbf{c}/M$                  ▷ Monte-Carlo averaging for BERs (not necessary)
$\mathcal{F} = \mathbf{indices\_of\_greatest\_elements}\left(\mathbf{z}, N-K\right)$
  // Find indices of the highest $N-K$ elements
return $\mathcal{F}$
```

Function 18 UpdateL(i, j) : Recursive LR comp. of SCD

```
INPUT  : Element indices $i, j$
OUTPUT : Recursively updated matrix $\mathbf{L}$
$u = 2^{n-j}$ and $l = (i \bmod u)$
if $l < u/2$ then                                   ▷ Upper branch of a unit circuit
    if ($\mathbf{L}[i][j+1] =$ NaN ) then
        $\mathbf{UpdateL}(i, j+1)$; end;
    if ($\mathbf{L}[i+u/2][j+1] =$ NaN ) then
        $\mathbf{UpdateL}(i+u/2, j+1)$; end;
    $\mathbf{L}[i][j] = \frac{\mathbf{L}[i][j+1]\mathbf{L}[i+u/2][j+1] + 1}{\mathbf{L}[i][j+1] + \mathbf{L}[i+u/2][j+1]}$
else                                                ▷ Lower branch of a unit circuit
    if $\mathbf{B}[i-u/2][j] = 0$ then
        $\mathbf{L}[i][j] = \mathbf{L}[i][j+1]\mathbf{L}[i-u/2][j+1]$
    else
        $\mathbf{L}[i][j] = \frac{\mathbf{L}[i][j+1]}{\mathbf{L}[i-u/2][j+1]}$
    end
end
```

Algorithm 19 PCC-2: Tal & Vardy's $\Big($See Fig. 5.2$\Big)$

```
INPUT   : $N$, $K$, and $EdB$ = the design-SNR ($E_b/N_0$) in dB;
OUTPUT: $\mathcal{F} \subset \{0, 1, \ldots, N-1\}$ with $|\mathcal{F}| = N - K$
Allocate channels, an array of $N$ TPMs, of dimension $2 \times \mu$. $ch1, ch2$ hold intermediate TPMs.
$n = \log_2 N$, and $R = K/N$
$channels[0] = discretizeAWGN(\mu, EdB - 10\log_{10} R)$    //Initialized.
for $j = 1 : n$ do                                   ▷ for each stage in Fig. 2.2, right-to-left
    $u = 2^j$
    for $t = 0 : \frac{u}{2} - 1$ do
        $ch1$ = ConvolveA($channels[t]$)
        $ch2$ = ConvolveB($channels[t]$)
        $channels[t]$ = Quantize_to_size($ch1, \mu$)
          $channels[u/2 + t]$ = Quantize_to_size($ch2, \mu$)
    end
end
Allocate $\mathbf{z}$, a real vector dimension $N \times 1$
for $i = 0 : N - 1$ do
    $\mathbf{z}[i] = 0$
    $\mu'$ = no. of columns in $channels[i]$
    for $j = 0 : \mu' - 1$ do
        $\mathbf{z}[i] = \mathbf{z}^{(2)}[i] + (channels[i])[1][j]$
    end
end
$\mathcal{F}$ = indices_of_greatest_elements$\Big(\mathbf{z}, N - K\Big)$   // Find indices of the greatest $N - K$
  elements
Return $\mathcal{F}$
```

Function 20 discretizeAWGN(): The block Q_1

```
INPUT  : $\mu$ – the (integer) size of the quantized AWGN channel, $EdB$ = design-SNR ($E_b/N_0$)
         of AWGN channel in dB, and $R$ – the code rate
OUTPUT : TPM $\mathbf{P}$ of size $2 \times \mu$
$S = R \cdot 10^{EdB/10}$
$\lambda(y) \triangleq \exp\left(-2y\sqrt{2S}\right)$ and                                   ▷ (5.11)
$C(x) \triangleq 1 - \log_2(1+x) + x\log_2(x)/(1+x)$                                      ▷ (5.9)
$a \in \mathbb{R}^N$, initialize $a[0] = 0$ and $a[\mu] = \infty$
for $j = 1 : \mu - 1$ do
    $a[j] = \text{solve}\{C(\lambda(y)) = j/\mu\}$
end
for $j = 0 : \mu - 1$ do                                                                  ▷ Eq. (5.13)
    $\mathbf{P}[0][j] = Q\left(\sqrt{2S} + a[j]\right) - Q\left(\sqrt{2S} + a[j+1]\right)$
    $\mathbf{P}[1][j] = Q\left(-\sqrt{2S} + a[j]\right) - Q\left(-\sqrt{2S} + a[j+1]\right)$
end
Return $\mathbf{P}$
```

Function 21 ConvolveA(): The block C_A in Fig. 5.2

```
INPUT  : $\mathbf{P}$ – A $(2 \times \mu)$ TPM to be convolved by itself
OUTPUT : $\mathbf{Q}$ – A new TPM of size $2 \times \mu'$, $\mu' \leq \mu(\mu+1)/2$
Allocate $\mathbf{Q} \in \mathbb{R}^{2\times\mu(\mu+1)/2}$ and initialize $idx = -1$
for $i = 0 : \mu - 1$ do
    $idx = idx + 1$
    $\mathbf{Q}[0][idx] = (\mathbf{P}[0][i]^2 + \mathbf{P}[1][i]^2)/2$, and $\mathbf{Q}[1][idx] = \mathbf{P}[0][i]\mathbf{P}[1][i]$
    for $j = i + 1 : \mu - 1$ do
        $idx = idx + 1$
        $\mathbf{Q}[0][idx] = \mathbf{P}[0][i]\mathbf{P}[0][j] + \mathbf{P}[1][i]\mathbf{P}[1][j]$, and $\mathbf{Q}[1][idx] = \mathbf{P}[0][i]\mathbf{P}[1][j] + \mathbf{P}[1][i]\mathbf{P}[0][j]$
        if $(\mathbf{Q}[0][idx] < \mathbf{Q}[1][idx])$ then swap($\mathbf{Q}[0][idx]$, $\mathbf{Q}[1][idx]$) end;
    end
end
Merge the columns of $\mathbf{Q}$ with same LR (add columns)
Return $\mathbf{Q}$
```

Function 22 ConvolveB(): The block C_B in Fig. 5.2

```
INPUT  : $\mathbf{P}$ – A $(2 \times \mu)$ TPM to be convolved by itself
OUTPUT : $\mathbf{Q}$ – A new TPM of size $2 \times \mu'$, $\mu' \leq \mu(\mu+1)$
Allocate $\mathbf{Q} \in \mathbb{R}^{2\times\mu(\mu+1)}$ and initialize $idx = -1$
for $i = 0 : \mu - 1$ do
    $idx = idx + 1$
    $\mathbf{Q}[0][idx] = \mathbf{P}[0][i]^2/2$, and $\mathbf{Q}[1][idx] = \mathbf{P}[1][i]^2/2$
    $idx = idx + 1$
    $\mathbf{Q}[0][idx] = \mathbf{P}[0][i]\mathbf{P}[1][i]$, and $\mathbf{Q}[1][idx] = \mathbf{Q}[0][idx]$
    for $j = i + 1 : \mu - 1$ do
        $idx = idx + 1$
        $\mathbf{Q}[0][idx] = \mathbf{P}[0][i]\mathbf{P}[0][j]$, and $\mathbf{Q}[1][idx] = \mathbf{P}[1][i]\mathbf{P}[1][j]$
        if $(\mathbf{Q}[0][idx] < \mathbf{Q}[1][idx])$ then swap($\mathbf{Q}[0][idx]$, $\mathbf{Q}[1][idx]$); end
        $idx = idx + 1$
        $\mathbf{Q}[0][idx] = \mathbf{P}[0][i]\mathbf{P}[1][j]$, and $\mathbf{Q}[1][idx] = \mathbf{P}[1][i]\mathbf{P}[0][j]$
        if $(\mathbf{Q}[0][idx] < \mathbf{Q}[1][idx])$ then swap($\mathbf{Q}[0][idx]$, $\mathbf{Q}[1][idx]$); end
    end
end
Merge the columns of $\mathbf{Q}$ with same LR (add columns)
Return $\mathbf{Q}$
```

Function 23 Quantize_to_size(): The block Q_2 in Fig. 5.2

```
INPUT  : $\mathbf{P}$ – a TPM of size $2 \times L$; $\mu$ – quantization size, $\mu < L$
OUTPUT : H – a quantized TPM of size $2 \times \mu', \mu' \leq \mu$
Create a heaplist object $H$ and initialize with $\mathbf{P}$
while $H.size() > \mu$ or $H.has\_duplicates()$ do
    $idx = H.minloss\_index()$, $H.merge\_at\_index(idx)$
end
Return $H.TPM()$
```

Algorithm 24 PCC-3: The Gaussian approximation

```
INPUT    : $N$, $K$, and $EdB$ = the design-SNR ($E_b/N_0$) in dB
OUTPUT : $\mathcal{F} \subset \{0, 1, \ldots, N-1\}$ with $|\mathcal{F}| = N - K$
$n = \log_2 N$, $R = K/N$, and $S = R \cdot 10^{EdB/10}$
$\mathbf{z} \in \mathbb{R}^N$, initialize $\mathbf{z}[0] = 4S$
for $j = 1 : n$ do                                  ▷ for each stage in Fig. 2.2, right-to-left
    $u = 2^j$
    for $k = 0 : \frac{u}{2} - 1$ do
        $T = \mathbf{z}[k]$
        $\mathbf{z}[k] = \phi^{-1}\left(1 - (1 - \phi(T))^2\right)$          ▷ (5.15)
        $\mathbf{z}[u/2 + k] = 2T$                                  ▷ (5.16)
    end
end
$\mathcal{F} = $ indices_of_least_elements$\Big(\mathbf{z}, N - K\Big)$
   // Find indices of the least $N - K$ elements
Return $\mathcal{F}$
```

Function 25 indices_of_least_elements($\mathbf{v}$, l)

```
INPUT    : Vector $\mathbf{v}$ of dimension $|\mathbf{v}| \times 1$ and integer $l$
OUTPUT : An $l \times 1$ integer vector containing $l$ indices in $\{0, 1, \ldots, |\mathbf{v}| - 1\}$
Return $\Big($indices_of_greatest_elements$(-\mathbf{v},\ l)\Big)$
```

5.8 A New Optimal Quantizer of BI-DMC with Optimal Mutual Information Using Constrained Shortest Path Problem

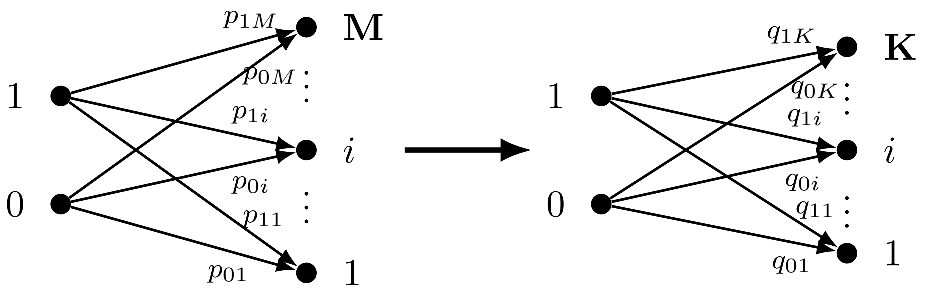
5.8.1 Quantizers for Polar Code Construction

Tal & Vardy [101] proved that by using appropriate quantizer as a unit, one can complete the construction in $\mathcal{O}(N)$ time itself. However the speed of the construction critically depends on the algorithm used for quantizer. Therefore they have proposed a low-complexity heuristic quantizer and used for their PCC that had theoretical guarantees on the resulting code performance.

It was known that there exist optimal quantizers in literature which had a cubic or quadratic complexity order, which is prohibitively large for a PCC. Reducing its complexity is an open problem. We address this problem precisely in this section and elaborate one novel optimal quantizer with same order of complexity but is upto 50% faster.

5.8.2 Introduction to Channel Quantization

Given a BI-DMC with output alphabet of size M, we want to find an alternative channel of size $K \leq M$, which is optimal in the given fidelity criteria (see Fig. 5.9). Such a reduction of the channel's output alphabet size is a classic problem arising naturally in communication systems and information theory, and it is also connected to some other problems in learning theory and rate-distortion theory [168].



	1	···	M
0	p_{01}	···	p_{0M}
1	p_{11}	···	p_{1M}

	1	···	K
0	q_{01}	···	q_{0K}
1	q_{11}	···	q_{1K}

Figure 5.9: Quantization of a BI-DMC to reduce channel's output alphabet size and corresponding finite probability transition matrices as tables

The origin of the problem dates back to 1960's where the objective was to obtain a finite alphabet DMC alternative to the given continuous alphabet channel, modulator and demodulator combination (i.e. Fig. 5.10 with infinite M), at optimal *cut-off rate* [169–171]. An iterative algorithm to design a quantizer at the optimal cut-off rate was proposed in [171]. These algorithms were liberally used for a long time, motivated by the contemporary popular belief that cut-off rate is the practically achievable maximum rate of transmission with reasonable complexity.

Over the past decade, with the advent of capacity achieving coding techniques, it became evident that the mutual information (MI) of the channel is a more appropriate fidelity criteria for quantization than cut-off rate. This led to numerous quantizer algorithms being proposed over the past decade in context of several coding techniques and different channels (See for e.g. [172–174]). Finally, it was found in [168, 175] that this problem can indeed be solved in $O(M^3)$ polynomial time using dynamic programming techniques. A more recent work in this direction is [176, 177], using an interesting technique called SMAWK algorithm. Our current work is as an independent development, based on graph theory and serves as an alternate to the original quantizer in [168].

In this section, we present an alternative graph theory based formulation that allows a range of new algorithms to be developed in the future. As a first step, we propose a new solution that solves the problem in a significantly less amount of time. In particular, we assume that the

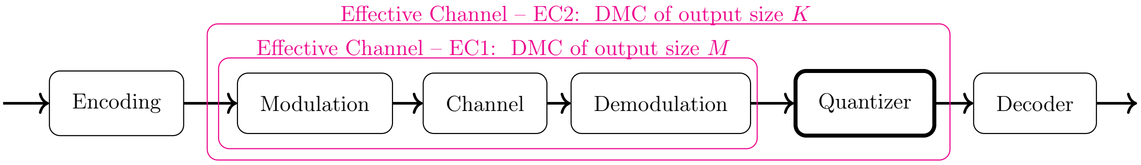

Effective Channel – EC2: DMC of output size K
Effective Channel – EC1: DMC of output size M
Encoding
Modulation
Channel
Demodulation
Quantizer
Decoder

Figure 5.10: System model for channel quantization

effective channel EC1 in Fig. 5.10 is a finite alphabet DMC but has a high alphabet size M, and we want to process the channel to obtain a new channel EC2 with a lower alphabet size K that has the optimal mutual information.

A list of contributions of this section is given below:

1. A new graph-based formulation of the quantizer problem is proposed that can aid better understanding of the problem and serve as a basis for more efficient algorithms in the future.
2. A new efficient alternative algorithm to [168] is proposed to find the optimal quantizer solution, based on classic Bellman-Ford algorithm and its enhancements.
3. Obtained over 50% reduced complexity compared to the original optimizer algorithm.

5.8.3 System Model and Problem Formulation

Let a BI-DMC with input alphabet $\mathcal{X} = \{0, 1\}$ and output alphabet $\mathcal{Y} = \{1, 2, \ldots, M\}$ be fully described by a transition probability matrix (TPM) $\mathbf{P}_{2\times M} \triangleq [p_{ij}]$, where the transition probability $p_{ij} \triangleq \Pr(j \in \mathcal{Y} | i \in \mathcal{X})$ and $\sum_j p_{ij} = 1, \forall i \in \mathcal{X}$. Let the input distribution be $\mathbf{r} \triangleq \{r_0, 1 - r_0\}$. Now, the mutual information of this channel is defined as,

$$\begin{aligned} I(\mathbf{P}; \mathbf{r}) \triangleq \sum_{j\in\mathcal{Y}} \Bigg\{ & r_0 p_{0j} \log_2 \left(\frac{p_{0j}}{r_0 p_{0j} + (1-r_0) p_{1j}} \right) + \\ & (1-r_0) p_{1j} \log_2 \left(\frac{p_{1j}}{r_0 p_{0j} + (1-r_0) p_{1j}} \right) \Bigg\} \\ = \sum_{j\in\mathcal{Y}} & I_j(\mathbf{P}; \mathbf{r}) \end{aligned} \tag{5.18}$$

We denote $I_j(\mathbf{P}; \mathbf{r})$ as the *partial mutual information* contributed by output symbol j.

A channel transformation is defined as a row-stochastic matrix $\mathbf{T}$, which transforms a given BI-DMC $\mathbf{P}$ to a new BI-DMC $\mathbf{Q}$ with output alphabet $\mathcal{Y}' \triangleq \{1, 2, \ldots, K\}$ as $\mathbf{Q} = \mathbf{PT}$. Let $\mathbb{1}_{N\times 1}$ denote a column vector of dimension $N \times 1$ with all elements equal to one.

Now, the channel quantization problem may be written as a simple optimization problem as below, to find an optimal channel transformation $\mathbf{T}^*$ as a solution to $\mathbf{T} \triangleq [t_{ij}]$.

$$\begin{aligned} &\max \ I(\mathbf{Q};\mathbf{r}) \\ \text{subject to:}& \\ &\mathbf{Q} = \mathbf{PT}, \\ &\mathbf{T}\cdot \underline{1}_{K\times 1} = \underline{1}_{M\times 1}, \\ &t_{ij} \geq 0, \forall i,j \text{ and } \mathbf{T} \in \mathbb{R}^{M\times K} \end{aligned} \tag{5.19}$$

Where $\mathbb{R}$ denotes set of real numbers. It is not difficult to see that it is a convex programming problem, given that the objective function $I(\mathbf{Q};\mathbf{r})$ is a concave function in its arguments and constraints are all linear. Note that, this formulation also extends easily to non-binary input channels with only a dimensional change in the elements of the problem. Furthermore, it was proved in [168] that the problem can be simplified for BI-DMC's, to a simple combinatorial search problem described as follows.

Let $\text{LR}(i) \triangleq \frac{P_{0i}}{P_{1i}}$, and assume a reordering of the output alphabet such that they are in ascending order of LR's i.e., $\text{LR}(i) < \text{LR}(j), \forall i < j$. The symbols with equal LR can be clubbed into one symbol, thereby reducing the output alphabet size at no loss in mutual information. Therefore, we assume that sufficient preprocessing is performed before we start the quantization, which results in strict inequalities. Finally, the output alphabet of the MI-optimal channel $\mathcal{Y}'$ is a partitioning of the original alphabet $\mathcal{Y}$ (taken in the above ascending order), partitioned into K partitions (i.e., a *K-partitioning*, in short). Then by combining each partition into a single large symbol, we obtain the new output alphabet $\mathcal{Y}'$.

The corresponding optimal channel transformation $\mathbf{T}^*$ is simply a binary matrix with ones at locations $\{(i,j) : i \in \mathcal{Y}, j \in \mathcal{Y}'\}$ that match the new symbols in $\mathcal{Y}'$ to the constituent symbols from $\mathcal{Y}$. By using $\mathbf{T}^*$, we can finally compute $\mathbf{Q} = \mathbf{PT}^*$, the TPM of the new channel. Alternatively, we can calculate each column of $\mathbf{Q}$ directly by noting that each column represents a symbol $y' \in \mathcal{Y}'$ and is simply the sum of a few consecutive columns in $\mathbf{P}$, which correspond to the symbols $y \in \mathcal{Y}$ that form y'.

Using a small combinatorial calculation, we may find that the number of all possible K-partitionings is equal to $\binom{M-1}{K-1}$. Such a value is very large even for the reasonable values of M and K, and hence the exhaustive search for finding an optimal quantizer is prohibitive. This motivates the need for more intelligent searching algorithms and therefore the authors of [168] have proposed a dynamic programming approach to find an optimal quantizer solution at a worst-case complexity of $\mathcal{O}(M^3)$. We, in the next section, propose an alternative formulation of the problem using graphs, and finally provide a new algorithm that can work significantly faster in practice. Further, the formulation is based on graph theory which we believe to serve as a stronger basis for better algorithms in the future, even though recent works such as [176, 177] Further investigations in this directions are relegated to our future work.

Relevant Graph Theoretic Terminology: Let $\mathcal{G}(\mathcal{V},\mathcal{E})$ be a directed graph with $M+1$ vertices

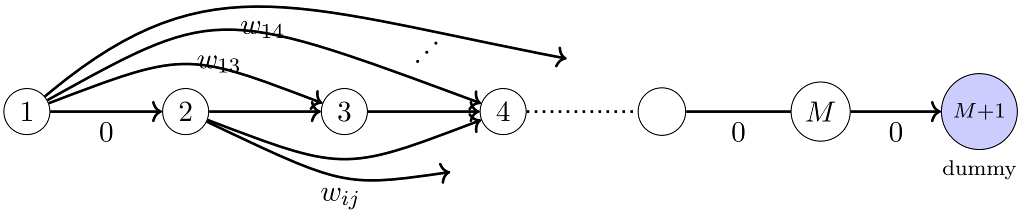


Figure 5.11: The directed acyclic graph representing the adjacency matrix definition in (5.20)

$\mathcal{V} \triangleq \{1, 2, \cdots, M+1\}$ and edges $\mathcal{E}$. We use notation (u, v) or $u \to v$ to denote an edge from node u to node v. We are interested in a specific graph with all forward edges and non-negative weights, which is completely specified by a weight-adjacency matrix $A = [a_{ij}]$ of $(M+1) \times (M+1)$ dimension, defined as follows.

$$a_{ij} = \begin{cases} w_{ij}, \text{ if } i < j \leq (M + 1 - \max\{0, K - i\}) \\ \\ \infty, \text{ elsewhere} \end{cases} \tag{5.20}$$

Every finite value a_{ij} in A implies that there exists an edge between node i to j having the value as edge-weight. With the above definition, the resulting graph looks similar to Fig. 5.11, having all forward edges and the number of edges $(\frac{M}{2}(M+1) - \frac{K}{2}(K-1))$. It may also be identified as a *directed acyclic graph* (DAG).

A *path* on a graph is defined as an ordered sequence of nodes among which, every consecutive pair has an edge between them in the same direction. Given a path of $K+1$ vertices, we say that the path has K *hops*, meaning there are K edges on that path to reach from one end to the other end. The sum of all edge weights on a given path is defined as its *path-weight* or *path-distance.*

A *single source shortest path problem* (SSSPP) is defined as a problem of finding a path between a given source node to every other node, having smallest path-distance. Such paths are termed as *shortest paths.* We recall that an algorithm that finds a shortest path from a given source node to every other node has the same complexity as that of a problem where we fix a destination node on a random graph.

A *hop-constrained SSSPP* is the one in which we need shortest paths that are constrained on the number of hops. In particular, a *K-hop single source shortest path problem* (K-hop SSSPP) is the one in which we need to find, from a given source node to all other nodes, the shortest path among all paths that have at most K edges.

It was established in detail in [178, 179] that any generic algorithm that finds the K-hop shortest path on any given graph with N nodes, and it is *path comparison based*, must have a worst-case complexity of $\Omega(N^3)$, ($N = M$ in graph $\mathcal{G}$). It implies that we cannot find a generic algorithm that is better than $\Theta(N^3)$. It was also shown in [178] that this asymptotic limit can indeed be achieved by an algorithm devised based on Bellman-Ford Algorithm for single source shortest path problem. However, it should be noted that the above result doesn't restrict the complexity of a dedicated algorithm designed for a given graph and weight structure.

In the next section we show that the problem of finding an optimal quantizer can be written as a K-hop SSSPP. This implies that the algorithm in [168] is asymptotically optimal in the class of generic algorithms for K-hop SSSPP and that we can only do better by constant factor improvements over $\mathcal{O}(M^3)$. We attempt to achieve one such improvement in this section. However, the usage of more specific properties of the graph such as edge-structure and cost-structure may lead to much faster algorithms. Devising such advanced algorithms is relegated to our future work, for which the current work would serve as a strong precursor.

5.8.4 A New Formulation of Optimal Quantizer Design Problem

Consider the graph $\mathcal{G}$ defined earlier, where nodes now denote symbols of the original alphabet to be quantized, but reordered in the strict ascending order of LR's as detailed in Section 5.8.3. The extra node $M+1$ is reserved as a dummy node to mark the end of a desired path.

We consider every edge $i \to j$ in $\mathcal{G}$ representing a partition of M symbols, containing symbols $\{i, i+1, \cdots, j-1\}$. This further implies that every end-to-end path $1 \to \cdots \to (M+1)$ in $\mathcal{G}$ is a complete partitioning of M symbols, where number of partitions is equal to the number of hops on the path. Recall that the optimal quantizer is also one such partitioning. Every partitioning derives a set of *super-symbols* (quantized alphabet) $\mathcal{Y}'$ and TPM according to $\mathbf{Q} = \mathbf{PT}$. The optimal quantizer would then correspond to the optimal channel transformation matrix $\mathbf{T}^*$.

To define the edge weights, we recall that every subcollection of symbols in $\mathcal{Y}$ or $\mathcal{Y}'$ contributes an additive component to the mutual information as defined in (5.18). With this observation we define below the edge weights w_{ij} as the loss of *partial mutual information* (PMI) contributed by each partition, after the quantization. Let $\boxed{i \to j}$ denote the super-symbol formed from the respective edge/partition.

$$w_{ij} \triangleq \left(\sum_{l=i}^{j-1} I_l(\mathbf{P};\mathbf{r}) \right) - I_{\boxed{i \to j}}(\mathbf{Q};\mathbf{r}) \tag{5.21}$$

Note that $I_l(\mathbf{P};\mathbf{r})$, the PMI value of symbol l depends only on the column l of $\mathbf{P}$ and a similar column in $\mathbf{Q}$ that is required to calculate the PMI of $\boxed{i \to j}$ is simply the sum of columns $\{i, i+1, \cdots, j-1\}$ in $\mathbf{P}$. We obtain, $w_{i,i+1} = 0, \forall i$, since they are the same symbols before and after the quantization.

We may verify that given any path $1 \to \cdots \to (M+1)$, the path-weight denotes the overall loss of mutual information due to quantization, i.e., $(I(\mathbf{P};\mathbf{r}) - I(\mathbf{Q};\mathbf{r}))$. Therefore, the objective "$\min\left(I(\mathbf{P};\mathbf{r}) - I(\mathbf{Q};\mathbf{r})\right)$" is equivalent to a K-hop shortest path from 1 to $M+1$ in the graph $\mathcal{G}$ that corresponds to the minimal loss of mutual information. Since the objective "$\min\left(I(\mathbf{P};\mathbf{r}) - I(\mathbf{Q};\mathbf{r})\right)$" is equivalent to "$\max I(\mathbf{Q};\mathbf{r})$" in (5.19), the optimal quantizer problem defined in (5.19) as a K-hop single source shortest path problem on $\mathcal{G}$ with source node equal to 1 and with edge weights as defined by (5.21).

5.8.5 A New Optimal Quantizer Design Algorithm

Recall the classic *Bellman-Ford algorithm* [137, 180] to solve SSSPP in a graph with additive but real-valued edge weights. Note that the Bellman-Ford algorithm is a general algorithm to solve SSSPP on any graph. At this point we refer to the wide literature available on Bellman-Ford algorithm to improve its performance, without change in complexity order $\mathcal{O}(M^3)$ [181–183]. However, for the particular case of $\mathcal{G}$ defined in Section 5.8.3, this algorithm is of no interest simply because we need to solve a *K-hop SSSPP* not an SSSPP. Also, it is easy to see that the shortest path in $\mathcal{G}$ solving SSSPP is simply the path $1 \to 2 \to \cdots M$ having zero path-weight.

However, inspired by Bellman-Ford algorithm, we present our Optimal Quantizer Algorithm (OQA) 26, including the improvements from [178] to solve K-hop SSSPP on $\mathcal{G}$. We may interpret it as two significant modifications to Bellman-Ford algorithm — (1) Simultaneous/parallel relaxation of edges, (2) Truncating the number of iterations to $K-1$. Also, we store all intermediate states, by using two $(K-1) \times M$ matrices Prev and Dist, each holding $K-1$ separate row-vectors instead of one as in classic Bellman-Ford algorithm. Before we start the algorithm we need to initialize the adjacency matrix by applying (5.21) for each of $|\mathcal{E}| = (\frac{M}{2}(M+1) - \frac{K}{2}(K-1))$ edges. By efficiently reusing the intermediate calculations, we can easily show that this is $\mathcal{O}(|\mathcal{E}|)$.

26 OQA: K-hop SSSPP solution based on Bellman-Ford algorithm (Guerin'02)

```
INPUT  : $\mathcal{G}(\mathcal{V},\mathcal{E})$, $\mathcal{V} = \{1,\dots,M+1\}$, Weight-adjacency matrix $A = [a_{ij}]$ from (5.20) and
         source node $s = 1$
OUTPUT : $K$-hop shortest path in $PATH[\,]$ and its weight $W$
Initialize two $(K-1) \times M$ matrices Prev[ ][ ] and Dist[ ][ ]
Prev$[i][j] = 0$, Dist$[i][j] = \infty,\ \forall i,j$
Dist$[1][i] = a_{1i}$ and Prev$[1][i] = 1\ \forall i$                 ▷ First hop
for $i = 2 : K-1$ do                                                   ▷ $K-2$ hops
    for $v = (i+1) : (M-K+i+1)$ do
        for $u = i : (v-1)$ do
            //("Relaxation" of the edge $u \to v$)
            if Dist$[i-1][u] + a_{uv} <$ Dist$[i][v]$ then
                Prev$[i][v] = u$
                Dist$[i][v] =$ Dist$[i-1][u] + a_{uv}$
            end
        end
    end
end
$W = \infty$ and $firstPrev = 0$
for $u = K : M$ do                                                     ▷ $K^{th}$hop
    if Dist$[K-1][u] + a_{u(M+1)} < W$ then
        $firstPrev = u$
        $W =$ Dist$[K-1][u] + a_{u(M+1)}$
    end
end
Allocate $(K+1) \times 1$ vector Path[ ]
Path$[K+1] = M+1$, Path$[K] = firstPrev$
for $i = K-1, K-2, \cdots, 1$ do                                       ▷ Backtracing
    Path$[i] =$ Prev$[i][$Path$[i+1]]$
end
```

This Algorithm 26 takes a complexity $\mathcal{O}((K-2)(M-K)^2/2)$ iterations, which is already slightly less than the reported $\mathcal{O}(K(M-K)^2/2)$ complexity in [168]. We also note that the above algorithm has several interesting similarities with the algorithm in [168] (for example, both having an initialization component of the same complexity $O(|\mathcal{E}|)$ and [168] having a similar main loop that runs K iterations instead of $K-2$), which we believe to be due to the fact that Bellman-Ford algorithm is also a classic example of dynamic programming in a graph theoretic context.

We finally propose a few more modifications, as illustrated in the algorithm titled 'Modification to Algorithm 26', that lead to a further constant factor reduction in complexity. We do this using a classic enhancement proposed by J. Y. Yen [181, 183]. It is well-known to enhance Bellman-Ford algorithm to a factor of at least 1/2. The changes are explained in below, which replaces the lines 4-14 in Algorithm 26, which should finally result in 1/2 complexity to that of the above algorithm which already has less complexity than [168] (i.e. an overall gain of > 50%). Following our discussion in Section 5.8.3, we do not expect a gain more than a constant factor as the algorithm is based on Bellman-Ford algorithm.

The above modifications are explained as follows. The pseudocode of these modifications is also provided. $\mathcal{S}$ and $\mathcal{C}$ are arbitrary collections of vertices, as used in the pseudocode. If a vertex v has not changed its weight from the last iteration, it is not expected to change in the future as well. So simply we remove that node from the list of nodes to update in the next iteration. For this we continuously update a set $\mathcal{S}$ considering only those nodes. Another set $\mathcal{C}$ is the set of vertices that don't change their value, which aids to compute $\mathcal{S}$ (See the pseudocodes Algorithms 26 and 27). This method eventually reduces the complexity to at least 1/2 of the original algorithm as proved formally in [181, 183]. We also note that the latest improvements to Bellman-Ford algorithm proposed in [183] are unfortunately not applicable for the current K-hop SSSPP. This is due to the loss of the intermediate optimality of paths (with no. of hops equal to the iteration index), which is critical for the K-hop SSSPP.

27 Modifications to Algorithm 26 (replaces lines 4-14 in Algorithm 26)

```
$\mathcal{C} = \phi$
for $i = 2 : K-1$ do
    $\mathcal{S} = \{i+1, i+2, \cdots, M-K+i+1\} \backslash \mathcal{C}$
    for each $v \in \mathcal{S}$ do
        for $u = i : (v-1)$ do
            ("Relaxation" of the edge $u \to v$)
            if Dist$[i-1][u] + a_{uv} <$ Dist$[i][v]$ then
                Prev$[i][v] = u$
                Dist$[i][v] =$ Dist$[i-1][u] + a_{uv}$
            end
        end
        if Dist$[i][v] =$ Dist$[i-1][v]$ & Prev$[i][v] =$ Prev$[i-1][v]$ then
            $\mathcal{C} = \mathcal{C} \cup \{v\}$
        end
    end
end
```

5.9 Summary

In this chapter, we have first considered four major PCCs over BI-AWGN channels and presented their full pseudocode implementations. Furthermore, we have identified the importance of design-SNRs due to the non-universality of polar codes and reported the best design-SNRs for these four PCCs with various code rates and code lengths. By extensive simulations, we have found that all the four PCCs are equally good for BI-AWGN, once the design-SNR is optimized for the best performance.

We have presented a comprehensive survey and full pseudocode implementations of all the well known construction algorithms. We then proposed a simple discrete search to find the best design-SNR for any given polar code construction algorithm. We then compared various polar code constructions and concluded that all are equally good in AWGN if the design-SNR is optimized for the best performance. Thus in future, we may use simple algorithms only.

Finally, we considered new optimal channel quantizers for using with Tal & Vary's PCC in order to improve their performance as a new PCC. We have presented a new formulation to the MI-optimal quantizers for reducing the output alphabet of a BI-DMC, as a K-hop single source shortest path problem on graphs. Using this new formulation, we have proposed a new solution algorithm based on classic Bellman-Ford algorithm that solves the problem at slightly less than the complexity of the latest optimal algorithm in [168]. Finally, by using a well-known enhancement of Bellman-Ford algorithm in the literature, we improved the complexity of the algorithm over 50% compared to [168]. We also noted that more powerful algorithms with reduced complexity order are possible, only if we use graph-specific algorithms to the specific graph proposed in this section. Further investigations of improving the complexity are relegated to our subsequent future work. This is critical for their usage in PCC algorithms producing longer codes.

Chapter 6

Performance Estimation of Polar Codes

This chapter continues the discussion of polar code construction algorithms with new applications. We also continue to use the same PCC notation and the channel model. As you may recall, the basic idea of Arıkan is that PCC is a way of choosing K among N bit channels, denoted by the set $\mathcal{F}$. However, a more accurate definition of the quality of bit channels is required for this.

Let N bit channels be denoted by $\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_{N-1}$. Let $Z(\mathcal{W}_i)$ and $P_e(\mathcal{W}_i)$ denote the Bhattacharyya parameter and the BER of the bit channel respectively $\mathcal{W}_i$ under ML (thresholding) decision. Initially, Arıkan proposed that one can simply choose the best K among N bit channels, corresponding to the least K values of the Bhattacharyya parameters $Z(\mathcal{W}_i)$ of the bit channels at a given value of E_b/N_0 defined as the *design-SNR*. Such a selection minimizes an *upper bound* of the overall BLER (or some times, the *frame error rate* (FER)) of the corresponding polar code.

Definition 1 *A PCC at a given design-SNR is a choice of the set $\mathcal{F}$ of K bit channel indices such that it corresponds to the bit channels with the least K values among $\{Z(\mathcal{W}_i), 0 \le i \le N-1\}$.*

In this chapter, we will improve the Arıkan's standard definition of Bhattacharyya parameter based PCC as follows. This is motivated by the fact that the actual objective is to lower BLER and not to reduce the upper bounds alone.

Definition 2 *A PCC at a given design-SNR is a choice of the set $\mathcal{F}$ of K bit channel indices such that it corresponds to the bit channels with the least K values among $\{P_e(\mathcal{W}_i), 0 \le i \le N-1\}$.*

Throughout this chapter, we tend to base our proposals using more accurate expressions of BLER, as we will discuss in detail in Section 6.1. We will also show in Section 6.1 that the new Definition 2 can only improve the performance. Further, the Bhattachryya parameters may

---

Parts of this chapter are based on [149, 151] be thought of as one of the many possible estimates of the values $P_e(\mathcal{W}_i)$ and hence Arıkan's definition follows as a special case.

Since the exact $Z(\mathcal{W}_i)$ or $P_e(W_i)$ of the bit channels are not easy to compute, several approximations are proposed as PCC algorithms that we saw in Chapter 5. Note that a polar code resulted from any of the PCCs may be used at an SNR different from the design-SNR, known as the *operating-SNR*. The dependence of the code structure on the choice of design-SNR in a PCC is termed as the non-universality of polar codes. Due to this, applications that frown upon changing the code often require designing a unique polar code at a good design-SNR such that it works well at a range of operating-SNRs of interest. We therefore have seen one simple elementary heuristic algorithm in Chapter 5 which helps us choose the design-SNR. This chapter extends this algorithm further. Before we describe the same, we discuss several useful byproducts of PCC algorithms.

The following is a quick summary of the main contributions from this chapter:

1. An accurate BLER formula and ways to compute it using existing PCC algorithms.
2. A more accurate BLER estimation using a set of PCC algorithms together, as a minimum of the PCCs.
3. A new comparison strategy for PCCs, extending the heuristic discussion in the last chapter.
4. A new algorithm for find the reliable design-SNR of any PCC, based on the accurate BLER estimation that we proposed above. This completely avoids the use of any high complexity Monte-Carlo simulations.

6.1 Bit Channel Metrics and Estimation of BLER from Polar Code Construction

Let the design-SNR be s and operating-SNR be t. Now, let $\mathbb{P}(s,t)$ denote the exact BLER of the polar code being used. Accurate BLER formulas serve as a starting point for our final and accurate BLER estimation proposed in Section 6.1.6, with the help of various PCCs discussed in Chapter 5.

6.1.1 Accurate Formulas for Computing BLER

Arıkan proposes that the BLER of a polar code can be estimated by the sum of Bhattacharyya bounds of bit channels.

$$\mathbb{P}(s,t) \leq \sum_{i \in \mathcal{F}^c} Z(\mathcal{W}_i) \tag{6.1}$$

An alternative expression was suggested in [101, Eq.(1)], which considers bit channel BERs instead of their respective Bhattacharyya parameters. This can only improve the BLER estimate.

$$\mathbb{P}(s,t) \leq \sum_{i \in \mathcal{F}^c} P_e(\mathcal{W}_i) \tag{6.2}$$

The most accurate BLER expression is also known and is given below. It was originally suggested in [103, Eq. (6)] and was also discussed recently in [106, Eq. (3)]. The above two bounds can be trivially derived from this expression.

$$\mathbb{P}(s,t) = 1 - \prod_{i \in \mathcal{F}^c} (1 - P_e(\mathcal{W}_i)) \tag{6.3}$$

All earlier results may be summarized as a single straightforward theorem as follows. A proof of the theorem is an easy extension of the above formulas.

Theorem 1 *Let $\mathbb{P}(s,t)$ denote the exact BLER of a polar code with design-SNR s and operating-SNR t. Then,*

$$\mathbb{P}(s,t) = 1 - \prod_{i \in \mathcal{F}^c} (1 - P_e(\mathcal{W}_i))$$

and

$$\mathbb{P}(s,t) \leq \sum_{i \in \mathcal{F}^c} P_e(\mathcal{W}_i) \leq \sum_{i \in \mathcal{F}^c} Z(\mathcal{W}_i)$$

Clearly (6.3) is an accurate estimate compared to the other two bounds. Furthermore, the bounds (6.1) and (6.2) often exceed 1 (at low SNRs), rendering them useless. Such a problem never arises with (6.3). Therefore we will use (6.3) for calculating the BLER, through the rest of the chapter. This has improved the accuracy significantly. However, it should be noted that when the values $P_e(\mathcal{W}_i)$ are loosely estimated (with estimates such as Bhattacharyya parameters), a gap between the actual BLER and the estimate (6.3) will continue to exist, just like the rest of the bounds.

6.1.2 A Robust Iterative Formula for BLER Computation

We now see some practical concerns with computing (6.3). We first find that due to polarization effect, many channels exist with their bit channel BERs that exceed the numerical precision limits of a floating point number when represented in linear domain. A simple solution will then be to use a log-domain representation of all BERs. However, computations similar to (6.3) will again lead to underflow problems when applied directly. We therefore propose the following simple iterative formula to compute (6.3) efficiently in log-domain.

Now we propose a simple iterative strategy to compute the BLER (6.3), in a numerically stable form, especially useful for computations in log-domain. This is summarized in the below self-explanatory theorem. The additional numerical stability of this formula may be regarded as a result of the availability of numerically robust addition and difference formulas in log-domain, by using the well-known *log-sum-exp* function [184, Sec. 3.1.5], [134].

Theorem 2 (Efficient computation of BLER) *Let K bit channel BERs $\{P_e(\mathcal{W}_i), i \in \mathcal{F}^c\}$ be denoted arbitrarily as $p_1, p_2, \ldots, p_K$. Now, the BLER from* (6.3) *becomes*

$$\mathbb{P}(s,t) = 1 - \prod_{i=1}^{K}(1-p_i)$$

Let a function $f(l)$ be defined as follows.

$$f(l) \triangleq 1 - \prod_{i=1}^{l}(1-p_i), \quad \forall 1 \leq l \leq K \tag{6.4}$$

Then, we have $f(1) = p_1$ and $f(K) = \mathbb{P}(s,t)$, with the function satisfying the following recursion.

$$f(l+1) = f(l) + p_{l+1} - f(l) \cdot p_{l+1} \tag{6.5}$$

Now, with an initialization of $f(1) = p_1$ and computing $f(2), \ldots, f(K)$ sequentially using the above formula, one may compute $\mathbb{P}(s,t)$ efficiently.

6.1.3 Improving Polar Codes by Using Definition 2 of PCC

Arıkan defines polar codes to minimize the sum of Bhattacharyya parameters of bit channels as an efficient alternative for minimizing the BLER of the code at a given design-SNR. In other words, he estimates the BLER of a polar code with the sum of Bhattacharyya parameters of information bit channels. The following theorem states that by using Definition 2 one can only improve the performance of the polar codes.

Theorem 3

1. *Definition 2 solves the following problem where s denotes the design-SNR.*

$$\min_{\mathcal{F}} \mathbb{P}(s,t)$$

subject to:

$$\mathbb{P}(s,t) = 1 - \prod_{i \in \mathcal{F}^c}(1 - P_e(\mathcal{W}_i)) \text{ and } t = s$$

2. *Definition 2 obtains the optimal BLER performance of the polar code and hence can only improve the polar code when compared to Arıkan's definition (a selection based on $Z(\mathcal{W}_i)$, unlike Definition 2).*

Hence the new Definition 2 is justified. However, a couple of issues may be noted as follows. Theorem 3 only states that whenever the bit channel estimates $P_e(\mathcal{W}_i)$ are accurate, Definition 2 is better compared to Arıkan's. However, when the bit channels are loosely estimated (such as using Bhattacharyya parameters), the result will continue to be suboptimal.

6.1.4 BLER Estimates as a Byproduct of PCCs

As far as the main objective of a PCC (selection of bit channels such that BLER is minimized) is concerned, the full expression of (6.3) never needs to be computed. However, once the polar code is constructed, (6.3) becomes an important byproduct because it provides an estimate of the final performance of the resulting code, which is usually found only after a Monte-Carlo simulation. In fact, such an estimate helps enormously in devising efficient algorithms for the choice of a reliable design-SNR, as we see in Section 6.3.

6.1.5 Bit Channel BERs vs. Bhattacharyya Parameters

Even though asymptotically tight, at finite block lengths, we find that the Bhattacharyya bounds often tend to be very loose. On the other hand, most of the available PCCs can often estimate the bit channel BERs directly. Further, it is clear from the earlier discussion that using bit channel BERs will lead to a more accurate PCC. Therefore whenever possible, we suggest to compute only the bit channel BERs and use them instead of Bhattacharyya parameters as we do in the rest of the chapter.

6.1.6 A New Accurate Estimation of FER Using PCC Algorithms

From above discussion it is clear that one can use *any* of the available PCCs for estimating the BLER of *any* given polar code. Further, we readily discussed in detail four different PCCs, so we can have four independent BLER estimates. Naturally, it is nice to find the best estimate among these independent estimates of the same quantity. Such an estimate will behave as a new estimate that is at least as good as any of the individual estimates. Let us denote this estimate as approximate-FER (AFER).

AFER is relatively more useful because of the following reason. In simulations such as Figs. 6.2 and 6.3, we find that none of the estimates is best always, however at least one of the estimates is close enough to the exact BLER, at all SNRs. Which means different PCCs can be accurate at different combinations of rates, block lengths, and SNRs. Clearly, the combined estimate will be much better than any of the individuals and serve as an alternative to highly complex Monte-Carlo simulations.

With the above motivation, we simply propose to use the minimum of the four, as the new BLER estimate. That is, if $\alpha_1(s,t), \alpha_2(s,t), \ldots$ are the estimates of the true BLER $\mathbb{P}(s,t)$, a new estimate $\alpha^*(s,t)$ we propose is simply the minimum:

$$\mathbb{P}(s,t) \approx \alpha^*(s,t) \triangleq \min\{\alpha_1(s,t), \alpha_2(s,t), \ldots\}$$

A couple of issues may be observed with the proposed estimate. First, we can not guarantee that this estimate is an upper bound. Second, whenever an estimation algorithm fails to estimate the BLER (for example PCC-1 can often fail or produce unreliable BLER estimates due to the limited sample size), one must drop it from the list before the minimum is computed.

Following are some advantages of the proposed BLER estimate.

1. One may estimate BLERs of codes of very long block lengths such as $N > 10^6$, which helps making a more detailed analysis of the asymptotic behaviour of a polar code. This is unlike a Monte-Carlo which has limited ability depending on the platform.
2. One can estimate upto very low BLERs such as 10^{-20}, unlike a Monte-Carlo estimation where the dynamic range of estimated BLER (along with its precision) is much more limited ($< 10^{-6}$, for example) especially by the practical sample sizes that are used.
3. One need not build an explicit encoder or decoder to know the BLER performance.

An example of the above BLER estimation is shown in Fig. 6.1, which illustrates its accuracy. The latest BLER estimation curve essentially overlapped with the exact Monte-Carlo estimation of BLER. Further, the estimation enabled to foresee the performance of the polar code to much lower BLERs.

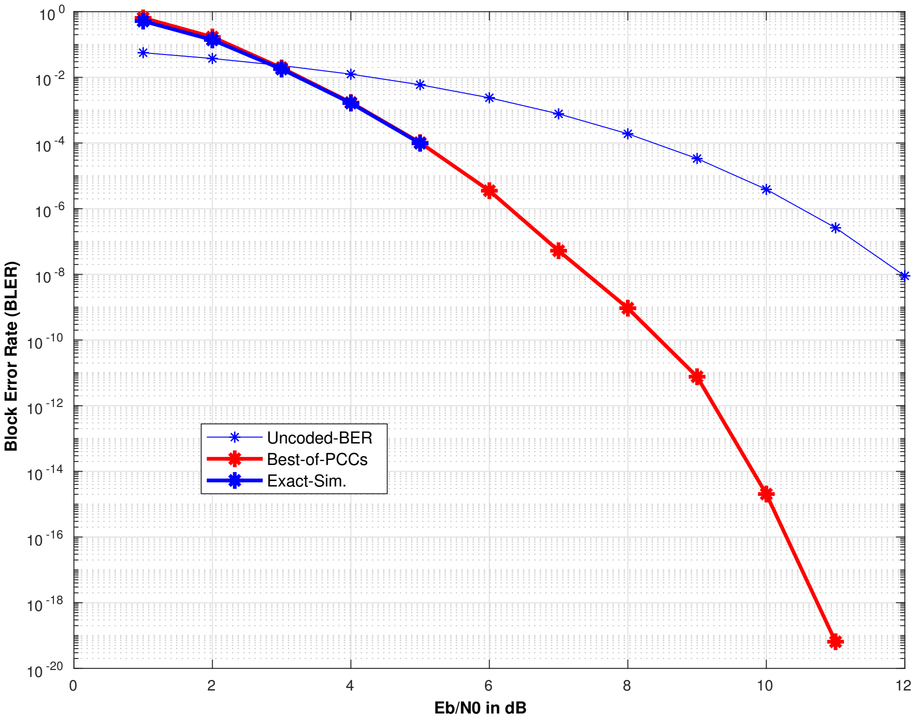


Figure 6.1: An accurate BLER estimation obtained using all PCCs $N = 256$ and $R = 0.5$

6.2 New Comparison Strategies for PCCs

Given two arbitrary PCC algorithms, it is often difficult to compare them unless we perform Monte-Carlo simulations of the actual BLER. For curves that crossover, the decision is ambiguous even after the efforts of performing a Monte-Carlo simulation. In this section, we propose two approaches for fairly comparing various PCCs, along with relevant simulation results.

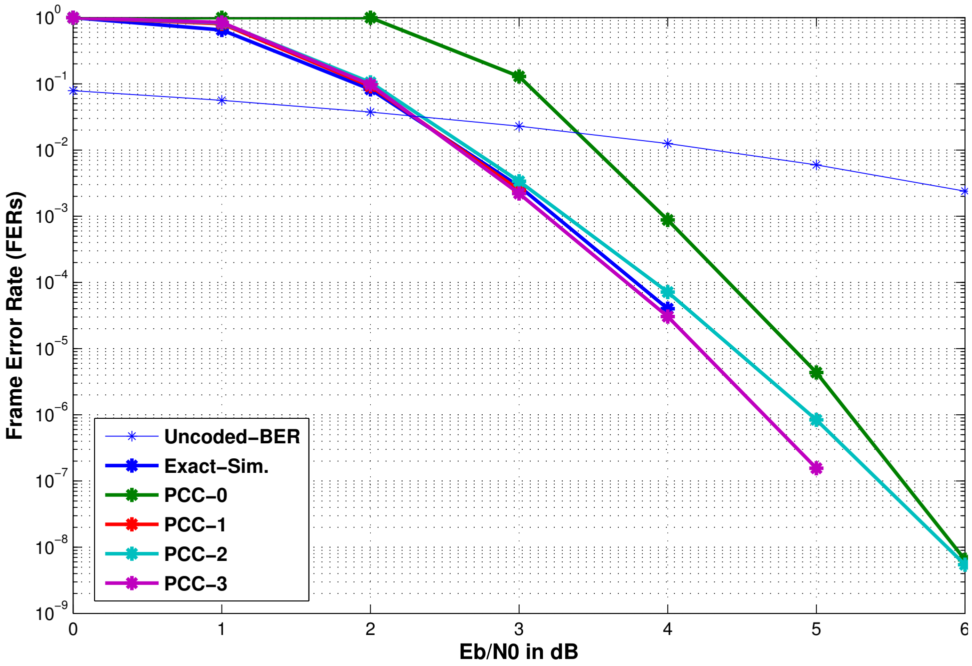


Figure 6.2: *Approach 1*: the acccuracy of FER or BLER estimates from various PCCs at $N = 1024, R = 0.9$

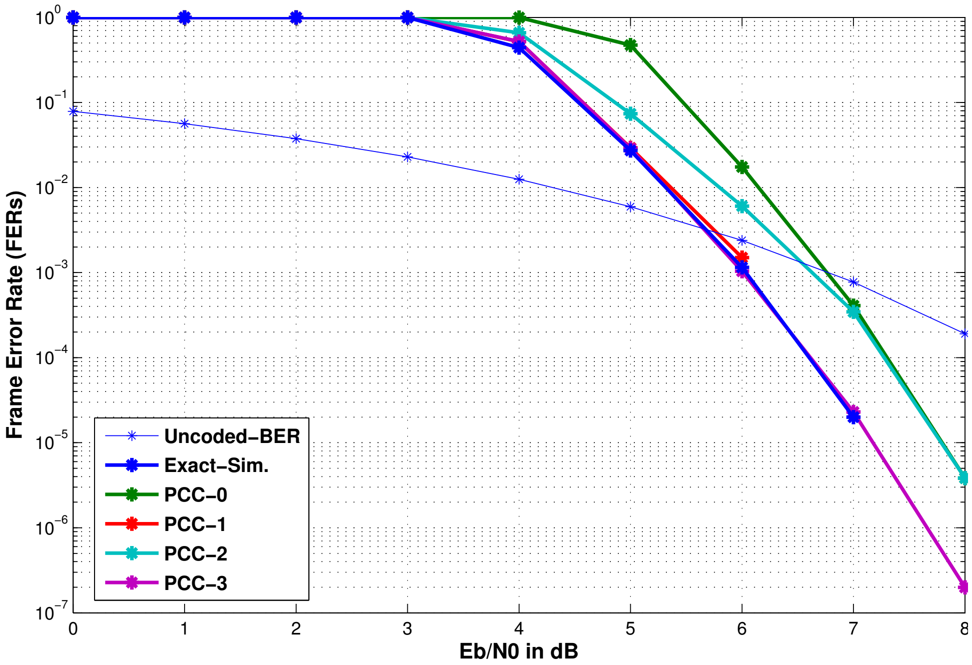


Figure 6.3: *Approach 1*: the acccuracy of FER or BLER estimates from various PCCs at $N = 2048, R = 0.9$

6.2.1 Approach 1:

Our first approach is to find how accurate are the upper bounds on BLER, that are estimated by a PCC. This is useful because better upper bounds on the error performance lead to codes that better optimize the code performance. This is similar to the comparison available up to a limited extent in [104–106]. We will extend the same to all the PCCs at relatively higher block lengths.

A fairness of this comparison, however, needs to be established. For this, we first consider a random PCC algorithm, and use it to obtain a set of frozen bit indices $\mathcal{F}$. Now, we may use all four PCC algorithms at any given SNR, with a sole purpose of estimating the BLER of the polar code defined by $\mathcal{F}$. We may repeat the same at every SNR and obtain the corresponding four plots of BLER (estimated). Finally we may compare them against the exact BLER plot obtained via Monte-Carlo simulations and find how accurate is each BLER estimation.

For each construction we use the exact expression of BLER from (6.3). As was discussed in Section 6.1 we perform all our computations in log-domain and use the iterative BLER computation in Section 6.1.4. The results are shown in Figs. 6.2 and 6.3.

Some interesting observations can be made from the comparisons in Figs. 6.2 and 6.3. The PCC-3, based on Gaussian approximation, estimates the BLER quite accurately. The PCC-1, based on Monte-Carlo estimation, is also accurate but is limited by the sample size of Monte-Carlo and fails to estimate BLERs below a threshold. PCC-2 on the other hand estimates the BLER at low SNRs, but essentially coincides with PCC-0 at higher SNRs (due to its inherent use of PCC-0 and choosing the best).

Interestingly, while the above comparison of the PCCs is true at low SNRs, at very high SNRs (leading to BLERs $< 10^{-9}$, as seen in Fig. 6.1) we find that the estimations provided by the simplest PCC-0 are outstanding, and fully dominate the rest of the PCCs.

An important observation with respect to the accuracy of BLER estimates can be made as follows. First note that optimizing the exact BLER should produce the best possible polar code in the sense of BLER and hence obtaining accurate estimates of BLER is indeed important. But, optimizing a relatively better BLER approximation need not always produce a better polar code in the sense of BLER. Simply because, the optimizing point can be different for two different approximations and the exact BLERs obtained at those points can be in any order. In other words, even though a PCC produces a poorer estimate of BLER, the polar code resulted from it can still have a better exact BLER. This motivates our next approach for the comparison of various PCCs.

6.2.2 Approach 2:

Recall that our main objective is to improve the exact BLER of the polar code. So a straightforward approach is to compare the exact BLER of different polar codes resulted from different PCCs. However, a Monte-Carlo estimation may be essential to know such an exact BLER (unless we have an accurate-enough algorithm to estimate BLER, such as the one we will see in Section 6.1.6).

A fairness of the comparison is again an important concern because of the design-SNR parameter which effects each PCC very differently (see Section 6.2). Fortunately, we can take care of this by simply choosing the design-SNR according to one of the series of algorithms in Section 6.3. More specifically, we will use the efficient *Algo-Simple* to decide what design-SNR to be used for each PCC. The final results are shown in Fig. 6.4. Also, the same result is obtained when the naive search is used hence validating *Algo-Simple* to choose a reliable design-SNR.

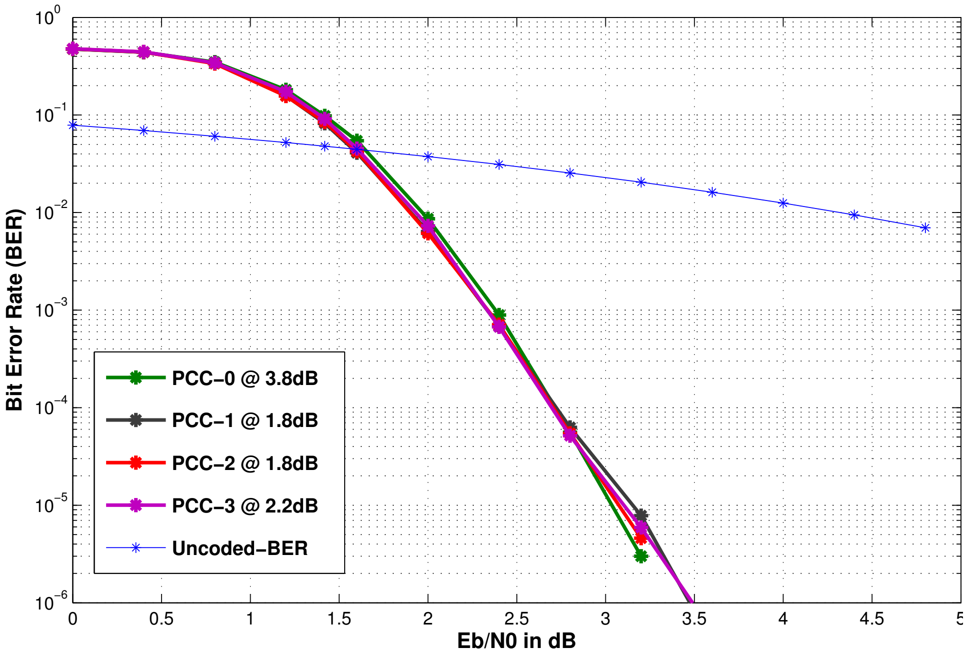


Figure 6.4: *Approach 2*: BLER estimates after choosing design-SNRs at $N = 2048$ and $R = 0.5$

6.3 The Efficient Choice of a Reliable Design-SNR

Let an arbitrary PCC be given and be fixed throughout this section. Assume the following notation.

- $\mathscr{F}(s)$: The set of frozen bit indices obtained by running the PCC at a design-SNR s
- $\mathcal{C}(s)$: The corresponding polar code completely described by the frozen bit indices $\mathscr{F}(s)$.
- $\mathbb{P}(s,t)$: The exact BLER probability of a polar code which was constructed at a design-SNR s and used at an operating-SNR t
- $\mathcal{X}(s)$: The exact BLER plot defined by the points $\{(t, \mathbb{P}(s,t)) : t \in \mathbb{R}^{+}\}$, at design-SNR s.
- $\hat{\mathcal{X}}(s)$: The BLER plot estimation of the polar code, estimated by the given PCC at design-SNR s.

We now propose a series of search algorithms that find a reliable design-SNR by searching over a finite number of m design-SNRs, except our last algorithm which will avoid this and searches over a continuous real line using a binary search. However, our search over a finite set of design-SNRs is usually sufficient because there exists a neighborhood around any choice of design-SNR, such that for any design-SNR value in the neighborhood, the ranking of bit channels (and hence the polar code) estimated by a given PCC remains the same. We find that a value of $m = 10$ to 20 is usually sufficient for all our algorithms. Also, as a part of all our search algorithms, a full run of the PCC is performed at several design-SNRs, including the reliable design-SNR that we find. Therefore we may simply store and reuse these intermediate results, to produce our final code.

We first propose a naive search as a generalization to the earlier heuristic in Chapter 5, which finds a reliable design-SNR over m possible design-SNR values. Even though this may not be used in practice, it provides a good motivation for our next algorithms that are quite efficient.

6.3.1 A Naive Search

1. Consider a finite set of design-SNRs $\{s_1, s_2, \ldots, s_m\}$ that covers the full range of operating-SNRs.
2. Run the PCC m times at the design-SNRs to design m polar codes fully described by $\mathscr{F}(s_1), \mathscr{F}(s_2), \ldots, \mathscr{F}(s_m)$.
3. Using Monte-Carlo, find the corresponding BLER plots of the resulting codes $\mathcal{X}(s_1)$, $\mathcal{X}(s_2)$, …, $\mathcal{X}(s_m)$.
4. Select the BLER plot $\mathcal{X}(s_j)$ that best suits the needs of the target application and its corresponding design-SNR s_j.
5. Declare s_j as the reliable design-SNR for the given PCC. Also output the corresponding $\mathcal{F}_j$, obtained at s_j in step-2. (need not run the PCC again)

We may quickly note a few problems with this naive search especially as with steps 3 and 4.

1. Running a full Monte-Carlo based simulation at step 3, for the BLER plots of polar codes at a range of design-SNRs, is highly complex. Further, it may not be computationally feasible to obtain the estimates of BLERs falling below a threshold (for e.g. 10^{-7}).
2. A comparison of different BLER plots according to step 4 is not straightforward, especially when the curves intersect with each other.

Unfortunately, there is no solution available in literature, to address these nontrivial issues. We therefore propose our next algorithms to resolve them.

6.3.2 Algo-Master: The General Form Algorithm

We will now define a general algorithmic framework, which may then be used to derive a variety of application-specific rules. In fact, our next algorithm is an improved version of a specific variation of this algorithm.

We first define a general strategy to distinguish two BLER plots, addressing the issue (2) above with the earlier naive search. Let two design-SNRs s_1 and s_2 be given, resulting in two codes with BLER plots $\mathcal{X}(s_1)$ and $\mathcal{X}(s_2)$. Now, we say that $\mathcal{X}(s_1)$ is better than $\mathcal{X}(s_2)$, if $g(s_1) < g(s_2)$, where g denotes a *metric function.* The function g of a BLER plot is analogous to a probability of error function in the sense that the more the value of g, the worse the BLER plot.

Second, we propose to find the BLER plot estimations $\{\hat{\mathcal{X}}(s_i)\}$ from the PCCs themselves, instead of using a highly complex (but more accurate within certain limits of BLER) Monte-Carlo estimation to find the exact plots. The following observations will justify this. First, a BLER estimate is a natural byproduct of a PCC with only a few additional computations. This can be performed at any SNR, independent of the design-SNR at which polar code is constructed (details in Section 6.1). Second, such an BLER estimate is accurate within 1dB, as we find in simulations (see Sections 6.1.6 and 6.2). Hence, we propose the next algorithm, using these BLER estimates.

Later in Section 6.1.6 we will see how the accuracy of BLER estimates can be improved further by using their minimum. An extension of the below can then be made trivially, using such an improved BLER estimation instead. For simplicity, assume the original BLER estimates only for below.

Algo-Master:

1. Consider a finite set of SNRs $\{s_1, s_2, \ldots, s_m\}$ that covers the range of SNRs of interest.
2. Run the PCC m times at the design-SNRs to design m polar codes fully described by $\mathscr{F}(s_1), \mathscr{F}(s_2), \ldots, \mathscr{F}(s_m)$.
3. Obtain the BLER plot estimations $\hat{\mathcal{X}}(s_1)$, $\hat{\mathcal{X}}(s_2)$, $\ldots$, $\hat{\mathcal{X}}(s_m)$, using the PCC (as in Section 6.1).
4. Compute the m metrics $g(s_1)$, $g(s_2)$, $\ldots$, $g(s_m)$ (using the above plots) and compute
$$s^* = \arg\left(\min_{s\in\{s_i\}_{i=1}^m} g(s)\right)$$
5. Declare s^* as a reliable design-SNR and its corresponding $\mathscr{F}(s^*)$, available from step-2, which fully describes the polar code.

A special case: A specific example of the *Algo-Master* can be derived by defining a metric function $\eta(s, \epsilon)$, with a fixed user defined parameter ϵ. In particular, $\eta(s, \epsilon)$ denotes the SNR at which a *target BLER* of ϵ is achieved by the polar code constructed at design-SNR s.

By using this as a metric function in *Algo-Master*, we are indicating that we are interested in designing our polar code such that a BLER of ϵ is achieved at the smallest SNR possible. The algorithm will then output a reliable design-SNR and also the corresponding set of frozen bit indices that achieve the same. We soon see that this interesting definition of $\eta(s, \epsilon)$ helps simplifying the *Algo-Master* significantly.

Mathematically, we write $\eta(s, \epsilon)$ using the BLER function $\mathbb{P}(s, t)$, as follows.

$$\eta(s,\epsilon) \triangleq \begin{cases} \arg\min\limits_{t\in\mathbb{R}^+} \mathbb{P}(s,t) \\ \\ \text{s.t. } \mathbb{P}(s,t) \leq \epsilon \end{cases} \tag{6.6}$$

6.3.3 Algo-Simple: An Efficient Search Algorithm for a Reliable Design-SNR

Consider the above metric function $\eta(s,\epsilon)$, with a user-specified ϵ. We find several advantages with this choice of a metric function. First, the function can be computed very efficiently. And then it significantly simplifies the overall problem of finding a reliable design-SNR to a simple and efficient binary search (or a bisection method) with a significantly reduced runtime compared with a direct use of *Algo-Master*. Finally, it eliminates the explicit choice of the candidate design-SNRs in step-1. Such a simplified *Algo-Master* is precisely the latest algorithm *Algo-Simple*. This becomes our recommended algorithm from this chapter, towards the choice of a reliable design-SNR. Our main result is summarized in the below theorem. We will then see the full details of the algorithm.

Theorem 4 *Consider the metric function $\eta(s,\epsilon)$ in* Algo-Master*. Then the best possible estimate of the reliable design-SNR s^* is simply a solution of the following simple problem.*

$$s^* = \arg\min_t \{t : \mathbb{P}(t,t) \leq \epsilon\} \tag{6.7}$$

Further, assuming that the BLER function $P(t,t)$ decreases monotonically with t, the realiable-design-SNR s^* may be simply obtained by using a bisection method *solving:*

$$P(s^*,s^*) = \epsilon \tag{6.8}$$

Proof:

Consider the minimization in step-4 of *Algo-Master* specialized with our metric function $\eta(s,\epsilon)$.

$$s^* = \arg\left(\min_{s\in\{s_i\}_{i=1}^m} \eta(s,\epsilon)\right) \tag{6.9}$$

On substitution of the definition of η, we obtain

$$(s^*,t^*) = \arg\left(\min_{s\in\{s_i\}_{i=1}^m} \min_{t\in\mathbb{R}^+} \mathbb{P}(s,t)\right) \text{ s.t. } \mathbb{P}(s,t) \leq \epsilon \tag{6.10}$$

The above (6.10) presents us an interesting opportunity. Previously, we proposed to run the PCC at only a finite number of design-SNRs to reduce the complexity. Now for instance, let us assume we run the given PCC at all possible design-SNRs (i.e. $\forall s \in \mathbb{R}^+$). This leads to a significant simplification avoiding the choice of s altogether, as follows.

$$\min_{s\in\mathbb{R}^+}\min_{t\in\mathbb{R}^+}\mathbb{P}(s,t) \text{ s.t. } \mathbb{P}(s,t)\leq\epsilon \tag{6.11}$$

$$= \min_{t\in\mathbb{R}^+}\min_{s\in\mathbb{R}^+}\mathbb{P}(s,t) \text{ s.t. } \mathbb{P}(s,t)\leq\epsilon \tag{6.12}$$

$$= \min_{t\in\mathbb{R}^+}\mathbb{P}(t,t) \text{ s.t. } \mathbb{P}(t,t)\leq\epsilon \tag{6.13}$$

The final step follows from our improved Definition 2 of a PCC: A PCC minimizes its BLER at a given design-SNR. In other words, when an operating-SNR t is fixed, an optimum performance is obtained when the PCC chooses a design-SNR $s = t$. ■

Theorem 4 provides a drastic simplification compared to a direct use of *Algo-Master* even under the use of a simple metric function such as $\eta(s,\epsilon)$, because we no longer need to obtain the full BLER plots of many different polar codes (obtained at many design-SNRs) as before. We just need to run the PCC at a few values of t and estimate $\mathbb{P}(t,t)$ (see Section 6.1). Once we find the reliable design-SNR, we may also output the code parameters $\mathcal{F}$ that are already found by running the PCC at that design-SNR, avoiding any further runs of the PCC.

As suggested in the theorem, a significant reduction in the number of runs of the specific PCC may be obtained by simply running a binary search (or a *bisection method*) for the corresponding solution. Such a search is justified by the general observation of monotonicity of $\mathbb{P}(t,t)$ in t, as it represents the minimum of several BLER curves obtained by various design-SNRs.

An interesting and alternative interpretation of the Theorem 4 is as follows. Consider an *ideal polar code* that changes dynamically at every point of SNR. In other words, we rerun the PCC at the operating SNR and change the code whenever the operating-SNR changes (note that this is in contrast to our main theme of building a unique polar code that works at all SNRs). Now, our *Algo-Simple* simply states that a reliable design-SNR according to the metric function $\eta(s,\epsilon)$ is simply the SNR at which the ideal polar code achieves a BLER of ϵ.

Our final algorithm is denoted *Algo-Simple*, with the parameters $(\epsilon, s_0, \Delta, \delta)$. For the purposes of applying this algorithm, all SNRs may be considered in dB scale. An example of the parameter values is $(10^{-4}, 0\,\text{dB}, 5\,\text{dB}, 0.05\,\text{dB})$. Here,

- ϵ : the target BLER in metric function η,
- s_0 : the initial SNR in dB
- Δ : the initial step-size of the SNR, and
- δ : the tolerence such that an error in reliable design-SNR that we find, if any, should be at most $\delta/2$ dB

The *Algo-Simple*$(\epsilon, s_0, \Delta, \delta)$:

1. Run the PCC at a design-SNR s_0 and obtain the BLER estimate $\mathbb{P}(s_0, s_0)$.

2. - If $\mathbb{P}(s_0, s_0) > \epsilon$, run the PCC at design-SNRs $t = s_0, s_0 + \Delta, \ldots$ and stop whenever $\mathbb{P}(t,t) \leq \epsilon$. Initialize $s_l = s_0$ and $s_r = t$.
   - If $\mathbb{P}(s_0, s_0) \leq \epsilon$, then run the PCC at design-SNRs $t = s_0, s_0 - \Delta, \ldots$ and stop whenever $\mathbb{P}(t,t) > \epsilon$. Initialize $s_l = t$ and $s_r = s_0$.
3. Initialize $\mathbb{P}_l = \mathbb{P}(s_l, s_l)$ & $\mathbb{P}_r = \mathbb{P}(s_r, s_r)$
4. $s_{mid} = \frac{s_l + s_r}{2}$
5. Run the PCC at design-SNR s_{mid} and obtain $\mathbb{P}_{mid} = \mathbb{P}(s_{mid}, s_{mid})$ and $\mathscr{F}(s_{mid})$.
6. If $\mathbb{P}_{mid} \leq \epsilon$, then set $\mathbb{P}_r = \mathbb{P}_{mid}$. Otherwise, set $\mathbb{P}_r = \mathbb{P}_{mid}$
7. - If $s_r - s_l > \delta$, then repeat step-4.
   - Otherwise, output $\big(s_{mid}, \mathscr{F}(s_{mid})\big)$ & STOP.

6.4 Summary

In this chapter, we made some new extensions to PCC algorithms extending the mere estimation of the set $\mathcal{F}$. We started with the immediate byproduct of the PCC algorithms, namely the BLER estimation. This requires using an exact formula that we derived from the literature. The bit channel estimates that are already computed are the input to this formula. We also provided a robust iterative formula to avoid overflow, a problem which often arises. In fact this let us slightly redefine the polar code construction algorithm for more accuracy, proposing to use the exact bit channel BERs, instead of Z-parameters. In other words, replacing Bhattacharyya parameters with bit channel BERs as bit channel estimates will aid both the BLER computation as well as an improved PCC definition. When we have multiple BLER estimates from multiple PCC algorithms, we proposed a combined estimate for better accuracy than any of the individual BLER estimates. This has the potential to avoid high complexity Monte-Carlo simulations of polar codes altogether.

Next, we revisited the problem of comparison of different PCC algorithms and proposed a general framework. Using the above BLER computation methods, we proposed a much simpler way to directly find the reliable design-SNR following two approaches that we formalized and provided theoretical results. We substantiated our theory with simulations.

Chapter 7

An FPGA Implementation and a Software Package for Polar Codes

7.1 Introduction to the FPGA Implementation

We have undertaken a project to see some of algorithms in FPGA implementation. We proposed a project with carefully designing the approach, and finally supervised the implementation of the IMFSCD of polar codes on a *software defined radio* platform. This work is in conjunction with Yechao She, from Southeast university, China, as a part of her final year project at Monash University. The results were also presented as a poster in [186], which is reproduced in its entirety in Fig. 7.4.

The main result of this work is that the use of a folded successive cancellation decoder can also reduce the hardware resource requirement (number of gates) significantly from $\mathcal{O}(N)$ to $\mathcal{O}(\sqrt{N})$, if we allow an increase in decoding latency. It is in fact, a complementary result to what we have shown in [139]. In [139], we have shown that the decoder can reduce the delay significantly if we implement the decoder in full scale. The above result shows that if we allow an increase in the delay, hardware requirement of the decoder can be reduced significantly.

The core of the work is a VHDL implementation, whose detailed report is available separately [185]. Due to the domain of the topic falling out of the scope of this thesis, we restrict our discussion to relevant parts only. Parts of it may be quickly inferred from Fig. 7.4.

7.2 The IMFSCD on FPGA

Recall the concept of the IMFSCD, discussed earlier in Section 4.4. The basic technique relies on the decomposition of an SCD graph of larger dimensions in terms of smaller SCD circuits. For

---

Parts of this chapter are based on [185, 186] example, an SCD of size $N = 2^n$, can be thought of as a combination of several smaller SCDs of sizes $N = 2^{n-m}$ and $N = 2^m$ where m is any integer less than n, as was illustrated in Fig. 4.5. An interesting case among them is when n is an even number, where all smaller SCDs can be of same size $2^{n/2}$ as shown in Fig. 7.1. We will denote this special case as *square folding*.

The original objective of the IMFSCD was to reduce the latency by parallelizing some of the above component SCDs. However, when it comes to hardware platforms we may be limited with additional constraints such as limited hardware resources which is more important than the latency. Our decomposition presents a new opportunity in this case due to the fact that the decomposition, as seen in Fig. 7.1, many identical SCD copies are required, in two different dimensions. Further in a square folding, all the smaller SCDs have the same size. The idea is simply that we may implement only one copy for each dimension and reuse.

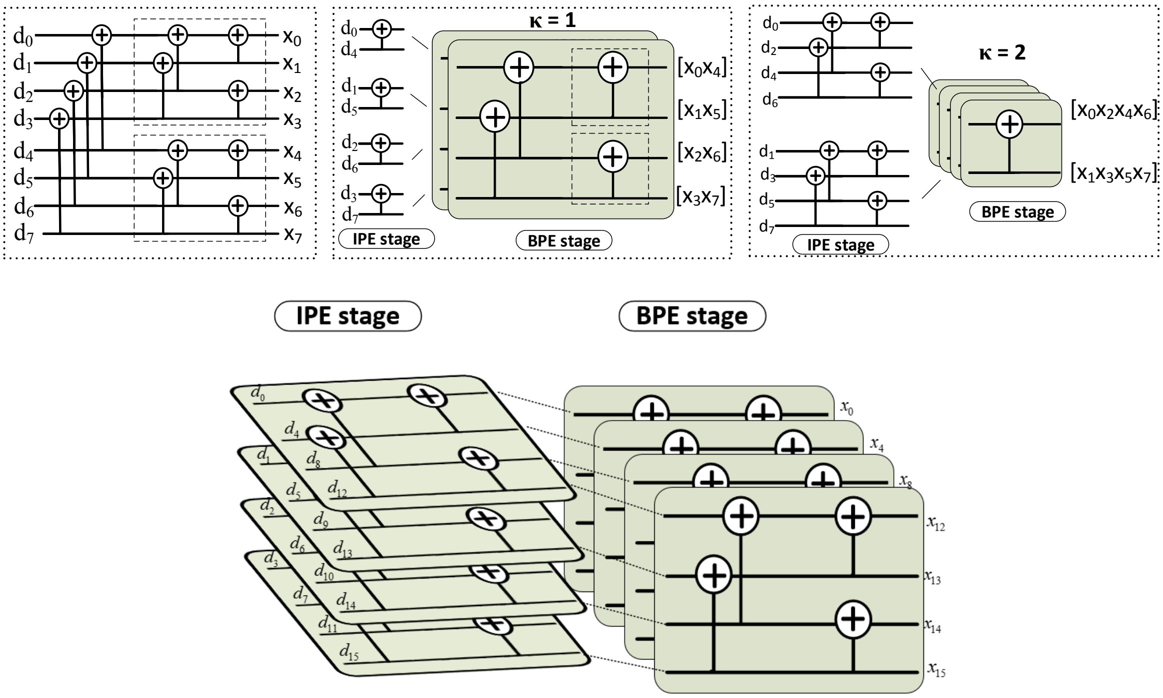


Figure 7.1: Illustrating folding for $N = 8$ and $N = 16$

The gains in hardware resources are immediate due to this reuse and may be quantified as follows. You may recall from Section 4.1 that the original SCD in its space-efficient form requires $\mathcal{O}(N)$ or $\mathcal{O}(2^n)$ nodes. We call the nodes as *processing elements* (PE) in this chapter, and consume a dominant majority of the hardware resources. After folding and reusing of one or two SCD copies, the size of the SCDs have reduced and we will now require $\mathcal{O}(\max 2^{n-m}, 2^m)$ PEs only. In the case of square folding, the number reaches its minimum of $\mathcal{O}(2^{n/2})$ PEs, therefore it is important. A more accurate comparison is given in Table 7.1

The gains are significant, however, at the cost of additional latency as well as additional requirement of register memory. A new hardware architecture is also required to implement the appropriate scheduling, input, and the interconnection of the results from the same SCD.

	IMFSCD	SCD
PEs	$2^m + 2^{n-m} - 2$	$2^n - 1$
Registers	$2^{m+2} + 2^{n-m} - 5$	$3(2^n - 1)$
Decoding Latency (clocks)	$2^n + 2^{2n-2m-1} - 2^{n-m-1} - 1$	$2^n - 1$

Table 7.1: Comparison of hardware resources required for IMFSCD and SCD

A sample architectural changes of the $N = 16$ SCD before and after the square folding is as shown in Figs. 7.2 and 7.3. The upper half represents the tree of computations as we discussed in Section 4.1. The bottom half of the circuits represent the broadcasting of the binary decisions from message nodes. In other words, the log-likelihood portion of the nodes of Section 4.1 is in the above half and the binary decisions portion is in the bottom half. After folding the PEs reduced but get larger with more register memory, forming a single $N = 4$ SCD being reused according to Fig. 7.1.

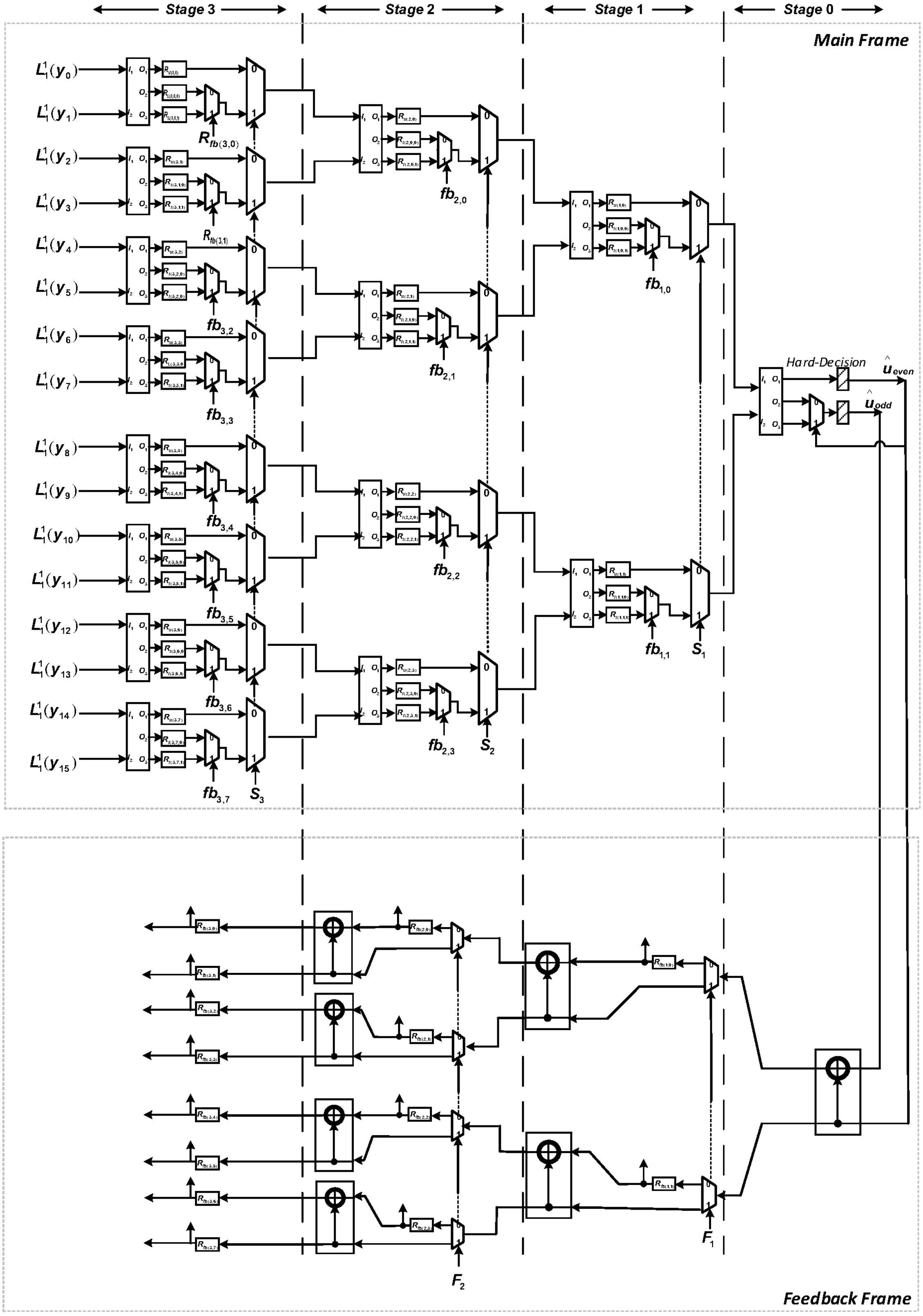


Figure 7.2: Hardware architecture of an SCD of $N = 16$

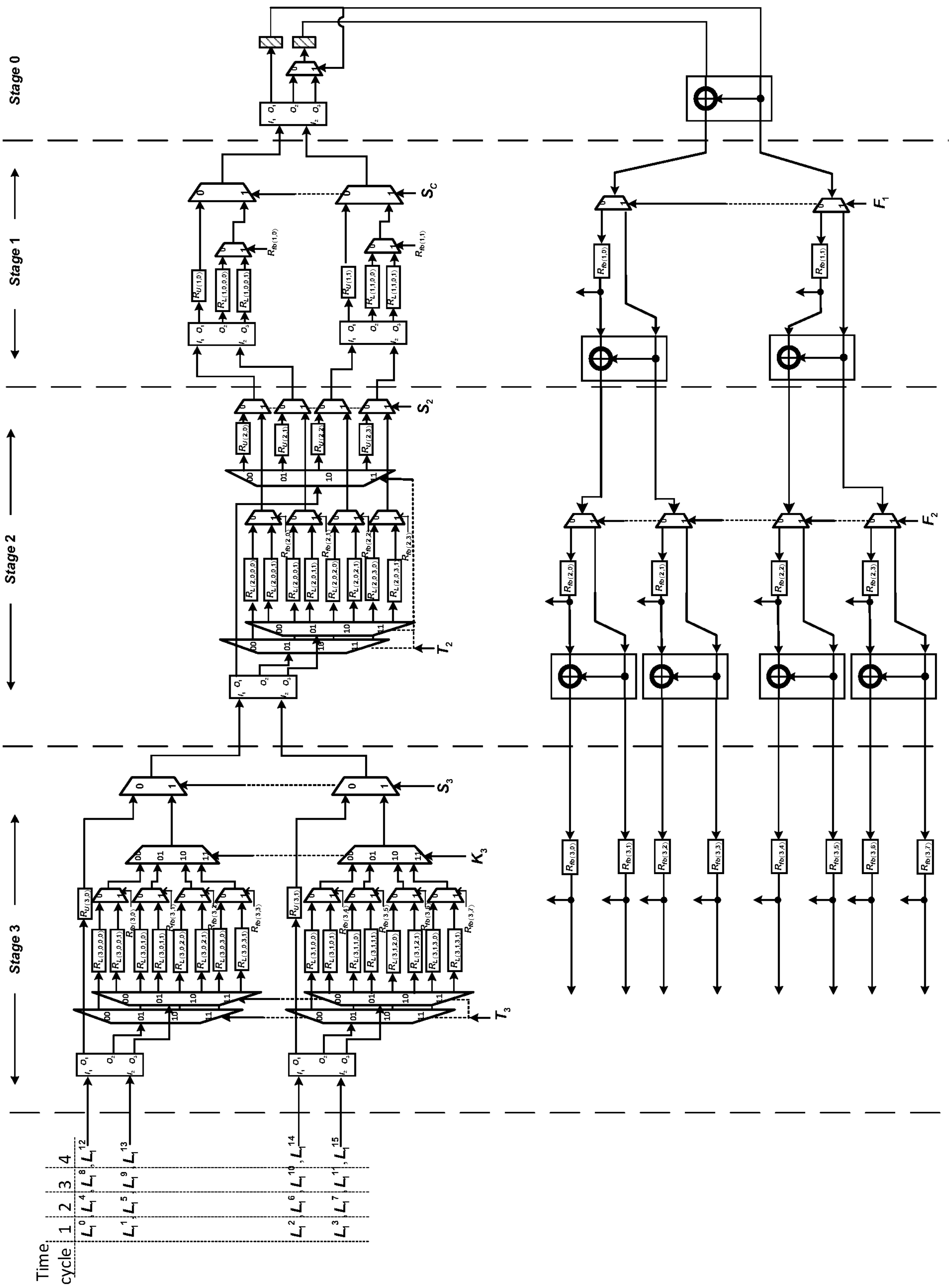
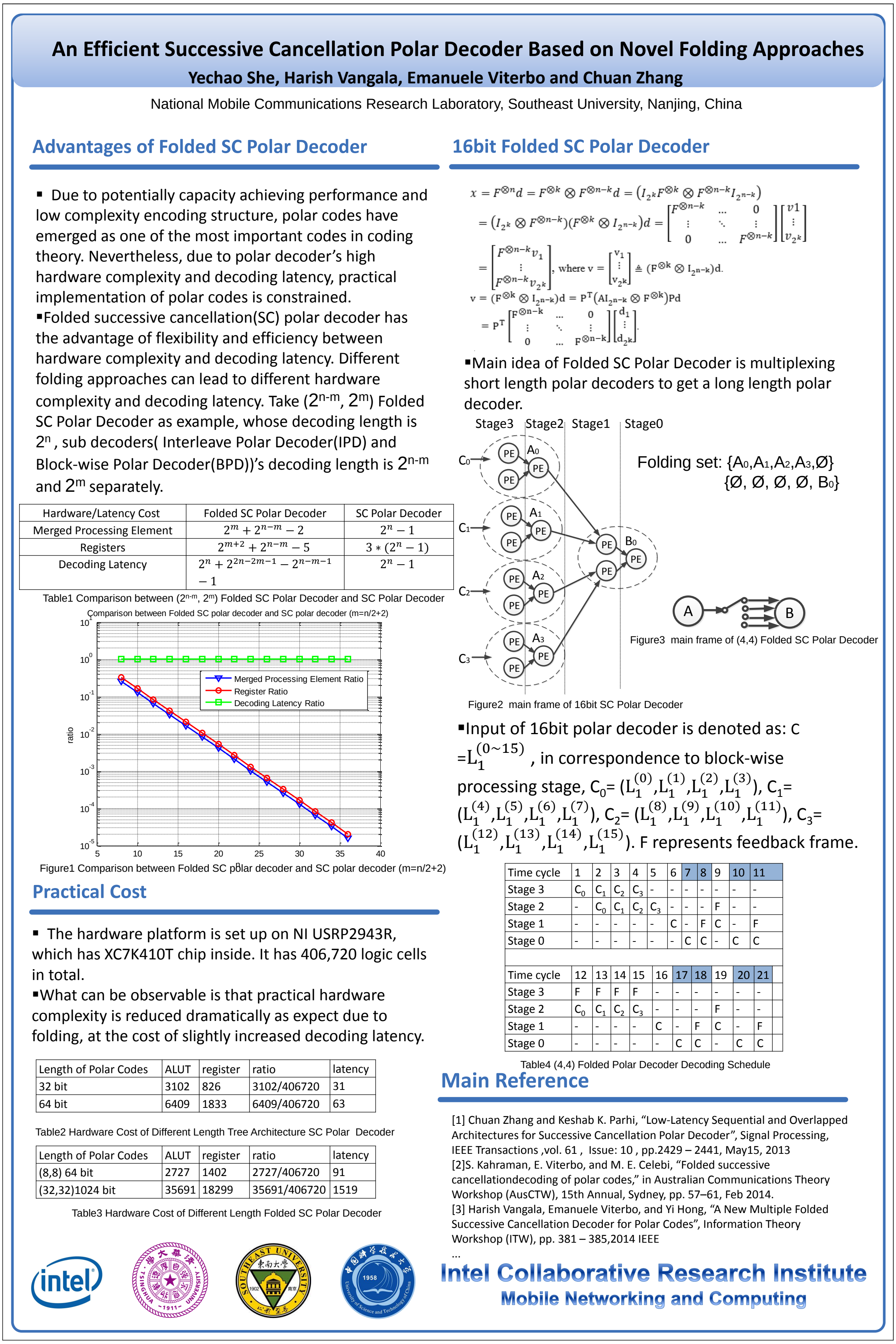

Stage 0
Stage 1
Stage 2
Stage 3
S_C
S_2
S_3
F_1
F_2
K_3
T_2
T_3
Time cycle
1
2
3
4
L_1^0, L_1^4, L_1^8, L_1^12
L_1^1, L_1^5, L_1^9, L_1^13
L_1^2, L_1^6, L_1^10, L_1^14
L_1^3, L_1^7, L_1^11, L_1^15

Figure 7.3: Hardware architecture of the IMFSCD with square folding a reminiscent of $N = 4$



An Efficient Successive Cancellation Polar Decoder Based on Novel Folding Approaches

Yechao She, Harish Vangala, Emanuele Viterbo and Chuan Zhang

National Mobile Communications Research Laboratory, Southeast University, Nanjing, China

Advantages of Folded SC Polar Decoder

- Due to potentially capacity achieving performance and low complexity encoding structure, polar codes have emerged as one of the most important codes in coding theory. Nevertheless, due to polar decoder's high hardware complexity and decoding latency, practical implementation of polar codes is constrained.
- Folded successive cancellation(SC) polar decoder has the advantage of flexibility and efficiency between hardware complexity and decoding latency. Different folding approaches can lead to different hardware complexity and decoding latency. Take (2^{n-m}, 2^m) Folded SC Polar Decoder as example, whose decoding length is 2^n , sub decoders(Interleave Polar Decoder(IPD) and Block-wise Polar Decoder(BPD))'s decoding length is 2^{n-m} and 2^m separately.

Hardware/Latency Cost	Folded SC Polar Decoder	SC Polar Decoder
Merged Processing Element	$2^m + 2^{n-m} - 2$	$2^n - 1$
Registers	$2^{m+2} + 2^{n-m} - 5$	$3 * (2^n - 1)$
Decoding Latency	$2^n + 2^{2n-2m-1} - 2^{n-m-1} - 1$	$2^n - 1$

Table1 Comparison between (2^{n-m}, 2^m) Folded SC Polar Decoder and SC Polar Decoder

Figure1 Comparison between Folded SC polar decoder and SC polar decoder (m=n/2+2)

Practical Cost

- The hardware platform is set up on NI USRP2943R, which has XC7K410T chip inside. It has 406,720 logic cells in total.
- What can be observable is that practical hardware complexity is reduced dramatically as expect due to folding, at the cost of slightly increased decoding latency.

Length of Polar Codes	ALUT	register	ratio	latency
32 bit	3102	826	3102/406720	31
64 bit	6409	1833	6409/406720	63

Table2 Hardware Cost of Different Length Tree Architecture SC Polar Decoder

Length of Polar Codes	ALUT	register	ratio	latency
(8,8) 64 bit	2727	1402	2727/406720	91
(32,32)1024 bit	35691	18299	35691/406720	1519

Table3 Hardware Cost of Different Length Folded SC Polar Decoder

16bit Folded SC Polar Decoder

$$x = F^{\otimes n}d = F^{\otimes k} \otimes F^{\otimes n-k}d = \left(I_{2^k}F^{\otimes k} \otimes F^{\otimes n-k}I_{2^{n-k}}\right)$$

$$= \left(I_{2^k} \otimes F^{\otimes n-k}\right)\left(F^{\otimes k} \otimes I_{2^{n-k}}\right)d = \begin{bmatrix} F^{\otimes n-k} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & F^{\otimes n-k} \end{bmatrix} \begin{bmatrix} v1 \\ \vdots \\ v_{2^k} \end{bmatrix}$$

$$= \begin{bmatrix} F^{\otimes n-k}v_1 \\ \vdots \\ F^{\otimes n-k}v_{2^k} \end{bmatrix}, \text{ where } \mathrm{v} = \begin{bmatrix} \mathrm{v}_1 \\ \vdots \\ \mathrm{v}_{2^k} \end{bmatrix} \triangleq (\mathrm{F}^{\otimes \mathrm{k}} \otimes \mathrm{I}_{2^{n-k}})\mathrm{d}.$$

$$\mathrm{v} = (\mathrm{F}^{\otimes \mathrm{k}} \otimes \mathrm{I}_{2^{n-k}})\mathrm{d} = \mathrm{P}^{\mathrm{T}}\left(\mathrm{AI}_{2^{n-k}} \otimes \mathrm{F}^{\otimes \mathrm{k}}\right)\mathrm{Pd}$$

$$= \mathrm{P}^{\mathrm{T}} \begin{bmatrix} \mathrm{F}^{\otimes \mathrm{n-k}} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathrm{F}^{\otimes \mathrm{n-k}} \end{bmatrix} \begin{bmatrix} \mathrm{d}_1 \\ \vdots \\ \mathrm{d}_{2^k} \end{bmatrix}.$$

- Main idea of Folded SC Polar Decoder is multiplexing short length polar decoders to get a long length polar decoder.

Folding set: {A_0,A_1,A_2,A_3,Ø}
{Ø, Ø, Ø, Ø, B_0}

Figure3 main frame of (4,4) Folded SC Polar Decoder

Figure2 main frame of 16bit SC Polar Decoder

- Input of 16bit polar decoder is denoted as: C $=L_1^{(0\sim15)}$, in correspondence to block-wise processing stage, C_0= ($L_1^{(0)}$,$L_1^{(1)}$,$L_1^{(2)}$,$L_1^{(3)}$), C_1= ($L_1^{(4)}$,$L_1^{(5)}$,$L_1^{(6)}$,$L_1^{(7)}$), C_2= ($L_1^{(8)}$,$L_1^{(9)}$,$L_1^{(10)}$,$L_1^{(11)}$), C_3= ($L_1^{(12)}$,$L_1^{(13)}$,$L_1^{(14)}$,$L_1^{(15)}$). F represents feedback frame.

Time cycle	1	2	3	4	5	6	7	8	9	10	11
Stage 3	C_0	C_1	C_2	C_3	-	-	-	-	-	-	-
Stage 2	-	C_0	C_1	C_2	C_3	-	-	-	F	-	-
Stage 1	-	-	-	-	-	C	-	F	C	-	F
Stage 0	-	-	-	-	-	-	C	C	-	C	C

Time cycle	12	13	14	15	16	17	18	19	20	21	
Stage 3	F	F	F	F	-	-	-	-	-	-	
Stage 2	C_0	C_1	C_2	C_3	-	-	-	F	-	-	
Stage 1	-	-	-	-	C	-	F	C	-	F	
Stage 0	-	-	-	-	-	C	C	-	C	C	

Table4 (4,4) Folded Polar Decoder Decoding Schedule

Main Reference

[1] Chuan Zhang and Keshab K. Parhi, "Low-Latency Sequential and Overlapped Architectures for Successive Cancellation Polar Decoder", Signal Processing, IEEE Transactions ,vol. 61 , Issue: 10 , pp.2429 – 2441, May15, 2013
[2]S. Kahraman, E. Viterbo, and M. E. Celebi, "Folded successive cancellationdecoding of polar codes," in Australian Communications Theory Workshop (AusCTW), 15th Annual, Sydney, pp. 57–61, Feb 2014.
[3] Harish Vangala, Emanuele Viterbo, and Yi Hong, "A New Multiple Folded Successive Cancellation Decoder for Polar Codes", Information Theory Workshop (ITW), pp. 381 – 385,2014 IEEE
...

Intel Collaborative Research Institute
Mobile Networking and Computing

Figure 7.4: Poster presented at *Semi-annual Conference of Intel Collaborative Research Institutes on Mobile Networking and Computing*, held at Beijing, 2015.

7.3 The Software Package and Tutorials

In addition to the rich literature on polar codes, in this thesis, we have proposed several new algorithms on polar codes. We found that a significant effort is involved in attempting to use polar codes by various researchers due to the difficulty of implementing them, especially for beginners. We believed that this could be counter-productive to the widespread application of polar codes, inspite of their usefulness.

We found that the needs of various researchers interested in polar codes could be of two types. First, it could be those who like to consider polar codes as a block without any further insights, and verify its performance before they go ahead adapting and incorporating them into their systems. Second, is those who would like to work on the theoretical aspects of polar codes and channel polarization.

A reference implementation of polar codes with the basic blocks of encoding, decoding, and construction in a ready-to-run form could therefore become great utility to many researchers interested in polar codes. With this motivation, we have undertaken a project to implement polar codes as a user-friendly MATLAB package and made it freely available online at `www.polarcodes.com` [125]. Its complete documentation is also made available, and is copied in the appendix.

We are happy to find that our objectives have been achieved as the response to our website has been overwhelming with hundreds of visits every week (See Fig. 7.5). Encouraged by this success, we have also made available a video tutorial on polar codes on youtube, and linked it on `www.polarcodes.com`. This also has already obtained a few thousands of views.

Recently, MathWorks, Inc. has approached us for an official project of redesigning our package so as to be included into the Communications System Toolbox™ of MATLAB®. This project has completed as of August 2017.

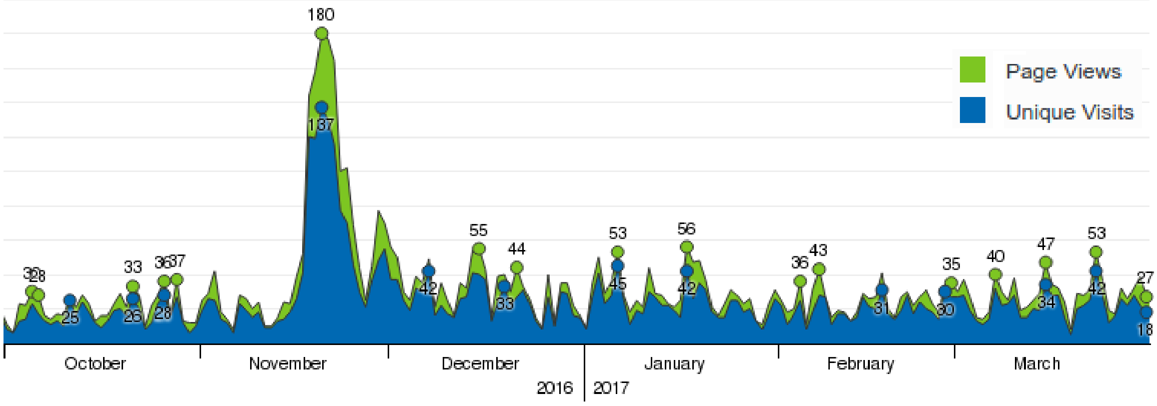


Figure 7.5: Visitor traffic to our website `www.polarcodes.com` between Oct 2016 – Mar 2017 (The peak visits were during the time when polar codes were accepted into 5G)

Chapter 8

Conclusions and Future Directions

8.1 Conclusions

In this thesis, we have conducted an extensive study on various aspects of polar codes and derived efficient algorithms with the main objectives of improving the error correction performance or latency. We have made several contributions which can be summarized as follows.

Initially, we have discussed the fundamentals and our motivations for studying polar codes in Chapters 1 and 2. Along side, we have also provided easy-to-use pseudocode implementations of encoding, decoding, and construction of polar codes, which are not easily available in literature. From here onwards our contributions spanned until the end of the thesis, as concluded below.

The Chapter 3 has illustrated the advantages and the need of systematic polar encoding and provided efficient algorithms equally as fast as the original non-systematic polar codes. This derives a remarkable conclusion that systematic polar codes are a better choice to the original polar codes since they have an improved BER with SCD and almost the all available decoders are extensible with a trivial overhead (at most one additional encoding operation per decoding). In the same chapter, we have also provided insights of how to make the encoding a flat non-recursive algorithm.

The Chapter 4 has presented several variants of the SCD, and the LSCD, namely the PSCD, IMFSCD, and the parellelized LSCD. These have distinct advantages such as altering the decoding order to natural order, reducing the latency of SCD and LSCD, and reducing the hardware requirements. An extensive discussion of these variants has been provided. In the same chapter, we have also provided a flat non-recursive SCD implementation, which is unavailable in literature. Such an implementation is of critical importance in general for its efficiency in both software and hardware.

The Chapter 5 has presented an extensive survey of several PCC algorithms available from the literature. We then performed a comprehensive comparison of the algorithms and concluded that one can elicit best performing polar codes from any algorithm provided we carefully tailor the design-SNR value. We also provided a heuristic to choose the best design-SNR that is appropriate for a consistently good performance over a range of operating-SNRs. A detailed table of good design-SNRs is given for polar codes of length upto 2^{16}. Efficient pseudocode implementations of various PCCs is also provided. Additionally, we discussed a new channel quantizer with optimal mutual information, which can be instrumental in novel construction algorithms, following the lead of Tal & Vardy.

Chapter 6 has extended our study on PCCs to use them for a different application of quickly estimating the BLER of polar codes without any Monte-Carlo simulations. We have derived an accurate BLER estimation algorithm that utilizes several PCCs that are available with the user. Using this accurate BLER estimation, we have extended the earlier heuristic design-SNR computation, and gave a much faster algorithm to find a good design-SNR, along with its theoretical characterization.

Finally, Chapter 7 has provided the details of an efficient FPGA implementation that we undertook as a collaboration. It also discussed a software package called "Polar Coding Algorithms in MATLAB" that we developed from scratch. As a service to the community we have freely made available our package at `www.polarcodes.com`. This contained several efficient algorithms that are useful for the researchers on polar codes, and has already been receiving a good response from the community with regular visits to the above online resource and everyday downloads.

Recently our software package has been taken by MathWorks, Inc., for inclusion into the Communication Toolbox™ of MATLAB®. A special project was initiated on a proposal from MathWorks to modify the package to suit the needs of the Communication Toolbox of MATLAB. The project has completed as of August 2017.

This concludes our work on polar codes during my doctoral studies. We will finally see some of the possible extensions that are worth investigating in future.

8.2 Future Directions

We will now see some problems that we believe will be worth pursuing for further gains of polar codes.

8.2.1 Application to Fading Channels

One can easily extend the ideas of polar encoding, SCD, and the construction to fading channels however, optimality of the code construction is no more guaranteed under an SCD or LSCD. Therefore, application of polar codes to fading channels is an exciting area of open problems with some elementary work already available [159, 187–192].

We already know that under suboptimal constructions itself, polar codes offer superior performance to some of the popular LDPC codes in literature. Extending the ideas of polar code construction algorithms for their optimality will therefore be worth pursuing, especially given that they are already into 5G.

8.2.2 New Decoders to Polar Codes

The most popularly acknowledged polar decoder with near-ML performance and low-complexity is an LSCD with CRC. SCD, LSCD, and LSCD with CRC form the extended family of decoders that have incrementally improved performance at the cost of much additional latency to the SCD (which is already sequential) and additional complexity. The higher latency of an LSCD is therefore currently a problem that is actively being pursued for their adoption in realtime implementations of polar codes. To this extent we have already made one proposal in Chapter 4.

Two trajectories can therefore be drawn to achieve better performance with lower cost.

1. Low latency LSCD with techniques such as parallelization
2. New decoders with low latency such as advanced belief propagation

8.2.3 Code Optimization Strategies

One important contribution that is often ignored from Arıkan is the way he automatically incorporated a code search in the name of a PCC algorithm. In short, the algorithm extends the well known Reed-Muller family of codes and picks a different code from the extended family that has better performance under a novel low complexity decoding than Reed-Muller codes. It is important to note the three components that made polar codes worth pursuing:

1. A large family (or larger if it is already one) of codes.
2. A low complexity decoder for any code in the family.
3. Theoretical estimate of the performance of the code, as a function of the choice of the code.

This has essentially setup a simple optimization problem whose solution is a code within the family of codes that already has a low complexity decoding. We believe this idea can be extended to other families of codes, which we are currently investigating. We strongly believe it has a greater potential in future.

8.2.4 Error Floors with Polar Codes

It was established over the BSC that polar codes won't have error floors. It was an important result for the practical consideration of polar codes, especially in applications that require very low BER, such as storage. It was not clear whether this result extends to AWGN channels and no simulation has ever been performed beyond $10^{-6} - 10^{-8}$ BLER. There are two approaches immediately available to investigate the error floors.

1. Develop methods to estimate the BLER values down to at least as low as 10^{-20}, by using rare event simulation models such as importance sampling, and subset sampling.
2. Theoretically estimating an efficient upper bound to the BLER of polar codes over AWGN channels.

Bibliography

[1] C. E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3, July 1948, pp. 379–423 [On pg: 1].

[2] Y. Polyanskiy. “Channel coding: non-asymptotic fundamental limits”. PhD dissertation. Dept. of electrical engineering: Princeton University, Nov. 2010 [On pg: 1, 3].

[3] Y. Polyanskiy, H. V. Poor, and S. Verdu. “Channel Coding Rate in the Finite Blocklength Regime”. In: *Information Theory, IEEE Transactions on* 56.5, May 2010, pp. 2307–2359 [On pg: 1].

[4] B. K. Butler. “Error Floors of LDPC Codes and Related Topics”. PhD dissertation. Electrical Engineering, University of California, San Diego, 2013 [On pg: 1].

[5] B. K. Butler and P. H. Siegel. “Error Floor Approximation for LDPC Codes in the AWGN Channel”. In: *IEEE Transactions on Information Theory* 60.12, Dec. 2014, pp. 7416–7441 [On pg: 1].

[6] R. Garello et al. “On error floor and free distance of turbo codes”. In: *IEEE International Conference on Communications (ICC).* Vol. 1. June 2001, pp. 45–49 [On pg: 1].

[7] R. Garello, P. Pierleoni, and S. Benedetto. “Computing the free distance of turbo codes and serially concatenated codes with interleavers: algorithms and applications”. In: *IEEE Journal on Selected Areas in Communications* 19.5, May 2001, pp. 800–812 [On pg: 1].

[8] T. J. Richardson and R. L. Urbanke. “Efficient Encoding of Low-Density Parity-Check Codes”. In: *IEEE Transactions on Information Theory* 47.2, Feb. 2001, pp. 638–656 [On pg: 1].

[9] M. Awais et al. “A Novel Architecture for Scalable, High Throughput, Multi-standard LDPC Decoder”. In: *Digital System Design (DSD), 2011 14th Euromicro Conference on.* Aug. 2011, pp. 340–347 [On pg: 1].

[10] S. u. Rehman et al. “On-chip implementation of memory mapping algorithm to support flexible decoder architecture”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* May 2013, pp. 2751–2755 [On pg: 1].

[11] E. Arıkan. “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels”. In: *IEEE Transactions on Information Theory* 55.7, July 2009, pp. 3051–3073 [On pg: 1, 2, 9, 15, 17, 34, 38, 43, 51, 56, 64, 65, 66, 68, 137, 138, 139].

[12] E. Şaşoğlu. *Polarization and Polar Codes.* Now publishers, 2012, p. 125 [On pg: 1].

[13] A. G. Sahebi and S. S. Pradhan. “Multilevel Channel Polarization for Arbitrary Discrete Memoryless Channels”. In: *IEEE Transactions on Information Theory* 59.12, Dec. 2013, pp. 7839–7857 [On pg: 1].

[14] E. Şaşoğlu and L. Wang. “Universal Polarization”. In: *IEEE International Symposium on Information Theory (ISIT).* Honolulu, HI, June 2014, pp. 1456–1460 [On pg: 1, 64].

[15] R. Nasser. “Ergodic theory meets polarization I: A foundation of polarization theory”. In: *2015 IEEE International Symposium on Information Theory (ISIT).* June 2015, pp. 2451–2455 [On pg: 1].

[16] R. Nasser. "Ergodic theory meets polarization II: A foundation of polarization theory for MACs". In: *2015 IEEE International Symposium on Information Theory (ISIT).* June 2015, pp. 2456–2460 [On pg: 1].

[17] S. Kudekar et al. "Threshold saturation on BMS channels via spatial coupling". In: *2010 6th International Symposium on Turbo Codes Iterative Information Processing.* Sept. 2010, pp. 309–313 [On pg: 2].

[18] S. Kudekar, T. J. Richardson, and R. L. Urbanke. "Threshold Saturation via Spatial Coupling: Why Convolutional LDPC Ensembles Perform So Well over the BEC". In: *IEEE Transactions on Information Theory* 57.2, Feb. 2011, pp. 803–834 [On pg: 2].

[19] S. Kudekar, T. J. Richardson, and R. L. Urbanke. "Spatially Coupled Ensembles Universally Achieve Capacity Under Belief Propagation". In: *IEEE Transactions on Information Theory* 59.12, Dec. 2013, pp. 7761–7813 [On pg: 2].

[20] S. Kudekar et al. "Reed-Muller Codes Achieve Capacity on Erasure Channels". In: *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing.* STOC '16. Cambridge, MA, USA: ACM, 2016, pp. 658–669 [On pg: 2].

[21] S. Kudekar et al. "Reed-Muller Codes Achieve Capacity on Erasure Channels". In: *arXiv:1601.04689 [cs.IT]*, 2016. (under review IEEE TIT), pp. 1–18 [On pg: 2].

[22] S. Kumar et al. "Threshold Saturation for Spatially Coupled LDPC and LDGM Codes on BMS Channels". In: *IEEE Transactions on Information Theory* 60.12, Dec. 2014, pp. 7389–7415 [On pg: 2].

[23] S. Kumar, R. Calderbank, and H. D. Pfister. "Reed-Muller codes achieve capacity on the quantum erasure channel". In: *2016 IEEE International Symposium on Information Theory (ISIT).* July 2016, pp. 1750–1754 [On pg: 2].

[24] S. Kumar, R. Calderbank, and H. D. Pfister. "Beyond double transitivity: Capacity-achieving cyclic codes on erasure channels". In: *2016 IEEE Information Theory Workshop (ITW).* Sept. 2016, pp. 241–245 [On pg: 2].

[25] S. Kudekar et al. "Comparing the bit-MAP and block-MAP decoding thresholds of reed-muller codes on BMS channels". In: *2016 IEEE International Symposium on Information Theory (ISIT).* July 2016, pp. 1755–1759 [On pg: 2].

[26] N. Stolte. "Rekursive Codes mit der Plotkin-Konstruktion und ihre Decodierung". (Translation available: *Recursive Codes with the Plotkin-Construction and Their Decoding*). PhD dissertation. University of Technology Darmstadt, Germany: Faculty of Electronics and Information Technology, 2002 [On pg: 2].

[27] M. El-Khamy, H. P. Lin, and J. Lee. "Binary polar codes are optimised codes for bitwise multistage decoding". In: *Electronics Letters* 52.13, 2016, pp. 1130–1132 [On pg: 2].

[28] P. Giard et al. "Hardware decoders for polar codes: An overview". In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS).* May 2016, pp. 149–152 [On pg: 2, 3].

[29] G. Sarkis and W. J. Gross. "Implementation of Polar Decoders". In: *Advanced Hardware Design for Error Correcting Codes.* Ed. by C. Chavet and P. Coussy. Cham: Springer International Publishing, 2015, pp. 33–45 [On pg: 2, 3].

[30] B. L. Gal, C. Leroux, and C. Jego. "Multi-Gb/s Software Decoding of Polar Codes". In: *IEEE Transactions on Signal Processing* 63.2, Jan. 2015, pp. 349–359 [On pg: 2, 3].

[31] B. L. Gal, C. Leroux, and C. Jego. "A scalable 3-phase polar decoder". In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS).* May 2016, pp. 417–420 [On pg: 2, 3].

[32] P. Giard et al. "237 Gbit/s unrolled hardware polar decoder". In: *Electronics Letters* 51.10, 2015, pp. 762–763 [On pg: 2, 3].

[33] G. Sarkis et al. "Fast Polar Decoders: Algorithm and Implementation". In: *IEEE Journal on Selected Areas in Communications* 32.5, May 2014, pp. 946–957 [On pg: 2, 3, 22, 28].

[34] O. Dizdar and E. Arıkan. “A High-Throughput Energy-Efficient Implementation of Successive Cancellation Decoder for Polar Codes Using Combinational Logic”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 63.3, Mar. 2016, pp. 436–447 [On pg: 2, 3, 10, 28].

[35] A. Pamuk and E. Arıkan. “A Two Phase Successive Cancellation Decoder Architecture for Polar Codes”. In: *IEEE International Symposium on Information Theory (ISIT).* Istanbul, July 2013, pp. 957–961 [On pg: 2, 3].

[36] T. Liu and L. F. Network. *Polar Code selected 5G standard, Huawei to get the short code control channel, what does that mean?* Online. http://www.globaltimes.cn/content/1019126.shtml. Nov. 2016 [On pg: 2].

[37] L. Liu and C. Zhang. “Circuits and systems for 5G network: Massive MIMO and advanced coding”. In: *2015 IEEE 11th International Conference on ASIC (ASICON).* Nov. 2015, pp. 1–4 [On pg: 2].

[38] D. Bushell-Embling. *Huawei achieves 27Gbps 5G speeds with Polar Code.* Online article. `http://www.telecomasia.net/content/huawei-achieves-27gbps-5g-speeds-polar-code`. Oct. 2016 [On pg: 2, 3].

[39] E. Arıkan. “A Performance Comparison of Polar Codes and Reed-Muller Codes”. In: *IEEE Communications Letters* 12.6, June 2008, pp. 447–449 [On pg: 3, 17, 45, 64, 65, 66, 67].

[40] N. Hussami, S. B. Korada, and R. L. Urbanke. “Performance of Polar Codes for Channel and Source Coding”. In: *IEEE International Symposium on Information Theory (ISIT).* Seoul, July 2009, pp. 1488–1492 [On pg: 3].

[41] A. Eslami and H. Pishro-Nik. “On Finite-Length Performance of Polar Codes: Stopping Sets, Error Floor, and Concatenated Design”. In: *IEEE Transactions on Communications* 61.3, Mar. 2013, pp. 919–929 [On pg: 3].

[42] Y. Zhang et al. “A Modified Belief Propagation Polar Decoder”. In: *IEEE Communications Letters* 18.7, July 2014, pp. 1091–1094 [On pg: 3, 65].

[43] Y. Zhang et al. “A Simplified Belief Propagation Decoder for Polar Codes”. In: *IEEE International Wireless Symposium (IWS).* Xian, Mar. 2014, pp. 1–4 [On pg: 3].

[44] S. M. Abbas et al. “Low Complexity Belief Propagation Polar Code Decoders”. In: *arXiv:1505.04979 [cs.IT]*, May 2015 [On pg: 3].

[45] O. Dogan. “An investigation on belief propagation decoding of polar codes”. Masters Thesis, Middle East Technical University, Ankara, Turkey: Department of Electrical and Electronics Engineering, Dec. 2015 [On pg: 3].

[46] S. Jin et al. “A memory efficient belief propagation decoder for polar codes”. In: *China Communications* 12.5, May 2015, pp. 34–41 [On pg: 3].

[47] J. Guo et al. “Enhanced Belief Propagation Decoding of Polar Codes through Concatenation”. In: *IEEE International Symposium on Information Theory (ISIT).* Honolulu, HI, June 2014, pp. 2987–2991 [On pg: 3].

[48] B. Yuan and K. K. Parhi. “Architectures for Polar BP Decoders using Folding”. In: *IEEE International Symposium on Circuits and Systems (ISCAS).* Melbourne, June 2014, pp. 205–208 [On pg: 3].

[49] A. Elkelesh et al. “Improving Belief Propagation decoding of polar codes using scattered EXIT charts”. In: *IEEE Information Theory Workshop (ITW).* Sept. 2016, pp. 91–95 [On pg: 3].

[50] Y. Ren et al. “Efficient early termination schemes for belief-propagation decoding of polar codes”. In: *2015 IEEE 11th International Conference on ASIC (ASICON).* Nov. 2015, pp. 1–4 [On pg: 3].

[51] J. Sha, J. Lin, and Z. Wang. “Stage-combined belief propagation decoding of polar codes”. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS).* May 2016, pp. 421–424 [On pg: 3].

[52] B. Yuan and K. K. Parhi. “Belief propagation decoding of polar codes using stochastic computing”. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS).* May 2016, pp. 157–160 [On pg: 3].

[53] I. Tal and A. Vardy. “List Decoding of Polar Codes”. In: *IEEE International Symposium on Information Theory (ISIT).* St. Petersburg, Aug. 2011, pp. 1–5 [On pg: 3, 33, 52, 56, 57, 59, 60, 139].

[54] G. Bonik, S. Goreinov, and N. Zamarashkin. “A variant of list plus CRC concatenated polar code”. In: *arXiv:1207.4661 [cs.IT]*, July 2012, pp. 1–4 [On pg: 3, 52, 54].

[55] K. Chen, K. Niu, and J. Lin. “List successive cancellation decoding of polar codes”. In: *Electronics Letters* 48.9, Apr. 2012, pp. 500–501 [On pg: 3, 52].

[56] B. Li, H. Shen, and D. Tse. “An Adaptive Successive Cancellation List Decoder for Polar Codes with Cyclic Redundancy Check”. In: *IEEE Communications Letters* 16.12, Dec. 2012, pp. 2044–2047 [On pg: 3, 52, 54].

[57] K. Chen, K. Niu, and J. Lin. “A Reduced-Complexity Successive Cancellation List Decoding of Polar Codes”. In: *Vehicular Technology Conference (VTC spring), IEEE 77th.* Dresden, June 2013, pp. 1–5 [On pg: 3, 52, 55].

[58] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg. “LLR-based successive cancellation list decoding of polar codes”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* Florence, May 2014, pp. 3903–3907 [On pg: 3, 52, 55, 56].

[59] C. Cao et al. “Low complexity list successive cancellation decoding of polar codes”. In: *IET Communications* 8.17, Nov. 2014, pp. 3145–3149 [On pg: 3, 52, 55].

[60] G. Sarkis et al. “Increasing the Speed of Polar List Decoders”. In: *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2014, pp. 1–6 [On pg: 3, 52, 55].

[61] C. Xiong, J. Lin, and Z. Yan. “Symbol-Based Successive Cancellation List Decoder for Polar Codes”. In: *IEEE Workshop on Signal Processing Systems (SiPS).* Belfast, UK, Oct. 2014, pp. 1–6 [On pg: 3, 52, 55].

[62] B. Yuan and K. K. Parhi. “Successive Cancellation List Polar Decoder using Log-likelihood Ratios”. In: *48th Asilomar Conference on Signals, Systems, and Computers.* Pacific Grove,CA, Nov. 2014, pp. 548–552 [On pg: 3, 52, 55, 56].

[63] B. Yuan and K. K. Parhi. “Low-Latency Successive-Cancellation List Decoders for Polar Codes With Multibit Decision”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.10, Oct. 2015, pp. 2268–2280 [On pg: 3, 52, 55].

[64] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg. “LLR-Based Successive Cancellation List Decoding of Polar Codes”. In: *Signal Processing, IEEE Transactions on* 63.19, Oct. 2015, pp. 5165–5179 [On pg: 3].

[65] Y. Fan et al. “Low-latency List Decoding Of Polar Codes With Double Thresholding”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2015, pp. 1042–1046 [On pg: 3].

[66] Y. Fan et al. “A Low-Latency List Successive-Cancellation Decoding Implementation for Polar Codes”. In: *IEEE Journal on Selected Areas in Communications* 34.2, Feb. 2016, pp. 303–317 [On pg: 3].

[67] H. F. Jing and L. E. Li. “A practical CRCs-ADSCL decoding scheme for systematic polar codes”. In: *2016 8th International Conference on Wireless Communications Signal Processing (WCSP).* Oct. 2016, pp. 1–5 [On pg: 3].

[68] S. A. Hashemi, C. Condo, and W. J. Gross. “A Fast Polar Code List Decoder Architecture Based on Sphere Decoding”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 63.12, Dec. 2016, pp. 2368–2380 [On pg: 3].

[69] J. V. Wonterghem et al. “Performance Comparison of Short-Length Error-Correcting Codes”. In: *arXiv:1609.07907 [cs.IT]*, 2016. (submitted to IEEE SCVT 2016 conference), pp. 1–6 [On pg: 3].

[70] N. Goela, S. B. Korada, and M. Gastpar. “On LP Decoding of Polar Codes”. In: *IEEE Information Theory Workshop(ITW).* Dublin, Sept. 2010, pp. 1–5 [On pg: 3].

[71] V. Taranalli and P. H. Siegel. “Adaptive Linear Programming Decoding of Polar Codes”. In: *IEEE International Symposium on Information Theory (ISIT).* Honolulu, HI, June 2014, pp. 2982–2986 [On pg: 3].

[72] H. Mahdavifar et al. “Fast Multi-dimensional Polar Encoding and Decoding”. In: *Information Theory and Applications Workshop.* Yet to confirm whether it is from ITA’2014... not possible on IEEE because it updates very late and this is very recent. San Diego, CA, Feb. 2014, pp. 1–5 [On pg: 3].

[73] K. Niu and K. Chen. “Stack decoding of polar codes”. In: *Electronics Letters* 48.12, June 2012, pp. 695–697 [On pg: 3].

[74] K. Niu and K. Chen. “CRC-Aided Decoding of Polar Codes”. In: *IEEE Communications Letters* 16.10, Oct. 2012, pp. 1668–1671 [On pg: 3].

[75] V. Miloslavkaya and P. Trifonov. “Sequential Decoding of Polar Codes”. In: *IEEE Communications Letters* 18.7, July 2014, pp. 1127–1130 [On pg: 3].

[76] G. Trofimiuk and P. Trifonov. “Block sequential decoding of polar codes”. In: *2015 International Symposium on Wireless Communication Systems (ISWCS).* Aug. 2015, pp. 326–330 [On pg: 3].

[77] D. Wu et al. “Ordered Statistic Decoding for Short Polar Codes”. In: *IEEE Communications Letters* 20.6, June 2016, pp. 1064–1067 [On pg: 3].

[78] U. U. Fayyaz and J. R. Barry. “Polar codes for partial response channels”. In: *IEEE International Conference on Communications (ICC).* June 2013, pp. 4337–4341 [On pg: 3].

[79] U. U. Fayyaz and J. R. Barry. “A Low-Complexity Soft-Output Decoder for Polar Codes”. In: *IEEE Global Communications Conference (GLOBECOM).* Atlanta, GA, USA, Dec. 2013, pp. 2692–2697 [On pg: 3].

[80] A. Hadi and E. Alsusa. “On the application of the fast Hadamard transform in Polar codes”. In: *IEEE 17th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC).* July 2016, pp. 1–5 [On pg: 3].

[81] S. Kahraman, E. Viterbo, and M. E. Celebi. “Folded Tree Maximum-Likelihood Decoder for Kronecker Product-Based Codes”. In: *Allerton Conference on Communication, Control and Computing, 51st Annual.* Monticello, IL, Oct. 2013, pp. 629–636 [On pg: 3, 48].

[82] S. Kahraman and M. E. Celebi. “Code Based Efficient Maximum-Likelihood Decoding of Short Polar Codes”. In: *IEEE International Symposium on Information Theory (ISIT).* Cambridge, MA, July 2012, pp. 1967–1971 [On pg: 3].

[83] A. Alamdar-Yazdi and F. R. Kschischang. “A Simplified Successive-Cancellation Decoder for Polar Codes”. In: *IEEE Communications Letters* 15.12, Dec. 2011, pp. 1378–1380 [On pg: 3].

[84] Z. L. Huang, C. J. Diao, and M. Chen. “Latency reduced method for modified successive cancellation decoding of polar codes”. In: *Electronics Letters* 48.23, Nov. 2012, pp. 1505–1506 [On pg: 3].

[85] Z. Huang, C. Diao, and M. Chen. “Multiple Candidates Successive-Cancellation Decoding of Polar Codes”. In: *International Conference on Wireless Communications and Signal Processing (WCSP).* Huangshan, 2012, pp. 1–4 [On pg: 3].

[86] G. Sarkis and W. Gross. “Polar codes for data storage applications”. In: *Computing, Networking and Communications (ICNC), 2013 International Conference on.* Jan. 2013, pp. 840–844 [On pg: 3].

[87] B. Li, H. Shen, and D. Tse. “Parallel Decoders of Polar Codes”. In: *arXiv:1309.1026 [cs.IT]*, Sept. 2013, pp. 1–4 [On pg: 3, 55].

[88] Z. Huang et al. “An Improvement of Modified Successive-Cancellation Decoder for Polar Codes”. In: *IEEE Communications Letters* 17.12, Dec. 2013, pp. 2360–2363 [On pg: 3].

[89] S. Kahraman, E. Viterbo, and M. E. Celebi. "Folded Successive Cancelation Decoding of Polar Codes". In: *Australian Communications Theory Workshop (AusCTW), 15th Annual.* Sydney, NSW, Feb. 2014, pp. 57–61 [On pg: 3, 48, 50, 51, 55].

[90] B. Li et al. "Low-Latency Polar Codes via Hybrid Decoding". In: *International Symposium on Turbo Codes and Iterative Information Processing (ISTC), 8th.* Bremen, Germany, Aug. 2014, pp. 223–227 [On pg: 3, 55].

[91] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg. "A Low-Complexity Improved Successive Cancellation Decoder for Polar Codes". In: *Asilomar Conference on Signals, Systems, and Computers, 48th.* Pacific Grove,CA, Nov. 2014, pp. 2116–2120 [On pg: 3].

[92] S. Kahraman, E. Viterbo, and M. E. Celebi. "Multiple Folding for Successive Cancelation Decoding of Polar Codes". In: *IEEE Wireless Communication Letters* 3.5, Oct. 2014, pp. 545–548 [On pg: 3, 48, 50, 51].

[93] H. Yoo and I.-C. Park. "Efficient Pruning for Successive-Cancellation Decoding of Polar Codes". In: *IEEE Communications Letters* 20.12, Dec. 2016, pp. 2362–2365 [On pg: 3].

[94] C. Leroux et al. "Hardware architectures for successive cancellation decoding of polar codes". In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* May 2011, pp. 1665–1668 [On pg: 3, 33].

[95] A. Mishra et al. "A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS". In: *Solid State Circuits Conference (A-SSCC), 2012 IEEE Asian.* Nov. 2012, pp. 205–208 [On pg: 3].

[96] C. Leroux et al. "A Semi-Parallel Successive-Cancellation Decoder for Polar Codes". In: *IEEE Transactions on Signal Processing* 61.2, Jan. 2013, pp. 289–299 [On pg: 3].

[97] G. Sarkis and W. J. Gross. "Increasing the Throughput of Polar Decoders". In: *IEEE Communications Letters* 17.4, Apr. 2013, pp. 725–728 [On pg: 3].

[98] P. Giard et al. "Low-Latency Software Polar Decoders". In: *arXiv:1504.00353 [cs.IT]*, Apr. 2015. (submitted to IEEE TSP), pp. 1–16 [On pg: 3].

[99] B. L. Gal, C. Leroux, and C. Jego. "Memory reduction techniques for successive cancellation decoding of polar codes". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* Mar. 2016, pp. 986–990 [On pg: 3].

[100] G. Sarkis et al. "Flexible and Low-Complexity Encoding and Decoding of Systematic Polar Codes". In: *IEEE Transactions on Communications* 64.7, July 2016, pp. 2732–2745 [On pg: 3].

[101] I. Tal and A. Vardy. "How to Construct Polar Codes". In: *IEEE Transactions on Information Theory* 59.10, Oct. 2013, pp. 6562–6582 [On pg: 3, 17, 64, 65, 68, 82, 92, 139].

[102] R. Mori and T. Tanaka. "Performance of Polar Codes with the Construction using Density Evolution". In: *IEEE Communications Letters* 13.7, July 2009, pp. 519–521 [On pg: 3, 65].

[103] R. Mori and T. Tanaka. "Performance and Construction of Polar Codes on Symmetric Binary-Input Memoryless Channels". In: *IEEE International Symposium on Information Theory (ISIT).* 2009, pp. 1496–1500 [On pg: 3, 65, 93].

[104] P. Trifonov. "Efficient Design and Decoding of Polar Codes". In: *IEEE Transactions on Communications* 60.11, Nov. 2012, pp. 1–7 [On pg: 3, 17, 64, 65, 71, 98].

[105] H. Li and J. Yuan. "A practical construction method for Polar Codes in AWGN channels". In: *TENCON Spring Conference.* Sydney, NSW, Apr. 2013, pp. 223–226 [On pg: 3, 17, 45, 64, 65, 67, 71, 98].

[106] D. Wu, Y. Li, and Y. Sun. "Construction and Block Error Rate Analysis of Polar Codes Over AWGN Channel Based on Gaussian Approximation". In: *IEEE Communications Letters* 18.7, July 2014, pp. 1099–1102 [On pg: 3, 64, 65, 68, 71, 93, 98].

[107] S. B. Korada and E. Şaşoğlu. "A Class of Transformations that Polarize Binary-Input Memoryless Channels". In: *IEEE International Symposium on Information Theory (ISIT).* July 2009, pp. 1478–1482 [On pg: 3].

[108] S. B. Korada, E. Şaşoğlu, and R. L. Urbanke. "Polar Codes: Characterization of Exponent, Bounds, and Constructions". In: *IEEE International Symposium on Information Theory (ISIT).* Seoul, July 2009, pp. 1483–1487 [On pg: 3].

[109] Z. Liu et al. "Performance analysis of polar codes based on 3×3 kernel matrix". In: *2015 10th International Conference on Communications and Networking in China (ChinaCom).* Aug. 2015, pp. 382–386 [On pg: 3].

[110] H.-P. Lin, S. Lin, and K. A. S. Abdel-Ghaffar. "Linear and Nonlinear Binary Kernels of Polar Codes of Small Dimensions with Maximum Exponents". In: *IEEE Transactions on Information Theory* 61.10, Oct. 2015, pp. 5253–5270 [On pg: 3].

[111] M. Bakshi, S. Jaggi, and M. Effros. "Concatenated Polar Codes". In: *IEEE International Symposium on Information Theory (ISIT).* Austin, Texas, USA, June 2010 [On pg: 3].

[112] A. Eslami and H. Pishro-Nik. "A Practical Approach to Polar Codes". In: *IEEE International Symposium on Information Theory (ISIT).* St. Petersburg, Aug. 2011, pp. 16–20 [On pg: 3].

[113] P. Trifonov and P. Semenov. "Generalized Concatenated Codes Based on Polar Codes". In: *8th International Symposium on Wireless Communication Systems.* 2011 [On pg: 3, 66].

[114] H. Mahdavifar et al. "On the Construction and Decoding of Concatenated Polar Codes". In: *IEEE International Symposium on Information Theory (ISIT).* Istanbul, July 2013, pp. 952–956 [On pg: 3, 66].

[115] H. Mahdavifar et al. "Performance Limits and Practical Decoding of Interleaved Reed-Solomon Polar Concatenated Codes". In: *IEEE Transactions on Communications* 62.5, May 2014, pp. 1406–1417 [On pg: 3].

[116] D. Wu et al. "Concatenated polar codes based on selective polarization". In: *12th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP).* Dec. 2015, pp. 436–442 [On pg: 3].

[117] S.-M. Zhao, S.-p. Xu, and C. Xing. "Concatenated polar-coded multilevel modulation". In: *10th International Conference on Communications and Networking in China (ChinaCom).* Aug. 2015, pp. 153–157 [On pg: 3].

[118] Q. Zhang et al. "Practical Design and Decoding of Parallel Concatenated Structure for Systematic Polar Codes". In: *IEEE Transactions on Communications* 64.2, Feb. 2016, pp. 456–466 [On pg: 3].

[119] E. Arıkan. "Systematic Polar Coding". In: *IEEE Communications Letters* 15.8, Aug. 2011, pp. 860–862 [On pg: 3, 22, 23, 24].

[120] G. T. Chen et al. "A Low Complexity Encoding Algorithm for Systematic Polar Codes". In: *IEEE Communications Letters* 20.7, July 2016, pp. 1277–1280 [On pg: 3].

[121] H. Vangala, Y. Hong, and E. Viterbo. "Efficient Algorithms for Systematic Polar Encoding". In: *IEEE Communication Letters* 20.1, Jan. 2016, pp. 17–20 [On pg: 3, 21, 139].

[122] D. Wu et al. "Parallel concatenated systematic polar codes". In: *Electronics Letters* 52.1, Jan. 2016, pp. 43–45 [On pg: 3].

[123] P. Trifonov et al. "Fast Encoding of Polar Codes With Reed-Solomon Kernel". In: *IEEE Transactions on Communications* 64.7, July 2016, pp. 2746–2753 [On pg: 3].

[124] H. Huawei. *R1-164039: Polar codes - encoding and decoding.* Online: `http://www.3gpp.org/ftp/tsg_ran/WG1_RL1/TSGR1_85/Docs/`. May 2016 [On pg: 4].

[125] H. Vangala, Y. Hong, and E. Viterbo. *Polar Coding Algorithms in MATLAB.* `www.polarcodes.com` and `www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html`. Nov. 2015 [On pg: 4, 15, 111].

[126] H. Vangala, E. Viterbo, and Y. Hong. “Permuted Successive Cancellation Decoder for Polar Codes”. In: *International Symposium on Information Theory and Applications (ISITA).* Melbourne, Oct. 2014, pp. 438–442 [On pg: 7, 33, 51, 52, 53, 139].

[127] Wikipedia. *Zero-based numbering — Wikipedia, The Free Encyclopedia.* [Online; accessed 17-May-2017]. 2017 [On pg: 8, 24].

[128] J. W. Cooley and J. W. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90, May 1965, pp. 297–301 [On pg: 10].

[129] B. J. Fino and V. R. Algazi. “Unified Matrix Treatment of the Fast Walsh-Hadamard Transform”. In: *IEEE Transactions on Computers* C-25.11, Nov. 1976, pp. 1142–1146 [On pg: 10].

[130] Ç. Çalık and A. Doğanaksoy. “Computing the Weight of a Boolean Function from Its Algebraic Normal Form”. In: *Sequences and Their Applications – SETA 2012: 7th International Conference, Waterloo, ON, Canada, June 4-8, 2012. Proceedings.* Ed. by T. Helleseth and J. Jedwab. Vol. 7280. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 89–100 [On pg: 10].

[131] M. Fürer. “Faster Integer Multiplication”. In: *SIAM J. Comput.* 39.3, Sept. 2009, pp. 979–1005 [On pg: 10].

[132] A. De et al. “Fast Integer Multiplication Using Modular Arithmetic”. In: *SIAM Journal on Computing* 42.2, 2013, pp. 685–699 [On pg: 10].

[133] R. T. Moenck. “Practical Fast Polynomial Multiplication”. In: *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation (SYMSAC).* SYMSAC ’76. Yorktown Heights, New York, USA: ACM, 1976, pp. 136–148 [On pg: 10].

[134] Wikipedia. *LogSumExp — Wikipedia, The Free Encyclopedia.* [Online; accessed 3-September-2015]. 2015 [On pg: 13, 93].

[135] G. Sarkis et al. “Flexible and Low-Complexity Encoding and Decoding of Systematic Polar Codes”. In: *arXiv:1507.03614 [cs.IT]*, 2015, pp. 1–9 [On pg: 22].

[136] N. Presman and S. Litsyn. “Recursive Descriptions of Polar Codes”. In: *arXiv:1209.4818 [cs.IT]*, 2012, pp. 1–60 [On pg: 22].

[137] T. H. Cormen et al. *Introduction to algorithms.* 3rd ed. MIT press and McGraw-Hill, 2009 [On pg: 26, 88].

[138] H. Vangala, Y. Hong, and E. Viterbo. “A New Parallelized Algorithm for List Successive Cancellation Decoding of Polar Codes”. In: 2017. (draft) [On pg: 33].

[139] H. Vangala, E. Viterbo, and Y. Hong. “A New Multiple Folded Successive Cancellation Decoder for Polar Codes”. In: *IEEE Information Theory Workshop (ITW).* Hobart, Australia, Nov. 2014, pp. 381–385 [On pg: 33, 55, 105].

[140] H. Vangala, E. Viterbo, and Y. Hong. “Improved Multiple Folded Successive Cancellation Decoder for Polar Codes”. In: *XXXIth International Union of Radio Science - General Assembly and Scientific Symposium (URSI-GASS).* (invited paper). Beijing, China, Aug. 2014, pp. 1–4 [On pg: 33].

[141] I. Tal and A. Vardy. “List Decoding of Polar Codes”. In: *IEEE Transactions on Information Theory* 61.5, May 2015, pp. 2213–2226 [On pg: 33].

[142] J. Lin and Z. Yan. “Efficient List Decoder Architecture for Polar Codes”. In: *IEEE International Symposium on Circuits and Systems (ISCAS).* Melbourne, June 2014, pp. 1022–1025 [On pg: 52, 54].

[143] J. Lin, C. Xiong, and Z. Yan. “A reduced latency list decoding algorithm for polar codes”. In: *2014 IEEE Workshop on Signal Processing Systems (SiPS).* Oct. 2014, pp. 1–6 [On pg: 52, 54].

[144] J. Lin and Z. Yan. “An Efficient List Decoder Architecture for Polar Codes”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.11, Nov. 2015, pp. 2508–2518 [On pg: 52, 54].

[145] A. Balatsoukas-Stimming et al. “Hardware Architecture for List Successive Cancellation Decoding of Polar Codes”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 61.8, Aug. 2014, pp. 609–613 [On pg: 52, 54, 57].

[146] C. Zhang, X. You, and J. Sha. “Hardware Architecture for List Successive Cancellation Polar Decoder”. In: *IEEE International Symposium on Circuits and Systems (ISCAS).* Melbourne, June 2014, pp. 209–212 [On pg: 52, 54].

[147] M. Mondelli, S. H. Hassani, and R. L. Urbanke. “Scaling Exponent of List Decoders with Applications to Polar Codes”. In: *IEEE Information Theory Workshop (ITW).* Sevilla, Sept. 2013, pp. 1–5 [On pg: 52].

[148] P. Elias. *List decoding for noisy channels.* Technical Report 335. (Reprinted from 1957 IRE WESCON Convention Record, Part 2). MIT, CA: Research Laboratory of Electronics, Sept. 1957 [On pg: 56].

[149] H. Vangala, Y. Hong, and E. Viterbo. “A Comparitive Study of Polar Code Constructions for the AWGN channel”. In: 2017. (draft). Available: http://arxiv.org/abs/1501.02473 [On pg: 63, 91].

[150] H. Vangala, E. Viterbo, and Y. Hong. “Quantization of Binary Input DMC at Optimal Mutual Information Using Constrained Shortest Path Problem”. In: *International Conference on Telecommunications (ICT), 22nd.* Sydney, Australia, Apr. 2015, pp. 151–155 [On pg: 63].

[151] H. Vangala, Y. Hong, and E. Viterbo. “Elements of Polar Code Construction and Accurate Block Error Rate Estimation”. In: 2017. (draft) [On pg: 63, 91].

[152] S. H. Hassani and R. L. Urbanke. “Universal Polar Codes”. In: *IEEE International Symposium on Information Theory (ISIT).* Honolulu, HI, June 2014, pp. 1451–1455 [On pg: 64, 66].

[153] M. Alsan. “Universal Polar Decoding with Channel Knowledge at the Encoder”. In: *IEEE Information Theory Workshop (ITW).* Hobart, Australia, Nov. 2014, pp. 371–375 [On pg: 64].

[154] M. Alsan. “Re-proving Channel Polarization Theorems: An Extremality and Robustness Analysis”. PhD dissertation. Switzerland: Ecole Polytechnique Federale de Lausanne, Jan. 2015 [On pg: 64].

[155] D. Kern, S. Vorkoper, and V. Kuhn. “A New Code Construction for Polar Codes Using Min-Sum Density”. In: *International Symposium on Turbo Codes and Iterative Information Processing (ISTC), 8th.* Bremen, Germany, Aug. 2014, pp. 228–232 [On pg: 64, 65].

[156] G. Bonik, S. Goreinov, and N. Zamarashkin. “Construction and analysis of polar and concatenated polar codes: practical approach”. In: *arXiv:1207.4343 [cs.IT]*, 2012, pp. 1–17 [On pg: 64, 65].

[157] S. Zhao, P. Shi, and B. Wang. “Designs of Bhattacharyya Parameter in the Construction of Polar Codes”. In: *Wireless Communications, Networking and Mobile Computing (WiCOM), 7th international conference on.* Wuhan, Sept. 2011, pp. 1–4 [On pg: 64, 65].

[158] N. Goela, E. Abbe, and M. Gastpar. “Polar Codes for Broadcast Channels”. In: *IEEE International Symposium on Information Theory (ISIT).* Istanbul, July 2013, pp. 1127–1131 [On pg: 66].

[159] A. Bravo-Santos. “Polar Codes for the Rayleigh Fading Channel”. In: *IEEE Communications Letters* 17.12, Dec. 2013, pp. 2352–2355 [On pg: 66, 114].

[160] K. Chen, K. Niu, and J.-R. Lin. “Practical polar code construction over parallel channels”. In: *IET Communications* 7.7, May 2013, pp. 620–627 [On pg: 66].

[161] S. Cayci, O. Arıkan, and E. Arıkan. “Polar code construction for non-binary source alphabets”. In: *Signal Processing and Communications Applications Conference (SIU).* (in Turkish). Mugla, Apr. 2012, pp. 1–4 [On pg: 66].

[162] L. Zhang, Z. Zhang, and X. Wang. “Polar Code with Block-length $N = 3^n$”. In: *Wireless Communications & Signal Processing (WCSP), International Conference on.* Huangshan, Oct. 2012, pp. 1–6 [On pg: 66].

[163] V. Miloslavskaya and P. Trifonov. “Design of binary polar codes with arbitrary kernel”. In: *IEEE Information Theory Workshop.* Lausanne, Sept. 2012, pp. 119–123 [On pg: 66].

[164] B. Serbetci and A. E. Pusane. “Practical polar code construction using generalised generator matrices”. In: *IET Communications* 8.4, Mar. 2014, pp. 419–426 [On pg: 66].

[165] S.-Y. Chung, R. T. J., and R. Urbanke. “Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation”. In: *IEEE Transactions on Information Theory* 47.2, Feb. 2001, pp. 657–670 [On pg: 71].

[166] R. W. Floyd and R. L. Rivest. “Algorithm 489: The Algorithm SELECT – for finding the ith Smallest of n Elements[M1]”. In: *Communications of the ACM* 18.3, Mar. 1975, p. 173 [On pg: 77].

[167] K. C. Kiwiel. “On Floyd and Rivest’s SELECT algorithm”. In: *Theoretical Computer Science* 347.1–2, 2005, pp. 214–238 [On pg: 77].

[168] B. M. Kurkoski and H. Yagi. “Quantization of Binary-Input Discrete Memoryless Channels”. In: *IEEE Transactions on Information Theory* 60.8, Aug. 2014, pp. 4544–4552 [On pg: 83, 84, 85, 87, 89, 90].

[169] J. M. Wozencraft and R. S. Kennedy. “Modulation and Demodulation for Probabilistic Coding”. In: *IEEE Transactions on Information Theory* 12.3, July 1966, pp. 291–297 [On pg: 83].

[170] J. L. Massey. “Coding and modulation in digital communications”. In: *Proceedings International Zurich Seminar on Digital Communication.* 1974, E2(1)–E2(4) [On pg: 83].

[171] L.-N. Lee. “On Optimal Soft-Decision Demodulation”. In: *IEEE Transactions on Information Theory* 22.4, July 1976, pp. 437–444 [On pg: 83].

[172] T. Koch and A. Lapidoth. “At Low SNR, Asymmetric Quantizers are Better”. In: *IEEE Transactions on Information Theory* 59.9, Sept. 2013, pp. 5421–5445 [On pg: 83].

[173] A. Winkelbauer, G. Matz, and A. Burg. “Channel-Optimized Vector Quantization with Mutual Information as Fidelity Criterion”. In: *Asilomar Conference on Signals, Systems and Computers.* Pacific Grove, CA, Nov. 2013, pp. 851–855 [On pg: 83].

[174] R. Mathar and M. Dorpinghaus. “Threshold Optimization for Capacity-Achieving Discrete Input One-Bit Output Quantization”. In: *IEEE International Symposium on Information Theory (ISIT).* Istanbul, July 2013, pp. 1999–2003 [On pg: 83].

[175] B. M. Kurkoski and H. Yagi. “Concatenation of a Discrete Memoryless Channel and a Quantizer”. In: *IEEE Information Theory Workshop (ITW).* Cairo, Jan. 2010, pp. 1–5 [On pg: 83].

[176] K. Iwata and S. Ozawa. “Quantizer design for outputs of binary-input discrete memoryless channels using SMAWK algorithm”. In: *IEEE International Symposium on Information Theory (ISIT).* Honolulu, HI, July 2014, pp. 191–195 [On pg: 83, 85].

[177] Y. Sakai and K. Iwata. “Suboptimal quantizer design for outputs of discrete memoryless channels with a finite-input alphabet”. In: *International Symposium on Information Theory and Its Applications (ISITA).* Melbourne, Oct. 2014, pp. 120–124 [On pg: 83, 85].

[178] R. Guerin and A. Orda. “Computing Shortest Paths for Any Number of Hops”. In: *IEEE/ACM Transactions on Networking* 10.5, Oct. 2002, pp. 613–620 [On pg: 86, 88].

[179] G. Cheng and N. Ansari. “Finding All Hops Shortest Paths”. In: *IEEE Communications Letters* 8.2, Feb. 2004, pp. 122–124 [On pg: 86].

[180] R. Bellman. “On a routing problem”. In: *Quarterly of Applied Mathematics* 16, 1958, pp. 87–90 [On pg: 88].

[181] J. Y. Yen. “An algorithm for finding shortest routes from all source nodes to a given destination in general networks”. In: *Quarterly of Applied Mathematics* 27.1.1, 1969/70, pp. 536–530 [On pg: 88, 89].

[182] J. Y. Yen. *Shortest Path Network Problems.* Series: Mathematical Systems in Economics;18, Publishers: Verlag Anton Hain - Meisenheim Am Glan, Jan. 1975, p. 169 [On pg: 88].

[183] M. J. Bannister and D. Eppstein. “Randomized Speedup of the Bellman-Ford Algorithm”. In: *Meeting on Analytic Algorithmics and Combinatorics (ANALCO12).* The Westin Miyako, Kyoto, Japan, Jan. 2012, pp. 41–47 [On pg: 88, 89].

[184] S. Boyd and L. Vandenberghe. *Convex Optimization.* New York, NY, USA: Cambridge University Press, 2004 [On pg: 93].

[185] Y. She. *A Hardware Implementation of Folded Successive Cancellation Decoder for Polar Codes.* Tech. rep. (supervision of full final year project). Southeast university, China and Monash university, Australia, June 2015 [On pg: 105].

[186] Y. She et al. “An efficient successive cancellation polar decoder based on novel folding approaches”. In: *Conference of Intel Collaborative Research Institutes on Mobile Networking and Computing (ICRI-MNC).* (poster). Beijing, July 2015 [On pg: 105].

[187] S. Liu, Y. Hong, and E. Viterbo. “Polar Codes for Block Fading Channels”. In: *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW).* Mar. 2017, pp. 1–6 [On pg: 114].

[188] P. Trifonov. “Design of polar codes for Rayleigh fading channel”. In: *2015 International Symposium on Wireless Communication Systems (ISWCS).* Aug. 2015, pp. 331–335 [On pg: 114].

[189] R. Deng, L. Li, and Y. Hu. “On the Polar Code Encoding in Fading Channels”. In: *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall).* Sept. 2016, pp. 1–5 [On pg: 114].

[190] M. K. Islam and R. Liu. “Polar Coding for Fading Channel”. In: *International Conference on Information Science and Technology (ICIST), 3rd.* Yangzhou, Mar. 2013, pp. 1096–1098 [On pg: 114].

[191] H. Si, O. O. Koyluoglu, and S. Vishwanath. “Polar Coding for Fading Channels”. In: *IEEE Information Theory Workshop (ITW).* Sevilla, Spain, Sept. 2013, pp. 1–5 [On pg: 114].

[192] J. J. Boutros and E. Biglieri. “Polarization of Quasi-Static Fading Channels”. In: *IEEE International Symposium on Information Theory (ISIT).* Istanbul, July 2013, pp. 769–773 [On pg: 114].

[193] C. Zhang, B. Yuan, and K. K. Parhi. “Reduced-Latency SC Polar Decoder Architectures”. In: *International Conference on Communications (ICC).* Ottawa, ON, June 2012, pp. 3471–3475 [On pg: 139].

[194] H. Vangala, E. Viterbo, and Y. Hong. “A Comparative Study of Polar Code Constructions for the AWGN Channel”. In: *arXiv:1501.02473 [cs.IT]*, 2015. (draft) [On pg: 139, 140].

Appendix

A.1 Introduction

This is the full documentation of a MATLAB package dedicated to help simulating the polar codes in various channel models such as a BSC, BEC, and AWGN.

The package focuses to provide the most fundamental blocks related to polar codes, to aid researchers start working with polar codes right away. The package is expected to save a significant amount of valuable time required to develop modules related to polar codes such as encoding, decoding, code-construction, and performance analysis such as plotting FER/BER curves.

More importantly, the package provides an implementation of greater efficiency to the users which is tested, benchmarked, and improved over a long period of time. As we find, the only way to see any significant improvement in the speed of the routines in this package is to rewrite them in a lower level programming language such as C/C++, with a similar logic.

Due care has been taken to provide the users a code with detailed comments and help written all over so that they can easily build over the modules. During the writing of this code, a significant importance has been given to be concise and user-friendly. The result is a very small, portable package of fully readable matlab files that you can easily download from the below link.

`www.polarcodes.com`

A.2 A Quick Reference

A.2.1 Installation

The MATLAB package is openly available at below link:

`www.polarcodes.com`

It is the enhanced online resource of our earlier site at: `http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html`, hosting the code along side of the other tutorial videos and many other useful resources related to polar codes.

The installation of the package is achieved by probably the simplest way possible: Just copy the few source files (extracted from the zip file) to a your local directory.

More preferrably, adding the location of the extracted files to the matlab's *path* (File → Set Path → Add folder) will allow the user to use the modules from any working directory.

A.2.2 Quick basics

Here is a matlab package, that you can start using right away to work with polar codes, by simply copying them on to your hard drive (see Appendix A.2.1).

More specifically, the packages allows you the following fundamental operations.

1. Choice of the three popular channels: BEC, BSC, and AWGN
2. Polar encoding
3. Polar decoding
4. Polar code construction
5. Plot the performance curves

Finally, it is made sure such that all the above will only empower the users to upgrade the package and build more advanced simulation environments for their research with polar codes.

For a quick illustration of what this package is capable of doing, we will simply show some pictures, tabular forms, and cheat sheets in this chapter. An impatient user should be able to start right away by simply reading a couple of diagrams in this chapter.

In the next chapters, we will provide a traditional and complete documentation of the package, which will unleash the full features of the package for a more advanced use.

A.2.3 Illustrations and Tables

This section contains several illustrations to help understand the architecture of the package quickly and start using it immediately. In the next chapter we will provide a detailed and complete documentation.

The Fig. A.1 describes a mind-map of what constitutes this package. It also describes the functional components of polar codes. The terms 'universal' and 'non-universal' refer to the dependence of code structure on the channel conditions. If the code structure is changed everytime the channel conditions change (by running the code construction again), then the code is called a non-universal code. If the code is not changed with channel conditions, then it is called a universal code.

The Fig. A.2 describes a division of the package into two categories: the basic routines and the derived routines. It is a very convenient division to understand the package.

Finally, the Tables A.1 and A.2 quickly describe all the main routines and their arguments that a user should be interested in practice. A more detailed and complete documentation of the same along side of many more utilities will be given in the next chapter.

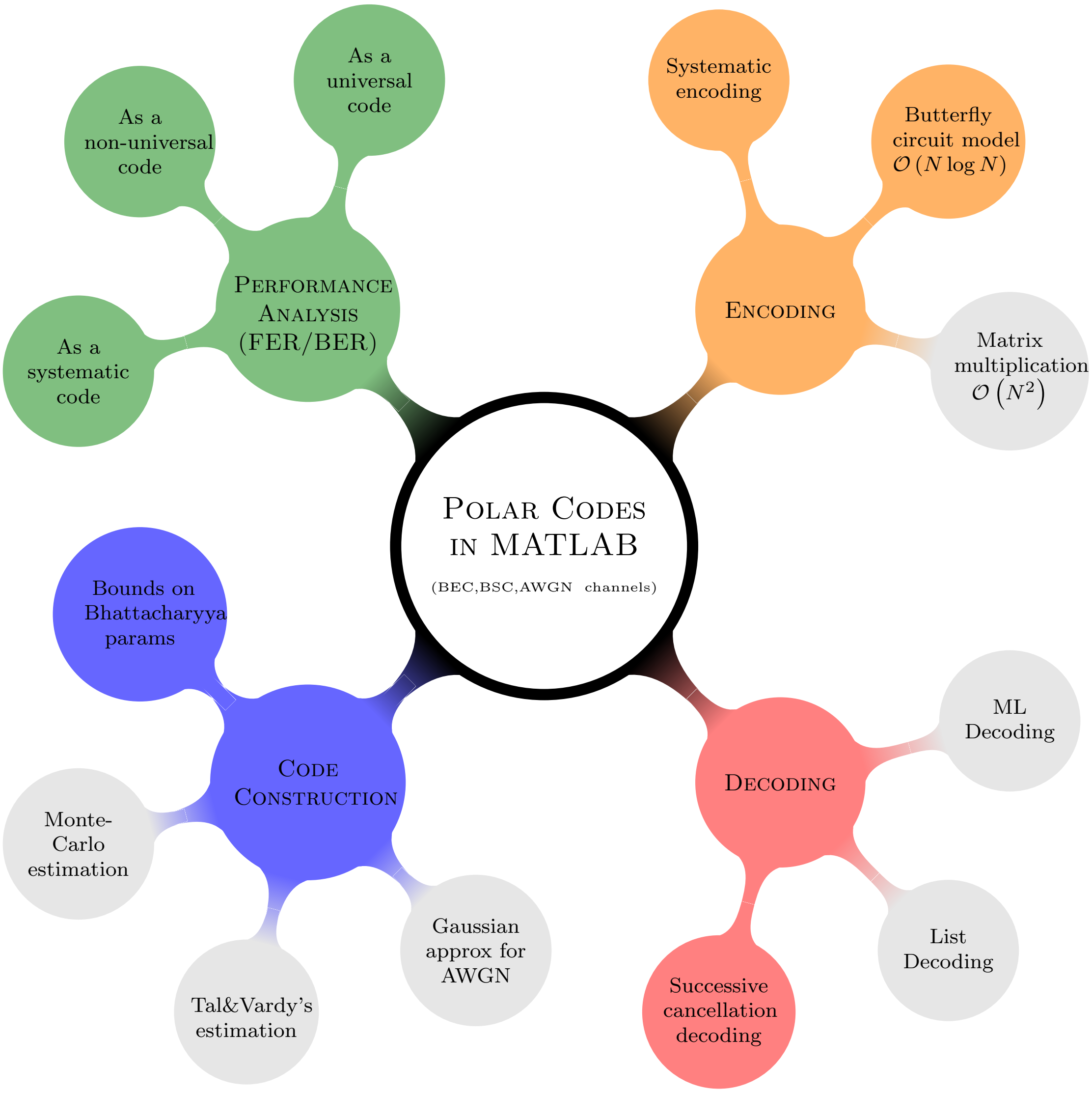
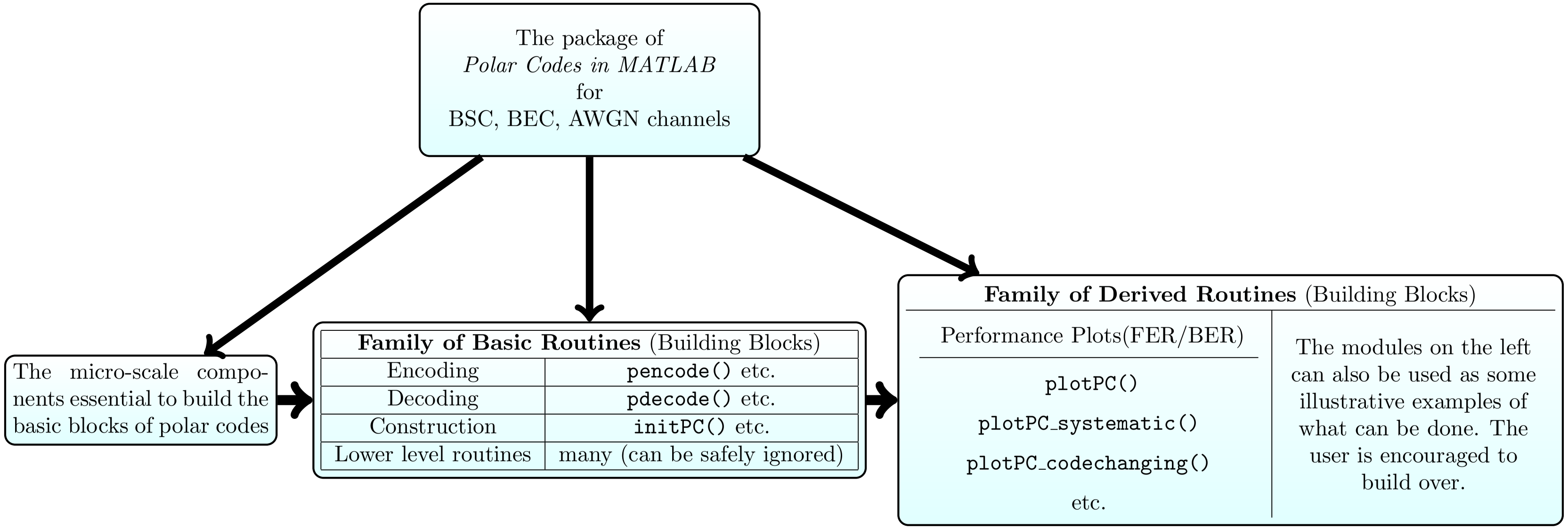


Figure A.1: The overall conceptual structure of the MATLAB package (parts in gray are unavailable, may be available in future versions)


The package of
Polar Codes in MATLAB
for
BSC, BEC, AWGN channels
The micro-scale components essential to build the basic blocks of polar codes
Family of Derived Routines (Building Blocks)
Performance Plots(FER/BER)
`plotPC()`
`plotPC_systematic()`
`plotPC_codechanging()`
etc.
The modules on the left can also be used as some illustrative examples of what can be done. The user is encouraged to build over.

Family of Basic Routines (Building Blocks)	
Encoding	`pencode()` etc.
Decoding	`pdecode()` etc.
Construction	`initPC()` etc.
Lower level routines	many (can be safely ignored)

Figure A.2: Operational division of the package contents

Table of Functions

(Note: Arguments in *underscored italics* are optional)

Family of Basic Utilities
initPC(N,K,channelname,channelstate,*silentflag*,*frozenbits*)
x = pencode(u,*myfrozenlookup*)
u = pdecode(y,channelname,channelstate,*myfrozenlookup*)
x = systematic_pencode(u,*myfrozenlookup*,*algorithmname*)
u = systematic_pdecode(y,channelname,channelstate,*myfrozenlookup*)
y = OutputOfChannel(x,channelname,channelstate)
myfrozenlookup = build_a_lookup(frozenbits,frozenbitindices)

Family of Plotting Utilitites
plotPC(N,K,channelname,designstate,channelrange, *MaxIters*)
plotPC_codechanging(N,K,channelname,channelrange, *MaxIters*)
plotPC_systematic(N,K,channelname,designstate,channelrange, *MaxIters*)
plotPC_systematic_codechanging(N,K,channelname,channelrange, *MaxIters*)

Table A.1: The core utilities of the MATLAB package

A.3 The More Detailed Reference

As Fig. A.2 describes, the contents of our package *Polar Codes in MATLAB* fall into three categories as below. This division follows a top-down approach, in the same respective order.

1. Plotting utilities (derivatives)
2. Encoding/Decoding functions (basic routines)
3. Core functions (the actual engines used by basic routines)

Argument Glossary

N — The blocklength, always a power of 2 (if not, it will be adjusted by `ceil`’ing to the immediate power-of-2)

K — The message length, a positive integer $\leq N$ ($\implies$ the rate of the code $R = K/N$)

`algorithmname` – The name of the systematic-polar-encoding algo: one of ’A’ or ’B’ or ’C’ (in-quotes)

`channelname` – One of the strings (single-quotes) `'BSC'` or `'BEC'` or `'AWGN'` with the respective channel-state parameters p or ϵ or E_b/N_0(in dB) respectively. (`'AWGN'` assumes $\sigma^2 = \frac{N_0}{2}$ and BPSK signalling).

`channelrange` – A vector of channel-state values

`designstate` – The single value of the channel-state to be used during the polar code construction

`frozenbits` – A $(N-K) \times 1$ size binary vector of $N-K$ frozen bits (usually are all zeros by default)

`frozenbitindices` – A $(N-K) \times 1$ size integer vector of $N-K$ unique indices from $\{1, 2, \ldots, N\}$

`MaxIters` – (Always optional) Maximum number of iterations to be used with Monte-Carlo estimation (default: 10000). More its value, more the accuracy and wider the range of parameters it supports

`myfrozenlookup` – (Always optional) A $N \times 1$ size lookup vector, overriding the internal one. It has only the values $-1, 0, 1$, representing whether the corresponding element index is an information-bit (-1) or a frozen-bit (0 or 1, to which it is frozen to). One should typically use `build_a_lookup()`, to build a lookup vector of userdefined values. It is a very critical parameter for polar codes.

`silentflag` – (Always optional) A binary value (default: 0), requesting whether to run silently without printing any o/p

Table A.2: Quick descriptions of all the arguments the earlier table

Due to a significant overlap in the arguments of various functions, we would like to save ourselves from a heavy repeatation. We therefore request the readers to refer back to the Table A.2, in order to understand various arguments that are missing in the descriptions over here.

Also, users are encouraged to use the syntax `help function-name` within MATLAB environment to understand various elements of the package, where `function-name` can be the name of any MATLAB script (`.m` files) that you notice in the package. It should be noted that the descriptions in this document essentially follow the same `help` descriptions, while improving their presentation for a better readability.

A.3.1 Channel Selection

One of the important features of the package is to give users a flexibility among all the three popular channels of:

1. A *binary symmetric channel* (BSC) with transition probability p
2. A *binary erasure channel* (BEC) with erasure probability ϵ
3. An *additive white Gaussian noise* (AWGN) channel with signal-to-noise-ratio (SNR) in decibels defined as $10\log_{10}\left(\frac{E_b}{N_0}\right)$, which implicitly assumes two components of a BPSK modulation of $\{0,1\} \to \{\pm\sqrt{R \cdot E_b}\}$ (R is the rate of the code $\triangleq \frac{K}{N}$) and a variance of $\frac{N_0}{2}$ for the additive noise.

When required to simulate such a channel, the user is encouraged to make use of '`OutputOfChannel()`', in order avoid any mismatch.

Typically, such a channel is specified to various functions simply as a string and a real value pair: `channelname` and `channelstate`, which take the following special cases:

1. `'BSC'` and p, $0 \leq p \leq 0.5$
2. `'BEC'` and ϵ, $0 \leq \epsilon \leq 1$
3. `'AWGN'` and $SNRdB$, $SNRdB \triangleq 10\log_{10}\left(\frac{E_b}{N_0}\right) \in (-\infty, +\infty)$

We will now proceed to the specific details of all possible functions in the earlier three categories.

A.3.2 The Plotting Utilities

These utilities are the examples of various experiments that a user is typically interested to perform, while working with polar codes. These functions essentially estimate the *frame error rate* (FER) and the *bit error rate* (BER) of polar codes in various parametric cases.

The common functionality of all the below functions is to produce the following results, while preserving the current settings of the polar coding set by an earlier run of `initPC()`.

i.	`plotPC()`
ii.	`plotPC_codechanging()`
iii.	`plotPC_systematic()`
iv.	`plotPC_systematic_codechanging()`

1. Live-in status of the running simulation (which is typically longer than a few seconds) printed in a user-friendly output format (in steps of 100 samples).
2. At the end, it outputs a figure window, containing two plots:
   - FER vs. Channel state
   - BER vs. Channel state
3. Creates two global variables `FER` and `BER` in the current workspace, containing the main results.

The core technique of all the functions below is to simply simulate a large number ($\leq$ `MaxIters`) of sample encoding-decoding operations under individual noisy channel conditions, in order to Monte-Carlo estimate the FER and the BER.

Supplying an explicit integer value of `MaxIters` is optional, whenever the default value of `MaxIters`= 10000 is sufficient. But, an explicit larger value must be supplied whenever the simulations produce a zero FER/BER at an essential channel-state value.

A.3.2.1 i. `plotPC(` N,K`,channelname,designstate,channelrange,` *`MaxIters`*`)` :

Obtains the performance curves **FER/BER** vs. **channelstate** of a (N,K) polar code when **channelstate** takes values in `channelrange`.

The design/construction of the polar code is performed under `channelname` channel at a channel-state value `designstate`, with blocklength of N bits, encoding K message bits at once.

e.g. `plotPC(1024,512,'BEC',0.1,0.05:0.05:0.3,20000)`

A.3.2.2 ii. `plotPC_codechanging(` N,K`,channelname,channelrange,`*`MaxIters`*`)` :

Obtains the performance curves **FER/BER** vs. **channelstate** in a similar way to `plotPC()`, but the code construction is repeated at every value of **channelstate** in `channelrange`. This way, the polar code changes at every time channel changes and such a code is called a non-universal code. (Note: polar codes are non-universal by definition)

Note the absense of the argument `designstate` as in `plotPC()`, denoting the use of a unique code designed by running the code-construction at `designstate` and keeping the code unchanged for all channel conditions.

e.g. `plotPC_codechanging(1024,512,'BEC',0.05:0.05:0.3,20000)`

A.3.2.3 iii and iv.

`plotPC_systematic(` N,K`,channelname,designstate,channelrange,` *`MaxIters`*`)` & `plotPC_systematic_codechanging(` N,K`,channelname,channelrange,` *`MaxIters`* `)` :

Exactly same as the earlier two functions except for the use of "systematic polar codes" instead of the original polar codes by Arıkan. Such a variant should exhibit the same FER as the original but is known to have a better BER under similar channel conditions.

A.3.3 Encoding/Decoding/Code-Construction Functions (Basic Modules)

i.	`initPC()`
ii.	`pencode()`
iii.	`pdecode()`
iv.	`systematic_pencode()`
v.	`systematic_pdecode()`

A common feature of all the last four functions is an optional argument of `myfrozenlookup` vector. It is an $N \times 1$ vector conveying a lot of information at once, about the frozen bit indices, and information bit indices within the indices of $\{1, 2, \ldots, N\}$, and also the frozen bits used at the frozen bit indices, as follows:

$$\texttt{myfrozenlookup}(i) = \begin{cases} 0, \text{ if } i^{th} \text{ bit is frozen to } 0 \\ 1, \text{ if } i^{th} \text{ bit is frozen to } 1 \\ -1, \text{ if } i^{th} \text{ bit is information} \end{cases}$$

Such a vector is the output of a polar code construction algorithm, and is already available to the function by a prior run of `initPC()` (is a must before running any function in this section).

By supplying an alternative `myfrozenlookup`, user will be able to bypass the default settings and analyze the consequences. This is critical for example in cryptography, where the frozen-bits and their indices can carry a *secret key*, which might be partially known/unknown to an evesdropper.

The bonus functions at the end of this section will the users help generating the lookup vector `myfrozenlookup`.

A.3.3.1 i. `initPC(N,K,channelname,designstate,`*`silentflag`*`,`*`frozenbits`*`)`

This stores all the information about the polar codes in a global structure named "`PCparams`", that we want to simulate. It runs a full run of the polar-code-construction algorithm and chooses the frozen bits indices and sets those frozen bits to `frozenbits` (optional with default: all zeros).

All this information is critical for the rest of the functions. Further, the function also pre-allocates memory for performing various decoding operations. Therefore, one must have run this function with appropriate initial values, before using any other function in this section.

> **TIP:** One can manually view/change the contents of `PCparams` in order to manipulate the behavior of the encoder and decoder blocks. This is very useful in adanced simulations related to polar codes. For example in cryptography, `frozenbits` and their indices are changed very arbitrarily. In that case, one can run `initPC()`, manipulate the resulting structure `PCparams` (especially the element `FZlookup`), and run the subsequent `pencode()` and `pdecode()` functions (similar to `plotPC()`) to investigate the change in performance. The whole set of changes can then be conveniently written, for example, as a new variant of `plotPC_xxxx()` function.

A.3.3.2 ii. `x=pencode(u,`*`myfrozenlookup`*`)`

Encodes the K message bits in `u`, into N bits in `x`. `myfrozenlookup` is an optional argument, used to bypass the default frozenbits/indices setting.

A.3.3.3 iii. `u=pdecode(y,channelname,channelstate,`*`myfrozenlookup`*`)`

Decodes original `u` from a received $N \times 1$ (noisy) vector y, assuming the underlying channel `channelname` and its channel-state parameter value `channelstate`. Optionally, it can use a user-defined frozenbit information embedded in `myfrozenlookup`, which is otherwise known to the receiver during the run of `initPC()` itself.

A.3.3.4 iv. and v.
`x=systematic_pencode(u,`*`myfrozenlookup`*`)` &
`u=systematic_pdecode(y,channelname,channelstate,`*`myfrozenlookup`*`)`

Exactly same as the earlier two functions except for the use of "systematic polar codes" instead of original polar codes by Arıkan. Such a variant should exhibit the same FER as the original but is known to have a better BER under similar channel conditions.

A.3.3.5 A Couple of Additional Functions:

1. To make sure we have no mis-match in explicitly simulating various channels such as BSC,BEC, and AWGN, we provide the following function:

```
y = OutputOfChannel(x,channelname,channelstate)
```

i.	`OutputOfChannel()`
ii.	`build_a_lookup()`

It simply passes `x` via an instantiation of the noisy channel named `channelname` under its channel-state parameter value equal to `channelstate`, and returns the noisy output of the channel. (involving a few bit-flips of a BSC, or a few erasures of a BEC, or a noise-adding AWGN)

2. When required, one usually suggests the more intuitive `frozenbits` and their locations (`frozenbitindices`). But all our functions take a lookup vector (`myfrozenlookup`) that has this information embedded. Forming such a vector requires a bit of code. The following function bridges this gap, by helping to generate the vector quickly.

```
myfrozenlookup = build_a_lookup(frozenbits,frozenbitindices)
```

A.3.4 Core Functions Used by the Basic Encoding/Decoding Functions

The below are smaller MATLAB functions which are used within the basic blocks discussed in the earlier functions. In general, a user may not be interested in explicitly using these functions, but will be of a great utility for some researchers who need to work with finer details. We will give a concise description of all these routines in Table A.3.

A.4 A Discussion on the Algorithms Used

We have provided a video tutorial at the below link:

`www.polarcodes.com`

The videos explain the basic concepts of polar codes to a beginner in a detailed fashion within an hour. We will refer to them and skip the most of the algorithmic discussion.

This section will provide an additional information related to the algorithms that we used in implementing these different modules.

A.4.1 The Encoding

A.4.1.1 The Classic Encoding

The original polar codes are non-systematic, and being a linear code, the encoding simply needs a matrix multiplication. Due to the larger size of the code, matrix multiplication becomes computationally expensive ($\mathcal{O}(N^2)$). An alternative implementation (similar to FFT's butterfly circuit implementation) comes to the rescue with a significantly reduced computational-complexity of $\mathcal{O}(N \log N)$. This is available in Arıkan's seminal paper [11].

Function Name	**Arguments & A Short Description**
`j = bitreversed(i)`	Using the N from `initPC()`, it returns decimal equivalent `j` of the reversed the binary $n = log_2(N)$ bit representation of the index `i`, by looking-up a table of N integers $\{0, 1, \ldots, N-1\}$
`j = bitreversed_slow(i)`	An exactly similar function as above, but uses no lookup tables (and hence is slower). This is used to build the lookup table during `initPC()`.

`EncoderA()`
`EncoderB()`
`EncoderC()`
	These are the various systematic-polar-encoding algorithms optionally used by `systematic_pencode(u)`. The arguments are slightly involved. Interested readers can read the `help` descriptions of these functions.

`FN_transform(`$\mathbf{d}$`)`	Returns the $N \times 1$ vector resulted as a product of an $N \times 1$ binary vector $\mathbf{d}$ with the n-fold kronecker product $\mathbf{F}^{\otimes n}$ of $\mathbf{F} \triangleq \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ in binary field

`c = logdomain_diff(a,b)`
`c = logdomain_sum(a,b)`
	Adding or subtracting a given two real numbers in log-domain, which is essential for avoiding overflow/underflow problems with likelihood ratios.

`lowerconv(upperbit,`l_1`,`l_2`)`
`lowerconv_BEC0()`
`lowerconv_BEC1()`
	The second likelihood-transformation of Arıkan (see [11, Eq.(76)]) on two LLRs, which also uses decisions from the earlier decoding steps. In a BEC channel, these operations are optimized with a couple of very small 3×3 lookup-table. Calling the appropriate function is taken care of by `pdecode()` etc functions.

`frozenlookup = pcc(N,K,channelstring,channelstate,frozenbits)` — An independent polar code construction function, returning a lookup-vector holding everything necessary for defining a polar code	
`pdecode_BEC()`	Optimized version of `pdecode_LLRs()` for a BEC. It is implicitly called by `pdecode()` etc.
`pdecode_LLRs()`	The core utility called after computing LLRs of received symbols `y` within `pdecode()` etc., in a BSC or AWGN channel
`updateBITS(latestbit,i)`	Broadcast and update various bits after the knowledge of the latest bit decision `latestbit` at index `i`
`updateLLR(i)`	Compute the `i`-th bit-likelihood by using a computational-tree of different likelihood transformation operations applied on the N received likelihoods
`updateLLR_BEC()`	Optimized equivalent of the above function in a BEC setting

`upperconv(`l_1`,`l_2`)`
`upperconv_BEC()`
	The first likelihood-transformation of Arıkan (see [11, Eq.(75)]) on two LLRs. In a BEC channel, these operations are optimized with a couple of very small 3×3 lookup-table. Calling the appropriate function is taken care of by `pdecode()` etc functions.

Table A.3: Table of micro-scale functions constituting the basic modules in Appendix A.3.3

Our package provides only the efficient implementation, and the other implementation is usually not interesting in any case.

A.4.1.2 The Systematic Encoding

On the other hand, systematic variants of polar encoding do exist, but their usual formulation takes the form of solving a bunch of linear equations. Fortunately, efficient solutions are available in [121].

In our package, we provide the implementation of all the three algorithms from [121], and when not stated explicitly, our interface considers the least complexity algorithm by default (uses a bit more memory than other two algorithms, but is always manageable).

A.4.2 The Decoding

Our package provides an implementation of the basic successive cancellation decoder, which we believe to be the *most efficient* implementation possible as of today, and is also distributed freely for all educational and research purposes. It is an all-MATLAB implementation, and as we are aware of, the only way to improve its efficiency is via reimplementing it in a lower-level programming language such as C++, with exactly same architecture. (We noticed more than 100x improvement in its run-time)

Note that, the celebrated decoder for polar codes is SCD, even though it exhibits a relatively poorer performance compared to state-of-the-art codes. Being fundamental for all the later advanced decoders that exhibit superior performance, one cannot avoid having an SCD.

A quick understanding of how to implement an SCD in MATLAB is possible by reading the sample pseudocode in [126].

Even though we started off with the above implementation, we discovered more powerful implementations both in terms of speed and memory efficiency. The ideas stem from the discussions in [53, 101, 193], encouraging us to use a tree architecture for an SCD instead of a rectangular memory architecture. Even though these tree-based implementation ideas are slightly involved and complicated to implement compared to Arikan's original exposition in [11], it got significantly simplified and now it became a very natural implementation of an SCD.

Our package implements precisely the most efficient tree-based software architecture of a successive cancellation decoding. All those interested can take time to read the script files with abundant comments to easily understand the underlying concepts.

A.4.3 The Code Construction

There are a number of PCC algorithms, one can read [194] to get a glimpse of the major construction algorithms.

One critical conclusion of [194] is that at all practical blocklengths ($N \leq 2^{16}$), the simplest of all constructions is as good as the highly complex PCC algorithm. Taking the lead of this interesting conclusion, we provide only the simplest construction and nothing more. It also avoids an additional ambiguity in the choice of a construction algorithm.

However, one issue that every user should be aware of is the non-universality part of a PCC, where any PCC algorithm assumes a specific channel condition to run the algorithm (say a *design condition*). To obtain the best code performance for the same code at a range of channel conditions, one should try to optimize over several design-conditions, as illustrated in [194], requiring to simulate FER vs. channel-state curves for each code obtained at each design-condition.

A.4.4 The Use of Log-domain calculations

The SCD involves calculations using all likelihood ratio (LR) values (or Bhattacharyya parameter or probability of error values in the construction phase). The LRs or Bhattacharyya parameters are usually stored directly in double-precision floating-point variables. However, it is well-known to cause an underflow or an overflow, after several likelihood or Bhattacharyya parameter transformation operations. In otherwords, the available computer variables are either in-capable or in-efficient of storing the exact LR/Bhattacharyya parameter values and hence the simulations fail.

A popular solution to this problem is to store LLRs instead of likelihood ratios, i.e., a log-domain representation of all the LRs or the Bhattacharyya parameters. Throughout the package, we will perform our real valued calculations in log-domain. This completely avoids the problem of an underflow/overflow.

Throughout the package we prefer to use natural (or base e) logarithms, except during the dB calculations of SNR (E_b/N_0).