Fast and Simplified Successive-Cancellation Based Decoders for Polar Codes

by

Maryam Haghighi Ardakani

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Communications

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Polar codes are the first class of error correction codes with explicit construction that provably achieve the channel capacity. In addition, polar codes enjoy low-complexity encoding and low hardware and computational complexities under successive-cancellation (SC) decoding [1]. Therefore, they have been selected to be used for the control channel in the fifth generation of mobile communication standards (5G) [2]. Although polar codes are asymptotically capacity-achieving under SC decoding, the SC decoding fails to provide reasonable error correction performance for short to moderate code length. This limitation is due to the performance gap between the SC decoder and the maximum-likelihood (ML) decoder. Successive-cancellation list (SCL) and successive-cancellation flip (SCF) decoding are proposed to improve the performance of polar codes. However, their serial decoding nature results in significant decoding latency. To reduce this latency some operations can be done in parallel. Specifically, special nodes are identified in the decoding tree of polar codes that can be decoded without serially traversing the decoding tree.

In this thesis, we present fast implementations of the SCL and SCF decoders. In particular, we propose fast parallel list decoders for five newly-identified nodes in the decoding tree of a polar code, which significantly reduces the decoding latency. We also present novel fast SCF decoders that decode some special nodes in the decoding tree of a polar code without serially computing bit log-likelihood ratios. Our proposed fast decoders significantly reduce the decoding latency without sacrificing the error-rate performance.

Another limitation of polar codes is lack of length flexibility. Polar codes are constructed by the recursive Kronecker product of a size-2 polarizing matrix, which limits their code length to integer powers of two. Incorporating larger size polarizing matrices in conjunction with the size-2 kernel enables the construction of multi-kernel polar codes. Multi-kernel codes are also decoded with a SC decoder and thus suffer its long decoding latency. This made devising fast decoding solutions for larger size kernels necessary. The size-3 kernels have drawn much attention due to their sufficiently high polarization exponents and the lowest decoding complexity among non-binary kernels. Hence, we identify a new node in the decoding tree of polar codes constructed by two commonly used size-3 kernels and propose a low-complexity decoder for it. Moreover, we adapt the generalized-repetition (G-REP) node introduced for the binary kernel to be used in the fast SC decoding of the size-3 kernel polar codes. The proposed fast decoders reduce the decoding latency at the cost of a slight degradation in error correction performance. Furthermore, the two specific size-3 kernels that can achieve the optimal polarization have a zero in their last rows. This results in the error-rate performance degradation of the repetition (REP) and G-REP nodes in addition to increased memory requirements for their fast decoders. Thus, we propose modifications to the REP and G-REP nodes that simultaneously result in improved error-rate performance and reduced memory requirements for their fast decoders.

Preface

The main results presented in Chapter 3 were published in IEEE Wireless Communications and Networking Conference Workshops (WCNCW) [3] and IEEE Transactions on Communications [4] in April 2016 and July 2019, respectively. Also, the results in Chapter 4 were published in [4]. Furthermore, the results in Chapter 5 will be published in the 2020 IEEE Vehicular Technology Conference. Finally, the results presented in Chapter 6 have been submitted to IEEE Wireless Communications Letters.

To Mom, Dad,

and Alireza.

Acknowledgements

I want to use this opportunity to thank everyone who has helped me in preparation of this work. I would like to express my deepest appreciation to Prof. Masoud Ardakani for his continuous support during my study and research, and for his patience, motivation, and immense knowledge. My special thanks go for Prof. Chintha Tellambura for his kind support which made the completion of this work possible.

Also, I would like to thank the rest of my thesis committee: Prof. T. Aaron Gulliver, Prof. Bruce Cockburn, and Prof. Majid Khabbazian, dedicating their precious time to read my thesis and providing invaluable comments. Furthermore, I want to thank Dr. Muhammad Hanif for his engagement through the learning process of my research and support on the way.

I would like to thank my loved ones. Thanks to my parents, brother and sister for their enduring love. Also, thanks to my dear husband, Alireza, for his love, support and encouragement, without them, I could never come thus far.

Acronyms

3GPP	3rd Generation Partnership Project
5G	Fifth Generation
AWGN	Additive White Gaussian Noise
B-DMC	Binary Discreet Memoryless Channel
BEC	Binary Erasure Channel
BER	Bit Error-Rate
BLER	BLock Error-Rate
BPSK	Binary Phase Shift Keying
CRC	Cyclic Redundancy Check
G-PC	Generalized Parity-check
G-REP	Generalized Repetition
GA	Gaussian Approximation
LLR	Log-Likelihood Ratio
ML	Maximum Likelihood
PM	Path Metric
REP	Repetition
SC	Successive-Cancellation

SCF	Successive-Cancellation Flip
SCL	Successive-Cancellation List
SNR	Signal-to-Noise Ratio
SPC	Single Parity-check
t**D-SPC**	t-Dimensional Single Parity-check

Table of Contents

List of Tables

List of Figures

Chapter 1

Motivation

1.1 Introduction

In any communication link, information is transmitted via a medium referred to as the communication channel. The channel is usually imperfect and noisy resulting in the erroneous received data. However, the transmission reliability and robustness to disturbances present on the transmission channel can be increased by adding redundant bits to the original data bits. This process is a part of any modern communication system and is called error correction coding or channel coding.

How much redundancy should one add? This is the fundamental question about channel codes and was answered by Shannon [12] in 1948. Shannon's theory stated that regardless of the advent of technology, reliable communication over a noisy channel is limited by a maximum information rate, referred to as the channel capacity. An ideal channel code is one that is capacity-achieving, meaning that it provides error-free communication at an information rate equal to the capacity. For decades, tremendous effort has been made to find channel codes that achieve the capacity or can approach it. By the turn of the century, these efforts resulted in the discovery of two powerful coding schemes, low-density parity check (LDPC) codes [13, 14] and Turbo codes [15].

Although the LDPC and Turbo codes approach the channel capacity for specific cases under iterative decoding, they can not be considered capacity-achieving in general.

Only for the case of the binary erasure channel, LDPC codes are provably capacity achieving [16]. Furthermore, the construction of both codes is complicated due the required randomness in the process.

Polar codes, discovered by Arıkan [1], are the first class of channel codes that are provably capacity-achieving. This new class of error correcting codes has an explicit non-random construction and can achieve the symmetric capacity of any binary-input memoryless channel under successive-cancellation (SC) decoding as the code length goes to infinity [1]. In particular, polar codes are constructed based on the phenomenon of channel polarization [1]. The polarization synthesizes bit channels such that certain bits are deemed reliable and used to transfer data while others are deemed completely unreliable and frozen to a known value. As the code length goes to infinity, the fraction of reliable bits converges to the channel capacity.

Due to these unique features, polar codes have received significant attention in recent years and have been selected to be used for the control channels in the fifth generation mobile communication standards (5G) [17]. However, a number of challenges are associated with their practical application. First, polar codes are only asymptotically capacity-achieving under SC decoding. Second, the SC decoder suffers from relatively long decoding latency. Third, polar codes lack length flexibility.

1.1.1 Error Correction Performance

Polar codes achieve the capacity under SC decoding as the code length goes to infinity. However, the error correction performance of polar codes for short to moderate block lengths with the SC decoder is inferior to the other state-of-the-art LDPC and Turbo codes. This is due to the significant performance gap between the SC decoder and the maximum-likelihood (ML) decoder [18]. Unfortunately, ML decoding has a very high computational complexity, even for moderate-length polar codes. Thus, to improve the performance of polar codes, lower complexity decoders such as the successive-cancellation list (SCL) decoder and the successive-cancellation flip (SCF) decoder have been proposed.

The SCL decoder, similar to the original SC decoder, operates serially as it estimates one bit at a time. However, unlike the latter which keeps only a single decoding path, the SCL decoder maintains a list of L most probable paths while decoding each bit [18]. The SCL decoder achieves near-ML performance even with moderate list sizes [18]. Enhancing it with a cyclic redundancy check (CRC) further improves the error correction performance beyond that of LDPC and Turbo codes [18].

Although the SCL decoder achieves near-ML performance, compared to the SC decoder, it has higher computational and memory complexities that grow linearly with the list size. Therefore, a different approach called SCF decoding was introduced in [10] to improve the performance of the SC decoder.

The SCF decoder, unlike the SCL decoder that parallelizes multiple SC decoders, relies on multiple sequential applications of the SC decoder. In particular, the SCF decoder [10] allows up to a given number of decoding trials. The first trial is essentially the same as SC decoding, but it also forms a list of bit-flip positions which will be flipped in the next trials if the CRC is not satisfied. The bit-flip positions correspond to the information bits with the smallest absolute values of the decision log-likelihood ratios (LLRs). If the CRC is not satisfied in the initial SC decoding, then one bit is flipped in the next trial, and the SC decoder is used to decode the subsequent positions. A new decoding trial is launched unless the CRC is satisfied or the maximum number of trials is reached.

It is shown in [10] that the error-rate performance of the SCF decoder with up to 32 trials is almost identical to that of the SCL decoder with $L = 2$ with half the memory requirement and half the computational complexity for reliable channels. More specifically, unlike the SCL decoder, the memory requirements of the SCF decoder do not scale linearly with the number of codewords considered. Furthermore, the computational complexity of the SCF decoder is similar to that of the SC decoder when the channel has high reliability. As such, the SCF decoding has gained interest recently in the research community [11, 19–24].

1.1.2 Decoding Latency

The SC decoder sequentially decodes the bits by traversing a decoding tree from top to bottom and from left to right. This serial decoding operation of the SC decoder results in high decoding latency. To increase the decoding speed, researchers have developed multiple schemes [5–8]. The main underlying idea behind these schemes is to parallelize some operations to reduce the decoding latency. More specifically, these schemes identify special nodes in the decoding tree of the polar code and implement fast parallel decoders for them. This way the decoder avoids the code tree traversal and outputs multiple bits in parallel.

In particular, [5] identifies nodes with all frozen bits and nodes with all information bits called Rate-0 (no information bits) and Rate-1 (all information bits) nodes, respectively, and provides low-complexity parallel SC decoders for them. Similarly, [6] identifies single-parity-check (SPC) and repetition (REP) nodes along with their fast decoders. Low-complexity decoders of five nodes (Type-I, Type-II, Type-III, Type-IV, and Type-V nodes) are proposed in [7] to further increase the SC decoding speed. Moreover, [8] provides general rules for the fast decoding of polar codes by identifying generalized-REP (G-REP) and generalized parity-check (G-PC) nodes, which generalize most of the other nodes. The aforementioned proposed fast SC decoders mainly increase the throughput and reduce the latency without any error correction performance loss with respect to the conventional SC decoder.

To increase the SCL decoding speed, a similar strategy to that of the fast SC decoder is followed. However, unlike the fast SC decoders [5–7] that output only the most-probable output sequence, the fast SCL decoders maintain and output a list of the most probable paths.

Similar to the SC and SCL decoders, the decoding latency of the SCF decoder can be improved by implementing fast decoders of different nodes. In particular, a fast parallel decoder of a node, in addition to outputting the codeword, should also compute and update the list of most probable bit-flip positions.

1.1.3 Length-flexibility

Another challenge associated with polar codes is their lack of length flexibility. In particular, the seminal work on polar codes [1] proposed to construct these codes by a recursive Kronecker product of a size-2 (binary) kernel. This limits the codeword lengths of the polar codes to the integer powers of two. However, arbitrary-length codes are desirable for varying channel conditions. Thus, different length-matching techniques, such as puncturing [25, 26] and shortening [27, 28] have been proposed for polar codes. While, in the 5G standard, the puncturing and shortening methods have been selected as the rate-matching (length-matching) schemes for polar codes [2], they suffer two main drawbacks. First, punctured or shortened polar codes suffer from high decoding complexity and latency as they are decoded on their mother code tree, whose size might be significantly larger than the code length. Second, some optimizations are required to determine the best patterns for the punctured and shortened bits to achieve the best performance for a given length [29, 30].

An alternative solution to achieving length flexibility is to use different sized kernels in the construction of the polar codes. In particular, larger size polarizing matrices can be used along with the binary kernel [29, 31, 32] to achieve more flexible codeword lengths. The resulting codes are aptly named multi-kernel polar codes.

It is shown in [29, 33] that in some cases the multi-kernel polar codes constructed with combinations of the size-2 and size-3 (ternary) kernels outperform the selected rate-matching scheme in the 5G standard. Therefore, multi-kernel polar codes, especially those constructed with ternary kernels, have received significant attention [29, 30, 33–36]. Similar to the binary kernel polar codes, multi-kernel polar codes are decoded by the SC decoder and hence they suffer from the shortcomings of the SC decoders.

1.2 Contribution

Despite the capacity-achieving property of polar codes, they have mediocre error correction performance at short to moderate lengths. The SCL and SCF decoders can be used to improve the error correction performance of polar codes. Similar to the SC decoder, these decoders have relatively long decoding latency as they operate serially. However, their decoding latency can be improved by implementing fast parallel decoders for different kind of nodes, as discussed in Section 1.1.2.

The proposed fast SC decoders in [5–7] only output a single codeword. The SCL decoder, on the other hand, maintains a list of L codewords for each bit. As such, the fast SC decoders of the special nodes require modifications to be used for the SCL decoder. Fast SCL decoders for Rate-0, REP, SPC and Rate-1 nodes have been proposed in [37–39]. To further increase the SCL decoding speed, we present fast SCL decoders for five other nodes (Type-I, Type-II, Type-III, Type-IV, and Type-V) identified in the decoding tree of a polar code [7]. We further study how the fast SCL decoders can be adopted to be used in the case of distributed parity-check aided SCL decoding. It is shown that implementing the proposed fast decoders, along with the existing fast SCL decoders of special nodes, can increase the decoder throughput significantly with identical bit error-rate (BER) performance.

Since the SCF decoder uses SC decoding in each trial, implementing fast SC decoders will improve the decoding latency of the SCF decoder. However, the SCF decoder must also maintain a list of the most probable bit-flip positions and should be able to flip the bits at these positions. Therefore, the existing fast SC decoders cannot be used directly to increase the decoding speed. Fast SCF decoders for the REP, Rate-1, SPC, and Type-I nodes have been proposed in [11]. However, except for the REP node, the decision metrics used in these decoders to compute the most-probable bit-flip positions, result in a BER performance loss compared to the original SCF decoder [10]. To resolve this issue, we propose a new procedure to compute and update the list of most-probable bit-flip positions. More specifically, we propose a decision metric that reflects the effect of a bit/bits flip on the log likelihood of the estimated codeword. The proposed decision metric can be computed without traversing the code tree, hence facilitating fast SCF decoding. We investigate Rate-1, SPC and Type-I nodes with our proposed procedure. Moreover, in order to further reduce the decoding latency, we adapt the existing fast SC decoders of Type-II, Type-III, Type-IV and

Type-V nodes for SCF decoding. Our proposed fast decoder significantly reduces the SCF decoder latency without sacrificing its error correction performance.

As was mentioned in Section 1.1.3, lack of length-flexibility is another shortcoming of polar codes and one way to combat it is to combine the binary kernel with larger size kernels to construct multi-kernel polar codes. The size-3 kernels have drawn much attention due to their sufficiently high polarization exponents and the lowest decoding complexity among non-binary kernels [29, 30, 34–36]. Unlike the binary kernel, there are several choices for a ternary kernel and they achieve different levels of polarization. In particular, $\mathbf{T_3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ and $\mathbf{T'_3} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ were introduced and proved to be optimal for polarization in [36] and [40], respectively.

Like binary polar codes, multi-kernel polar codes suffer from high decoding latency due to the serial decoding nature of the SC decoder. Recently, [30] described Rate-0, REP, Rate-1 and SPC nodes and their decoding methods for multi-kernel polar codes constructed with combination of the binary and $\mathbf{T_3}$ kernels. It is shown that implementing fast decoding of these four nodes significantly improves the decoding speed. To further increase the decoding speed, we identify a family of special nodes called t-dimensional SPC nodes (tD-SPC), where t stands for the number of dimensions, and propose corresponding fast parallel decoding algorithms. It is observed that the proposed nodes significantly improve the decoding latency at the expense of a slightly degraded error-rate performance. We also adapt G-REP nodes, introduced for the binary kernel in [8], to be used in the fast SC decoding of polar codes constructed with $\mathbf{T_3}$ and $\mathbf{T'_3}$ kernels.

Unlike the size-2 kernel, $\mathbf{T_3}$ and $\mathbf{T'_3}$ have a zero in their last rows. As such, the codeword of a REP node, the node corresponding to only one message bit, has some zero elements. This results in error-rate performance degradation. Moreover, different kernel orders in multi-kernel polar codes yield different zero locations for a REP node [30]. Thus, the fast decoders for an REP node in a multi-kernel polar code require storing all the zero-location patterns, which increases the memory requirements for the decoders. In order to reduce the memory requirements, [30] proposes a fast decoder that decodes the REP node up to a certain maximum size. This, in turn, results in reduced decoding speed.

To address the above issues, we propose a modified REP pattern for polar codes constructed using size-3 kernels. The proposed modification eliminates zeros in the coded bits resulting in a unified REP pattern in multi-kernel codes. We also present a low-complexity optimal decoder for the modified REP node. In addition, we apply our proposed modification to the generalized repetition (G-REP) nodes [8] to further improve the error-rate performance.

1.3 Organization of the Thesis

The following chapter provides the required background to understand polar codes. In particular, channel polarization, polar code construction, encoding, and SC decoding are reviewed. Furthermore, it summarizes SCL and SCF decoders, and introduces the special nodes in the decoding tree of polar codes. The chapter ends by reviewing multi-kernel polar codes.

Fast SCL and fast SCF decoders are proposed in Chapter 3 and Chapter 4, respectively. The decoding latency and error correction performance of the proposed fast decoders are also presented.

We propose tD-SPC nodes in Chapter 5 to increase the decoding speed of the polar codes constructed by ternary kernels. Their latency and performance are also compared with the existing scheme. We also adapt G-REP nodes fast decoding to the ternary kernels in this chapter.

In Chapter 6 we propose the modifications of REP and G-REP node for ternary kernels and compare the error correction performance of the modified codes with the original ones.

Finally, a summary of the dissertation and suggestions for future work are provided in Chapter 7.

Conventions for notation: Matrices and vectors are denoted by capital bold face letters and small bold face letters, respectively. y_k denotes the kth entry of vector $\mathbf{y}$. Furthermore, $\mathbf{y}_{\mathbf{K}}$ are all the y_k entries with $k \in \mathbf{K}$ and $\mathbf{y}_i^j$ are y_k where $i \leq k \leq j$. Moreover, $\{\!\{R\}\!\}$ denotes the set $\{0, 1, \cdots, R-1\}$. $\mathbf{M}_N$ is a square matrix of size $N \times N$. Also, $\mathbf{0}_N$ is an all-zero square matrix of size $N \times N$.

Symbol $\otimes$ denotes the Kronecker product and $\mathbf{F}^{\otimes n}$ is the n times Kronecker product of $\mathbf{F}$ with $\mathbf{F}^{\otimes 0} = 1$. Furthermore, $\oplus$ is the binary XOR and $\boxplus$ is the check node operation defined as $a \boxplus b = 2 \operatorname{arctanh}\left(\tanh\left(\frac{a}{2}\right) \tanh\left(\frac{b}{2}\right)\right)$. Also, $\mathbf{M}^{-1}$ is the inverse of a matrix $\mathbf{M}$.

Chapter 2

Background

This chapter provides the required background for understanding polar codes. It starts with a description of channel polarization, which is the main idea behind polar coding. It then presents polar code construction and the encoding method for both non-systematic and systematic codes. Next, the SC decoding method is reviewed, as well as the SCL and SCF decoding algorithms. Then the simplified and fast SC decoders, which provide a significant increase in decoding speed, are reviewed. Finally, the chapter proceeds with a description of multi-kernel polar codes.

2.1 Channel Polarization

In his seminal work, Arıkan introduced the concept of channel polarization [1] where N independent copies of a given channel $\mathcal{W}$ are synthesized to a set of N new channels, $\{\mathcal{W}_N^{(i)} : 1 \leq i \leq N\}$. Here $\mathcal{W} : \mathcal{X} \rightarrow \mathcal{Y}$ denotes a binary discrete memoryless channel (B-DMC), where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output alphabets, respectively. The new set $\{\mathcal{W}_N^{(i)} : 1 \leq i \leq N\}$ exhibits a polarization effect in the sense that, as $N \rightarrow \infty$, except for a vanishing fraction of indices i, the symmetric capacities, $I(\mathcal{W}_N^{(i)})$, tend toward either one or zero.

The symmetric channel capacity, $I(\mathcal{W})$, and the Bhattacharyya parameter, $Z(\mathcal{W})$, are two measures of the channel quality. The symmetric capacity is the highest rate at which reliable communication is possible over $\mathcal{W}$ when the inputs are used with equal frequencies. Let $\mathcal{W}(y|x)$ denotes the transition probability, where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, then $I(\mathcal{W})$ is defined as

$$I(\mathcal{W}) = \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} \frac{1}{2}\mathcal{W}(y|x) \log \frac{\mathcal{W}(y|x)}{\frac{1}{2}\mathcal{W}(y|0) + \frac{1}{2}\mathcal{W}(y|1)}. \tag{2.1}$$

The Bhattacharyya parameter is an upper bound on the probability of decision error when a maximum-likelihood decoder is used to estimate a single y and is expressed as

$$Z(\mathcal{W}) = \sum_{y\in\mathcal{Y}} \sqrt{\mathcal{W}(y|0)\mathcal{W}(y|1)}. \tag{2.2}$$

If we use a base-2 logarithm, then both $Z(\mathcal{W})$ and $I(\mathcal{W})$ take a value in the interval $[0, 1]$ and are related with the following two inequalities

$$I(\mathcal{W}) \geq \log \frac{2}{1 + Z(\mathcal{W})}, \tag{2.3}$$

and

$$I(\mathcal{W}) \leq \sqrt{1 - Z(\mathcal{W})^2}. \tag{2.4}$$

From (2.3) and (2.4) it can be observed that $I(\mathcal{W}) \approx 1$ if and only if $Z(\mathcal{W}) \approx 0$ and vice-versa. In particular, $I(\mathcal{W}) = 1$ indicates a perfect channel with error-free transmission, and $I(\mathcal{W}) = 0$ is a completely unreliable channel where the probability of correctly detecting a bit approaches 0.5.

The basic channel polarization can be explained as follows. Consider the two bits $\mathbf{u}_0^1 \in \mathcal{X}^2$ transmitted in two independent instances of $\mathcal{W}$, as in Fig. 2.1(a), with the corresponding output vectors $\mathbf{y}_0^1 \in \mathcal{Y}^2$. The mutual information values will be

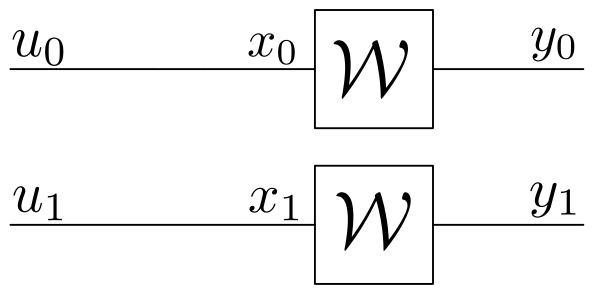
$$I(\mathbf{y}_0^1; u_0) = I(\mathcal{W}) = I(\mathbf{y}_0^1; u_1). \tag{2.5}$$

Now consider that $\mathbf{u}_0^1$ is transformed to $\mathbf{x}_0^1$ such that $x_0 = u_0 \oplus u_1$ and $x_1 = u_1$, as depicted in Fig. 2.1(b). In this case, two independent channels $\mathcal{W}$ are combined to create a vector channel $\mathcal{W}_2 : \mathcal{X}^2 \to \mathcal{Y}^2$ with

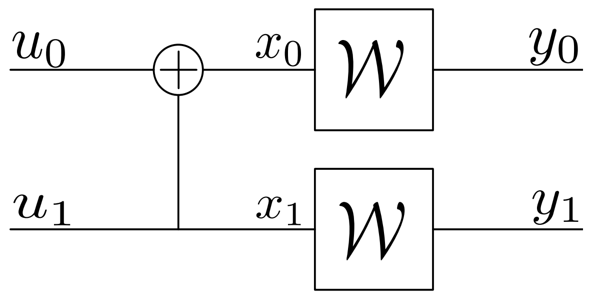
$$I(\mathbf{y}_0^1; \mathbf{u}_0^1) = I(\mathbf{y}_0^1; \mathbf{x}_0^1) = 2I(\mathcal{W}). \tag{2.6}$$

This step is called channel combining in [1]. Using the chain rule, the left hand side of (2.6) can be written as

$$I(\mathbf{y}_0^1; \mathbf{u}_0^1) = I(\mathbf{y}_0^1; u_0) + I(\mathbf{y}_0^1|u_0; u_1) = I(\mathbf{y}_0^1; u_0) + I(\mathbf{y}_0^1, u_0; u_1). \tag{2.7}$$



(a)



(b)

Fig. 2.1: Channel polarizing transformation: (a) No transformation, and (b) Basic transformation

We can define bit channel $\mathcal{W}_2^{(0)} : \mathcal{X} \rightarrow \mathcal{Y}^2$ with the mutual information $I(\mathbf{y}_0^1; u_0)$. $I(\mathbf{y}_0^1; u_0)$ can be interpreted as the mutual information of the channel between the input u_0 and the output $\mathbf{y}_0^1$, with u_1 being random. Similarly, $\mathcal{W}_2^{(1)} : \mathcal{X} \rightarrow \mathcal{Y}^2$ can be defined with the mutual information $I(\mathbf{y}_0^1, u_0; u_1)$. This can be interpreted as the mutual information of the channel between u_1 and the output $\mathbf{y}_0^1$, when u_0 is available at the decoder. In [1], this step is called channel splitting. Then the following relations, which are proved in [1], can be written.

$$I(\mathcal{W}_2^{(0)}) \leq I(\mathcal{W}) \leq I(\mathcal{W}_2^{(1)}), \tag{2.8}$$

$$I(\mathcal{W}_2^{(0)}) + I(\mathcal{W}_2^{(1)}) = 2I(\mathcal{W}). \tag{2.9}$$

It can be observed that by channel combining and splitting, two independent copies of $\mathcal{W}$ are synthesized to two new channels. The new channels $\mathcal{W}_2^{(0)}$ and $\mathcal{W}_2^{(1)}$ are a worse and a better channel than $\mathcal{W}$, respectively. In other words, the probability of correctly estimating u_0 decreases while that of u_1 increases. As the number of transformed

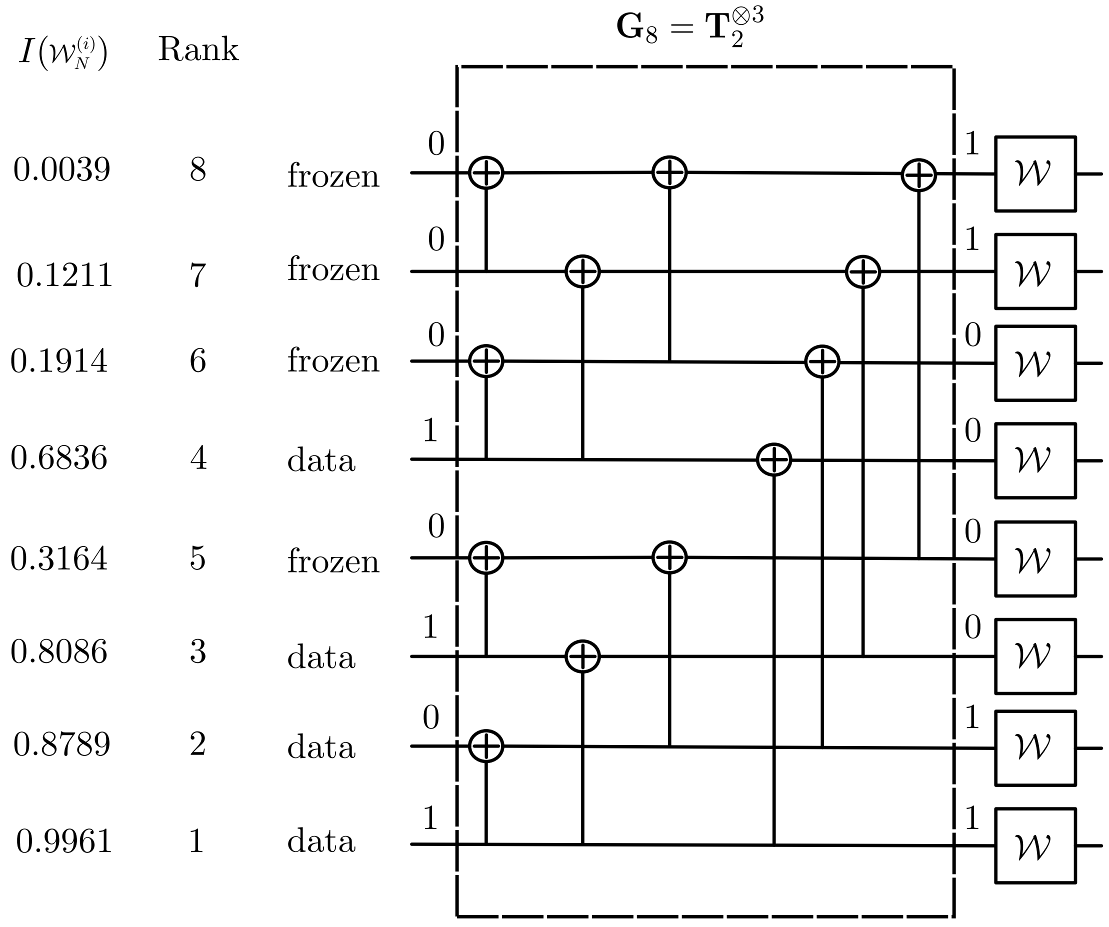


Fig. 2.2: Eight instances transformation

channels increases, the probability of correctly estimating a bit either approaches 1.0 (completely reliable) or 0.5 (completely unreliable) with the proportion of reliable bits approaching the channel capacity [1].

By recursive applications of the polarizing transformation, multiple instances of $\mathcal{W}$ can be synthesized. The polarizing transformation for eight instances of the channel is illustrated in Fig. 2.2.

2.2 Polar Code Encoding

2.2.1 Linear Block Codes Encoding

An (N, k) linear block code of length N and rate k/N can be defined by a generator matrix $\mathbf{G}_{k\times N}$. In particular, in an (N, k) linear block code each information sequence, $\mathbf{u}_{1\times k}$, is a block of length k symbols, which is converted to a codeword of length N, $\mathbf{x}_{1\times N}$ using $\mathbf{G}_{k\times N}$. Mathematically, this can be written as

$$\mathbf{x} = \mathbf{u}\mathbf{G}. \tag{2.10}$$

An example of linear block codes are Hamming codes. For a $(7, 4)$ Hamming code, each information sequence is in the form of $\mathbf{u} = \{m_0\,, m_1,\ m_2,\ m_3\}$. Assuming a binary Hamming code with $\mathbf{u} = \{1\,, 0,\ 0,\ 1\}$ and generator matrix

$$\mathbf{G}_{4\times 7} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.11}$$

the codeword will be $\mathbf{x} = \mathbf{u}\mathbf{G} = \{0,\ 1,\ 1,\ 1,\ 0,\ 0,\ 1\}$.

2.2.2 Polar Code Encoding

In a binary polar code of length $N = 2^n$, where n is a positive integer, a channel transformation matrix is defined as

$$\mathbf{G}_N = \mathbf{T_2}^{\otimes n}, \tag{2.12}$$

where $\mathbf{T_2}$ is the matrix representation of the basic two-bit transformation, depicted in Fig. 2.1(b), which can be expressed as

$$\mathbf{T_2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.13}$$

The transformation matrix $\mathbf{G}_N$ applies the polarization transformation and synthesizes N bit channels from N independent copies of a given channel. For a $P(N, k)$ polar code, amongst the N synthesized channels, the k most reliable ones are used to carry the information and the $N-k$ least reliable synthesized channels are frozen to a known value, usually 0. We denote the index set of k information bits by $\mathcal{I}$, and $\mathcal{F}$ is used to denote the index set of frozen bits.

The transformation matrix has the property of $\mathbf{G}_N = \mathbf{G}_N^{-1}$ and can be recursively computed as

$$\mathbf{G}_N = \begin{bmatrix} \mathbf{G}_{N/2} & \mathbf{0}_{N/2} \\ \mathbf{G}_{N/2} & \mathbf{G}_{N/2} \end{bmatrix}. \tag{2.14}$$

The generator matrix of the polar code is a sub-matrix of $\mathbf{G}_N$ including the rows whose indices are in $\mathcal{I}$. However, the encoding complexity can be reduced by exploiting the recursive structure of $\mathbf{G}_N$ in (2.14). To do so, an auxiliary input vector $\mathbf{u}$ of length N is introduced. In particular, $\mathbf{u} = \{u_0, u_1, \cdots, u_{N-1}\}$ is generated by setting $u_i = 0$ if $i \in \mathcal{F}$, and assigning the information bits to the indices in $\mathcal{I}$. Note that the number and location of the frozen bits affect the input vector $\mathbf{u}$, while $\mathbf{G}_N$ does not change. The input vector $\mathbf{u}$ is then mapped to the codeword $\mathbf{x} = \{x_0, x_1, \cdots, x_{N-1}\}$ such that

$$\mathbf{x} = \mathbf{u}\mathbf{G}_N. \tag{2.15}$$

Therefore, a $P(N,k)$ polar code is completely described by its channel transformation matrix $\mathbf{G}_N$ and the set of information bit indices $\mathcal{I}$. Encoding of a $P(8,4)$ polar code with data bits $\{1\,1\,0\,1\}$ is illustrated in Fig. 2.2, where the auxiliary input vector is $\mathbf{u} = \{0,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 1\}$.

Considering the recursive nature of the generator matrix $\mathbf{G}_N$, the time complexity of a serial implementation of this encoding algorithm is $\mathcal{O}(N \log N)$ [1].

2.2.3 Systematic Encoding

The encoding scheme presented above results in a non-systematic codeword, i.e. the information bits do not appear unaltered in the codeword $\mathbf{x}$. However, one might be interested in generating a systematic polar codeword which can be separated into information and parity bits. A systematic encoding scheme for polar codes was introduced in [41]. It was shown that the systematic encoding preserves the low-complexity nature of non-systematic polar codes, improves the BER performance, and retains the same block-error-rate (BLER) performance.

In the systematic encoding algorithm, first the information bits, $\mathbf{x}_{\mathcal{I}}$, are located at the information indices. Then the parity bits, which will be placed at the frozen bit indices, are calculated as

$$\mathbf{x}_{\mathcal{F}} = \mathbf{x}_{\mathcal{I}}(\mathbf{G}_{\mathcal{I}\mathcal{I}})^{-1}\mathbf{G}_{\mathcal{I}\mathcal{F}}. \tag{2.16}$$

Here $\mathbf{G}_{\mathcal{I}\mathcal{I}}$ and $\mathbf{G}_{\mathcal{I}\mathcal{F}}$ are sub-matrices of $\mathbf{G} = \mathbf{T_2}^{\otimes n}$. The rows and columns of $\mathbf{G}_{\mathcal{I}\mathcal{I}}$ correspond to the set of indices in I, and the rows and columns of $\mathbf{G}_{\mathcal{I}\mathcal{F}}$ corresponds to the $\mathcal{I}$ and $\mathcal{F}$, respectively.

2.3 Construction of Polar Codes

One of the challenges associated with the polar coding is determination of the synthesized channel reliability, referred to as the polar code construction. Although the encoding of polar codes is explicit in theory, the precise estimation of the synthesized channels' reliability is intractable in practice. In particular, with different underlying channels and signal-to-noise ratios (SNRs) the indices of the reliable and unreliable synthesized bit channels change [1, 42]. Therefore, selection of the k most reliable channels out of N synthesized ones requires an algorithm for ranking them according to their reliability.

In [1], Arıkan proposed an efficient construction method for a binary-erasure channel (BEC). Specifically, he showed that if the Bhattacharyya parameter (or the erasure probability) of the underlying BEC is z then the two synthesized channels are BECs with the Bhattacharyya parameters of $2z - z^2$ and z^2. Since the synthesized channels are formed by the repeated Arıkan transformation, their Bhattacharyya parameters can be calculated recursively. Finally, the channels with the smallest Bhattacharyya parameters are used for data transmission. As an example, Fig. 2.2 presents the polar code construction for a BEC with an erasure probability of 0.5.

In general, the polar code construction can be solved by computing the Bhattacharyya parameters of the synthesized channels. However, unlike for a BEC, for other symmetric channels the closed form expressions of the Bhattacharyya parameters of the synthesized channels are nonexistent. Thus, their construction is not as efficient as that of a BEC [1, 42].

To construct polar codes for general channels, different approximate methods are proposed in [1, 42–48]. The earliest construction method evaluates simple bounds on the Bhattacharyya parameters of the synthesized channels [43]. In particular, for a given channel with capacity I, the Bhattacharyya parameters of the synthesized channels are computed by applying the channel transformation to a BEC with capacity I. However, this heuristic approach only leads to capacity-achieving codes for the BECs. Later, estimation of the Bhatacharyya parameters through a Monte-Carlo approach was suggested in [1]. While this method can be applied to a wide range of channels, including the additive white Gaussian noise (AWGN) channel, it comes at the cost of very large computational complexity. Therefore, density evaluation [49] was utilized in [44] and [45] to construct polar codes. Further, to reduce the computational complexity of density evaluation, for the AWGN channel intermediate LLRs were approximated as Gaussian random variables in [46–48]. This is referred to as Gaussian approximation (GA) and found to approximate well the reliability of the synthesized channels [47, 48]. GA has the second least complexity amongst the aforementioned constructions algorithms [50].

2.3.1 Gaussian Approximation

To compute the bit channel reliability for an AWGN channel, the GA method [46] is used. In the GA process it is assumed that all the intermediate LLRs have a Gaussian distribution. With this assumption, the absolute mean value corresponding to each index can be updated through the LLR transformation from one stage to the next. To begin, all N indices are distributed as the channel $\mathcal{W}$. In particular, if we consider a zero mean AWGN channel with variance σ^2 then each index is initialized as $z_i^N = 2/\sigma^2$. The mean values are recursively computed by [46]

$$z_{2i-1}^{N/2} = \phi^{-1}(1-(1-\phi(z_i^N))^2), \tag{2.17}$$

$$z_{2i}^{N/2} = 2z_i^N, \tag{2.18}$$

where

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{\infty} \tanh\frac{u}{2} e^{\frac{-(u-x)^2}{4x}} du, & x > 0 \\ 1, & x = 0. \end{cases} \tag{2.19}$$

The main computational burden in the GA method is the non-linear calculation in (2.17). Thus, [51] proposed a piece-wise quadratic approximation of $\Xi(x) = \phi^{-1}(1 - (1 - \phi(x))^2)$, as

$$\Xi(x) \approx \begin{cases} 0.9861x - 2.3152, & x > 12 \\ x(0.009005x + 0.7694), & 3.5 < x \geq 12 \\ x(0.062883x + 0.3678) - 0.1627, & 1 < x \geq 3.5 \\ x(0.2202x + 0.06448), & \text{otherwise.} \end{cases} \tag{2.20}$$

With the proposed approximation in [51] the complexity of GA polar code construction is essentially the same as the recursive Bhattacharyya computation for a BEC [1].

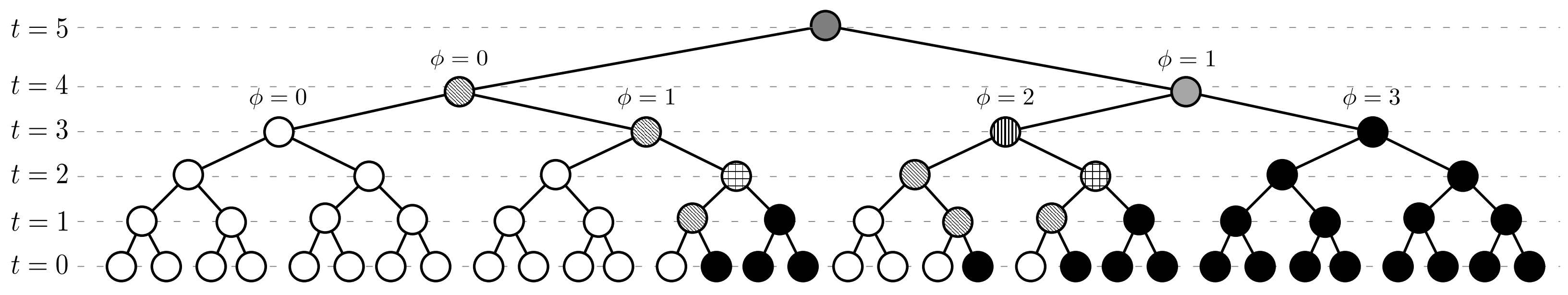
2.4 The Successive-Cancellation Decoder

In a SC decoder, as indicated by its name, bits are estimated sequentially starting from u_0. The real decoding task, however, is to estimate $\mathbf{u}_I$ since errors can be avoided in the frozen part of the code. Once the estimate $\hat{\mathbf{u}}_0^{i-1}$ is available, the SC decoder decides an information bit u_i by computing

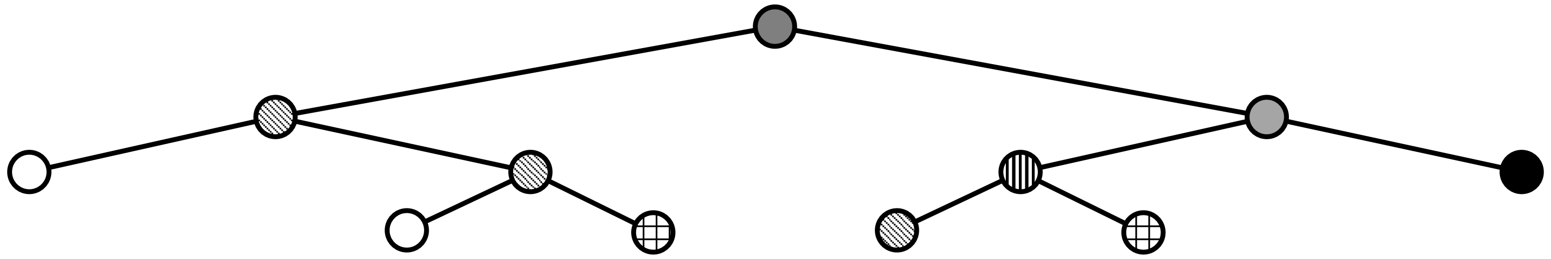
$$\hat{u}_i = \begin{cases} 0, & \text{if } \frac{\mathcal{W}_N^{(i)}(\mathbf{y}_0^N, \hat{\mathbf{u}}_0^{i-1}|u_i=0)}{\mathcal{W}_N^{(i)}(\mathbf{y}_0^N, \hat{\mathbf{u}}_0^{i-1}|u_i=1)} \geq 1; \\ 1, & \text{otherwise,} \end{cases} \tag{2.21}$$

where $\frac{\mathcal{W}_N^{(i)}(\mathbf{y}_0^N, \hat{\mathbf{u}}_0^{i-1}|u_i=0)}{\mathcal{W}_N^{(i)}(\mathbf{y}_0^N, \hat{\mathbf{u}}_0^{i-1}|u_i=1)}$ is the likelihood of u_i and can be computed recursively. It was shown in [52] that the SC decoding can be implemented in the logarithmic domain and likelihood can be replaced by LLR. This eliminates the multiplication and division operations and significantly reduces the complexity of each processing element.

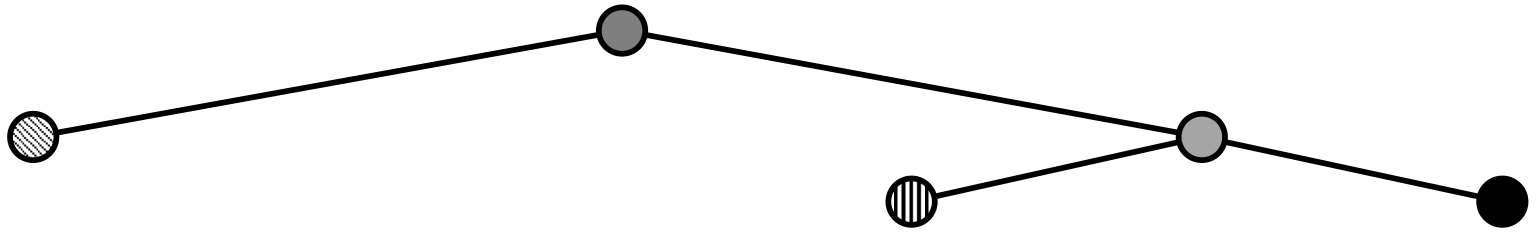
The SC decoding can be better understood using a binary-tree representation of the polar code. Fig. 2.3(a) depicts the binary-tree representation of a $P(32, 15)$, where



(a)



(b)



(c)

Fig. 2.3: Binary-tree representations of a polar code $P(32, 15)$: (a) SC decoding tree, (b) Fast-SC decoding tree with nodes in [5, 6], and (c) Fast-SC decoding tree with nodes in [5–8]

black and white leaf nodes are the information and frozen bits, respectively. Also, shaded nodes denote different special nodes, which will be introduced later in this chapter.

In Fig. 2.3(a), t denotes the level in the decoding tree and $0 \leq \phi < 2^{n-t}$ is the count from left to right at level t. $R = 2^t$ is used to denote the length of a node rooted at level t. Observe that each node in the code tree of a polar code has a bijective relationship with pair (ϕ, t). For example, the root node corresponds to $(0, n)$, whereas the ith leaf node can be represented as $(i, 0)$. Further, with the exception of the leaf nodes, each node (ϕ, t) has two children: the left child $(2\phi, t-1)$ and the right child $(2\phi+1, t-1)$.

The SC decoding involves a tree traversal from top to bottom and left to right and can be viewed as information exchange between nodes in the decoding tree. In particular, a soft information vector, $\mathbf{y}^{\phi,t}$, is sent to each (ϕ, t) node from their parents, except the root node which receives the channel LLRs, i.e., $\mathbf{y}^{0,n} = \mathbf{y}_{\text{ch}}$. The node (ϕ, t) then computes hard bit estimates $\mathbf{x}^{\phi,t}$ from $\mathbf{y}^{\phi,t}$, and send these estimates to its parent. More specifically, with the exception of the leaf nodes, each node upon receiving $\mathbf{y}^{\phi,t}$ generates LLR vectors for its children as

$$y_k^{2\phi,t-1} = 2\operatorname{arctanh}\Big(\tanh(\frac{y_k^{\phi,t}}{2})\tanh(\frac{y_{k+2^{t-1}}^{\phi,t}}{2})\Big) \tag{2.22}$$

$$y_k^{2\phi+1,t-1} = y_{k+2^{t-1}}^{\phi,t} + (1 - 2x_k^{2\phi,t-1})y_k^{\phi,t}, \tag{2.23}$$

where $0 \leq k < 2^{t-1}$ is used to denote the kth entry of a vector. On the other hand, $\mathbf{x}^{\phi,t}$ is computed as

$$x_i^{\phi,t} = \begin{cases} x_i^{2\phi,t-1} \oplus x_i^{2\phi+1,t-1}, & \text{if } i < 2^{t-1}; \\ x_{i-2^{t-1}}^{2\phi+1,t-1}, & \text{otherwise,} \end{cases} \tag{2.24}$$

where $\oplus$ denotes the binary XOR operation, and $0 \leq i < 2^t$. The decoding process starts from the root node by setting $\mathbf{y}^{0,n} = \mathbf{y}_{ch}$. Each node, upon receiving its LLR vector, computes and passes LLR vectors to its children until the leaf nodes receive their LLR values. At the ith leaf node $(i, 0)$, the ith input bit u_i is estimated as

$$\hat{u}_i = \mathbf{x}^{i,0} = \begin{cases} 0, & \text{if } i \in \mathcal{F}; \\ H(y^{i,0}), & \text{otherwise,} \end{cases} \tag{2.25}$$

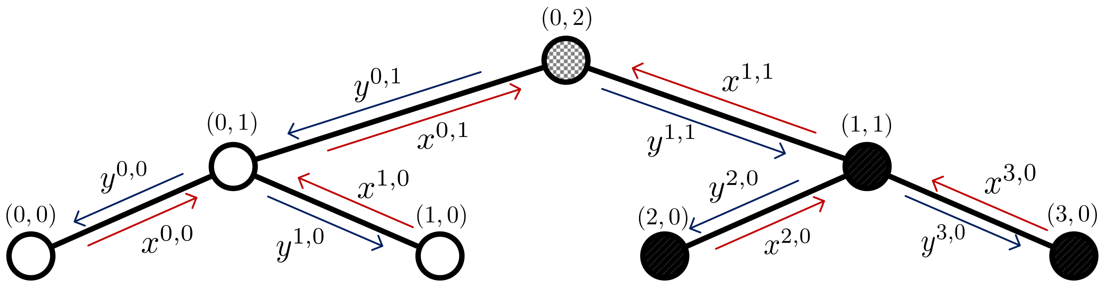


Fig. 2.4: The SC decoding tree of a length-four polar code.

where $H(\mathbf{y})$ makes a hard decision on each element of $\mathbf{y}$ as

$$H(y_i) = \begin{cases} 0, & \text{when } y_i \geq 0; \\ 1, & \text{otherwise.} \end{cases} \tag{2.26}$$

The leaf node $(i, 0)$ then sends $\mathbf{x}^{i,0}$ to their parents. Each node (ϕ, t) upon receiving $\mathbf{x}^{2\phi,t-1}$ and $\mathbf{x}^{2\phi+1,t-1}$ from its children, computes $\mathbf{x}^{\phi,t}$ using (2.24). Finally, the decoding process is completed when the hard decision estimate of $\mathbf{x}$, $\hat{\mathbf{x}} = \mathbf{x}^{0,n}$ is computed at the root node.

Example 2.1. *Fig. 2.4 depicts the decoding tree of a length four polar code and the corresponding information exchanges between the nodes. In particular, after getting the channel LLRs, the root node* $(0, 2)$ *computes* $\mathbf{y}^{0,1}$ *using* (2.22) *and sends it to its left child node* $(0, 1)$. *The node* $(0, 1)$ *then computes and passes* $y^{0,0}$ *to the node* $(0, 0)$. *This is a leaf node and it does not have any children. Thus it estimates* $x^{0,0}$ *by* (2.25) *and sends it to its parent. Upon receiving* $x^{0,0}$, *the node* $(0, 1)$ *computes* $y^{1,0}$ *using* (2.23) *and sends it to its right child. Then the leaf node* $(1, 0)$ *estimates* $x^{1,0}$ *and passes it to its parent. Now that the node* $(0, 1)$ *has received both* $x^{0,0}$ *and* $x^{1,0}$, *it can estimate* $\mathbf{x}^{0,1}$ *through* (2.24). *The decoding proceeds by traversing the remaining branches sequentially from top to bottom and left to right. This continues until the root node* $(0, 2)$ *receives* $\mathbf{x}^{1,1}$ *from its right child and estimates* $\mathbf{x}^{0,2}$.

2.4.1 Successive-Cancellation List Decoding

A well-known method to improve the error correction performance of block codes is list decoding [53,54]. This was first applied to polar codes in [18], where a likelihood-based

SCL decoder was introduced. An LLR-based SCL decoder was presented in [9], which shows identical error correction performance, while has lower computational complexity and memory requirements.

The SCL decoder operates in a similar fashion to the SC decoder. Instead of outputting only one codeword estimate, the SCL decoder maintains a list of L candidate codewords with their corresponding path metrics (PM) while decoding each bit. In particular, after estimating the ith bit, the path metric associated with the lth candidate codeword PM_i^l, is computed as [9]

$$\mathrm{PM}_i^l = \sum_{k=0}^{i-1} \ln\left(1 + e^{-(1-2\hat{u}_{k^l})y^{k^l,0}}\right), \tag{2.27}$$

where $\hat{u}_{k^l}$ is the estimate of the $(k+1)$-th bit, and $y^{k^l,0}$ is the LLR received by the $(k+1)$-th leaf node in the path l.

If the $(i+1)$-th bit is a frozen bit then $\hat{u}_{i^l} = 0$ for each list, and the path metrics are updated accordingly. But if the $(i+1)$-th bit is an information bit, then each path generates two paths corresponding to $\hat{u}_{i^l} = 0$ and $\hat{u}_{i^l} = 1$. The path metrics are updated accordingly, and a total of $2L$ paths are generated. Amongst them, only those L paths are maintained that have the lowest path metrics (largest reliability).

It was observed in [18] that aiding the SCL with a CRC in the selection of the final candidate codeword further improves the error correction performance. A CRC of length r, as an outer code, is concatenated to the polar code in serial. The CRC-aided SCL decoder operates as standard SCL decoder of polar codes except for the final output selection. The decoder selects the candidate codeword with the lowest path metric that satisfies the CRC constraint. If no such candidate is found, the output is the codeword with lowest path metric.

Although the SCL decoder achieves near-ML performance, compared to the SC decoder, it has higher computational and memory complexities that grow linearly with the list size. More specifically, for a code of size N the memory and computational complexities of the SC decoder are $\mathcal{O}(N)$ and $\mathcal{O}(N \log N)$, respectively, whereas for a list of size L, they increase to $\mathcal{O}(LN)$ and $\mathcal{O}(LN \log N)$, respectively, in the SCL decoder.

2.4.2 Successive-Cancellation Flip Decoding

Another polar code decoding scheme that outperforms the SC decoder is the SCF decoder. Similar to the SCL decoder, the SCF decoder selects the most-probable codeword amongst multiple codewords. However, unlike the SCL decoder that parallelizes multiple SC decoders, the SCF decoder can be viewed as multiple SC decoder operating in a sequential manner. In particular, in the initial phase, the codeword is decoded through the standard SC decoder. In addition to the decoding, the SC decoder identifies $T_{\max}$ bit-flip positions. The bit-level LLRs, $|y^{i,0}|$, corresponding to the bit-flip positions have the smallest absolute values.

Let $\mathbf{t} = \{t_0, t_1, ..., t_{T_{\max}-1}\}$ denote the set of bit-flip positions, where $\mathbf{t} \subset \mathcal{I}$ and $|y^{t_i,0}| \leq |y^{t_j,0}|$ for $0 \leq i \leq j < T_{\max}$. After the initial decoding, if the estimated codeword satisfies the CRC, the decoding process stops. Otherwise, another SC decoding trial is carried out. However, in this trial $\hat{u}_{t_0}$ is flipped and the subsequent bits are decoded with the standard SC decoder. The decoder outputs the estimated codeword if it satisfies the CRC. Otherwise, a new SC decoding trial is launched. But this time $\hat{u}_{t_1}$ is flipped instead. Again the CRC is checked, and a new trial is carried out if it is not satisfied. This process continues until the CRC of the estimated codeword is satisfied or the maximum number of trials $T_{\max}$ is reached. Note that the standard SC decoding corresponds to $T_{\max} = 0$.

The aforementioned procedure of the SCF decoder has $\mathcal{O}(N)$ memory complexity, and its average computational complexity is $\mathcal{O}(N \log N)$ at high SNR [10], which are the same complexities as those of the SC decoder.

2.5 Fast Successive-Cancellation Based Decoding

The sequential nature of the SC-based decoders and the tree traversal from top to bottom and left to right result in a high decoding latency. The decoding speed can be improved by implementing fast parallel decoders to avoid the code tree traversal and output multiple bits in parallel. Based on this idea, researchers have identified special nodes in the polar code tree and proposed their fast parallel decoders which improve the decoding speed significantly [5–7].

To better understand the special nodes, we find the following notations useful: For a node (ϕ, t), we define $\tilde{A}_{\phi,t} = \{i : i \in \{\!\!\{R\}\!\!\}, \text{ and } 2^t\phi + i \in \mathcal{I}\}$, and $\tilde{A}^c_{\phi,t} = \{i : i \in \{\!\!\{R\}\!\!\}, \text{ and } 2^t\phi + i \in \mathcal{F}\}$, where $R = 2^t$.

In particular, [5, 6] identified the following four nodes and proposed their fast SC decoding.

- Rate-0 node: This is a node (ϕ, t) correspond to $\tilde{A}_{\phi,t} = \{\}$. It has a tree with all leaf nodes being frozen bits (shown by white circles in Fig. 2.3(a)). This node is directly estimated as
$$x_i^{\phi,t} = 0. \tag{2.28}$$
- Rate-1 node: This node corresponds to a tree with only information leaf nodes and $\tilde{A}_{\phi,t} = \{\!\!\{R\}\!\!\}$. A Rate-1 is estimated by making hard decision on its soft-information vector, without any additional calculations, as
$$\mathbf{x}^{\phi,t} = H(\mathbf{y}^{\phi,t}). \tag{2.29}$$
- REP node: This node corresponds to $\tilde{A}_{\phi,t} = \{R - 1\}$ and is identified when only the rightmost leaf node is an information bit. In a REP node the data is repeated on all the coded bits. This node is decoded by performing a hard decision on the sum of the LLRs, as
$$x_i^{\phi,t} = H(\sum_{i=0}^{R-1} y_i^{\phi,t}). \tag{2.30}$$
- SPC node: This node correspond to the $\tilde{A}^c_{\phi,t} = \{0\}$ and is identified in the code tree where only the leftmost leaf node is a frozen bit. A SPC node can be estimated by Wagner decoding [55]. In particular, we set $\mathbf{x}_i^{\phi,t} = H(\mathbf{y}_i^{\phi,t})$ for $0 \le k < R$. If the frozen bits are fixed to zero, the SPC node will have an even-parity constraint, where the parity is computed as
$$\gamma = \bigoplus_{k=0}^{R-1} \mathbf{x}_i^{\phi,t}, \tag{2.31}$$

If the constraint is not satisfied, the bit associated with the least-reliable LLR value (the LLR with the smallest absolute value) should be flipped. The location of this bit is found as

$$i_{\min} = \operatorname*{argmin}_{0 \leq k < R} |\mathbf{y}_k^{\phi,t}|, \tag{2.32}$$

Further, we set $\mathbf{x}_{i_{\min}}^{\phi,t} = \mathbf{x}_{i_{\min}}^{\phi,t} \oplus \gamma$ to satisfy the even-parity constraint.

In addition to the aforementioned nodes, low-complexity decoders of five other nodes are proposed in [7] to further increase the SC decoding speed. More specifically, Type-I node corresponds, to $\tilde{A}_{\phi,t} = \{R-2, R-1\}$; Type-II node, to $\tilde{A}_{\phi,t} = \{R-3, R-2, R-1\}$; Type-III node, to $\tilde{A}^c_{\phi,t} = \{0,1\}$; Type-IV node, to $\tilde{A}^c_{\phi,t} = \{0,1,2\}$; and Type-V node, to $\tilde{A}_{\phi,t} = \{R-5, R-3, R-2, R-1\}$.

Moreover, [8] provides general rules to fast decoding of polar codes by identifying G-REP and G-PC nodes, which generalize most of the aforementioned nodes. In particular, the G-REP node is identified when only the rightmost descendent contains information bits. Also, the G-PC node is recognized when the leftmost descendent is a Rate-0 node.

By implementing these fast decoders, the decoding tree can be pruned. This reduces the number of node visitations and, as a result, the decoding latency. As an example, Fig. 2.3(b) depicts the fast decoding tree of $P(32, 15)$ when the nodes in [5, 6] are considered. This will further be pruned to Fig. 2.3(c) if the nodes in [7, 8] are also taken into account. While 62 time steps are required to traverse the SC decoding tree in Fig. 2.3(a), traversing the decoding trees in Fig. 2.3(b) and Fig. 2.3(c) take 13 and 7 time steps, respectively.

2.6 Multi-kernel Polar Codes

Length-flexibility of polar codes can be achieved by puncturing or shortening the code. However, an alternative way to enable polar codes of any block length is incorporating different sized kernels in the code construction. In particular, [40] introduced square polarizing kernels of size larger than two and [36, 56] proposed the construction of polar codes with mixed kernels, called multi-kernel polar codes.

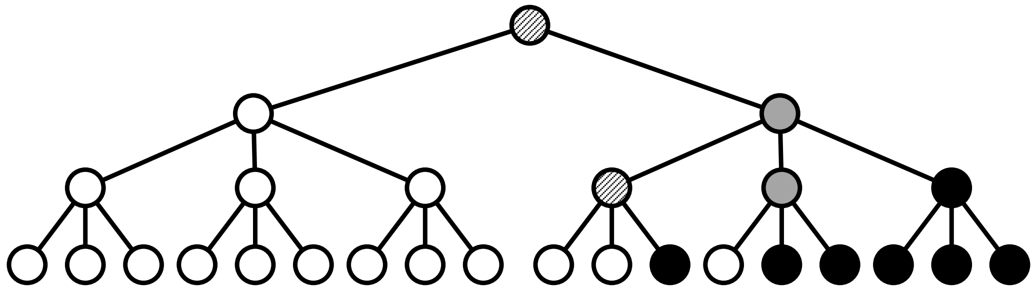


Fig. 2.5: A multi-kernel polar code $P(18, 6)$ with $\mathbf{k} = (2,\ 3,\ 3)$.

Among non-binary polarizing kernels, ternary (3×3) kernels received more attention [29, 30, 33–36] as they do not increase the decoding complexity significantly and have sufficiently high polarization exponent. Unlike the binary kernel $\mathbf{T_2}$, there are several choices for a ternary kernel [34, 35]. In particular, $\mathbf{T_3} = \left[\begin{smallmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right]$ and $\mathbf{T'_3} = \left[\begin{smallmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{smallmatrix}\right]$ were introduced in [36] and [40], respectively, and shown to be optimal for polarization with a polarization exponent of 0.42, slightly less than 0.5 which is the polarization exponent of $\mathbf{T_2}$.

A multi-kernel polar code, $P(N, k)$, that is constructed with mixed binary and ternary kernels has a length $N = 2^{n_1} \times 3^{n_2}$. In this case, the generator matrix $\mathbf{G}_N$ is constructed by n_1 times the Kronecker product of the binary kernel $\mathbf{T_2}$ and n_2 times the Kronecker product of ternary kernel with any order. Here, the number of the levels in the decoding tree is $S = n_1 + n_2$. As an example, a multi-kernel code of length 18 can be constructed with three different kernel orders, i.e., $\mathbf{k} = (2,\ 3,\ 3),\ (3,\ 3,\ 2), \text{or}\ (3,\ 2,\ 3)$.

To find the kernel order with most reliable set of information bits, [29] suggests searching over all possible kernel orders. For long polar codes the kernel order selection can be simplified by placing the ternary kernels at the first or last positions in the Kronecker product without any significant performance degradation [30]. However, the error rate performance of medium to high rate and low-rate polar codes can be improved by placing the ternary kernels at the first and last positions, respectively [30]. Fig. 2.5 depicts a multi-kernel polar code of length 18 and code rate of 1/3, where the ternary kernels are at last positions.

The bit channel reliability when the $\mathbf{T_3}$ and $\mathbf{T_3'}$ kernels are used, can be obtained by the GA method [46] as [30,33]

$$z_{3i-2}^{N/3} = \phi^{-1}\Big(1 - (1 - \phi(\phi^{-1}(1 - (1 - \phi(z_i^N))^2)))(1 - \phi(z_i^N))\Big), \tag{2.33}$$

$$z_{3i-1}^{N/3} = \phi^{-1}(1 - (1 - \phi(z_i^N))^2) + z_i^N, \tag{2.34}$$

$$z_{3i}^{N/3} = 2z_i^N. \tag{2.35}$$

Furthermore, the approximations of ϕ and ϕ^{-1} are obtained as [30]

$$\phi(x) = \begin{cases} e^{0.0564x^2 - 0485x}, & x < 0.8678 \\ e^{-0.4527x^{0.86} + 0.0218}, & \text{otherwise}, \end{cases} \tag{2.36}$$

$$\phi^{-1}(x) = \begin{cases} 4.3049(1 - \sqrt{1 + 0.9567\text{log}x}), & x > 0.6846 \\ (-2.209\text{log}x + 0.482)^1.163, & \text{otherwise}. \end{cases} \tag{2.37}$$

Also, any node (ϕ, t) with $t > 0$, at a ternary level of the decoding tree has $r = (3\phi, t-1)$, $c = (3\phi+1, t-1)$ and $l = (3\phi+2, t-1)$ as the left, center and right children, respectively.

The SC decoder for multi-kernel polar codes follows the similar steps as in the binary polar code, except for the LLR computations at the ternary levels. In particular, with the exception of the leaf nodes, each node at a ternary level computes LLR vectors of its left, center and right children using (2.38), (2.39) and (2.40), respectively, for $\mathbf{T}_3$.

$$y_k^l = y_k^{\phi,t} \boxplus y_{k+3^{t-1}}^{\phi,t} \boxplus y_{k+2\times 3^{t-1}}^{\phi,t}, \tag{2.38}$$

$$y_k^c = (-1)^{x_k^l} y_k^{\phi,t} + y_{k+3^{t-1}}^{\phi,t} \boxplus y_{k+2\times 3^{t-1}}^{\phi,t}, \tag{2.39}$$

$$y_k^r = (-1)^{x_k} y_{k+3^{t-1}}^{\phi,t} + (-1)^{x_k^l \oplus x_k^c} y_{k+2\times 3^{t-1}}^{\phi,t}, \tag{2.40}$$

In the case of $\mathbf{T}_3'$ the LLR for the left child is computed using (2.38). However, the optimal equations for LLR computation of center and right children are [57]

$$y_k^c = (-1)^{x_k^l} y_k^{\phi,t} \boxplus y_{k+2\times 3^{t-1}}^{\phi,t} + y_{k+3^{t-1}}^{\phi,t}, \tag{2.41}$$

$$y_k^r = (-1)^{x_k^l \oplus x_k^c} y_k^{\phi,t} + y_{k+2\times 3^{t-1}}^{\phi,t}. \tag{2.42}$$

Multi-kernel polar codes significantly improve the length flexibility of polar codes. For instance, consider up to nine levels in the code tree, where $n_1 + n_2 \leq 9$. Binary kernel polar codes, can attain only nine code lengths, while 54 different code lengths can be achieved using both binary and ternary kernels.

Chapter 3

Fast Successive-Cancellation List Decoding

The SCL decoder was proposed to improve the error-rate performance of polar codes. However, it parallelizes multiple SC decoders and suffers high decoding latency. Fast SCL decoders for Rate-0, REP, SPC, and Rate-1 nodes are proposed in [38, 39]. To further improve the SCL decoding speed, in this chapter, we propose the fast SCL decoders for the five special nodes (Type-I, Type-II, Type-III, Type-IV, and Type-V) [7] identified in the decoding tree of polar codes [3, 4]. Next, we study how the fast SCL decoders can be adopted to be used in the case of distributed parity-check-aided SCL decoding. Moreover, we compare the decoding latencies of the proposed decoders and the existing ones. Finally, simulation results are provided to verify the performance of the proposed decoders.

Before presenting our fast SCL decoders for the five nodes, we first examine the contribution of a node (ϕ, t) to the path metric of a codeword. Observe that the leaf nodes corresponding to the node (ϕ, t) are the nodes $(2^t\phi + k, 0)$, where $0 \leq k < 2^t$. Consequently, using (2.27), the contribution of the node (ϕ, t) to the path metric, denoted by $\mathrm{PM}^{\phi,t}$, is given by

$$\mathrm{PM}^{\phi,t} = \sum_{k=2^t\phi}^{2^t(\phi+1)-1} \ln\Big(1 + e^{-(1-2\hat{u}_k)y^{k,0}}\Big). \tag{3.1}$$

Our proposed decoders, similar to the list decoders of the Rate-0, Rate-1, REP and SPC nodes [38], compute $\mathrm{PM}^{\phi,t}$ without traversing the whole subtree corresponding to a node. To this end, for the path metric computation we use an important result introduced in [38, Theorem 1], which specifically proves that[1]

$$\mathrm{PM}^{\phi,t} = \sum_{k=0}^{2^t-1} \ln\left(1 + e^{-(1-2x_k^{\phi,t})y_k^{\phi,t}}\right). \tag{3.2}$$

Another result, that will be used extensively in our proposed decoders to simplify the calculations, is the following identity.

$$\ln\left(1+e^{a}\right) - \ln\left(1+e^{-a}\right) = a. \tag{3.3}$$

3.1 Proposed Fast SCL decoders

In the following, we present fast decoders for the Type-I, Type-II, Type-III, Type-IV and Type-V nodes. For better readability, we use $R = 2^t$ to denote the node length and drop the indexes ϕ and t from the notation. Further, we add l, where $0 \leq l < L$, in the notation to indicate that the calculations pertain to the path l.

3.1.1 Type-I Node

For a Type-I node, $\mathbf{x}^l = \{x_{R-2}, x_{R-1}, \cdots, x_{R-2}, x_{R-1}\}$ [7]. As such, $\mathbf{x}^l$ equals only one of the four codewords: $C_0 = \{0, 0, \cdots, 0, 0\}$, $C_1 = \{0, 1, \cdots, 0, 1\}$, $C_2 = \{1, 0, \cdots, 1, 0\}$, and $C_3 = \{1, 1, \cdots, 1, 1\}$.

In the proposed decoder, we compute the PM contribution of the Type-I node corresponding to each codeword for each list. In particular, the PMs corresponding to

[1]Although [38] presented the equivalence of PMs for a Rate-1 node, the same result is valid for any node and can be proved using a similar approach.

the four codewords for the list l, $0 \leq l < L$, are

$$\mathrm{PM}_0^l = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{-\mathbf{y}_{2k}^l}\right) + \ln\left(1+e^{-\mathbf{y}_{2k+1}^l}\right), \tag{3.4}$$

$$\mathrm{PM}_1^l = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{-\mathbf{y}_{2k}^l}\right) + \ln\left(1+e^{+\mathbf{y}_{2k+1}^l}\right), \tag{3.5}$$

$$\mathrm{PM}_2^l = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{+\mathbf{y}_{2k}^l}\right) + \ln\left(1+e^{-\mathbf{y}_{2k+1}^l}\right), \tag{3.6}$$

$$\mathrm{PM}_3^l = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{+\mathbf{y}_{2k}^l}\right) + \ln\left(1+e^{+\mathbf{y}_{2k+1}^l}\right). \tag{3.7}$$

Using (3.3), the above calculations can be simplified as

$$\mathrm{PM}_0^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1+e^{-\mathbf{y}_k^l}\right), \tag{3.8}$$

$$\mathrm{PM}_1^l = \mathrm{PM}_{00}^l + \varsigma_1, \tag{3.9}$$

$$\mathrm{PM}_2^l = \mathrm{PM}_{00}^l + \varsigma_0, \tag{3.10}$$

$$\mathrm{PM}_3^l = \mathrm{PM}_{00}^l + \varsigma_0 + \varsigma_1, \tag{3.11}$$

where $\varsigma_i = \sum_{k=0}^{R/2-1} \mathbf{y}_{2k+i}^l$ for $i = 0, 1$.

As a result of these calculations, a total of $4L$ path metrics are computed. Amongst them, only the smallest L ones are retained, and the corresponding codewords are assigned to $\mathbf{x}^l$.

3.1.2 Type-II Node

For a Type-II node, $\mathbf{x}^l = \{x_{R-4}, x_{R-3}, x_{R-2}, x_{R-1}, \cdots, x_{R-4}, x_{R-3}, x_{R-2}, x_{R-1}\}$, where $x_{R-4} = x_{R-3} \oplus x_{R-2} \oplus x_{R-1}$ [7]. Hence, $\mathbf{x}^l$ can only be one of the eight codewords: $C_0 = \{0,0,0,0\cdots,0,0,0,0\}$, $C_1 = \{1,0,0,1,\cdots,1,0,0,1\}$, $C_2 = \{1,0,1,0,\cdots,1,0,1,0\}$, $C_3 = \{0,0,1,1,\cdots,0,0,1,1\}$, $C_4 = \{1,1,0,0,\cdots,1,1,0,0\}$, $C_5 = \{0,1,0,1,\cdots,0,1,0,1\}$, $C_6 = \{0,1,1,0\cdots,0,1,1,0\}$, and $C_7 = \{1,1,1,1\cdots,1,1,1,1\}$.

Following the approach used in the Type-I list decoder, we compute

$$\mathrm{PM}_0^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-\mathbf{y}_k^l}\right), \tag{3.12}$$

$$\mathrm{PM}_1^l = \mathrm{PM}_0^l + \eta_0 + \eta_3, \tag{3.13}$$

$$\mathrm{PM}_2^l = \mathrm{PM}_0^l + \eta_0 + \eta_2, \tag{3.14}$$

$$\mathrm{PM}_3^l = \mathrm{PM}_0^l + \eta_2 + \eta_3, \tag{3.15}$$

$$\mathrm{PM}_4^l = \mathrm{PM}_0^l + \eta_0 + \eta_1, \tag{3.16}$$

$$\mathrm{PM}_5^l = \mathrm{PM}_0^l + \eta_1 + \eta_3, \tag{3.17}$$

$$\mathrm{PM}_6^l = \mathrm{PM}_0^l + \eta_1 + \eta_2, \tag{3.18}$$

$$\mathrm{PM}_7^l = \mathrm{PM}_0^l + \eta_0 + \eta_1 + \eta_2 + \eta_3. \tag{3.19}$$

where $\eta_i = \sum_{k=0}^{R/4-1} \mathbf{y}_{4k+i}^l$ for $0 \leq i < 4$.

Afterwards, L PMs are retained amongst the $8L$ computed PMs, and the corresponding codewords are assigned to $\mathbf{x}^l$.

3.1.3 Type-III Node

A Type-III node corresponds to $\hat{A}^c = \{0, 1\}$; i.e., there are only two frozen bits in the node [7]. Therefore, $\mathbf{x}^l$ can be one of the $2^{R-2} = 2^{2^t-2}$ valid Type-III node codewords. Since the size of a Type-III node can be large, it is impractical to compute a total of $2^{R-2}L$ path metrics and choose the best L ones from them. Hence, we follow the approach of generating candidate codewords used in [37, 39].

For each path l, we first find the ML codeword and compute its corresponding PM. Since the even-indexed and odd-indexed bits of a Type-III codeword constitute two separate SPC codes [7], we compute the ML codeword by using Wagner decoding [55]. In particular, we set $x_k^l = H(y_k^l)$, where $H(y)$ is defined in (2.26), and $0 \leq k < R$. We then find the location of the least-reliable even-indexed and odd-indexed LLR values as

$$(e_l, o_l) = (\underset{0 \leq k < R/2}{\operatorname{argmin}} |y_{2k}^l|, \underset{0 \leq k < R/2}{\operatorname{argmin}} |y_{2k+1}^l|), \tag{3.20}$$

and compute the parities of both SPC codes as

$$(\gamma_e^l, \gamma_o^l) = \left(\bigoplus_{k=0}^{R/2-1} x_{2k}^l, \bigoplus_{k=0}^{R/2-1} x_{2k+1}^l \right). \tag{3.21}$$

Further, to satisfy the even-parity constraint we set

$$x_{2e_l}^l = x_{2e_l}^l \oplus \gamma_e^l \tag{3.22}$$

and

$$x_{2o_l+1}^l = x_{2o_l+1}^l \oplus \gamma_o^l \tag{3.23}$$

Lastly, we compute the PM corresponding to the ML codeword as

$$\mathrm{PM}_{\mathrm{ML}}^l = \Delta^l + \gamma_e^l |y_{2e_l}^l| + \gamma_o^l |y_{2o_l+1}^l|, \tag{3.24}$$

where $\Delta^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-|y_k^l|}\right)$.

We then define modified LLRs, α_i^l's, as $\alpha_{2k}^l = |y_{2k}^l| + (1 - 2\gamma_e^l)|y_{2e_l}^l|$, and $\alpha_{2k+1}^l = |y_{2k+1}^l| + (1 - 2\gamma_o^l)|y_{2o_l+1}^l|$ for $0 \leq k < R/2$. Next, we sort the modified LLRs (excluding $\alpha_{2e_l}^l$ and $\alpha_{2o_l+1}^l$) in ascending order and denote the sorted indexes by $(i)_l$, where $0 \leq i < R-2$. Mathematically, $\alpha_{(k)_l}^l \leq \alpha_{(k+1)_l}^l$ for $0 \leq k < R-3$.

After computing the sorted indexes for each path, we start generating candidate codewords by flipping bits of the ML codeword. In particular, starting from $k = 0$, we generate two codewords corresponding to the bit $(k)_l$ being 0 or 1. In order to satisfy the even parity condition, the bits with indexes $2e^l$ and $2o^l+1$ are modified accordingly. The PM corresponding to the generated codewords are calculated as

$$\mathrm{PM}_{(k)_l}^l = \begin{cases} \mathrm{PM}_{(k-1)_l}^l, & \text{if } x_{(k)_l}^l = H(y_{(k)_l}^l); \\ \mathrm{PM}_{(k-1)_l}^l + |y_{(k)_l}^l| + \beta_{(k)_l}, & \text{otherwise,} \end{cases} \tag{3.25}$$

where $\mathrm{PM}_{(-1)_l}^l = \mathrm{PM}_{\mathrm{ML}}^l$, $\beta_{(k)_l} = (1 - 2\gamma_{e,k-1}^l)|y_{2e_l}^l|$ if $(k)_l$ is even, and $\beta_{(k)_l} = (1 - 2\gamma_{o,k-1}^l)|y_{2o_l+1}^l|$ otherwise. The bit $\gamma_{e,k}^l$ equals $\gamma_{e,k-1}^l$ when $(k)_l$ is odd. When $(k)_l$ is even then

$$\gamma_{e,k}^l = \begin{cases} \gamma_{e,k-1}^l, & \text{if } x_{(k)_l}^l = H(y_{(k)_l}^l); \\ 1 - \gamma_{e,k-1}^l, & \text{otherwise.} \end{cases} \tag{3.26}$$

Likewise, $\gamma_{o,k}^l = \gamma_{o,k-1}^l$ when $(k)_l$ is even. When $(k)_l$ is odd, $\gamma_{o,k}^l$ is updated as likewise, $\gamma_{o,k}^l = \gamma_{o,k-1}^l$ when $(k)_l$ is even. When $(k)_l$ is odd, $\gamma_{o,k}^l$ is updated as

$$c\gamma_{o,k}^l = \begin{cases} \gamma_{o,k-1}^l, & \text{if } x_{(k)_l}^l {=} H(y_{(k)_l}^l); \\ 1 - \gamma_{o,k-1}^l, & \text{otherwise.} \end{cases} \tag{3.27}$$

Here, $\gamma_{e,-1}^l = \gamma_e^l$, and $\gamma_{o,-1}^l = \gamma_o^l$.

As a result of the aforementioned operations, a total of $2L$ PMs are computed for each k. Amongst them, only the lowest L ones are retained. We continue this process of creating $2L$ codewords from the existing L ones and retaining only the L best ones in a successive manner as k varies from 0 to $\min\{L-2, R-3\}$. After that, we return the L surviving paths and their PMs. We do not consider all the codewords; i.e., we do not vary k from 0 to $R-3$, when $L < R-1$, because Theorem 3.1 asserts that the extra codewords created after $k = L-2$ are not included in the surviving paths.

Theorem 3.1. *In the proposed list decoder of the Type-III node, the codewords that have $x_{(k)_l}^l \neq H(y_{(k)_l}^l)$ for $k \geq L-1$ are not amongst the surviving codewords.*

Proof. We prove this theorem by contradiction. Suppose the codeword with $x_{(L-1)_l}^l \neq H(y_{(L-1)_l}^l)$ is among the surviving codewords. Consequently, the PM corresponding to such a codeword is one of the smallest L ones. Note that the minimum possible value of the corresponding PM is $\text{PM}_{\text{ML}}^l + \alpha_{(L-1)_l}^l$, which represents the case that all the bits of the generated codeword and the ML codewords are the same except the bit $(L-1)_l$ and the corresponding parity bit.

Now consider the codewords corresponding to $x_{(k)_l}^l = H(y_{(k)_l}^l)$ and $x_{(k)_l}^l = 1 - H(y_{(k)_l}^l)$ for $0 \leq k < L-1$. Amongst them, we consider the following L codewords. One of them is the ML codeword with the path metric of PM_{ML}^l. The other $L-1$ codewords are the ones that differ from the ML codeword only in two locations: the $(k)_l$ bit and the corresponding (even/odd) parity bit. The PM of such codeword is $\text{PM}_{\text{ML}}^l + \alpha_{(k)_l}^l$ for $0 \leq k < L-1$. Using the fact that $\alpha_i^l \geq 0$ and $\alpha_{(k)_l}^l \leq \alpha_{(L-1)_l}^l$, the PMs of the considered L codewords is less than or equal to $\text{PM}_{\text{ML}}^l + \alpha_{(L-1)_l}^L$. Consequently, the PM of the codeword corresponding to $x_{(L-1)_l}^l \neq H(y_{(L-1)_l}^l)$ is not amongst the best

L ones. Therefore, we do not need to consider multiple codewords corresponding to $x^l_{(L-1)_l}$ being equal to 0 or 1. Rather, we just set $x^l_{(L-1)_l} = H(y^l_{(L-1)_l})$.

Similar assertions can be made for $x^l_{(k)_l}$ for $k \geq L$. $\square$

3.1.4 Type-IV Node

The codeword in a Type-IV node satisfies the following relationship [7].

$$\sum_{i=0}^{R/4-1} x_{4i} = \sum_{i=0}^{R/4-1} x_{4i+1} = \sum_{i=0}^{R/4-1} x_{4i+2} = \sum_{i=0}^{R/4-1} x_{4i+3} = z, \tag{3.28}$$

where z can be 0 or 1.

In the proposed list decoder of a Type-IV node, we first compute the ML codewords for each list corresponding to $z = 0$ and $z = 1$ using a Wagner decoder. In particular, we compute the i_0^*, i_1^*, i_2^*, and i_3^* as

$$i_j^* = \underset{0 \leq k < R/4}{\operatorname{argmin}} |y^l_{4k+j}|, \tag{3.29}$$

where $j = 0, 1, 2, 3$.

Then we set $\mathbf{x}^l = H(\mathbf{y}^l)$ and compute γ_j^l as

$$\gamma_j^l = \bigoplus_{k=0}^{R/4-1} x^l_{4k+j}, \tag{3.30}$$

where $j = 0, 1, 2, 3$. Afterwards, we generate two ML codewords corresponding to $z = 0$ and $z = 1$ by setting $x^l_{4i_j^*+j} = x^l_{4i_j^*+j} \oplus \gamma_j^l \oplus z$. Their corresponding PMs are

$$\mathrm{PM}^l_{\mathrm{ML},z} = \Delta^l + \sum_{j=0}^{3} (\gamma_j^l \oplus z) |y^l_{4i_j^*+j}|, \tag{3.31}$$

where $\Delta^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-|y_k^l|}\right)$.

As a result, we compute $2L$ codewords and their corresponding PMs. Amongst them, we only retain those L codewords that have the smallest PM values. Afterwards, using z of the retained codewords, we update γ_j^l as $\gamma_j^l = \gamma_j^l \oplus z^l$.

Similar to the Type-III node decoder, we then define α_i's as $\alpha^l_{4k+j} = |y^l_{4k+j}| + (1 - 2\gamma_j^l)|y^l_{4i_j^*+j}|$ for $0 \leq k < R/4$, and $0 \leq j \leq 3$. Next, excluding $\alpha^l_{4i_j^*+j}$, we sort α_i^l's in an ascending order and denote the sorted indexes by $(i)_l$, where $0 \leq i < R-4$. Thus, $\alpha^l_{(k)_l} \leq \alpha^l_{(k+1)_l}$ for $0 \leq k < R-5$.

Starting from $k=0$, we then compute different codewords and their corresponding PMs by considering $x_{(k)_l} = 0$ and $x_{(k)_l} = 1$ and setting appropriate values to the $x^l_{4i^*_j+j}$, where $j = [\![(k)_l]\!]_2$ is the remainder after division of $(k)_l$ by $2^2 = 4$. The PMs are computed as

$$\mathrm{PM}^l_{(k)_l} = \begin{cases} \mathrm{PM}^l_{(k-1)_l}, & \text{if } \mathbf{x}^l_{(k)_l} = H(y^l_{(k)_l}); \\ \mathrm{PM}^l_{(k-1)_l} + |\mathbf{y}^l_{(k)_l}| + \beta_{(k)_l}, & \text{otherwise,} \end{cases} \tag{3.32}$$

where $\mathrm{PM}^l_{(-1)_l} = \mathrm{PM}^l_{\mathrm{ML},z}$ and $\beta_{(k)_l} = (1 - 2\gamma^l_{j,k-1})|\mathbf{y}^l_{4i^*_j+j}|$. Here, $\gamma^l_{j,k}$ is computed recursively as

$$\gamma^l_{j,k} = \begin{cases} \gamma^l_{j,k-1}, & \text{if } \mathbf{x}_{(k)_l} = H(\mathbf{y}_{(k)_l}); \\ 1 - \gamma^l_{j,k-1}, & \text{otherwise,} \end{cases} \tag{3.33}$$

with $\gamma^l_{j,-1} = \gamma^l_j$. Also, $\gamma^l_{i,k} = \gamma^l_{i,k-1}$ for $0 \leq i < 4$ and $i \neq j$, and $\gamma^l_{i,-1} = \gamma^l_i$.

The aforementioned procedure produces a total of $2L$ codewords for each k. We then save only those codewords and their corresponding PMs that have the lowest PM values. We continue this process till k reaches $\min\{L-2, R-5\}$. If $L < R-3$, we simply set $x^l_{(k)_l} = H(y^l_{(k)_l})$ for $L-2 < k \leq R-5$ as the codeword with $x^l_{(k)_l} \neq H(y^l_{(k)_l})$ has a larger PM and is not amongst the surviving codewords[2] .

3.1.5 Type-V Node

The list decoder for a Type-V node is quite similar to that of the Type-I and Type-II nodes. In particular,

$$\mathbf{x}^l = \mathbf{m} \begin{bmatrix} \mathbf{G}_0 & \cdots & \mathbf{G}_0 \end{bmatrix}, \tag{3.34}$$

where $\mathbf{m}$ is a binary row vector of length 4, and

$$\mathbf{G}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{3.35}$$

[2]The proof of this assertion is omitted because it is quite similar to the proof of Theorem 3.1.

Similar to the proposed decoders for Type-I and Type-II nodes, we compute 16 PMs corresponding to different values of $\mathbf{m}$[3] as

$$\text{PM}_0^l = \text{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-\mathbf{y}_k^l}\right), \tag{3.36}$$

$$\text{PM}_1^l = \text{PM}_0^l + \sigma_1 + \sigma_2 + \sigma_4 + \sigma_7, \tag{3.37}$$

$$\text{PM}_2^l = \text{PM}_0^l + \sigma_0 + \sigma_2 + \sigma_4 + \sigma_6, \tag{3.38}$$

$$\text{PM}_3^l = \text{PM}_0^l + \sigma_0 + \sigma_1 + \sigma_6 + \sigma_7, \tag{3.39}$$

$$\text{PM}_4^l = \text{PM}_0^l + \sigma_0 + \sigma_1 + \sigma_4 + \sigma_5, \tag{3.40}$$

$$\text{PM}_5^l = \text{PM}_0^l + \sigma_0 + \sigma_2 + \sigma_5 + \sigma_7, \tag{3.41}$$

$$\text{PM}_6^l = \text{PM}_0^l + \sigma_1 + \sigma_2 + \sigma_5 + \sigma_6, \tag{3.42}$$

$$\text{PM}_7^l = \text{PM}_0^l + \sigma_4 + \sigma_5 + \sigma_6 + \sigma_7, \tag{3.43}$$

$$\text{PM}_8^l = \text{PM}_0^l + \sigma_0 + \sigma_1 + \sigma_2 + \sigma_3, \tag{3.44}$$

$$\text{PM}_9^l = \text{PM}_0^l + \sigma_0 + \sigma_3 + \sigma_4 + \sigma_7, \tag{3.45}$$

$$\text{PM}_{10}^l = \text{PM}_0^l + \sigma_1 + \sigma_3 + \sigma_4 + \sigma_6, \tag{3.46}$$

$$\text{PM}_{11}^l = \text{PM}_0^l + \sigma_2 + \sigma_3 + \sigma_6 + \sigma_7, \tag{3.47}$$

$$\text{PM}_{12}^l = \text{PM}_0^l + \sigma_2 + \sigma_3 + \sigma_4 + \sigma_5, e \tag{3.48}$$

$$\text{PM}_{13}^l = \text{PM}_0^l + \sigma_1 + \sigma_3 + \sigma_5 + \sigma_7, \tag{3.49}$$

$$\text{PM}_{14}^l = \text{PM}_0^l + \sigma_0 + \sigma_3 + \sigma_5 + \sigma_6, \tag{3.50}$$

$$\text{PM}_{15}^l = \text{PM}_0^l + \sum_{i=1}^{7} \sigma_i, \tag{3.51}$$

where $\sigma_i = \sum_{k=0}^{R/8-1} \mathbf{y}_{8k+i}^l$ for $0 \leq i < 8$. Amongst the $16L$ computed path metrics, only the smallest L ones are kept, and their corresponding codewords are assigned to $\mathbf{x}^l$.

Remark 3.1. *In the aforementioned list decoders, we did not provide calculations for $\hat{\mathbf{u}}_k^l$, where $2^t\phi \leq k < 2^t(\phi + 1)$ for a node (ϕ, t). For systematic polar codes, we do not compute $\hat{\mathbf{u}}^l$ as the information bits appear directly in $\mathbf{x}$ [41]. For non-systematic*

[3]Here PM_0 corresponds to the codeword $\mathbf{x}$ when $\mathbf{m} = \{0, 0, 0, 0\}$, PM_1 corresponds to the codeword $\mathbf{x}$ when $\mathbf{m} = \{0, 0, 0, 1\}$, and so on.

polar codes, we can use $\hat{\mathbf{u}}^{\phi,t} = \mathbf{x}^{\phi,t}\mathbf{G}$ *to find the information bits corresponding to the codeword* $\mathbf{x}^{\phi,t}$*. Note that the above operation, in contrast to the conventional list decoding, involves only bit operations, which can be carried out expeditiously with very few hardware resources.*

For the Type-I, Type-II and Type-V nodes, we can further reduce the hardware complexity and decoding latency by using the special structure of $\mathbf{x}^{\phi,t}$ *and the frozen-bit pattern. For example, for a Type-I node,* $\hat{\mathbf{u}}$ *can be calculated as follows:* $\hat{u}_k = 0$ *for* $2^t\phi \leq k < 2^t(\phi+1)-2$, $\hat{u}_{2^t(\phi+1)-2} = x^{\phi,t}_{2^t-2} \oplus x^{\phi,t}_{2^t-1}$*, and* $\hat{u}_{2^t(\phi+1)-1} = x^{\phi,t}_{2^t-1}$.

Remark 3.2. *The computational and hardware complexity of PM calculations can be reduced using [9]*

$$\ln(1+e^a) \approx \begin{cases} a, & \text{if } a > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{3.52}$$

For example, using (3.52)*, we can compute* PM_0^l *for a Type-I node as*

$$\mathrm{PM}_0^l \approx \mathrm{PM}^l + \sum_{k:y_k^l<0} y_k^l. \tag{3.53}$$

3.2 A Case Study: Fast SCL Decoding in the Distributed Parity-check Aided Polar Codes

As mentioned earlier, the error correction performance of the SCL decoder can be improved with serial concatenation of a CRC after the non-frozen bits [18]. The idea of using a CRC is then extended to a more general case by distributing parity-check (PC) bits among the non-frozen bits [58, 59]. It was shown in [58, 59] that the distributed-PC-aided SCL decoder can outperform the CRC-aided SCL decoder since the scattered PC bits enables the SCL decoder to detect and prune error paths more effectively and timely. [58, 59] proposed two different methods for distributing the PC bits. In both methods each PC bit is the module 2 sum of a number of its preceding information bits and is estimated as

$$\hat{u}_i = (\sum_{j\in\mathcal{P}_i} \hat{u}_j) \bmod 2, \tag{3.54}$$

where $\mathcal{P}_i$ is the set of information bit indices corresponding to the ith PC bit. Note that all $j \in \mathcal{P}_i$ are smaller than the index of their corresponding PC bit, i.e., $j < i$.

In this section, we study fast SCL decoding for the distributed-PC-aided polar codes. In particular, in the distributed-PC-aided polar codes, the PC bits appear in the node patterns along with information and frozen bits and impose additional constraints. Thus, to see the impact of PC bits we consider different patterns (i.e., the REP, SPC, and Type-III patterns) and adapt their existing fast SCL decoders to be used in the case of distributed-PC-aided SCL decoding.

3.2.1 PC-REP Node

A PC-REP node is an REP node with the rightmost bit being a PC bit rather than an information bit. Thus, its codeword is $\mathbf{x}^l = \{\hat{u}^l_{\mathrm{PC}}, \hat{u}^l_{\mathrm{PC}}, ..., \hat{u}^l_{\mathrm{PC}}\}$, where $\hat{u}^l_{\mathrm{PC}}$ is the estimation of the PC bit for each list obtained by (3.54). Also, the PM contribution of this node is computed as

$$\mathrm{PM}^l = \sum_{k=0}^{R-1} \ln\big(1 + e^{-(1-2\hat{u}^l_{\mathrm{PC}})y^l_k}\big). \tag{3.55}$$

3.2.2 PC-SPC Node

A PC-SPC node is an SPC node except that the leftmost bit is a PC bit rather than being a frozen bit. Thus, the parity-check constraint is imposed by the estimated value of the PC bit. If $\hat{u}^l_{\mathrm{PC}} = 0$, the PC-SPC node is a SPC code with an even parity-check constraint. Otherwise, it is a SPC code with an odd parity-check constraint. As such, the fast SCL decoder for the SPC node [39] can be adopted to decode this node. In particular, the decoder first computes the $\hat{u}^l_{\mathrm{PC}}$ to identify the parity-check constraint for each path. The rest of the decoding process is identical to the decoding of the SPC node in [39].

3.2.3 PC-Type-III Node

In a Type-III node the first two bits are frozen bits. We observed that two different patterns can be described as a PC-Type-III node. In particular, in one of the patterns the first bit is frozen bit and the second bit is a PC bit. The corresponding codeword to this pattern satisfies the following constraint.

$$\sum_{i=0}^{R/2-1} x_{2i}^l = \sum_{i=0}^{R/2-1} x_{2i+1}^l = \hat{u}_{\mathrm{PC}}^l. \tag{3.56}$$

Here, the even-indexed and odd-indexed bits in the codeword form two separate SPC codes, where their parity check constraint is imposed by the value of $\hat{u}_{\mathrm{PC}}^l$.

In the second PC-Type-III pattern, the first and second bits are both PC bits. In this case, a valid codeword satisfies the following relations:

$$\sum_{i=0}^{R/2-1} x_{2i}^l = \hat{u}_{\mathrm{PC1}}^l \oplus \hat{u}_{\mathrm{PC2}}^l, \tag{3.57}$$

$$\sum_{i=0}^{R/2-1} x_{2i+1}^l = \hat{u}_{\mathrm{PC2}}^l, \tag{3.58}$$

where the subscripts PC1 and PC2 refer to the index of the first and second PC bits in this node, respectively. In particular, the even-indexed and odd-indexed bits in the codeword from two separate SPC codes.

Both PC-Type-III node patterns can be decoded using the proposed Type-III fast SCL decoder in Section 3.1.3, except that the (3.21) must be modified as

$$(\gamma_e^l, \gamma_o^l) = \left(\bigoplus_{k=0}^{R/2-1} x_{2k}^l \oplus \gamma_1^l, \bigoplus_{k=0}^{R/2-1} x_{2k+1}^l \oplus \gamma_2^l \right) \tag{3.59}$$

where for the first pattern $\gamma_1^l = \gamma_2^l = \hat{u}_{\mathrm{PC1}}^l$ and for the second pattern $\gamma_1^l = \hat{u}_{\mathrm{PC1}}^l \oplus \hat{u}_{\mathrm{PC2}}^l$ and $\gamma_2^l = \hat{u}_{\mathrm{PC2}}^l$.

3.3 Decoding Latency

In this section, we compare the decoding latency of our proposed fast SCL decoders with that of the existing ones [39] (referred to as FSCL decoders hereafter). To compare the decoding latencies we compute the required number of time steps to decode different nodes with the existing decoders and our proposed ones, under the following assumptions. First, we assume there are no resource limitations so that all the parallelizable instructions are performed in one time step [38, 39]. Second, addition/subtraction of real numbers and the check-node operation is assumed to consume one time step. Third, hard decisions on LLRs and bit operations are carried out instantaneously [5, 7, 9, 38, 39]. Fourth, Wagner decoding can be performed in a single time step. Last, the decoder duplicates all L paths, sorts the corresponding $2L$ PMs and selects the smallest L ones during a single time step [38, 39].

Note that [39] presented fast decoders for only Rate-0, REP, SPC and Rate-1 nodes, and the remaining special nodes are decoded using these nodes. In the following, we first present the decoding latency of the Rate-0, REP, SPC, and Rate-1 nodes as these calculations will be used when computing the decoding latencies of the FSCL decoders for the remaining nodes.

The FSCL decoder takes one time step to decode a Rate-0 node as the computation of the PM for active paths consumes a single time step. The FSCL decoder for a REP node duplicates each path and computes their corresponding PMs in a single time step. In the next time step, it sorts the PMs and selects the best L paths. As such, the decoding latency of the FSCL REP node decoder is two time steps. The Rate-1 node FSCL decoder computes the PM of the ML codeword in one time step and needs $\min\{L-1, R\}$ time steps to do the path and PM updates of the remaining codewords. As such, it consumes $1+\min\{L-1, R\}$ time steps to decode a Rate-1 node. Similarly, the FSCL decoder for an SPC node takes one time step to find the ML codeword and its corresponding PM and $\min\{L-1, R-1\}$ time steps for the path and PM updates of the remaining codewords. Thus, the decoding latency of the FSCL SPC node decoder is $1+\min\{L-1, R-1\}$ time steps.

For Type-I, Type-II and Type-V nodes, our proposed decoders consume one time step to compute the PMs corresponding to all codewords. Following the first and last assumptions from the beginning of this section, our decoders require two, three and four time steps to sort $4L$, $8L$ and $16L$ PMs, respectively. As such, the decoding latencies of the Type-I, Type-II and Type-V decoders are three, four and five time steps, respectively.

On the other hand, if the FSCL decoders are used to decode Type-I, Type-II and

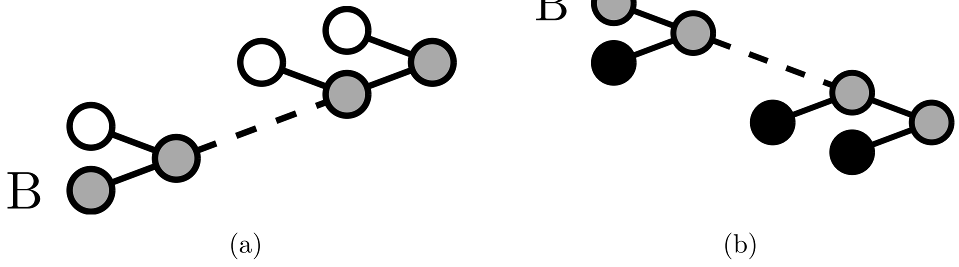


Fig. 3.1: Decoding trees of the proposed nodes in [7] correspond to either decoding tree (a) or decoding tree (b).

Type-V nodes then the decoding tree of these nodes corresponds to the decoding tree of Fig. 3.1(a). In particular, the left child of each node is a Rate-0 node, whereas node B is a Rate-1 node of size two for the Type-I node, an SPC node of size four for the Type-II, and a REP-SPC node of size eight for the Type-V node [7].

In general, the LLR computation for each child takes one time step. However, when the left children are Rate-0 nodes, the LLRs of both children can be calculated simultaneously in one time step. That results in a decoding latency of $\log_2(R/R_B)$ time steps to traverse to the node B, where R_B is the size of node B. Using the aforementioned decoding delay expressions, node B can be decoded in $\min\{L-1, 2\}+1$, $\min\{L, 4\}$, and $\min\{L, 4\}+4$ time steps for Type-I, Type-II and Type-V nodes, respectively. Using these calculations, the decoding latencies of these nodes can be computed and are given in Table 3.1.

The proposed decoder of a Type-III node first computes the ML codeword and its corresponding PM in one time step. Afterwards, it generates $2L$ codewords and select the best L ones for $\min\{L-1, R-2\}$ bits, which results in a decoding delay of $1+\min\{L-1, R-2\}$ time steps. Likewise, the decoding latency of the proposed Type-IV decoder can be shown to be $1+\min\{L-1, R-4\}$ time steps.

With the FSCL decoders, the decoding tree of a Type-III or Type-IV node corresponds to the decoding tree of Fig. 3.1(b). In particular, the right child of each node is a Rate-1 node. The node B is a Rate-0 node of size two, and a REP node of size four for the Type-III and Type-IV nodes, respectively. Decoding delays of $2\log_2(R/R_B)-1$ and $2\log_2(R/R_B)$ time steps are required to traverse to the level

Table 3.1: Decoding latencies of the FSCL and proposed fast SCL decoders.

Node	**FSCL [39]**	**Proposed fast SCL**
Type-I	$t + \min\{L-1, 2\}$	3
Type-II	$t - 2 + \min\{L, 4\}$	4
Type-III	$3t - 4 + \sum_{i=1}^{t-1} \min\{L-1, 2^i\}$	$\min\{L, R-1\}$
Type-IV	$3t - 4 + \sum_{i=2}^{t-1} \min\{L-1, 2^i\}$	$\min\{L, R-3\}$
Type-V	$t + 1 + \min\{L, 4\}$	5

of node B in the Type-III and Type-IV node, respectively. Using the decoding delay expression for the Rate-0, REP and Rate-1 nodes, the decoding latencies of the fast SCL decoders can be computed and are given in Table 3.1.

Observe that the decoding latencies of the proposed fast SCL decoders are less than those of the FSCL decoders. The latency improvement is more pronounced for Type-III and Type-IV nodes. For example, the proposed decoder will require four time steps to decode a Type-III node, whereas the FSCL decoder will consume a total of 22 time steps to decode the node when $L = 4$ and $R = 2^t = 2^5$.

Note that the aforementioned decoding delay calculations are valid when $L > 1$. This is because no PM computation and path duplication are required for $L = 1$.

3.4 Numerical Results

In this section, we compare the BER and block-error-rate (BLER) performances of the proposed fast SCL decoders with those of the existing decoders[4] . We used a 16-bit CRC defined by the generator polynomial $x^{16} + x^{15} + x^{12} + x^7 + x^6 + x^4 + x^3 + 1$ (0xC86C). Systematic polar codes [41] with random BPSK-modulated codewords were transmitted through the AWGN channel. Furthermore, the simple GA method [51] was used to design the polar codes.

[4]Performances of different decoders are compared under identical noise values and codewords in our simulations.

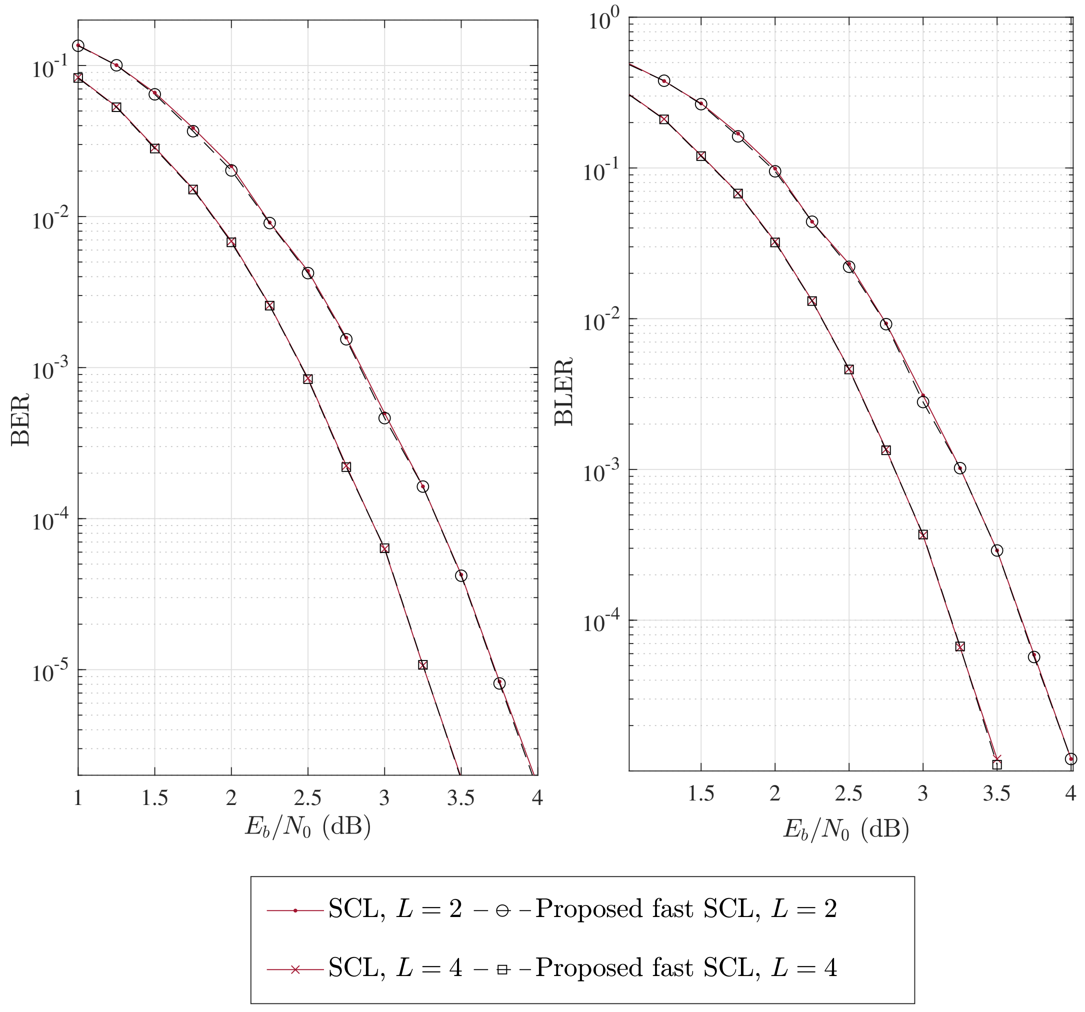


Fig. 3.2: BER and BLER performances of the SCL [9] and proposed fast SCL decoders for a $P(1024, 256)$.

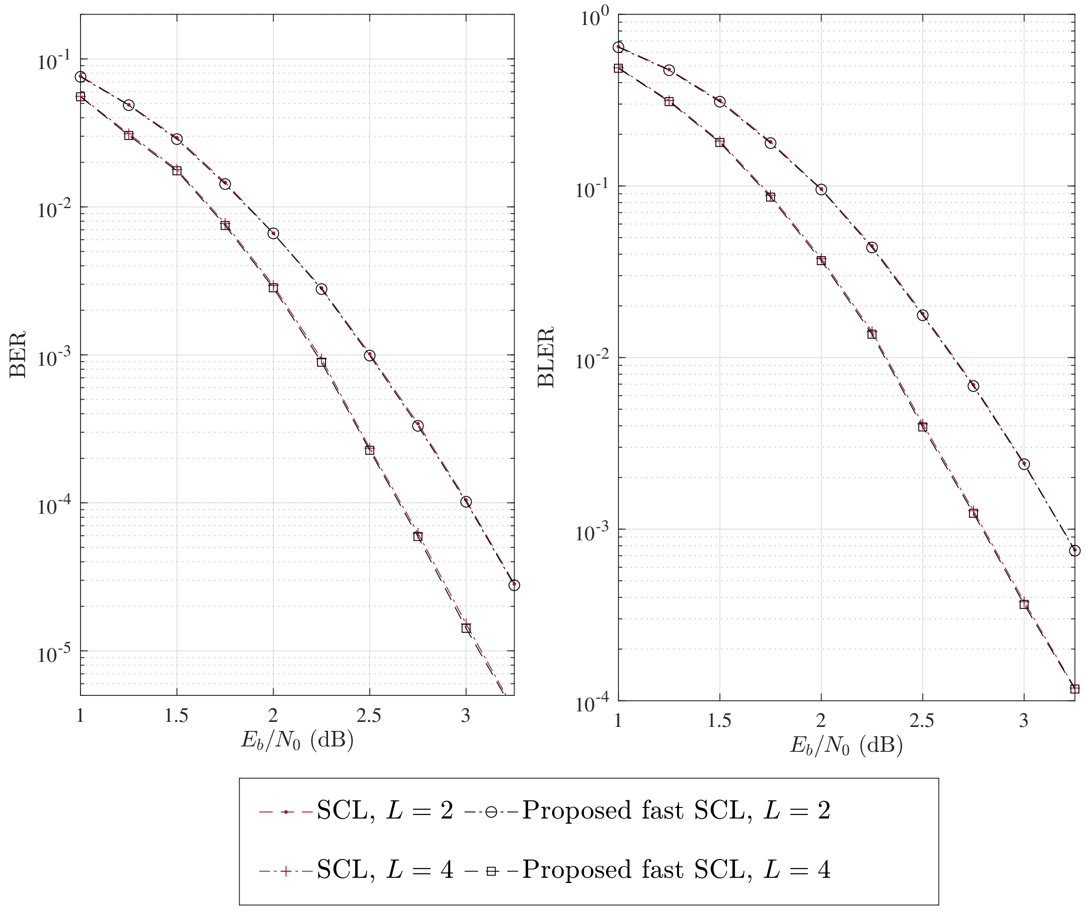


Fig. 3.3: BER and BLER performances of the SCL [9] and proposed fast SCL decoders for a $P(512, 256)$.

Table 3.2: Number of special nodes for different polar codes.

	Node Length	Rate-0	REP	Rate-1	SPC	Type-I	Type-II	Type-III	Type-IV	Type-V
$P(1024, 256)$	8	0	9	0	4	1	0	1	1	10
	16	1	6	0	2	1	0	1	0	0
	32	1	5	0	1	0	0	0	1	0
	64	1	2	0	1	0	0	0	0	0
	128	1	0	0	0	0	0	0	0	0
$P(512, 256)$	8	0	4	1	3	1	0	0	1	6
	16	0	2	0	2	0	1	0	1	0
	32	0	2	1	1	0	0	1	0	0
	64	0	1	1	0	0	0	0	0	0

Fig. 3.2 compares the BER and BLER performances of the proposed fast SCL decoder with those of the LLR-based SCL decoder [9] for a $P(1024, 256)$. Note that the proposed decoder, while matching the performance of the SCL decoder, has a decoding latency of almost 16% and 23% less than that of the FSCL decoder [39] for $L = 2$ and 4, respectively (see Table 3.3). In particular, considering the special nodes in the simulated $P(1024, 256)$ with 16-bit CRC provided in Table 3.2, the proposed fast SCL decoder consumes 226 and 250 time steps, whereas the FSCL decoder requires 269 and 323 time steps to complete the decoding when $L = 2$ and 4, respectively.

Furthermore, Fig. 3.3 illustrates the BER and BLER performances of the proposed fast SCL decoder for $P(512, 256)$. The required number of time steps of the proposed fast SCL decoder is 135 and 159, respectively, for $L = 2$ and 4 which is 19% and 25% less than that of the FSCL decoder [39].

3.5 Conclusion

The SCL decoder can be used to improve the performance of polar codes, especially for short to moderate length codes. However, their serial decoding nature results

Table 3.3: Decoding latency comparison of different SCL decoders

	Required number of time steps for the proposed fast SCL decoders		Saving with respect to the FSCL decoders [39]	
	$L = 2$	$L = 4$	$L = 2$	$L = 4$
$P(1024, 256)$	226	250	16%	23%
$P(512, 256)$	135	159	19%	25%

in significant decoding latency. Thus we proposed fast SCL decoders for five recently-identified nodes in the decoding tree of polar codes, i.e., Type-I, Type-II, Type-III, Type-IV and Type-V nodes. The new decoders, while achieving the same error-rate performance as the SCL decoders, significantly reduce the decoding latency.

Chapter 4

Fast Successive-Cancellation Flip Decoders

SCF decoding is another SC-based decoding scheme that outperforms SC decoding and can be used to improve the performance of polar codes. The SCF decoding relies on multiple sequential applications of the SC decoder. Thus, the sequential iterations of the SC decoder results in the relatively high decoding latency of the SCF decoder. Furthermore, the fast SC decoders in [5–7, 60–62] cannot be used directly to increase the speed because the SCF decoder needs to maintain the list of the least-reliable bit locations. Specifically, the conventional SCF decoder [10] uses the bit-level LLR, $|y^{i,0}|$, to determine the bit-flip locations. Note that the existing fast SC decoder of a node (ϕ, t) does not compute the bit-level LLRs to improve the throughput.

Since computing the bit-level LLRs will incur a high decoding latency, we propose a new decision metric to determine bit-flip locations. The proposed decision metric can be computed without traversing the decoding tree, hence speeding up SCF decoding.

In this chapter, we first introduce the new decision metric, and then describe the fast SCF decoders for the special nodes based on our proposed metric. We further compare the decoding latencies of the original SCF decoder [10], the existing fast SCF decoders [11], and our proposed fast SCF decoders. Simulation results are also provided to compare the performances of different SCF decoders. Using our proposed fast parallel SCF decoders, a speed-up of up to 81% is observed. This significant reduction in the decoding latency comes without sacrificing the error-rate performance of the code.

4.1 Proposed Decision Metric

Before we present our proposed decision metric, we first analyze the decision metric used in the conventional SCF decoder, i.e., $|y^{i,0}|$ [10]. To this end, we consider the path metric of a codeword decoded by the conventional SCL decoder for $L = 1$.

The conventional SCL decoder considers two possibilities for the bit u_i, i.e., 0 or 1, and selects the path with the least PM. Using (2.27), it can be verified that the difference between the PMs of the considered codewords is exactly $|y^{i,0}|$. That is, the decision metric used in the conventional SCF decoder equals the difference between the PMs of the codewords that share the same code bits except the code bits corresponding to the node $(i, 0)$.

We use the aforementioned observation to propose a decision metric for our proposed fast SCF decoders. Since our decoders decode the codewords corresponding to a node without traversing it, we propose a decision metric that can be computed without the traversal. In particular, we propose to use the following decision metric:

$$\lambda = \mathrm{PM}_{\mathbf{x}} - \mathrm{PM}_{\mathrm{ML}}. \tag{4.1}$$

Here, $\mathrm{PM}_{\mathbf{x}}$ and $\mathrm{PM}_{\mathrm{ML}}$ are the contributions of a codeword $\mathbf{x}$ and the ML codeword to the PM, respectively. The ML codeword is the codeword that is selected by the SCL decoders for $L = 1$, i.e., it has the least associated PM. Observe that our proposed metric λ coincides with $|y^{i,0}|$ for the nodes $(i, 0)$, which is used in the conventional SCF decoding [10].

It should be noted that [11] also proposed decision metrics for the REP, Rate-1, SPC and Type-I nodes. However, except the metric used for the REP node, the other metrics used in [11] result in performance degradation. Our proposed decoders, on the other hand, are based on the contribution of the selected codeword to the PM and, hence, show better BER performance than the decoders from [11].

4.2 Proposed Fast SCF Decoders

In the following, we first describe the fast SCF decoders of the REP, Rate-1, SPC and Type-I nodes proposed in [11]. We also compare them with the proposed SCF decoders highlighting the key differences, which eventually result in the improvement of the BER performance. Afterwards, we present fast SCF decoders for the remaining nodes (Type-II, Type-III, Type-IV, and Type-V nodes).

4.2.1 REP Node

A REP node contains only one information bit. Therefore, there are two valid REP node codewords: an all-zeros codeword and an all-ones codeword. The PM contributions for these codewords can be computed as

$$\mathrm{PM}_0 = \sum_{k=0}^{R-1} \ln\left(1 + e^{-y_k}\right), \tag{4.2}$$

and

$$\mathrm{PM}_1 = \sum_{k=0}^{R-1} \ln\left(1 + e^{y_k}\right). \tag{4.3}$$

When the ML codeword is an all-zeros codeword, $\mathrm{PM}_{\mathrm{ML}} = \mathrm{PM}_0$, and $\lambda = \mathrm{PM}_1 - \mathrm{PM}_0$; otherwise $\lambda = \mathrm{PM}_0 - \mathrm{PM}_1$. Equivalently, $\lambda = |\mathrm{PM}_1 - \mathrm{PM}_0|$, which can be simplified using (3.3) as

$$\lambda = \left|\sum_{k=0}^{R-1} y_k\right|. \tag{4.4}$$

Note that [11] also used the same decision metric for a REP node.

The decision metric in (4.4) is equivalent to that of the conventional SCF decoder [10] computed for the only non-frozen bit in $\mathbf{u}$ supported by a REP node.

After the first trial, in case the bit-flip position belongs to a REP node, all bits of the estimated ML codeword, x^{ML}, of the REP node are flipped. This process is equivalent to the flipping process described in [10], i.e., flipping the only non-frozen bit of the REP node at $\mathbf{u}$.

4.2.2 Rate-1 Node

The ML codeword of a Rate-1 node is obtained as $\mathbf{x}^{\mathrm{ML}} = H(\mathbf{y})$ for $0 \leq k < R$, and its corresponding PM is

$$\mathrm{PM}_{\mathrm{ML}} = \sum_{k=0}^{R-1} \ln\left(1 + e^{-|y_k|}\right). \tag{4.5}$$

The proposed SCF decoder for a Rate-1 node follows the approach used in its fast SCL decoder [39] to compute the contributions of other codewords to the PM. In particular, codewords are generated by flipping bits, individually and simultaneously, relative to the ML codeword, as is done in the decoding of a Type-III node. Specifically, the absolute values of LLRs are sorted. We denote the sorted indexes by (i), where $0 \leq i < R$. Next, PMs are computed in a successive manner by path splitting for each index (i). Since, we are generating $T_{\max}$ numbers of the most likely codewords, as we do in the SCL decoder with list size L [38, Theorem 1], we only consider path splitting for $0 \leq i < \min\{T_{\max}, R-1\}$. This limits the number of searches that are required to find the bit-flip positions. The contributions of the Rate-1 node codewords can be obtained as

$$\mathrm{PM}_{\mathbf{x}} = \mathrm{PM}_{\mathrm{ML}} + \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|. \tag{4.6}$$

Here, $\mathbf{f}_{\mathbf{x}}$ is the set of those indexes of codeword $\mathbf{x}$ at which the code bits differ that of the ML codeword. Consequently, the decision metric can be computed as

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|. \tag{4.7}$$

Observe that [11] introduced $\lambda_{\mathbf{x}} = |y_{(i)}|$ for $0 \leq i \leq R-1$ as the decision metric for a Rate-1 node. This metric computes the contribution of those codewords that differ from the ML codeword only in one bit location. For example, the decision metric used in [11] would compute $\lambda_2 = |y_{(2)}|$. On the other hand, we compute $\lambda_2 = |y_{(2)}|$ when $|y_{(2)}| < |y_{(0)}| + |y_{(1)}|$, and $\lambda_2 = |y_{(0)}| + |y_{(1)}|$ otherwise.

In the decoding trials following the initial one, in case the bit-flip positions belong to a Rate-1 node, we flip the bits in $\mathbf{f}_{\mathbf{x}}$ of the estimated ML codeword.

4.2.3 SPC Node

The ML codeword of an SPC node can be estimated using Wagner decoding [55]. In particular, we set $\mathbf{x}_k = H(\mathbf{y}_k)$ for $0 \leq k < R$. We then find the location of the least-reliable LLR value and compute the parity as

$$p = \operatorname*{argmin}_{0 \leq k < R} |y_k|, \tag{4.8}$$

and

$$\gamma_{\mathrm{ML}} = \bigoplus_{k=0}^{R-1} x_k, \tag{4.9}$$

respectively. Further, we set $\mathbf{x}_p = \mathbf{x}_p \oplus \gamma_{\mathrm{ML}}$ to satisfy the even-parity constraint. The PM corresponding to the ML codeword is given by

$$\mathrm{PM}_{\mathrm{ML}} = \Delta + \gamma_{\mathrm{ML}} |y_p|, \tag{4.10}$$

where $\Delta = \sum_{k=0}^{R-1} \ln\left(1 + e^{-|y_k|}\right)$.

In our proposed decoder, the absolute values of LLRs (excluding y_p) are sorted in ascending order. We use (i), where $0 \leq i < R-1$ to denote the sorted indexes. Next, we generate codewords by flipping bits relative to the ML codeword such that the parity constraint is satisfied. This can be done by 'path splitting' as described in [39] and in the proposed SCL decoders for the Type-III and Type-IV nodes.

The contribution of the generated codewords to the PM can be shown to be

$$\mathrm{PM}_{\mathbf{x}} = \Delta + \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|. \tag{4.11}$$

Here, $\mathbf{f}_{\mathbf{x}}$ is the set of indexes of the codeword $\mathbf{x}$ which differ from $H(\mathbf{y})$, where $H(\mathbf{y})$ is defined in (2.26). Consequently, the decision metrics for the node can be computed as

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j| - \gamma_{\mathrm{ML}} |y_p|. \tag{4.12}$$

Note that [11] introduced the following metric

$$\lambda_{\mathbf{x}} = |y_{(i)}| + (-1)^{\gamma_{\mathrm{ML}}} |y_p|, \tag{4.13}$$

which corresponds to only double bit flips relative to the ML codeword. Since (4.13) does not consider some valid codewords that have a greater likelihood than the considered codewords, it results in a BER performance loss compared to the conventional SCF decoder. Our proposed decoders, on the other hand, do not incur any BER performance loss compared to the conventional SCF decoder, as is evident from the simulation results presented in Section 4.4.

If a trial of the fast SCF decoder needs to flip bits in an SPC node, the bits stored in $\mathbf{f_x}$ are flipped relative to the hard-decision bit estimates.

4.2.4 Type-I node

A Type-I node contains two information bits, and its codeword is $\mathbf{x} = \{x_{R-2}, x_{R-1}, ..., x_{R-2},\ x_{R-1}\}$. Thus, the even and odd-indexed code bits constitute two repetition codes.

The proposed SCF decoder computes the PMs corresponding to $(x_{R-2}, x_{R-1}) = (0,0), (0,1),\ (1,0), (1,1)$, as mentioned in Section 3.1.1, and selects the one which has the least PM as the ML codeword. Next, it computes the decision metrics corresponding to other codewords using (4.1). It can be verified that the decision metric corresponding to the codeword that has all of its even-indexed bits flipped with respect to the ML codeword is

$$\lambda_0 = |\varsigma_0|. \tag{4.14}$$

Likewise, the decision metric corresponding to the codeword that differs from the ML codeword at odd-indexed bits is

$$\lambda_1 = |\varsigma_1|. \tag{4.15}$$

Finally, the codeword that differs from the ML codeword at all locations has the following decision metric:

$$\lambda_2 = |\varsigma_0| + |\varsigma_1|. \tag{4.16}$$

The conventional SCF decoder considers only two additional codewords: the codeword with the second last and last bits in the $\mathbf{u}$ vector flipped relative to the decoded SC codeword. These codewords correspond to the codewords with decision metrics (4.15) and (4.16), respectively. The Type-I node decoder presented in [11] only computes the codewords with decision metrics λ_0 and λ_1; i.e., it ignores a valid conventional SCF codeword. Our proposed decoder, on the other hand, computes the decision metric for both codewords. In addition, it considers a valid Type-I codeword that has both of its bits in $\mathbf{u}$ flipped relative to that of the ML codeword (codeword with decision metric (4.14)). This codeword has more likelihood (and hence has less contribution to the PM) than the codeword that has its last bit in $\mathbf{u}$ flipped.

4.2.5 Type-II node

A Type-II node contains three information bits and its codeword is $\mathbf{x} = \{x_{R-4}, x_{R-3}, x_{R-2}, x_{R-1}, ..., x_{R-4}, x_{R-3}, x_{R-2}, x_{R-1}\}$, where $x_{R-4} = x_{R-3} \oplus x_{R-2} \oplus x_{R-1}$ [7]. Hence, $(x_{R-3}, x_{R-2}, x_{R-1})$ can be one of the eight combinations: (000), (001), (010), (011), (100), (101), (110), and (111).

The proposed decoders for the Type-II node works similarly to that of a Type-I node. In particular, we first compute 8 PM values corresponding to the valid codewords of a Type-II node. Next, the ML codeword is found by selecting the code with the least PM. Then, we compute the decision metrics for the remaining codeword using (4.1).

The decision metric values of a Type-II node can be computed similar to an SPC node as in (4.12). In particular, $(x_{R-4}, x_{R-3}, x_{R-2}, x_{R-1})$ form a $(4, 3)$ SPC node with the corresponding LLR vector $\boldsymbol{\eta} = \{\eta_0, \eta_1, \eta_2, \eta_3\}$. Let η_p denote the minimum value of $\boldsymbol{\eta}$. We can compute the decision metric corresponding to a codeword $\mathbf{x}$ which disagrees with the hard decision on $\boldsymbol{\eta}$, $H(\boldsymbol{\eta})$, at locations in $\mathbf{f_x}$ as

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f_x}} |\eta_j| - \gamma_{\text{ML}} |\eta_p|. \tag{4.17}$$

Remark 4.1. *It should be noted that we do not need to compute the decision metric values using the aforementioned equations for Type-I and Type-II nodes. Rather, we compute the PMs for all the codewords and, based on* (4.1)*, subtract the PM of the ML codeword from that of the remaining codewords to compute the decision metrics.*

4.2.6 Type-III node

The even and odd-indexed bits of a Type-III node codeword constitute two separate SPC codes. Thus, they can be optimally decoded using two Wagner decoders [7].

In particular, we set $\mathbf{x}_i^{\mathrm{ML}} = H(\mathbf{y}_i)$ for $0 \leq i < R$. Next, we find the locations of the least reliable even-indexed and odd-indexed LLR values as $e = \underset{0 \leq i \leq R/2-1}{\arg\min} |y_{2i}|$ and $o = \underset{0 \leq i \leq R/2-1}{\arg\min} |y_{2i+1}|$ for $0 \leq i < R/2$, respectively. Then we check the parity condition for the SPC codes as

$$(\gamma_e,\gamma_o) = \left(\bigoplus_{i=0}^{R/2-1} x_{2i}^{ML}, \bigoplus_{i=0}^{R/2-1} x_{2i+1}^{ML} \right), \tag{4.18}$$

and set $x_{2e}^{\mathrm{ML}} = x_{2e}^{\mathrm{ML}} \oplus \gamma_e$ and $x_{2o+1}^{\mathrm{ML}} = x_{2o+1}^{\mathrm{ML}} \oplus \gamma_o$. The contribution of the ML codeword to the PM is given by

$$\mathrm{PM}_{\mathrm{ML}} = \Delta + \gamma_e |y_{2e}| + \gamma_o |y_{2o+1}|, \tag{4.19}$$

where $\Delta = \sum_{k=0}^{R-1} \ln\left(1 + e^{-|y_k|}\right)$.

Next, we generate different codewords by flipping bits relative to the ML codeword and compute their corresponding contribution to the PM, as discussed in Section 3.1.3. It can be shown that the contribution of a Type-III codeword to the PM is given by

$$\mathrm{PM}_{\mathbf{x}} = \Delta + \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|.$$

Here, $\mathbf{f}_{\mathbf{x}}$ is the set of bit indexes of the codeword $\mathbf{x}$ that differ from $H(\mathbf{y})$. Consequently, the decision metrics for the node are

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j| - \gamma_e |y_{2e}| - \gamma_o |y_{2o+1}|. \tag{4.20}$$

4.2.7 Remaining Nodes

The fast SCF decoders for the remaining two special nodes can be similarly implemented. For the Type-IV node, the proposed decoder works similarly to that of the proposed Type-III node. That is, it first computes the ML codeword and the corresponding PM. Next, it generates other codewords and their corresponding PMs (or equivalently the decision metrics) by flipping the bits relative to the ML codeword as detailed in Section 3.1.4.

For the Type-V node, the proposed decoder first generates 16 Type-V codewords and computes their corresponding PMs. Next, it selects the codeword with the least PM as the ML codeword and computes the decision metrics for the remaining codewords using (4.1).

4.3 Decoding Latency

The first trial of an SCF decoder, in addition to decoding the codeword, computes decision metrics and maintains a list of bit-flip positions. The remaining trials, on the other hand, do not compute any decision metrics. Therefore, the decoding latency of the first trial differs from that of the remaining trials. Following the assumptions in 3.3 and by means of similar calculations used for the fast SCL decoders, we provide the decoding latencies of our proposed fast SCF and the existing fast SCF decoders [11] (FSCF).

The decoding latency of the initial trials for the FSCF and proposed fast SCF decoders are tabulated in Table 4.1. Since the decoding latency calculations are very similar to those presented in Section 3.3, we briefly mention the results below.

The proposed decoders for the Rate-1 and SPC nodes require more time steps in first decoding trial than the FSCF decoders. This is because the FSCF decoders for both nodes flip only a single coded bit, whereas our proposed decoders flip multiple coded bits. Although the FSCF decoders consume slightly fewer time steps, they incur a significant error-rate performance loss, as it is evident from Fig. 4.1.

For the remaining decoding trials, we assume that bit flipping can be carried out instantaneously. Under this assumption, the decoding latency of both the FSCF and our proposed decoders are one, zero, one, and two time steps for the REP, Rate-1, SPC and Type-I nodes, respectively.

For the remaining nodes, our proposed decoders require fewer time steps than the FSCF decoders. In particular, the FSCF decoders consume $t-1$, $2t-2$, $2t-3$, and $t+1$ time steps for decoding Type-II, Type III, Type-IV and Type-V nodes, respectively. Our proposed fast SCF decoders, on the other hand, require only one or two time steps to decode these nodes.

Table 4.1: Decoding latencies of the FSCF and proposed fast SCF decoders in the first trial.

Node	**FSCF [11]**	**Proposed fast SCF**
Rate-0	0	0
REP	2	2
Rate-1	2	$\min\{T_{\max}, R\}$
SPC	2	$\min\{T_{\max}+1, R\}$
Type-I	2	2
Type-II	t	2
Type-III	$3t-4$	$1+\min\{T_{\max}, R-2\}$
Type-IV	$3t-4$	$1+\min\{T_{\max}, R-4\}$
Type-V	$t+3$	2

4.4 Simulation Results

In this section, we compare the BER and BLER performances of the proposed fast SCF decoders with those of the existing decoders. A 16-bit CRC defined by the generator polynomial $x^{16}+x^{15}+x^{12}+x^{7}+x^{6}+x^{4}+x^{3}+1$ (0xC86C) is used, unless otherwise stated. Systematic polar codes [41] with random BPSK-modulated codewords were transmitted through AWGN channel. Furthermore, we used the simple GA method [51]. We have included the performance of the SC and the oracle-assisted SC (SC-Oracle)[1] decoders in our simulation results to provide a better insight into the performance of different decoders.

Fig. 4.1 compares the BER and BLER performances of the proposed SCF decoder with those of the SCF [10] and FSCF [11] decoders for a $P(128,96)$ with 8-bit CRC (0xEA). Due to the possibility of multiple bits flipping in our proposed fast SCF decoder, it slightly outperforms the SCF decoder. Furthermore, our proposed decoder significantly outperforms the FSCF decoder, whose performance degradation is mainly due to the decision metrics used in the decoding of the SPC node. Note that multiple

[1]We employ an oracle-assisted SC decoder which is able to correct the first erroneous bit at the decision level.

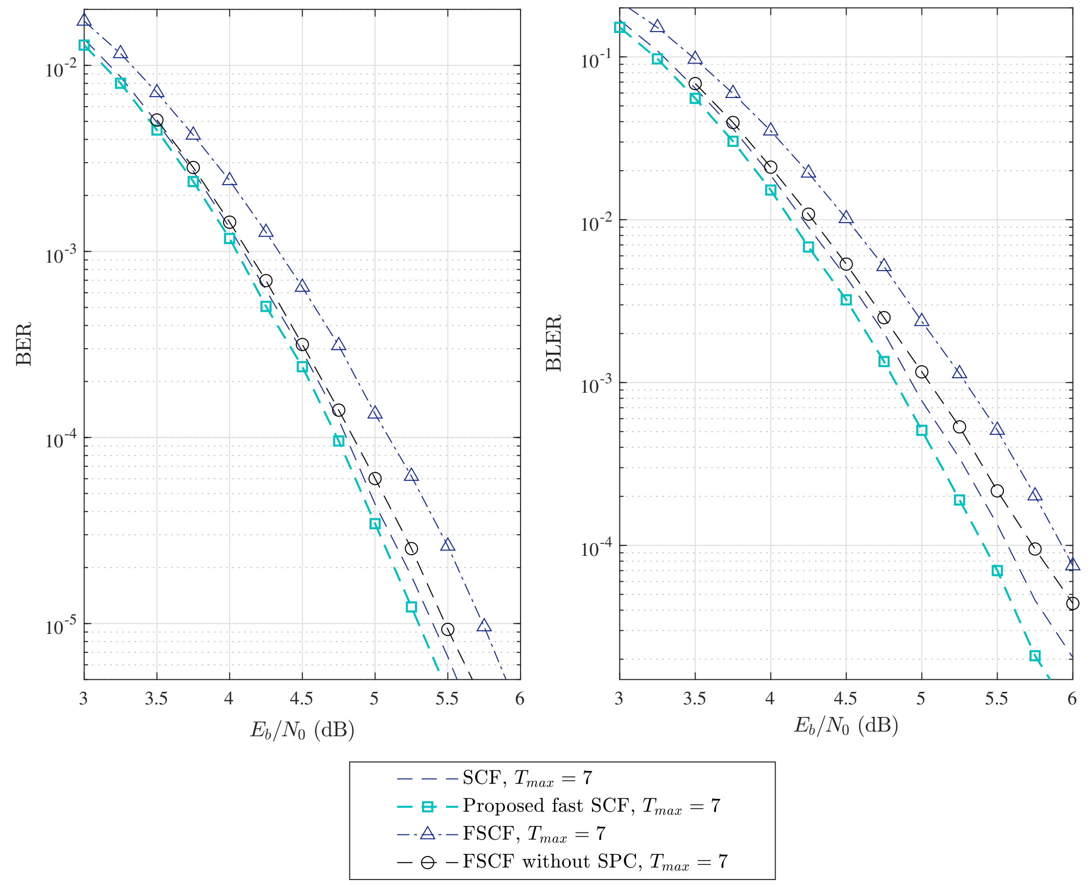


Fig. 4.1: BER and BLER performance of different SCF decoders (i.e., the SCF decoder [10], the FSCF decoders [11] and proposed fast SCF decoders) for a $P(128, 96)$ polar code and $T_{\max} = 7$.

Table 4.2: Number of special nodes for different polar codes.

	Node Length	Rate-0	REP	Rate-1	SPC	Type-I	Type-II	Type-III	Type-IV	Type-V
$P(512, 256)$	8	0	4	1	3	1	0	0	1	6
	16	0	2	0	2	0	1	0	1	0
	32	0	2	1	1	0	0	1	0	0
	64	0	1	1	0	0	0	0	0	0
$P(128, 96)$	8	0	0	0	2	0	0	0	0	2
	16	0	0	1	0	0	0	0	0	1
	64	0	0	0	0	0	0	1	0	0

Table 4.3: Decoding latency comparison of different SCF decoders in the first trial

	Required number of time steps		Saving with respect to SCF [10]
	FSCF [11]	Proposed fast SCF	
$P(128, 96)$, $T_{\max} = 7$	51	49	81%
$P(512, 256)$, $T_{\max} = 15$	158	234	77%

SPC nodes exist in the FSCF decoding tree of $P(128, 96)$ with 8-bit CRC. The FSCF decoder performance without SPC nodes is presented in Fig. 4.1. It can be seen that the performance of the FSCF decoder is significantly improved without SCP nodes.

The number and length of the special nodes in the proposed SCF code tree is presented in Table 4.2. Furthermore, the required number of time steps for the FSCF and proposed fast SCF decoders in their first trials, along with the reduction in the decoding latency using our proposed fast decoder instead of the SCF [10] decoder, are provided in Table 4.3.

The BER and BLER performances of different SCF decoders for $P(512, 256)$ are depicted in Fig. 4.2. Note that the performance of our proposed fast SCF decoder is identical to that of the SCF decoder with 77% fewer required time steps. More

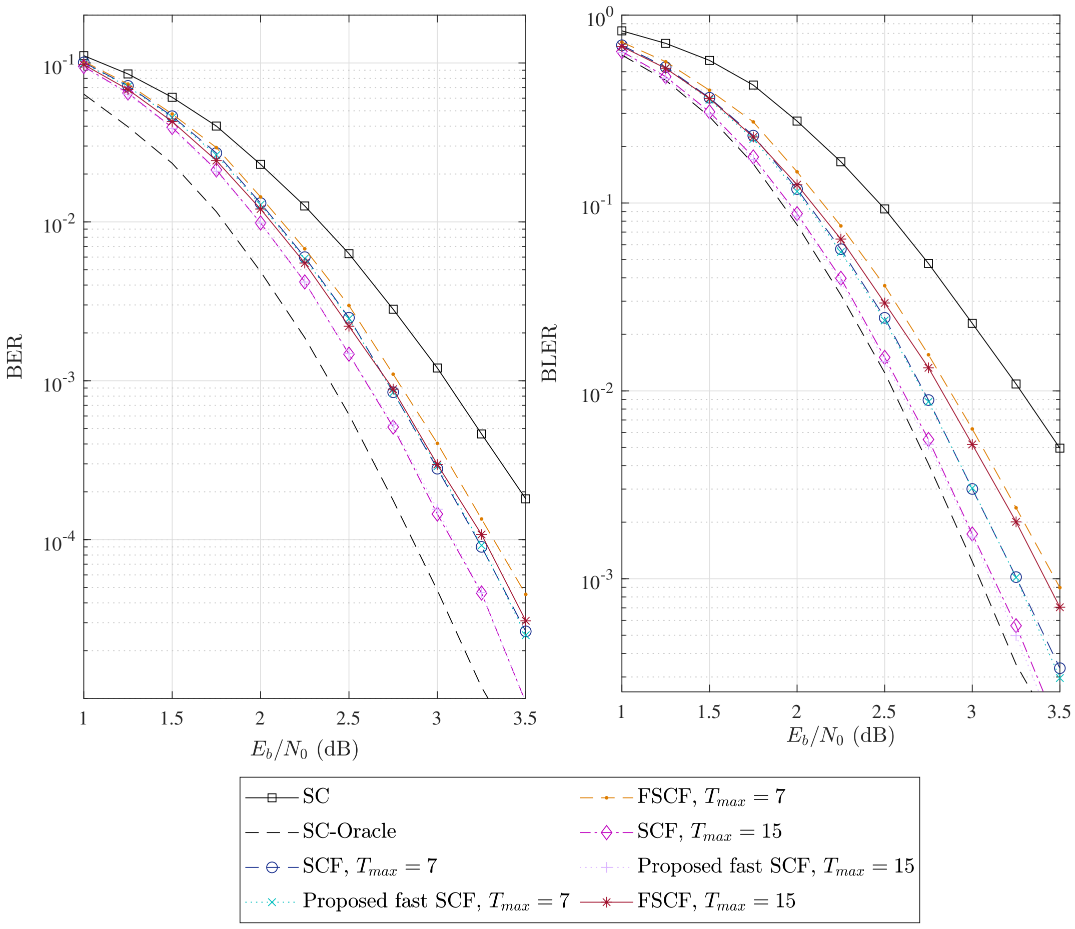


Fig. 4.2: BER and BLER performance of different SCF decoders for a $P(512, 256)$ polar code.

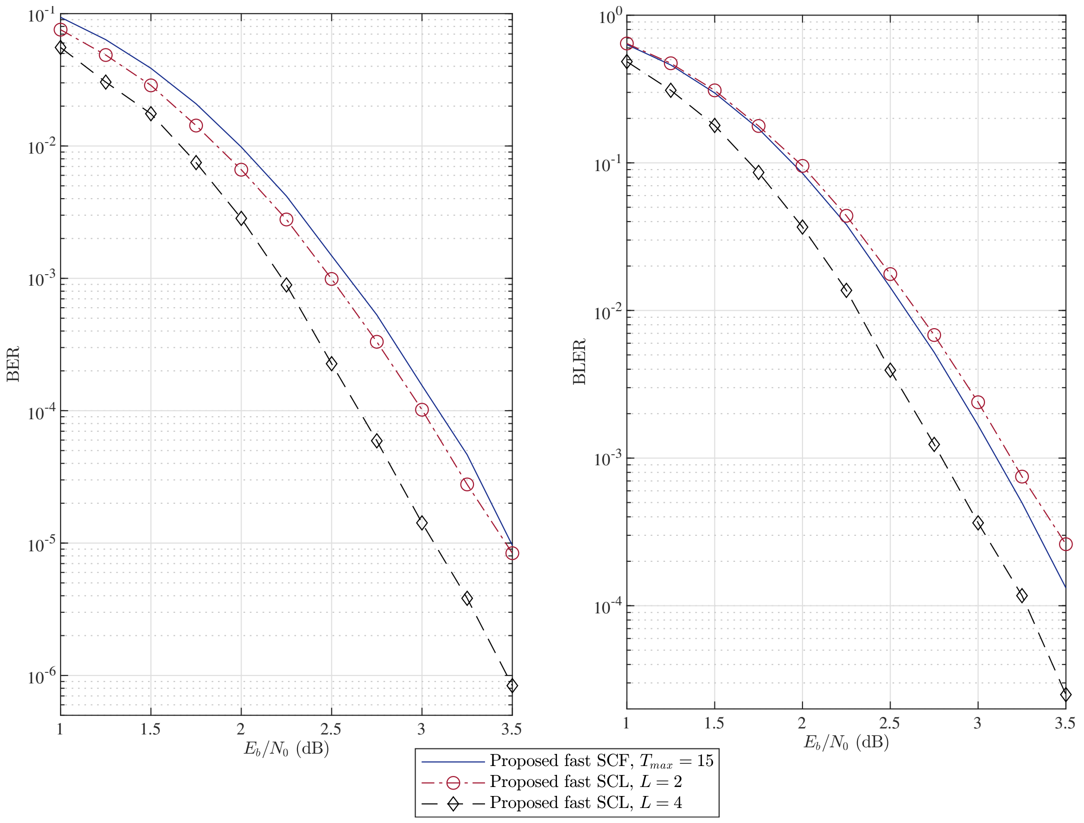


Fig. 4.3: BER and BLER performance of the proposed fast SCL and proposed fast SCF decoders for a $P(512, 256)$ polar code.

specifically, our proposed fast SCF decoder consumes 234 time steps in its initial trial. By contrast, the SCF decoder requires 1022 time steps. Here we have made the assumption that finding bit flip positions has no latency overhead. Thus the SCF decoder in each trial consumes a similar number of time steps as the SC decoder. On the other hand, the FSCF decoder consumes 158 time steps in its first trial, however, it fails to achieve the performance of the SCF decoder. Furthermore, in the next trials, our proposed decoder requires 90 time steps while the FSCF decoder takes 116 time steps.

We can compare the performance of the SCF and the SCL decoders in Fig. 4.3. It is observed that the performance of the proposed fast SCF decoder with $T_{\max} = 15$ is almost identical to that of the proposed fast SCL decoder with $L = 2$. However, for larger list sizes, such as $L = 4$, the proposed fast SCL decoder outperforms the proposed fast SCF decoder. Although, the proposed SCL decoder consumes fewer time steps (see Table 3.3) than the proposed SCF decoder, its memory complexity is two and four times that of the proposed SCF decoder when $L = 2$ and 4, respectively.

4.5 Conclusion

In this chapter, we proposed novel fast SCF decoders. To that end, we first proposed a new decision metric which can be computed without the decoding tree traversal. Using the proposed decision metric, we introduced fast parallel SCF decoders for many special nodes identified in the decoding tree of polar codes. The proposed fast SCF decoders achieve significant decoding speed improvement and, unlike the existing FSCF decoders [11], show the same error-rate performance as that of the conventional SCF [10] decoder.

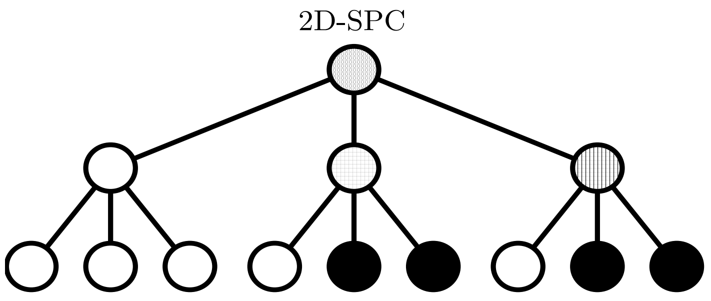
Chapter 5

New Fast Nodes for 3×3 Kernel Polar Codes

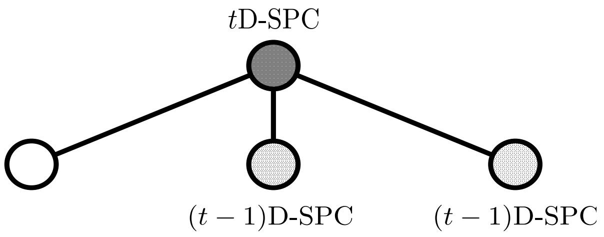
Non-binary kernels are used to improve the length flexibility of polar codes. Given the long decoding latency of the SC decoder, devising fast decoding solutions for non-binary kernels is necessary. In this chapter, we identify tD-SPC nodes in the decoding tree of polar codes which are constructed by $\mathbf{T_3}$ and $\mathbf{T_3'}$ kernels and then propose low-complexity decoders for them. Moreover, we adapt the G-REP node introduced for binary kernel to be used in the fast SC decoding of the ternary kernel polar codes. Numerical results are also provided to compare the decoding latency and performance of the proposed decoders with those of the existing ones. The results show that implementing the proposed fast decoders can reduce the decoding latency by more than 40% if an error-rate performance loss of just 0.5 dB can be tolerated.

5.1 tD-SPC node

An SPC node in the polar code tree is a node whose descendants are all Rate-1 nodes, except the leftmost bit which is frozen to zero. The only frozen bit in the SPC node imposes the even parity constraint on the codeword. This can be decoded through Wagner decoding [55] by making a hard decision on the received LLRs. If



(a)



(b)

Fig. 5.1: Tree representation of tD-SPC node.

the parity-check condition is satisfied, then the decoder outputs the hard-decision estimates; otherwise, the bit associated with the least reliable LLR is flipped to satisfy the constraint. The SPC node pattern was first identified in [6] in the polar code tree constructed by $\mathbf{T}_2$ kernel. Later [30] showed that the SPC pattern and its decoding method is valid while the $\mathbf{T}_3$ kernel is used, thus it can be implemented in the fast decoding of multi-kernel polar codes. This observation is also valid for the case of $\mathbf{T}'_3$ kernel.

In this section, we identify an extended SPC node in the polar code tree constructed by $\mathbf{T}_3$ or $\mathbf{T}_3^{'}$ kernel, called a t-dimensional SPC (tD-SPC) node. Specifically, in a tD-SPC node the left child is a Rate-0 node, whereas the center and the right children are $(t-1)$D-SPC nodes. The tD-SPC nodes[1] correspond to the node code rate of $(2/3)^t$. Also, note that a 1D-SPC is a SPC node of length three. To further investigate the tD-SPC nodes and their decoding, we first consider a 2D-SPC node, which is illustrated

[1]A tD-SPC node can be also described as a node where at each level all the left children are Rate-0 nodes.

in Fig. 5.1(a). It can be seen that this node consists of one Rate-0 node and two SPC (1D-SPC) nodes. The codeword $\mathbf{x} = \{x_0\ x_1\ x_2\ ...\ x_8\}$ corresponding to the 2D-SPC node can be structured in the form of a 3×3 matrix below

$$\mathbf{M_1} = \begin{bmatrix} x_0 & x_3 & x_6 \\ x_1 & x_4 & x_7 \\ x_2 & x_5 & x_8 \end{bmatrix}. \tag{5.1}$$

In this matrix each row and each column is an SPC code with even parity constraint. For example, the third row and second column suggests that $x_2 + x_5 + x_8 = 0$ and $x_3 + x_4 + x_5 = 0$, respectively. This is a SPC-product code and it can be decoded with soft iterative decoding [63]. However, to increase the decoding speed we adapt a hard decoding approach. In particular, first the parity of all SPC codes are computed to find the rows and columns in error. As each bit in error results in one erroneous row and one erroneous column, it can be readily observed that the sum of the number of rows and the number of columns in error is always an even number. As such, the error patterns can be categorized into five general cases. Fig. 5.2 illustrates sample cases of these five patterns.

The hard decoding of five error patterns are explained as follows. Fig. 5.2(a) illustrates the situation when one row and one column are in error. In this case flipping the common bit between the erroneous row and column will satisfy the even parity constraint for two erroneous SPC codes, resulting in all SPC constraints being satisfied. If the SPC codes in two rows/columns or two rows and two columns are in error, flipping a pair of bits, as shown in Fig. 5.2(b) and Fig. 5.2(c), respectively, will satisfy all parity constraints. Among these pairs, we suggest flipping the bit with the smallest sum of the absolute value of LLRs. Moreover, Fig. 5.2(d) shows when all rows/columns and only one column/row are in error. In this case the bit associated with the least reliable LLR in the single erroneous column/row will be flipped and the error pattern will changed to the case in Fig. 5.2(b). Furthermore, when all SPC codes do not satisfy the even parity constraint, Fig. 5.2(e), among all the bits, the one that has the least reliable LLR will be flipped. This yields to the error pattern in Fig. 5.2(c).

Now that 2D-SPC nodes and their hard decoding is well understood, we can extend

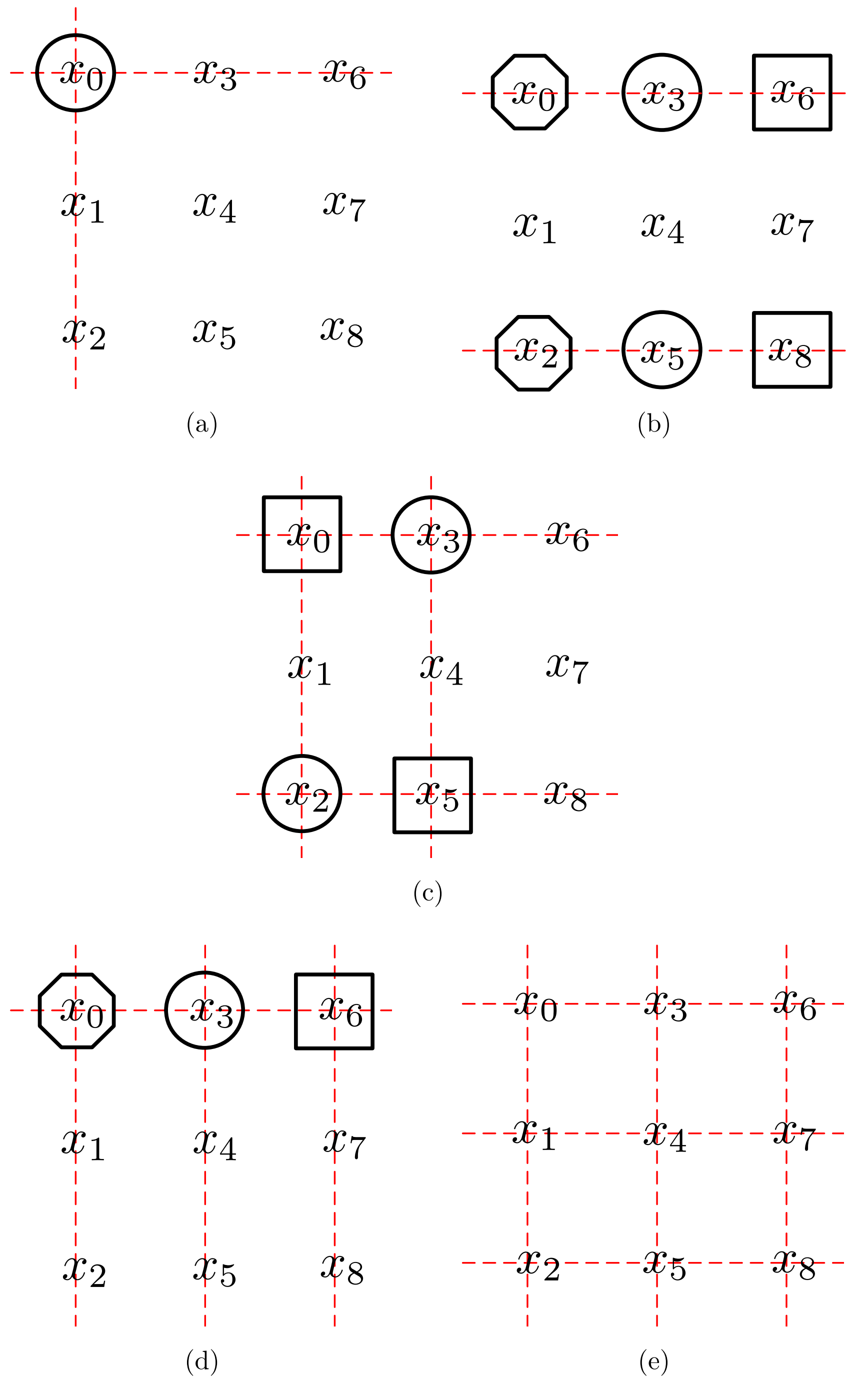


Fig. 5.2: The five different error patterns of a 2D-SPC node. Red dashed lines denote erroneous rows and columns. Also, circle, square, and octagonal shapes are used to denote bit/pair of bits that are suspected to be in error.

these ideas to higher dimensions. Note that a 3D-SPC consists of Rate-0 and 2D-SPC nodes as illustrated in Fig. 5.1(b). In this case nine 3×3 matrices with similar properties as $\mathbf{M}_1$ can be formed. In particular, the first three matrices are

$$\mathbf{M_1} = \begin{bmatrix} x_0 & x_3 & x_6 \\ x_1 & x_4 & x_7 \\ x_2 & x_5 & x_8 \end{bmatrix} \quad \mathbf{M_2} = \begin{bmatrix} x_9 & x_{12} & x_{15} \\ x_{10} & x_{13} & x_{16} \\ x_{11} & x_{14} & x_{17} \end{bmatrix} \quad \mathbf{M_3} = \begin{bmatrix} x_{18} & x_{21} & x_{24} \\ x_{19} & x_{22} & x_{25} \\ x_{20} & x_{23} & x_{26} \end{bmatrix}.$$

Furthermore, $\mathbf{M}_4$ to $\mathbf{M}_6$ are

$$\mathbf{M_4} = \begin{bmatrix} x_0 & x_9 & x_{18} \\ x_1 & x_{10} & x_{19} \\ x_2 & x_{11} & x_{20} \end{bmatrix} \quad \mathbf{M_5} = \begin{bmatrix} x_3 & x_{12} & x_{21} \\ x_4 & x_{13} & x_{22} \\ x_5 & x_{14} & x_{23} \end{bmatrix} \quad \mathbf{M_6} = \begin{bmatrix} x_6 & x_{15} & x_{24} \\ x_7 & x_{16} & x_{25} \\ x_8 & x_{17} & x_{26} \end{bmatrix}.$$

Note that, $\mathbf{M}_4$ consists of the first columns of $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{M}_3$. Also, $\mathbf{M}_5$ and $\mathbf{M}_6$, respectively, contain the second columns and third columns of $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{M}_3$. Finally, we have

$$\mathbf{M_7} = \begin{bmatrix} x_0 & x_3 & x_6 \\ x_9 & x_{12} & x_{15} \\ x_{18} & x_{21} & x_{24} \end{bmatrix} \quad \mathbf{M_8} = \begin{bmatrix} x_1 & x_4 & x_7 \\ x_{10} & x_{13} & x_{16} \\ x_{19} & x_{22} & x_{25} \end{bmatrix} \quad \mathbf{M_6} = \begin{bmatrix} x_2 & x_5 & x_8 \\ x_{11} & x_{14} & x_{17} \\ x_{20} & x_{23} & x_{26} \end{bmatrix}.$$

Observe that all the parity-check constraints in $\mathbf{M_7}$ to $\mathbf{M_9}$ are repetitions of those in $\mathbf{M_1}$ to $\mathbf{M_6}$. Thus, in the decoding of the 3D-SPC nodes only $\mathbf{M_1}$ to $\mathbf{M_6}$ are considered.

In the decoding of the 3D-SPC nodes, first the bit with the least reliability among those bits which are suspected to be in error in more than one set of matrices is flipped. The rest of the decoding works the same as the 2D-SPC decoding; i.e., among the bit or pair of bits suspected to be erroneous in all nine matrices, the one associated with the smallest sum of absolute value of LLRs is flipped.

In general, the decoding of tD-SPC nodes, when $t > 2$, follows similar rules as those of 2D-SPC node, except that we give priority to flipping bits which are suspected to be erroneous in multiple places.

Remark 5.1. *A node (ϕ, t) with the leftmost child at a level $p < t$ being Rate-0 and other children are Rate-1 nodes is proposed in [8] and is called G-PC. It is shown that* *this node can be optimally decoded using 2^p parallel Wagner decoders. The G-PC node is further extended to the relaxed G-PC nodes which is similar to the G-PC node except that some of the nodes have a rate close to one rather than being Rate-1 nodes. In this case, the additional frozen bits impose new constraints. Therefore, the proposed parallel Wagner decoding is not optimal and brings a trade off between the decoding latency and the error-rate performance [8]. Note that the 2D-SPC node can be considered as a relaxed G-PC node with two additional frozen bits. As such, we compare the performance of our proposed decoder and the parallel Wagner decoding in Section 5.3. It is observed that our proposed decoding method significantly outperforms the parallel Wagner decoding method [8].*

5.2 Generalized Repetition Nodes

The G-REP node is introduced in [8] as a generalization of the REP node. The G-REP node is any node (ϕ, t) for which all children are Rate-0 nodes, except the rightmost descendant at level $p < t$. Using $\mathbf{T_2}$ for polar code construction, the soft LLR vector of the rightmost descendant can be computed by

$$y_i^{\phi',p} = \sum_{j=0}^{2^{t-p}-1} y_{i+j2^p}^{\phi,t}, \tag{5.2}$$

where $\phi' = 2^{t-p}(\phi+1)-1$ and the node (ϕ', p) is decoded under SC or fast SC decoding. Then the corresponding codeword to the G-REP node is obtained by repeating the estimation of its rightmost descendant, $\mathbf{x}^{\phi',p}$, 2^{t-p} times as

$$\mathbf{x}^{\phi,t} = \{\underbrace{\mathbf{x}^{\phi',p}, ..., \mathbf{x}^{\phi',p}}_{2^{t-p}}\}. \tag{5.3}$$

The fast decoding of a G-REP node, when $\mathbf{T_3}$ or $\mathbf{T_3'}$ is used, can also be implemented. In this case, the LLR vector of the rightmost child at level p can be expressed as

$$y_i^{\phi',p} = \sum_{j=0}^{3^{t-p}-1} y_{i+3^p j}^{\phi,t} L_{i+3^p j}, \tag{5.4}$$

where $\phi' = 3^{t-p}(\phi+1)-1$ and $\mathbf{L} = \mathcal{R}_3^{\otimes t-p} \otimes \{\underbrace{1 \dots 1}_{3^p}\}$. Furthermore, $\mathcal{R}_3$ is $\{0\,1\,1\}$ for $\mathbf{T_3}$ and $\{1\,0\,1\}$ for $\mathbf{T'_3}$. Then $\mathbf{x}^{\phi',p}$ is estimated under SC or fast SC decoders using $\mathbf{y}^{\phi',p}$. Finally, the G-REP node codeword is obtained by

$$\mathbf{x}^{\phi,t} = \mathcal{R}_3^{\otimes t-p} \otimes \mathbf{x}^{\phi',p}. \tag{5.5}$$

5.3 Simulation results

In this section, the impact of the proposed nodes on fast SC decoding in terms of both speed and BER performances are investigated[2] . In our simulations, random BPSK-modulated codewords were transmitted through the AWGN channel. Also, "Fast-SC" refers to fast SC decoding with Rate-0, REP, Rate-1, and SPC nodes and "New Fast-SC" is the Fast-SC when G-REP, 2D-SPC and 3D-SPC nodes are also implemented. Here, we present the simulation results for the polar codes constructed by $\mathbf{T_3}$ only. A similar trend is depicted if $\mathbf{T'_3}$ is used instead.

To compare the decoding latencies, the number of time steps required to decode polar codes of different lengths and rates are provided in Table 5.1. Here, the G-REP, 2D-SPC and 3D-SPC nodes are progressively added to the Fast-SC decoding and their impact on the speed is evaluated. The last column provides the latency saving when the G-REP, 2D-SPC and 3D-SPC nodes are implemented. Note that as the channels reliability are computed based on (2.33), (2.34), and (2.35) for both $\mathbf{T_3}$ and $\mathbf{T'_3}$, the number and length of the special nodes in both cases are identical.

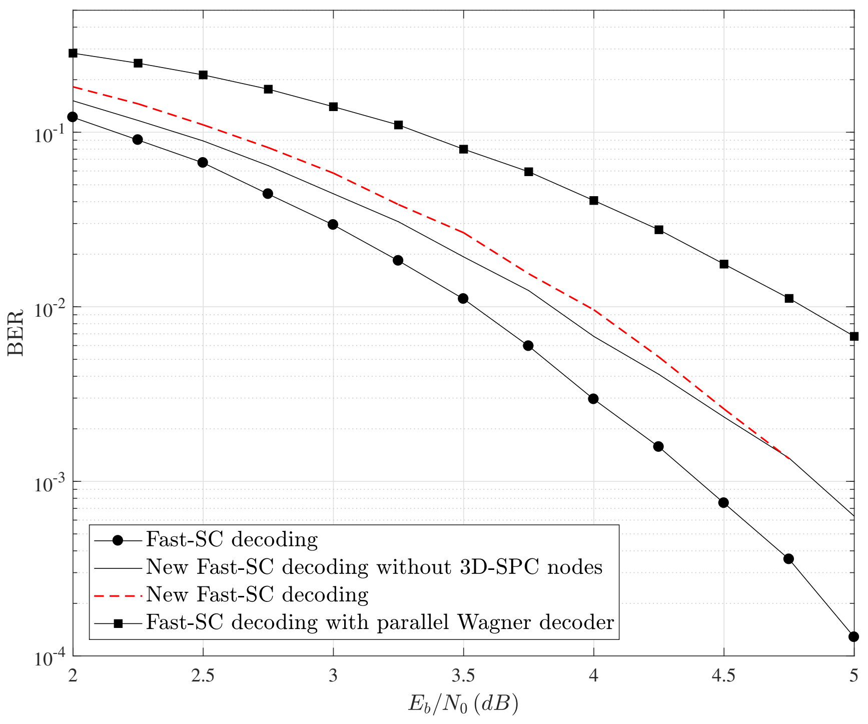
To compute the number of time steps we follow similar assumptions as in Section 3.4. With those assumptions, the Rate-0 and Rate-1 node will be decoded instantly. Moreover, REP and SPC nodes consumes one time step. The G-REP node takes one time step plus the time steps required to decode its rightmost children. Furthermore, the proposed 2D-SPC node consume one or two time steps, depending on the error pattern[3] . However, the number of time steps required to decode tD-SPC nodes when

[2]Due to the COVID-19 outbreak, the access to the high-speed computers at the university was limited. Hence, the simulations in Sections 5.3 and 6.3 were conducted using a limited number of trials. To achieve reliable results for lower BER values, a larger number of trials is required.

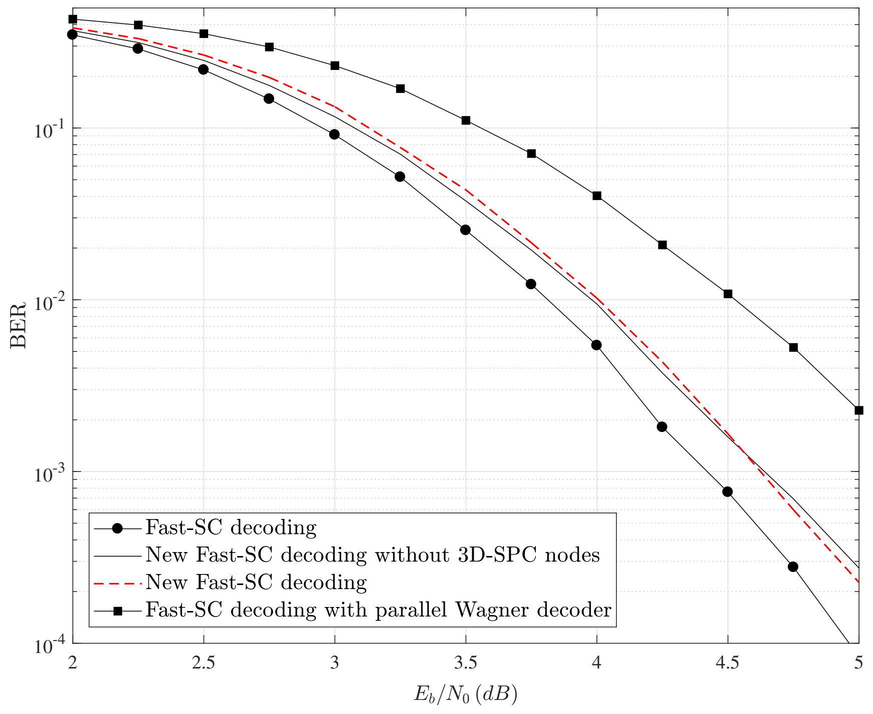
[3]In Table 5.1 the latency of 2D-SPC node decoding is considered to be 2 time steps.

Table 5.1: Required number of time steps for decoding.

N	**Rate**	**Fast-SC**	**+G-REP**	**+2D-SPC**	**+3D-SPC**	**Saving%**
81	1/3	24	24	21	21	12.5
	4/9	24	24	21	21	12.5
	1/2	34	32	29	29	14.7
	2/3	32	32	23	19	40.6
	5/6	18	16	16	16	11.11
243	1/3	62	62	53	53	14.5
	4/9	99	99	72	64	35.3
	1/2	91	91	64	52	42.8
	2/3	68	68	62	62	8.8
	5/6	52	52	37	33	36.5
729	1/3	188	181	148	136	27.6
	4/9	203	203	158	138	32
	1/2	185	185	149	137	25.9
	2/3	203	198	150	138	32.0
	5/6	118	118	94	86	27.1
2187	1/3	478	468	400	373	21.9
	4/9	485	485	395	371	23.5
	1/2	499	499	393	380	23.8
	2/3	532	532	385	321	39.6
	5/6	446	446	335	311	30.2



(a)



(b)

Fig. 5.3: The BER performances of the fast SC decoder with different fast nodes for (a) $P(243, 122)$ and (b) $P(729, 486)$.

$t > 2$ is variable as it depends on how many times the decoder revisits to correct the error patterns. Hence, we computed the average required time steps to decode a 3D-SPC nodes at BER around 10^{-3} for each case in Table5.1 through simulation. This average was found to be three time steps. Note that decoding the 3D-SPC node under Rate-0 and 2D-SPC nodes decoders will take five to seven time steps.

The G-REP nodes brought limited gain in terms of speed. This can be also observed in Table 5.1 as the number of time steps in the "+ G-REP" column is sometimes the same as the one in "Fast-SC" column. This is due to the fact that the number of G-REP nodes is small, regardless of the length and rate of the code. This is also observed in [8] when the $\mathbf{T}_2$ kernel is used.

On the other hand, the fast decoding of tD-SPC nodes can reduce the decoding latency significantly. It can be observed from Table 5.1 that the time step saving in most cases is more than 20%. As an example, adding 2D-SPC nodes increases the time step saving up to 29.7% when $N = 243$ and rate $= 1/2$; this saving increases further to 42.8% when 3D-SPC nodes are also added. This gain, however, comes at the cost of about 0.5 dB BER performance degradation, as seen in Fig. 5.3(a). Note that the performance loss is more significant when parallel Wagner decoders [8] are used to decode the 2D-SPC nodes. The performance loss of our proposed decoder is due to the hard decoding approach and is much smaller for $N = 729$. In particular, the BER performance loss brought by hard decoding of tD-SPC nodes is about 0.2 dB with $N = 729$ and rate 2/3, as illustrated in Fig. 5.3(b). In this case the combined time saving of adding G-REP and 2D-SPC nodes is 26.1% which grows to 32% by adding 3D-SPC nodes.

5.4 Conclusion

In this chapter, we identified a family of fast nodes in the decoding tree of polar codes constructed by two commonly used ternary kernels, $\mathbf{T_3}$ and $\mathbf{T_3'}$. We also presented fast parallel decoders for these nodes. The proposed decoders decrease the decoding latency with only a small performance degradation. Furthermore, we adapted the G-REP node, which was originally proposed for $\mathbf{T_2}$ kernel, to be fast decoded in the case of $\mathbf{T_3}$ and $\mathbf{T_3'}$ kernels.

The tD-SPC node can be also implemented in the fast SC decoding of multi-kernel polar codes, when the ternary kernels are ordered as the last components of the Kronecker product in the construction of multi-kernel codes.

Chapter 6

Modified REP pattern for 3×3 Kernel Polar Codes

Among the ternary kernels, $\mathbf{T_3} = \left[\begin{smallmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right]$ and $\mathbf{T'_3} = \left[\begin{smallmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{smallmatrix}\right]$, proved to be optimal for polarization in [36] and [40], respectively. Unlike the size-2 kernel, $\mathbf{T_3}$ and $\mathbf{T'_3}$ have a zero in their last rows. As such, in the corresponding codeword of a REP node some coded bits are zero rather than being a repetition of the data. This degrades the error correction performance of this node. Furthermore, different kernel orders in multi-kernel polar codes yield different zero locations for a REP node [30]. Thus, the fast decoders for a REP node in a multi-kernel polar code require storing all the zero-location patterns or computing them during the decoding procedure. This increases the memory requirements or the computational complexity of the decoders.

To address the above issues, we propose a modified REP node for polar codes constructed using $\mathbf{T_3}$ and $\mathbf{T'_3}$ kernels. The proposed modification eliminates zeros in the coded bits. This unifies the REP pattern in multi-kernel codes. We also present a low-complexity optimal decoder for the modified REP node. In addition, we apply our proposed modification to the G-REP nodes [8] to further improve the error-rate performance. Numerical results are provided to compare the performance of the multi-kernel polar codes and their modified ones.

6.1 Modified Repetition Nodes

A REP node in the decoding tree of the polar code has only one message bit m_0, at its rightmost leaf node. When the polar code is constructed using the $\mathbf{T}_2$ kernel, all coded bits of the REP node are repetitions of the m_0. Thus, the LLR of the rightmost leaf node is the sum of the LLRs in $\mathbf{y}^{\phi,t}$ as

$$y_{m_0} = \sum_{i=0}^{R-1} y_i^{\phi,t}, \tag{6.1}$$

where R is the REP node length and is equal to 2^t here. Further, the message bit is estimated by making a hard decision as $\hat{m}_0 = H\left(y_{m_0}\right)$ and $\mathbf{x}^{\phi,t}$ is obtained from the 2^t times repetition of $\hat{m}_0$.

When polar codes are constructed using $\mathbf{T_3}$ or $\mathbf{T'_3}$, the corresponding codeword to the REP node is $\mathbf{x}^{\phi,t} = m_0\mathcal{R}_3^{\otimes t}$, where $\mathcal{R}_3$ is $\{0\,1\,1\}$ for $\mathbf{T_3}$ and $\{1\,0\,1\}$ for $\mathbf{T'_3}$ and $\mathcal{R}_3^{\otimes t}$ is the t-th Kronecker product of $\mathcal{R}_3$. Further, y_{m_0} is given by

$$y_{m_0} = \sum_{i=0}^{R-1} y_i^{\phi,t}\mathcal{R}_3^{\otimes t}{}_i. \tag{6.2}$$

Here $R = 3^t$ and $\mathbf{x}^{\phi,t} = H\left(y_{m_0}\right)\mathcal{R}_3^{\otimes t}$.

In general, in a multi-kernel code constructed by binary and ternary kernels, the LLR of the rightmost bit of a REP node at stage $S = n_1 + n_2$ with length $R = 2^{n_1} \times 3^{n_2}$, is obtained by [30]

$$y_{m_0} = \sum_{i=0}^{R-1} y_i^{\phi,t}\mathcal{R}_i, \tag{6.3}$$

where $\mathcal{R}$ is the repetition pattern computed as

$$\mathcal{R} = \bigotimes_{i=0}^{S-1} \mathcal{R}_{k_i}. \tag{6.4}$$

In (6.4), $\mathcal{R}_{k_i}$ is $\mathcal{R}_2 = \{1\,1\}$ and $\mathcal{R}_3$ when i is a binary level or a ternary level, respectively.

From (6.4) it can be observed that in multi-kernel polar codes different kernel orders ($\mathbf{k}$) yield in several REP pattern. Some examples are provided in Table 6.1. Note that decoding different REP nodes requires the computation of $\mathcal{R}$ or storing all the patterns.

Table 6.1: Different REP patterns in multi-kernel codes.

R	**k**	**REP pattern with $\mathbf{T}_3$**	**REP pattern with $\mathbf{T'}_3$**
3	(3)	{0, 1, 1}	{1, 0, 1}
6	(2, 3)	{0, 1, 1, 0, 1, 1}	{1, 0, 1, 1, 0, 1}
6	(3, 2)	{0, 0, 1, 1, 1, 1}	{1, 1, 0, 0, 1, 1}
8	(2, 2, 2)	{1, 1, 1, 1, 1, 1, 1, 1}	{1, 1, 1, 1, 1, 1, 1, 1}
9	(3, 3)	{0, 0, 0, 0, 1, 1, 0, 1, 1}	{1, 0, 1, 0, 0, 0, 1, 0, 1}
12	(2, 2, 3)	{0, 1, 1, 0, 1, 1 0, 1, 1, 0, 1, 1}	{1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1}
12	(3, 2, 2)	{0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1}	{1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1}
18	(2, 3, 3)	{0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1}	{1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1}

To simplify the decoding, [30] proposes fast decoders that only decode REP patterns up to a certain maximum size and a certain number of ternary stages. This, in turn, reduces the decoding speed.

Furthermore, unlike $\mathbf{T_2}$, none of the coded bits are a repetition of the data when $\mathbf{T}_3$ or $\mathbf{T}_3^{'}$ are used. For instance, the corresponding codeword to a REP node of length 12 is $\{0, m_0, m_0, 0, m_0, m_0, 0, m_0, m_0, 0, m_0, m_0\}$ or $\{0, 0, 0, 0, m_0, m_0, m_0, m_0, m_0, m_0, m_0, m_0\}$, or $\{0, 0, m_0, m_0, m_0, m_0, 0, 0, m_0, m_0$ $, m_0, m_0\}$, depending on the order of $\mathbf{T_2}$ and $\mathbf{T_3}$ in the Kronecker product. Here four out of 12 coded bits are zero, thereby leading to error-rate performance degradation. This is because, instead of repeating the bit (in this case a maximum of 12 times), some zeros (in this case four zeros) are sent.

To eliminate zeros in the coded bits, we propose a modification to the REP node pattern. In particular, for the $\mathbf{T}_3$ kernel we suggest placing the data, m_0, in the leftmost descendant of the REP node rather than its rightmost one. Moreover, in the $\mathbf{T'}_3$ kernel we will set the frozen bits in the REP node to m_0 rather than zero. The suggested modifications eliminate zeros in the codeword and result in coded bits which are all repetitions of m_0, as in the case of binary kernel. With this modification, the Kronecker product computation in the fast estimation of REP node in (6.2) will be eliminated, thus eliminating the need for storing different patterns for the REP node in a multi-kernel polar code and, consequently, removing any limitation on the REP-node length in their fast decoding.

6.2 Modified G-REP node

The G-REP node is any node (ϕ, t) for which all children are Rate-0 nodes, except the rightmost descendant at level $p < t$. To fast decode the G-REP node based on (5.4) when only $\mathbf{T_3}$ or $\mathbf{T_3'}$ is used, either the Kronecker product must be computed or the pattern for different combinations of t and p must be stored. The number of possible patterns further increases based on the kernel orders in multi-kernel codes constructed with a combination of binary and ternary kernels. Moreover, similar to the REP node, some bits in the G-REP codeword are zeros and their LLRs are unused in the decoding procedure. For example, in a G-REP node of length 18, $\mathbf{k} = (2, 3, 3)$ and two message bits (m_1, m_0), the codeword is $\{0, 0, 0, m_1, m_0, m_0 + m_1, m_1, m_0, m_0 + m_1, 0, 0, 0, m_1, m_0, m_0 + m_1, m_1, m_0, m_0 + m_1\}$ when $\mathbf{T_3}$ is used. As such, six out of 18 coded bits are zero, which will degrade the error correction performance of the repetition code.

To address the above shortcomings we suggest modifying the G-REP node pattern. In particular, for the $\mathbf{T}_3$ kernel we exchange the pattern of the right child and left child at level p. Accordingly, the right child will become a Rate-0 node and the left child will contain some information bits. For the $\mathbf{T'}_3$ kernel, however, we suggest repeating the rightmost descendent pattern 3^{t-p} times, i.e., on all descendant at level p. In this case a number of frozen bits will repeat information bits rather than being frozen to zero. The suggested modification will thus eliminate all the zeros in the coded bits and simplify the decoding similar to that of binary polar codes by eliminating the computation of $\mathbf{L}$ in (5.4). In particular, for the ternary kernels polar codes the computation of the LLR vector of the rightmost descendent in (5.4) and the codeword estimation in (5.5) are simplified to

$$y_i^{\phi',p} = \sum_{j=0}^{3^{t-p}-1} y_{i+j3^p}^{\phi,t}, \tag{6.5}$$

and

$$\mathbf{x}^{\phi,t} = \{\underbrace{\mathbf{x}^{\phi',p}, ..., \mathbf{x}^{\phi',p}}_{3^{t-p}}\}. \tag{6.6}$$

As such, the LLR for G-REP nodes in the multi-kernel codes can be calculated as

$$y_i^{\phi',p} = \sum_{j=0}^{\frac{R}{R'}-1} y_{i+jR'}^{\phi,t}, \tag{6.7}$$

where R is the G-REP node length and R' is length of the rightmost descendent at level p and we have $\mathbf{x}^{\phi,t} = \{\underbrace{\mathbf{x}^{\phi',p}, ..., \mathbf{x}^{\phi',p}}_{\frac{R}{R'}}\}$.

Note that the rightmost descendent of a G-REP node may include REP or G-REP node of smaller length, which also must be modified. This case can be seen in the following example.

Example 6.1. *Fig. 6.1 illustrates an example of the suggested modifications. The original code tree is represented in Fig. 6.1(a). This G-REP node includes Rate-0, REP and SPC nodes. Here out of 27 coded bits, 11 bits are zeros. The modification in the code tree for the* $\mathbf{T}_3$ *kernel is shown in Fig. 6.1(b). It can be seen that both the REP and G-REP patterns are modified. On the other hand, Fig. 6.1(c) depicts the suggested changes for the* $\mathbf{T}_3^{'}$ *kernel where a crossed circle denotes a frozen bit that repeats an information bit.*

Remark 6.1. *Note that the proposed modifications are for the ternary-level REP and G-REP nodes only. As such, in a multi-kernel polar code, the binary-level REP and G-REP nodes are not modified. For example, in Fig. 6.2(a), the node* $(0, 3)$ *is a G-REP node. Since it is not a ternary-level node, we do not modify it. On the other hand, in Fig. 6.2(b) both the* $(0, 3)$ *and* $(1, 2)$ *nodes are G-REP nodes. In this case, the node* $(1, 2)$ *which corresponds to a ternary stage is modified only. The modified code tree for* $\mathbf{T_3}$ *kernel is illustrated in Fig. 6.2(c). Furthermore, in Fig. 6.2(d) nodes* $(0, 3)$, $(1, 2)$*, and* $(5, 1)$ *are REP nodes. However, only node* $(1, 2)$ *is modified because it is at a ternary level. This modification is provided in Fig. 6.2(e).*

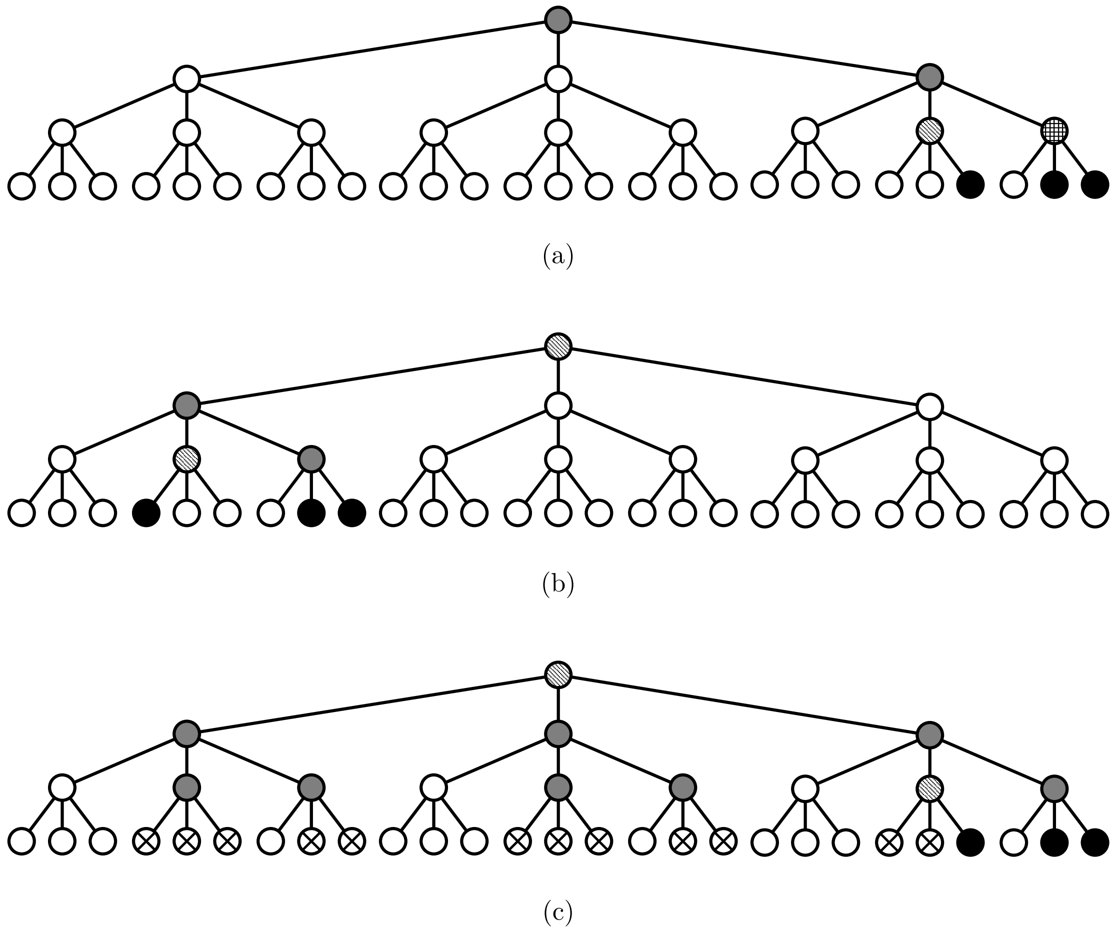
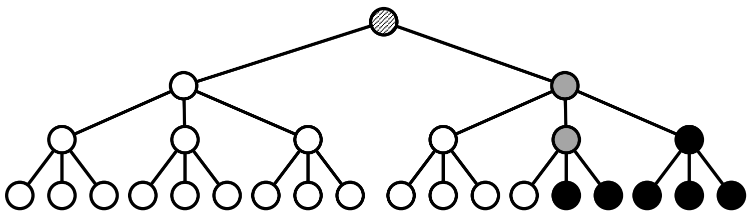
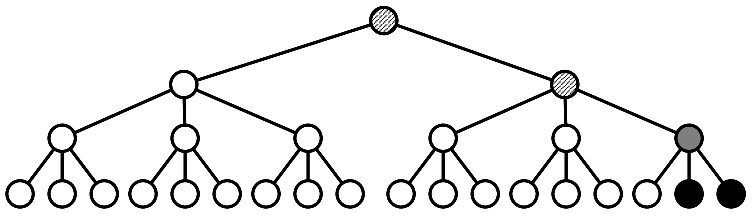


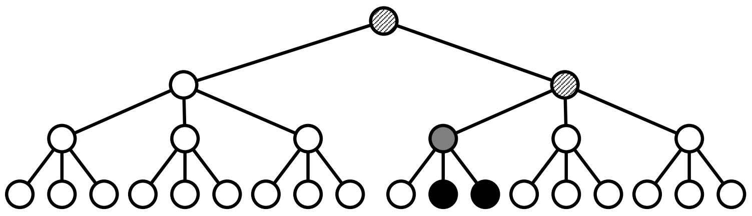
Fig. 6.1: An example of REP and G-REP (a) original pattern, (b) modified pattern for $\mathbf{T_3}$, and (c) modified pattern for $\mathbf{T_3'}$ in a ternary kernel polar code.



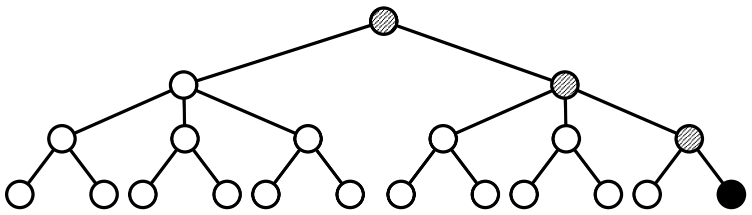
(a)



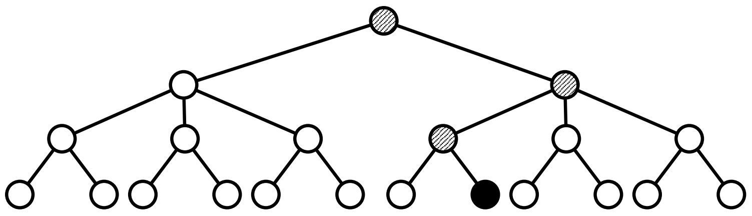
(b) .



(c)



(d)



(e)

Fig. 6.2: Examples of REP and G-REP nodes in multi-kernel code trees: (a) G-REP node which does not need modification, (b) a G-REP and (c) modified G-REP node in (b), (d) a REP node, and (e) modified REP node in (d).

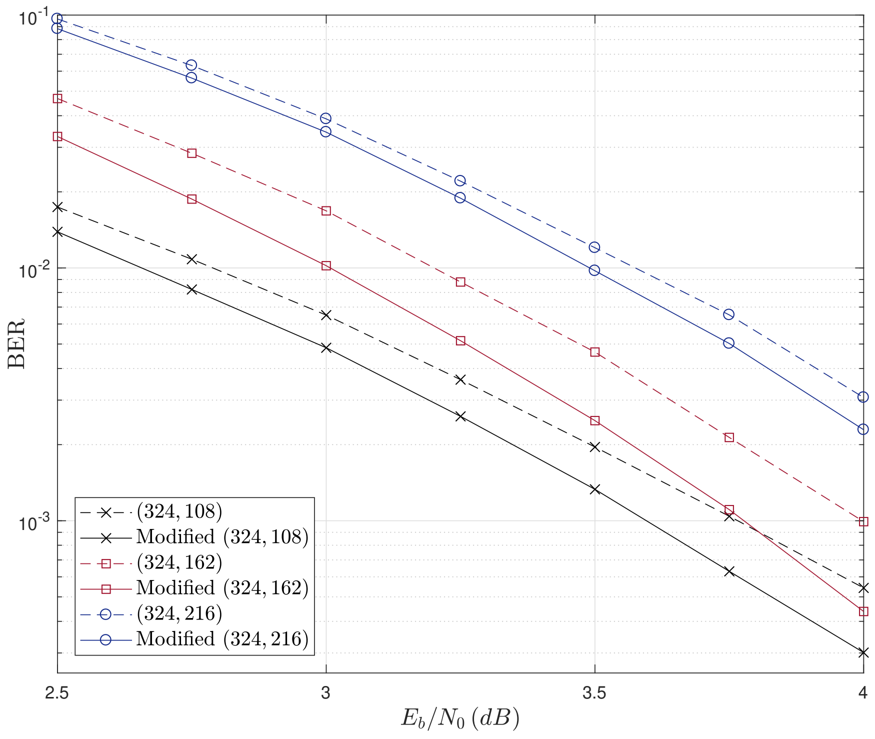
Table 6.2: Number and length of the modified REP and G-REP nodes.

$P(N,k)$	**Node length**	**REP**	**G-REP**
$(342,108)$	3	2	-
	9	1	-
$(342,162)$	3	4	-
	9	1	-
$(342,216)$	3	1	-
	9	1	-
$(576,192)$	3	5	-
	9	1	1
	18	1	-
$(576,288)$	3	3	-
	9	1	-
	36	1	-
	72	1	-
$(576,384)$	3	3	-
	9	1	-

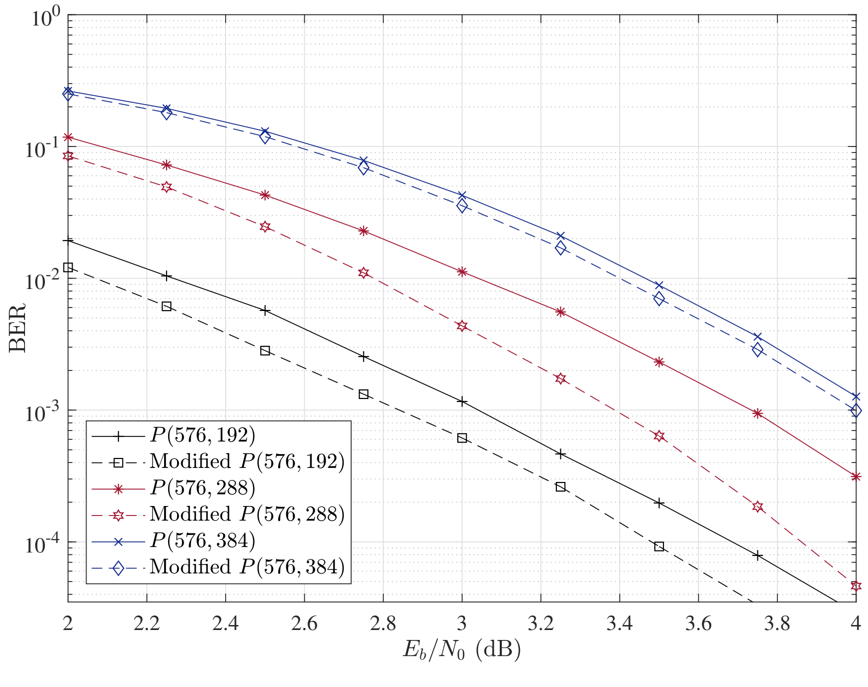
6.3 Simulation results

In this section, simulation results are provided to investigate the impact of the proposed modifications on the bit-error-rate (BER) performance. In our simulations, random BPSK-modulated codewords were transmitted through the AWGN channel. Here, we present the simulation results for the polar codes constructed by $\mathbf{T_3}$ only. A similar trend is depicted if $\mathbf{T_3'}$ is used instead.

Fig. 6.3 depicts the performance of two multi-kernel polar code with lengths $2^2 \times 3^4 = 324$ and $2^6 \times 3^2 = 576$ for different rates. It is seen that the modified codes at a BER= 10^{-3} show an improvement of 0.15 dB, 0.23 dB, and 0.1 dB when $N = 324$ and the code rate is 1/3, 1/2, and 2/3, respectively. Also, when $N = 576$ the modified codes show coding gains of 0.2 dB, 0.32 dB, and 0.05 dB when the code rate is 1/3,



(a)



(b)

Fig. 6.3: BER performance of a multi-kernel codes with length (a) 324 and (b) 576.

1/2, and 2/3, respectively. The amount of performance gain is related to the number and length of the REP and G-REP nodes which are modified, as shown in Table 6.2. Note that there are G-REP nodes in these multi-kernel code configurations which do not require modifications, as explained in Remark 6.1.

6.4 Conclusion

In this chapter, we identified an inefficiency in the polar codes constructed using the most commonly used ternary kernels ($\mathbf{T_3}$ and $\mathbf{T_3'}$) and proposed modifications for their REP and G-REP patterns. These modifications improve the error-rate performance of the code, unify the REP and G-REP node patterns in multi-kernel codes, and simplify the computations for their fast decoding.

Chapter 7

Summary and Future Work

7.1 Summary

The practical application of polar codes is beset by three main drawbacks: its mediocre error-rate performance for short to moderate length codes, its low decoding speed due to the serial nature of the SC decoder, and its lack of length flexibility.

The SCL and SCF decoders can be used to improve the performance of polar codes, especially for short to moderate length codes. However, their serial decoding nature results in high decoding latency. The decoding latency can be decreased by implementing some operations in parallel. In Chapter 3, we presented fast SCL decoders for five newly-identified nodes in the decoding tree of a polar code, i.e., Type-I, Type-II, Type-III, Type-IV and Type-V nodes. We also explained how fast SCL decoders can be used in the case of distributed PC-aided SCL decoders. The proposed decoders, while achieving the same error-rate performance, significantly reduce the decoding latency of the SCL decoder. Furthermore, in Chapter 4, we proposed fast SCF decoders. To that end, we first proposed a new decision metric that can be computed without a full decoding tree traversal. Using the proposed decision metric, we introduced fast parallel SCF decoders for many special nodes in the decoding tree of polar codes. The proposed fast SCF decoders achieve significant decoding speed improvement and, unlike the existing fast SCF decoders, show the same error correction performance as that of the original SCF decoder.

Length flexibility can be achieved in polar codes using a multi-kernel construction of the code, i.e., by using non-binary kernels in conjunction with the binary kernel. Among the non-binary kernels, the size-3 kernels have drawn much attention as they do not increase the decoding complexity significantly and have sufficiently high polarization exponents. Similar to the binary polar codes, multi-kernel codes are decoded by the SC decoder and suffer long decoding latency. Therefore, devising fast decoding solutions for non-binary kernels is necessary. In Chapter 5, we identify a new node, called the tD-SPC node, in the decoding tree of polar codes constructed by two commonly used size-3 kernels. We also propose a low-complexity hard decoder for this node. It was observed that the proposed node significantly improves the decoding latency at the expense of a slightly degraded error-rate performance. Furthermore, we adapt fast decoding of the G-REP nodes, introduced for the binary kernel, to the case of a ternary kernel.

The two specific size-3 kernels, $\mathbf{T_3}$ and $\mathbf{T_3'}$, which are known to have optimal polarization speed, contain one zero in their last rows. As such, the codeword corresponding to the REP and G-REP nodes have some zero elements. This results in some error-rate performance degradation. Moreover, different kernel orders in multi-kernel polar codes yield different zero locations for these two nodes. Thus, the fast decoders for the REP and G-REP nodes in a multi-kernel polar code are required to store all the zero-location patterns, which increases the memory requirements for the decoders. To eliminate these inefficiencies, in Chapter 6, we propose modifications to the encoding of the REP and G-REP patterns. The proposed modifications improve the error-rate performance, simplify the computations in the fast decoding of these two nodes and eliminate the need for storing different patterns for the REP and G-REP nodes in a multi-kernel polar code.

7.2 Future Work

7.2.1 Puncturing/Shortening of Multi-Kernel Polar Codes

The length flexibility of multi-kernel polar codes can be further improved by puncturing or shortening them. However, this subject is only briefly discussed in [33]. Both $\mathbf{T_3}$ and $\mathbf{T'_3}$ can be punctured, however, codeword shortening is only possible with the $\mathbf{T'_3}$ kernel. Also, unlike binary polar codes, the mother code is not strictly defined for puncturing/shortening of multi-kernel polar codes. It is possible to select longer codes with more binary stages or shorter codes where more ternary stages are incorporated. Different choices of mother code length will impact the decoding complexity and the error correction performance. Thus, more research needs to be carried out to find design criteria for selecting the mother code length and the patterns of punctured/shortened bits in multi-kernel codes.

7.2.2 Fast Decoding of Multi-Kernel Polar Codes

It is shown in [30] that the Rate-0, SPC, and Rate-1 nodes are compatible with multi-kernel polar codes constructed with the $\mathbf{T_2}$ and $\mathbf{T_3}$ kernels and can be decoded with fast decoders similar to those used for binary polar codes. Also, [30] presents different REP node patterns that can be seen in the multi-kernel code based on the order of the binary and ternary stages. We further adopt the G-REP node in the fast decoding of ternary polar codes. However, multiple other nodes are identified in the binary polar code tree and their compatibility with multi-kernel polar codes can be studied. Moreover, combining different size kernels may result in new node patterns, such as the tD-SPC node introduced in this work. Furthermore, shortening of the codes allows frozen bits on the right-hand side of the information bits, which generates new patterns. Thus, fast decoding of multi-kernel and shortened polar codes could be considered as a future research topic.

7.2.3 Machine Learning Application in Polar Code Decoding

A major focus of research in polar coding is to improve their complexity/latency. Machine learning techniques have proven successful in efficiently solving high-complexity decision-making problems. The problem of decoding polar codes can be cast as a decision-making problem. Hence, it could be solved using machine learning techniques. Recently, deep learning algorithms and specifically neural network-based decoders, have been considered as potential candidates to either replace or aid the existing polar codes decoders [64–71]. The one-shot decision-making approach of neural-network based decoders reduces the computational complexity and enables fast decoding of polar codes, however, their error-rate performance has been found to be not satisfactory [71]. Also, they are only feasible for short code lengths as the training complexity scales exponentially with the number of information bits [64]. Therefore, studies on the optimization of neural network based decoders to reduce the latency and complexity of polar code decoding while retaining the performance are of interest.

Bibliography

[1] E. Arıkan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] “Performance study of polar code candidates,” document R1-1703538, 1020 TSG-RANWG1 #88, 3GPP, Ericsson, Athens, Greece, Feb. 2017.

[3] M. Hanif, M. H. Ardakani, and M. Ardakani, “Fast list decoding of polar codes: Decoders for additional nodes,” in *IEEE Wireless Commun. Netw. Conf. Workshops*, Apr. 2018, pp. 37–42.

[4] M. H. Ardakani, M. Hanif, M. Ardakani, and C. Tellambura, “Fast successive-cancellation-based decoders of polar codes,” *IEEE Trans. on Commun.*, vol. 67, no. 7, pp. 4562–4574, 2019.

[5] A. Alamdar-Yazdi and F. R. Kschischang, “A simplified successive-cancellation decoder for polar codes,” *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011.

[6] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, “Fast polar decoders: Algorithm and Implementation,” *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 946–957, May 2014.

[7] M. Hanif and M. Ardakani, “Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes,” *IEEE Commun. Lett.*, vol. 21, no. 11, pp. 2360–2363, Nov. 2017.

[8] C. Condo, V. Bioglio, and I. Land, “Generalized fast decoding of polar codes,” in *IEEE Global Commun. Conf.*, Dec. 2018, pp. 1–6.

[9] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, “LLR-based successive cancellation list decoding of polar codes,” *IEEE Trans. Signal Process.*, vol. 63, no. 19, pp. 5165–5179, Oct. 2015.

[10] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, “A low-complexity improved successive cancellation decoder for polar codes,” in *IEEE Asilomar Conf. on Signals, Syst. Computers*, Nov. 2014, pp. 2116–2120.

[11] P. Giard and A. Burg, “Fast-SSC-flip decoding of polar codes,” in *IEEE Wireless Commun. Netw. Conf. Workshops*, Apr. 2018, pp. 73–77.

[12] C. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech.*, vol. 27, no. 3, pp. 379–423, Jul 1948.

[13] R. Gallager, “Low-density parity-check codes,” *IRE Trans. Info. Theory*, vol. 8, no. 1, pp. 21–28, Jan 1962.

[14] D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar 1997.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes.” in *IEEE Int. Conf. Commun.*, vol. 2, 1993, pp. 1064–1070.

[16] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[17] 3GPP, “NR; Multiplexing and Channel Coding,” http://www.3gpp.org/DynaReport/38-series.htm, Tech. Rep. TS 38.212, Tech. Rep. R1. 15, Jun. 2018.

[18] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[19] L. Chandesris, V. Savin, and D. Declercq, “Dynamic-SCFlip decoding of polar codes,” *IEEE Trans. Commun.*, vol. PP, no. 99, Jan. 2018.

[20] C. Condo, F. Ercan, and W. Gross, “Improved successive cancellation flip decoding of polar codes based on error distribution,” in *IEEE Wireless Commun. Netw. Conf. Workshops*, Apr. 2018, pp. 19–24.

[21] F. Ercan, C. Condo, S. A. Hashemi, and W. J. Gross, “Partitioned successive-cancellation flip decoding of polar codes,” in *IEEE Int. Conf. Commun.*, 2018, pp. 1–6.

[22] F. Ercan, T. Tonnellier, N. Doan, and W. J. Gross, “Simplified dynamic SC-flip polar decoding,” in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2020, pp. 1733–1737.

[23] C. Condo, V. Bioglio, and I. Land, “SC-flip decoding of polar codes with high order error correction based on error dependency,” in *IEEE Info. Theory Workshop*, 2019, pp. 1–5.

[24] Y. Pan, J. Li, and Y. Fang, “A new polar code SCF method based on effective flip bits,” in *IEEE Int. Conf. Electron. and Commun. Eng.*, 2019, pp. 108–111.

[25] K. Niu, K. Chen, and J. Lin, “Beyond turbo codes: Rate-compatible punctured polar codes,” in *IEEE Int. Conf. Commun.*, June 2013, pp. 3423–3427.

[26] S. Hong and M. Jeong, “An efficient construction of rate-compatible punctured polar (RCPP) codes using hierarchical puncturing,” *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5041–5052, 2018.

[27] R. Wang and R. Liu, “A novel puncturing scheme for polar codes,” *IEEE Commun. Lett.*, vol. 18, no. 12, pp. 2081–2084, Dec 2014.

[28] R. M. Oliveira and R. C. de Lamare, “Rate-compatible polar codes based on polarization-driven shortening,” *IEEE Commun. Lett.*, vol. 22, no. 10, pp. 1984–1987, 2018.

[29] F. Gabry, V. Bioglio, I. Land, and J. Belfiore, “Multi-kernel construction of polar codes,” in *IEEE Int. Conf. Commun. Workshops*, May 2017, pp. 761–765.

[30] A. Cavatassi, T. Tonnellier, and W. J. Gross, “Fast decoding of multi-kernel polar codes,” in *IEEE Wireless Commun. Netw. Conf.*, 2019, pp. 1–6.

[31] S. B. Korada, E. Şaşoğlu, and R. Urbanke, “Polar codes: Characterization of exponent, bounds, and constructions,” *IEEE Trans. on Info. Theory*, vol. 56, no. 12, pp. 6253–6264, 2010.

[32] V. Miloslavskaya and P. Trifonov, “Design of binary polar codes with arbitrary kernel,” in *IEEE Info. Theory Workshop*, 2012, pp. 119–123.

[33] T. Tonnellier, A. Cavatassi, and W. J. Gross, “Length-compatible polar codes: A survey,” in *Annual Conf. on Info. Sci. and Syst.*, 2019, pp. 1–6.

[34] L. Zhang, Z. Zhang, and X. Wang, “Polar code with block-length $N = 3^n$,” in *Int. Conf. Wireless Commun. and Signal Process.*, 2012, pp. 1–6.

[35] Y. Cao, H. Zhang, and Q. Tu, “Decoding performance analysis of good-channel relaxed polar codes with 3×3 kernel matrix,” *IET Commun.*, vol. 13, no. 8, pp. 1131–1139, 2019.

[36] M. Benammar, V. Bioglio, F. Gabry, and I. Land, “Multi-kernel polar codes: Proof of polarization and error exponents,” in *IEEE Info. Theory Workshop*, Nov. 2017, pp. 101–105.

[37] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, “Fast list decoders for polar codes,” *IEEE J. Sel. Areas Commun.*, vol. 34, no. 2, pp. 318–328, Feb. 2016.

[38] S. A. Hashemi, C. Condo, and W. J. Gross, "A fast polar code list decoder architecture based on sphere decoding," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 12, pp. 2368–2380, Dec. 2016.

[39] ——, "Fast and flexible successive-cancellation list decoders for polar codes," *IEEE Trans. on Signal Process.*, vol. 65, no. 21, pp. 5756–5769, Nov. 2017.

[40] H. Lin, S. Lin, and K. A. S. Abdel-Ghaffar, "Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents," *IEEE Trans. on Info. Theory*, vol. 61, no. 10, pp. 5253–5270, 2015.

[41] E. Arıkan, "Systematic polar coding," *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 860–862, Aug. 2011.

[42] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.

[43] E. Arıkan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Commun. Lett.*, vol. 12, no. 6, pp. 447–449, Jun. 2008.

[44] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," in *IEEE Int. Symp. Inf. Theory*, Jun. 2009, pp. 1496–1500.

[45] ——, "Performance of polar codes with the construction using density evolution," *IEEE Commun. Lett.*, vol. 13, no. 7, pp. 519–521, Jul. 2009.

[46] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3221–3227, Nov. 2012.

[47] H. Li and J. Yuan, "A practical construction method for polar codes in AWGN channels," in *IEEE Tencon*, Apr. 2013, pp. 223–226.

[48] D. Wu, Y. Li, and Y. Sun, "Construction and block error rate analysis of polar codes over AWGN channel based on Gaussian approximation," *IEEE Commun. Lett.*, vol. 18, no. 7, pp. 1099–1102, Jul. 2014.

[49] T. Richardson, T. Urbanke, and P. Urbanke, *Modern Coding Theory.* New York, NY, USA: Cambridge University Press, 2008.

[50] H. Vangala, E. Viterbo, and Y.Hong, “A comparative study of polar code constructions for the AWGN channel,” *arXiv:1501.02473.*

[51] P. Trifonov, “Randomized chained polar subcodes,” in *IEEE Wireless Commun. Netw. Conf. Workshops*, Apr. 2018, pp. 25–30.

[52] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” in *IEEE Int. Conf. on Acoust., Speech and Signal Process.*, May 2011, pp. 1665–1668.

[53] P. Elias, “List decoding for noisy channels,” Research Laboratory of Electronics, MIT, Tech. Rep. 335, 1957.

[54] V. Guruswami, “List decoding of error-correcting codes,” Ph.D. dissertation, MIT, Cambridge, MA, 2001.

[55] R. Silverman and M. Balser, “Coding for constant-data-rate systems,” *IRE Trans. Prof. Group Inform. Theory*, vol. 4, no. 4, pp. 50–63, Sep. 1954.

[56] N. Presman, O. Shapira, and S. Litsyn, “Mixed-kernels constructions of polar codes,” *IEEE J. Sel. Areas Commun.*, vol. 34, no. 2, pp. 239–253, 2016.

[57] V. Bioglio and I. Land, “On the marginalization of polarizing kernels,” in *IEEE Int. Symposium on Turbo Codes Iterative Info. Process.*, 2018, pp. 1–5.

[58] T. Wang, D. Qu, and T. Jiang, “Parity-check-concatenated polar codes,” *IEEE Commun. Lett.*, vol. 20, no. 12, pp. 2342–2345, Dec. 2016.

[59] H. Zhang, R. Li, J. Wang, S. Dai, G. Zhang, Y. Chen, H. Luo, and J. Wang, “Parity-check polar coding for 5G and beyond,” in *IEEE Int. Conf. Commun.*, May 2018, pp. 1–7.

[60] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Flexible and low-complexity encoding and decoding of systematic polar codes," *IEEE Trans. Commun.*, vol. 64, no. 7, pp. 2732–2745, Jul. 2016.

[61] K. Niu, K. Chen, J. Lin, and Q. T. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 192–203, Jul. 2014.

[62] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electron. Lett.*, vol. 51, no. 10, pp. 762–763, May 2015.

[63] D. M. Rankin and T. A. Gulliver, "Single parity check product codes," *IEEE Trans. on Commun.*, vol. 49, no. 8, pp. 1354–1362, 2001.

[64] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *IEEE Global Commun. Conf.*, 2017, pp. 1–6.

[65] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE J. Sel. Topics in Signal Process.*, vol. 12, no. 1, pp. 119–131, 2018.

[66] N. Doan, S. A. Hashemi, E. N. Mambou, T. Tonnellier, and W. J. Gross, "Neural belief propagation decoding of CRC-polar concatenated codes," in *IEEE Int. Conf. on Commun.*, 2019, pp. 1–6.

[67] W. Xu, Z. Wu, Y. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *IEEE Int. Workshop on Signal Process. Syst.*, 2017, pp. 1–6.

[68] N. Doan, S. A. Hashemi, and W. J. Gross, "Neural successive cancellation decoding of polar codes," in *IEEE Int. Workshop Signal Process. Adv. in Wireless Commun.*, Jun. 2018, pp. 1–5.

[69] N. Doan, S. A. Hashemi, F. Ercan, T. Tonnellier, and W. J. Gross, "Neural dynamic successive cancellation flip decoding of polar codes," in *IEEE Int. Workshop on Signal Process. Syst.*, 2019, pp. 272–277.

[70] J. Gao, K. Niu, and C. Dong, “Learning to decode polar codes with one-bit quantizer,” *IEEE Access*, vol. 8, pp. 27 210–27 217, 2020.

[71] H. Zhu, Z. Cao, Y. Zhao, and D. Li, “Learning to denoise and decode: A novel residual neural network decoder for polar codes,” *IEEE Trans. on Veh. Technol.*, pp. 1–1, 2020.