Low-Complexity Decoding of Short Linear Block Codes with Machine Learning

Nghia Doan

Department of Electrical and Computer Engineering
McGill University
Montreal, Canada

May 2022

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

The fifth Generation of Cellular Communications Standard (5G) consists of various application scenarios where each scenario prioritizes different performance requirements, including reliability, latency, and energy efficiency. A common design choice of the coding schemes in the control channels of 5G is to utilize short to moderate linear block codes to satisfy the stringent requirements of the 5G standard. In 5G short to moderate polar codes concatenated with Cyclic Redundancy Check (CRC) codes are used in the Enhanced Mobile Broadband (eMBB) and Ultra-Reliable Low-Latency Communication (URLLC) scenarios, and are being evaluated for the massive Machine-Type Communication (mMTC) scenario. Recently, Reed-Muller (RM) codes have regained significant research interests due to their similarity with polar codes and their excellent error-correction performance under (near) Maximum-Likelihood (ML) decoding. There are two main decoding algorithms for polar and RM codes, namely Successive-Cancellation (SC) and Belief Propagation (BP). Although SC decoding can provide a low-complexity implementation, its poor error-correction performance for short to moderate polar codes does not satisfy the requirements of the 5G standard. To improve the error-correction performance of SC decoding, Successive-Cancellation List (SCL) decoding was introduced. However, with a large list size, the good error-correction performance of SCL decoding comes at the cost of high computational complexity and memory requirement. Unlike the sequential nature of SC-based decoding algorithms, BP decoding is an iterative message passing algorithm that allows parallel computation enabling the decoder to reach high decoding throughput. However, with limited number of iterations, BP decoding of polar codes suffers from a poor error-correction performance. In this thesis, we address the underlying problem of SCL decoding by proposing novel bit-flipping and permutation decoding techniques tailored to SC-based decoding. We empirically show that compared to the state-of-the-art SCL-based decoders, the proposed SC-based decoders can achieve similar error-correction performance while significantly reducing the decoding latency, computational complexity, and memory requirements of the state-of-the-art SCL-based decoders. To address the poor error-correction performance of BP decoding, novel techniques that utilize the CRC factor-graph of the CRC-polar concatenated codes and codeword permutations are proposed. Different performance metrics of the proposed decoders are compared with those of state of the art. Furthermore, throughout the thesis, efficient machine leaning algorithms are used as the main optimization technique of the proposed decoders.

Résumé

La cinquième génération de la communication mobile (5G) se compose de divers scénarios d'applications où chaque scénario priorise différentes exigences de performance, y compris la fiabilité, la latence et le rapport énergétique. Un choix de conception commun des schémas de codage utilisés dans les canaux de contrôle de la 5G consiste à utiliser des codes de blocs linéaires courts ou modérés, qui produisent un compromis raisonnable pour satisfaire aux exigences strictes de la norme 5G. En particulier, les codes polaires courts ou modérés concaténés avec les codes de contrôle de redondance cyclique (CRC) sont utilisés dans les scénarios de débit mobile amélioré (eMBB) et de communication ultra-fiable à faible latence (URLLC) et sont en cours d'évaluation pour la communication massive de type machine (mMTC). Récemment, les codes de Reed-Muller (RM) ont regagné beaucoup d'intérêts dans la recherche en raison de leur similitude avec les codes polaires et de leurs excellentes performances de correction d'erreur sous le décodage à vraisemblance (presque) maximale (ML). Il existe deux principaux algorithmes de décodage pour les codes polaires et RM qui sont l'annulation successive (SC) et la propagation des croyances (BP). Bien que le décodage SC puisse produire une implémentation de faible complexité, sa mauvaise performance en correction d'erreurs pour les codes polaires courts ou modérés codes ne satisfait pas aux exigences de la norme 5G. Pour améliorer les performances en correction d'erreurs du décodage SC, le décodage par liste d'annulations successives (SCL) a été introduit. Cependant, avec une grande taille de listes, une bonne performance en correction d'erreurs du décodage SCL est atteinte au coût d'une complexité élevée des calculs et suffisamment de mémoire. Contrairement à la nature séquentielle des algorithmes de décodage basés sur le SC, le décodage BP est un algorithme de passage de message qui permet un calcul parallèle, il permet ainsi au décodeur d'atteindre un débit de décodage élevé. Cependant, avec un nombre limité d'itérations, le décodage BP appliqué aux codes polaires produit une mauvaise performance en correction d'erreurs. Dans cette thèse, nous abordons le problème sous-jacent du décodage SCL en proposant de nouvelles techniques de décodage par basculement de bits et par permutation adaptées au décodage SC. Nous montrons de façon empirique que par rapport aux décodeurs SCL de l'état de l'art, les décodeurs proposés basés sur le SC peuvent atteindre des performances de correction d'erreur similaires tout en réduisant considérablement la latence de décodage, la complexité de calcul et les exigences en mémoire des décodeurs SCL de l'état de l'art. Pour remédier aux mauvaises performances de correction d'erreur du décodage BP, de nouvelles techniques utilisant le graphe factoriel CRC des codes concaténés

polaires CRC et des permutations de codes sont proposées. Différentes métriques de performance des décodeurs proposés basés sur le BP sont comparées à celles de l'état de l'art. En outre, tout au long de la thèse, des algorithmes efficaces d'apprentissage automatique sont utilisés comme principale technique d'optimisation des décodeurs proposés.

Acknowledgments

First and foremost, I would like to thank my PhD supervisor, Professor Warren Gross, who has given me the chance to pursuit my PhD studies. Without his guidance and support, I will not be able to overcome unforeseen challenges during my PhD journey. Secondly, I would like to thank my PhD supervisory committee members, Professor Brett Meyer and Professor Ioannis Psaromiligkos, for their helpful and constructive advice, from which I can improve myself greatly to fulfill various challenging milestones of the degree. I would also like to thank Professor Hyuk-Jae Lee at Seoul National University, from whom I first learn how to conduct research.

I would like to thank the current and previous members of the Integrated Systems for Information Processing lab: Carlo Condo, Seyyed Ali Hashemi, Arash Ardakani, Furkan Ercan, Adam Cavatassi, Harsh Aurora, Jerry Ji, Loren Lugosch, Elie Mambou, Thibaud Tonnellier, Amir Ardakani, Syed Mohsin Abbas, Jiajie Li, Marwan Jalaleddine, and Charles Le. Thank you for your support and for the unforgettable time we shared at McGill.

I would like to give special thanks to Seyyed Ali Hashemi, who has been accompanying me in almost all of my research papers. He is a knowledgeable senior and a great friend, who I am very grateful to know and work with. I would also like to thank Elie Mambou again for his remarkable help in translating the thesis abstract into French.

Finally, my deep and sincere gratitude to my family for their continuous and unconditional love, help and support. No words are enough to express my love to them. Above all, I would like to thank my wife, Ngoc. Thank you mom, dad, Nancy, and Henry.

List of Acronyms

5G fifth Generation of Cellular Communications Standard

ARQ Automatic Request-for-Repeat

AWGN Additive White Gaussian Noise

BCH Bose-Chaudhuri-Hocquenghem

BP Belief Propagation

CPBP CRC-Polar BP

CRC Cyclic Redundancy Check

DL Deep Learning

DL-SCL Deep-Learning-Aided SCL

DSCF dynamic SC-Flip

eMBB Enhanced Mobile Broadband

Fast-SCF Fast SC-Flip

Fast-SCLF Fast SCL-Flip

FEC Forward Error Correction

FER Frame Error Rate

FHT Fast Hadamard Transforms

FHT-FSCL FHT-aided FSCL

FSC Fast SC

FSCL Fast Successive-Cancellation List

GANs Generative Adversarial Networks

KO Kronecker Operations

LDPC Low-Density Parity-Check

LLR Log-Likelihood Ratio

MIMO Multiple-Input Multiple Output

ML Maximum-Likelihood

mMTC massive Machine-Type Communication

NCPBP Neural CPBP

NNMS Neural Normalized Min-Sum

NNMS-RNN Neural Normalized Min-Sum Recurrent Neural Network

NSC Neural SC

OFDM Orthogonal Frequency Division Multiplexing

p-FHT-FSCL Permuted FHT-FSCL

RLD Recursive List Decoding

RM Reed-Muller

RPA Recursive Projection Aggregation

SC Successive-Cancellation

SCL Successive-Cancellation List

SCLF SCL-Flip

SNR Signal-to-Noise Ratio

SP Successive Permutations

SP-RLD SP-aided RLD

SRPA Sparse RPA

SSP-RLD simplified SP-RLD

URLLC Ultra-Reliable Low-Latency Communication

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In digital communications, signals are transmitted from a transmitter to a receiver through a channel, such as optical cables, electric wires, air, etc. Fig. 1.1 illustrates a basic block diagram of a digital communication system. In practice, the transmission channel is not ideal and often contains noise that corrupts the signals. To enable a reliable transmission, redundancy in terms of parity bits is added to the transmitted codeword, which allows for error-detection and/or error-correction of the received codeword. The majority of coding techniques for error prevention may be categorized into the set of Automatic Request-for-Repeat (ARQ) and the set of Forward Error Correction (FEC) schemes [5]. In ARQ schemes, the role of the codes is to detect whether or not the received codeword is corrupted. If the received codeword is corrupted, the receiver then requests a retransmission of the same codeword to the transmitter. The codes considered in this approach are referred as *error-detection codes*. In FEC schemes, the codes are designed to introduce redundancy in the form of an encoded codeword, which enables error correction through an efficient decoding algorithm. The codes in this approach are referred as *error-correction codes*. In practical scenarios, the *hybrid ARQ/FEC schemes* are often used in which a retransmission is requested if error correction is declared not successful on the received codeword.

Shannon's theorem [6] states that there is a maximum information rate at which a reliable communication can be established over a channel with a known error probability or Signal-to-Noise Ratio (SNR). There exist many FEC schemes in the literature that can achieve or closely approach the Shannon limit. Reed-Muller (RM) [7, 8] codes, discovered by Muller and Reed in 1954, were proven to achieve the capacity of erasure channels thanks to their large symmetry group [9]. Recently, RM codes are also proved to achieve the channel capacity of binary memoryless

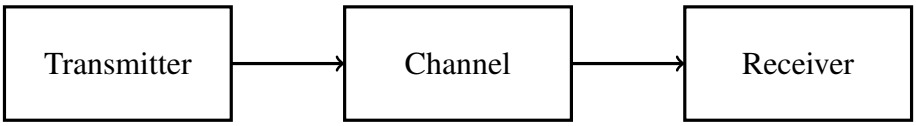


Figure 1.1: Basic digital communication system diagram.

channels [10]. Low-Density Parity-Check (LDPC) codes were first introduced by Gallager [11] in 1962 and then independently reintroduced by MacKay in 1995 [12], which have been shown to reach the channel capacity under some specific configurations [13]. In 1993, Turbo codes [14] were introduced by Berrou as a powerful FEC scheme that closely approaches the Shannon limit. In 2009, Arıkan proposed polar codes in his seminal paper [15], which are the first class of error-correction codes proven to achieve the channel capacity of any binary memoryless channel under efficient encoding and decoding algorithms. Because of this property, polar codes have drawn a great deal of attention from industry and academia and were adapted in the fifth Generation of Cellular Communications Standard (5G).

The 5G standard consists of various application scenarios where each scenario prioritizes different performance requirements, namely reliability, latency, and energy efficiency. A common design choice of the coding schemes in the control channels of 5G is to utilize short to moderate linear block codes, which provide a reasonable trade-off among the stringent requirements of low latency and high reliability specified by the 5G standard. In particular, short to moderate length polar codes are used in the Enhanced Mobile Broadband (eMBB) and Ultra-Reliable Low-Latency Communication (URLLC) scenarios, and are being evaluated for the massive Machine-Type Communication (mMTC) scenario [16, 17]. Fig. 1.2 shows the three scenarios of the 5G standard with some of their main characteristics and applications [18].

Recently, Reed-Muller (RM) codes have regained significant research interests due to their similarity with polar codes and their excellent error-correction performance under (near) Maximum-Likelihood (ML) decoding. RM codes are similar to polar codes in the sense that the generator matrices of both codes are constructed by selecting rows from a Hadamard matrix. The row selection of polar codes minimizes the error probability under SC decoding, while the row selection of RM codes maximizes the minimum distance of the codes. As a result, polar codes outperform RM codes under SC decoding and RM codes outperform polar codes under ML decoding, which can be approximated by an SCL decoder with a large list size.

The Successive-Cancellation (SC) decoding algorithm of polar and RM codes can provide

a low complexity implementation. However, SC decoding falls short in providing a reasonable error-correction performance for short to moderate length polar codes. SC list (SCL) decoding was introduced in [1,19,20,21,22] to improve the error-correction performance of SC decoding by keeping a list of candidate message words at each decoding step. In addition, it was observed that under SCL decoding, the error-correction performance is significantly improved when the polar code is concatenated with a cyclic redundancy check (CRC) [1, 21, 22]. Furthermore, SC-based decoding of polar codes can be represented as a binary tree traversing problem [23] and it was shown that the decoders in [1, 15, 21, 22] experience a high decoding latency as they require a full binary tree traversal. Several fast decoding techniques were introduced to improve the decoding latency of the conventional SC and SCL decoding algorithms [3, 24, 25, 26, 27]. The decoding operations of special constituent codes under the fast SC-based decoding algorithms proposed in [3, 24, 25, 26, 27] can be carried out at the parent node level, thus reducing the decoding latency caused by the tree traversal.

As the memory requirement of SCL decoding grows linearly with the list size [28], it is of great interest to improve the decoding performance of SCL decoding with a small list size. A solution for the aforementioned problem is to improve the error-correction performance of SCL decoding with a small list size by performing the SC and SCL decoding algorithms multiple times. Specifically, given that the first SC or SCL decoding attempt is not successful, a search set of possible erroneous decoding decisions is constructed for the first SC-based decoding attempt [29, 30, 31]. Then, the decoder performs the secondary decoding attempts in series, in which the estimated erroneous decisions are reversed at each additional decoding attempt. The decoding terminates if a codeword found in the secondary decoding attempts satisfies the CRC verification or if a predefined number of decoding attempts has been reached. This line of decoding algorithms is referred to as bit-flipping algorithms for SC [29] and SCL [30, 31] decoders. Note that shifting from searching in the list dimension to searching in the erroneous decoding decisions results in an increase in the number of maximum decoding attempts. However, it was observed in [29, 30, 31] that at moderate to high SNRs the decoders in [29, 30, 31] only incur a negligible increase in the number of average decoding attempts compared to that of the conventional SC and SCL decoders. This allows for a high-performance and low-power decoding algorithm which is highly suitable for the mMTC use case of the 5G standard [32]. Nevertheless, the bit-flipping algorithms in [29, 30, 31] require costly exponential and logarithmic computations that prevent the algorithms to be attractive for practical applications. In addition, all the decoders in [29, 30, 31] fully traverse the polar code decoding tree

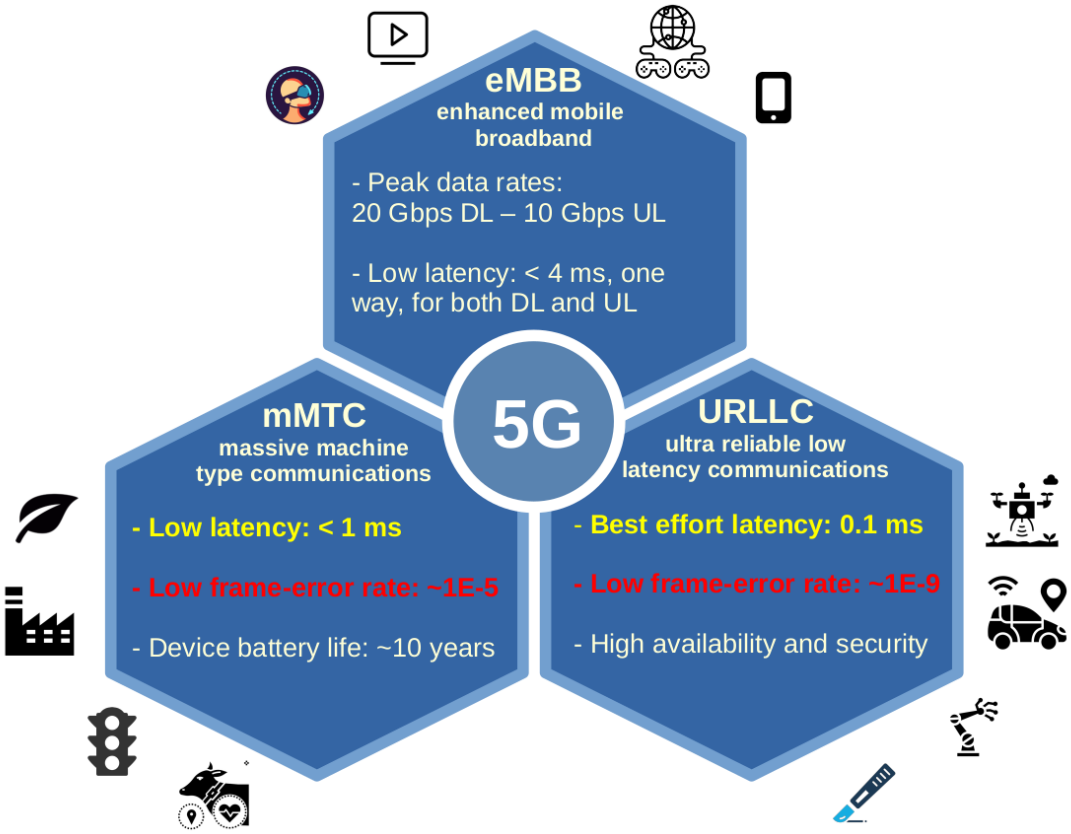


Figure 1.2: 5G application categories.

as required by the conventional SC-based decoding, thus resulting in a high decoding latency.

Unlike the sequential nature of SC-based decoding algorithms, Belief Propagation (BP) decoding [33] is a message passing algorithm that allows parallel computations enabling the decoder to reach high decoding throughput [34]. Furthermore, in some communication systems that require a Turbo channel equalizer [14] to improve the error correction performance, a soft-input soft-output BP decoder is often required instead of a soft-input hard-output decoder [35]. However, with limited number of iterations, BP decoding of polar codes suffers from a poor error-correction performance. Several attempts have been carried out to improve the performance of BP decoding for polar codes used in the 5G standard, where a CRC is concatenated with polar codes. In [36], the CRC is used as an early termination criterion to prevent the BP decoder from processing unnecessary iterations when the correct codeword is found. In [37], a post-processing algorithm is presented that uses a CRC to detect false-converged errors. The problem associated with [36,37] is

that the CRC factor-graph is only considered for error-detection and not for error-correction, which resulted in a negligible error-correction performance improvement compared to the conventional BP decoding algorithm.

Decoding algorithms for polar codes can also be used to decode RM codes. However, they do not provide near-ML-decoding performance. In fact, state-of-the-art near-ML decoding algorithms, namely SC-based decoding [19, 20, 21, 38] and Recursive Projection Aggregation (RPA) [39], suffer from high computational complexity, decoding latency, and memory consumption, preventing RM codes to be suitable for practical applications that require high reliability with reasonable computational complexity.

Previous research concerning the decoding of polar codes mainly focuses on the eMBB use case of 5G with SCL decoding being the state-of-the-art decoder. However, when targeting different use cases of 5G, SCL decoding experiences a high computational complexity, decoding latency, and memory requirements, making polar codes less attractive to the mMTC and URLLC scenarios. Throughout the thesis, we introduce various novel decoding techniques to practically extend the use of polar and RM codes for different 5G use cases. In Chapter 3 of the thesis, a low-complexity and low-latency bit-flip decoding algorithm of polar codes is first introduced to address the high complexity issue of SCL decoding. This algorithm enables a low complexity decoder as opposed to SCL decoding, thus potentially provides a low-power decoding algorithm, which enables the use of polar codes under the mMTC scenario. In Chapter 4 of the thesis, an improvement of the decoders proposed in Chapter 3 is introduced. In particular, a fast list flip decoder is proposed to significantly reduce the worst-case latency of decoding polar codes under the conventional bit-flipping based algorithms. In Chapter 5, we further extend the use case of polar codes under a soft-input soft-output BP decoder as required by a Turbo-like communication system. Specifically, the CRC is concatenated with the polar codes and is utilized for both error-detection and error-correction to significantly improve the reliability and decoding latency of the polar-CRC concatenated codes. On the other hand, Chapter 6 and Chapter 7 of the thesis consider RM decoding, where the RM codes are referred to as a special case of polar codes. It is worth to note that RM codes are a potential coding scheme for the URLLC use case of 5G due to their excellent error-correction performance under near-ML decoding and their symmetry properties that enable highly-parallel decoding algorithms. In particular, novel decoding techniques utilizing Fast Hadamard Transforms (FHT) and codeword permutations of RM codes are introduced in Chapter 6. Furthermore, in Chapter 7, an improvement scheme of the decoding algorithm presented in

Chapter 6 is introduced, which proposes a novel permutation selection scheme for RM codes under an improved recursive list decoding algorithm.

1.1 Summary of Contributions

Throughout the thesis we consider the Additive White Gaussian Noise (AWGN) channel model. Nevertheless, the proposed algorithms can be utilized in other channel models, such as fading channels or channels with correlated noise as such non-Gaussian channel models are often coupled with precoding and channel equalizer techniques. The detailed contributions are summarized as follows:

1.1.1 Machine-Learning-Aided Successive-Cancellation Flip Decoding of Polar Codes

We introduce a hardware-friendly SC-Flip decoding algorithm to address existing issues of the state-of-the-art dynamic SC-Flip (DSCF) decoder [29], which utilizes an additive trainable parameter to estimate the error bit indices under SC decoding. A Fast SC-Flip (Fast-SCF) decoder is then introduced to obviate the binary-tree traversal of the conventional SC-Flip decoders. In addition, the bit-flipping model of the Fast-SCF decoder is parameterized by a correlation matrix, which provides a superior estimation accuracy of the error indices compared to the methods used by state-of-the-art SC-Flip decoders. The parameters of the proposed SC-Flip based decoders are optimized using efficient supervised and reinforcement learning techniques.

1.1.2 Fast Successive-Cancellation List Flip Decoding of Polar Codes

We develop a Fast SCL-Flip (Fast-SCLF) decoding algorithm for polar codes that addresses the high latency issue associated with the SCL-Flip (SCLF) decoding algorithm. We first propose a bit-flipping strategy tailored to the state-of-the-art Fast Successive-Cancellation List (FSCL) decoding that avoids tree-traversal in the binary tree representation of SCLF, thus reducing the latency of the decoding process. We then derive a parameterized path-selection error model to accurately estimate the bit index at which the correct decoding path is eliminated from the initial FSCL decoding. The trainable parameter is optimized online based on an efficient supervised learning framework. By using online learning, the parameter can be directly trained at the operating SNR of the decoder while completely removing the need of pilot signals.

1.1.3 Improved Belief Propagation Decoding of CRC-Polar Concatenated Codes

We propose novel decoding techniques to significantly improve the error-correction performance of CRC-polar concatenated codes under BP decoding. In particular, the CRC factor-graph is first utilized to provide extrinsic information to the polar factor-graph. Trainable weights are then assigned to the edges of the concatenated graphs to reduce the decoding latency. In addition, the code permutations are utilized to further reduce the error probability of the CRC-aided BP decoder. We formalize the factor-graph selections of polar codes under CRC-aided BP decoding as a multi-armed bandit problem and use state-of-the-art bandit algorithms to select the set of good permutations on the fly.

1.1.4 Decoding Reed-Muller Codes with Fast Hadamard Transforms

We propose a novel permuted fast successive-cancellation list decoding algorithm with fast Hadamard transform, which is referred as FHT-aided FSCL (FHT-FSCL). In particular, the proposed decoder performs the decoding operations in both the information bit and codeword permutation domains of RM codes to significantly improve the error-correction performance of a previously introduced FHT-FSCL decoding algorithm. First, the proposed decoder initializes L ($L \geq 1$) active decoding paths with L random codeword permutations sampled from the full symmetry group of the codes. The path extension in the permutation domain is then carried out until the first constituent RM code of order 1 is visited, followed by the conventional path extension occurred only in the information bit domain. Furthermore, as different subsets of the codeword permutations are utilized for the permuted FHT-FSCL decoder, the error-correction performance of RM codes can be significantly improved by running M ($M > 1$) permuted FHT-FSCL decoders in parallel.

1.1.5 Decoding Reed-Muller Codes with Successive Codeword Permutations

A novel Recursive List Decoding (RLD) algorithm of RM codes based on Successive Permutations (SP) of the codeword is presented. An SP scheme that performs maximum likelihood decoding on a subset of the symmetry group of RM codes is first proposed to carefully select a good codeword permutation on the fly. Then, the proposed SP technique is applied to an improved RLD algorithm that initializes different decoding paths with random codeword permutations, which are sampled from the full symmetry group of RM codes. Finally, an efficient latency reduction scheme is introduced that virtually preserves the error-correction performance of the proposed decoder.

1.2 Related Publications

This doctoral research has resulted in the following publications.

Book Chapter

1. W. J. Gross, **N. Doan**, E. N. Mambou, and S. A. Hashemi, "Deep Learning Techniques for Decoding Polar Codes", Wiley, 2019.

   This chapter provides the background and motivation for the use of deep learning in various forward error correction schemes used for wireless communication systems. My contributions to this book chapter were to review related papers, implement the state-of-the-art algorithms, obtain the simulation results, and help in writing the book chapter.

Journal Papers

1. **N. Doan**, S. A. Hashemi, M. Mondelli, and W. J. Gross, "Decoding Reed-Muller Codes with Successive Codeword Permutations", IEEE Transactions on Communications *(under review)*.

   This paper presents a novel approach to select a good codeword permutation of RM codes to significantly improve the error-correction performance of RLD decoding with a small list size. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 7.

2. **N. Doan**, S. A. Hashemi, and W. J. Gross, "Successive-Cancellation Decoding of Reed-Muller Codes with Fast Hadamard Transform", IEEE Transactions on Vehicular Technologies *(under review)*.

   This paper introduces an efficient near-ML decoding algorithm of RM codes by utilizing FHT and codeword permutations under FSCL decoding. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 6.

3. **N. Doan**, S. A. Hashemi, and W. J. Gross, “Fast Successive-Cancellation List Flip Decoding of Polar Codes,” IEEE Access, 2022.

   This paper addresses the underlying high decoding problem of the SCLF decoding algorithm [30]. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 4.

4. **N. Doan**, S. A. Hashemi, F. Ercan, T. Tonnellier, and W. J. Gross, "Neural Successive-Cancellation Flip Decoding of Polar Codes", Journal of Signal Processing Systems, 2021.

   This paper introduces a training parameter and an approximation scheme that completely removes the need to perform transcendental computations in DSCF decoding [29], with almost no error-correction performance degradation. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 3.

5. F. Ercan, T. Tonnellier, **N. Doan**, W. J. Gross, "Practical Dynamic SC-Flip Polar Decoders: Algorithm and Implementation", IEEE Transactions on Signal Processing, 2020.

   This paper proposes a fast SC-Flip decoding algorithm of polar codes and its hardware implementation. My contribution to the paper was to help in the preparation of the manuscript.

Conference Papers

1. **N. Doan**, S. A. Hashemi, F. Ercan, and W. J. Gross, "Fast SC-Flip Decoding of Polar Codes with Reinforcement Learning", IEEE International Conference on Communications (ICC), Montreal, Canada, 2021.

   This paper introduces a novel bit-flipping algorithm tailored to FSC decoding, which tackles the high decoding latency problem of SCF decoding [40]. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 3.

2. S. A. Hashemi, **N. Doan**, W. J. Gross, J. Cioffi, and A. Goldsmith, "A Tree Search Approach for Maximum-Likelihood Decoding of Reed-Muller Codes", IEEE Globecom: Workshop on Channel Coding beyond 5G (GLOBECOM-Workshop), Madrid, Spain, 2021.

   This paper presents an ML decoding algorithm of RM codes based on a tree search algorithm. My contributions to the paper were to help in developing the idea, implement the algorithm and obtain the simulation results.

3. T. Tonnellier, M. Hashemipour, **N. Doan**, W. J. Gross, and A. Balatsoukas-Stimming, "Towards Practical Near-Maximum-Likelihood Decoding of Error-Correcting Codes: An Overview", IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, Canada, 2021.

   This paper reviews recent advanced decoding algorithms that can obtain (near) ML performance of linear block codes. My contribution to this paper was to write a section of the manuscript, which reviews recent machine-learning aided decoding algorithms.

4. **N. Doan**, S. A. Hashemi, and W. J. Gross, "Decoding of Polar Codes with Reinforcement Learning", IEEE Global Communications Conference (GLOBECOM), Taipei, Taiwan, 2020.

   This paper addresses the problem of selecting factor-graph permutations of CRC-polar concatenated codes under BP decoding to significantly improve the error-correction performance of the codes. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 5.

5. F. Ercan, T. Tonnellier, **N. Doan**, and W. J. Gross, "Simplified Dynamic SC-Flip Polar Decoding", IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020.

   This paper proposes a fast SC-Flip decoding algorithm of polar codes. My contribution to the paper was to help in the preparation of the manuscript.

6. **N. Doan**, S. A. Hashemi, E. N. Mambou, T. Tonnellier, and W. J. Gross, "Neural Belief Propagation Decoding of CRC-Polar Concatenated Codes", IEEE International Conference on Communications (ICC), Shanghai, China, 2019.

This paper first proposes a CRC-Polar BP (CPBP) decoder by exchanging the extrinsic information between the factor graph of the polar code and that of the CRC. It then proposes a Neural CPBP (NCPBP) algorithm which improves the CPBP decoder by introducing trainable normalizing weights on the concatenated factor graph. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 5.

7. **N. Doan**, S. A. Hashemi, F. Ercan, T. Tonnellier, and W. J. Gross, "Neural Dynamic Successive Cancellation Flip Decoding of Polar Codes", IEEE International Workshop on Signal Processing Systems (SiPS), Nanjing, China, 2019.

   This paper tackles the high computational complexity of the state-of-the-art DSCF decoding algorithm [29]. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 3.

8. S. A. Hashemi, **N. Doan**, and W. J. Gross, "Deep-Learning-Aided Successive-Cancellation Decoding of Polar Codes", Asilomar Conference on Signals, Systems, and Computers (ASILOMAR), Pacific Grove, USA, 2019.

   In this paper, a Deep-Learning-Aided SCL (DL-SCL) decoding algorithm for polar codes is introduced. The DL-SCL decoder works by allowing additional rounds of SCL decoding when the first SCL decoding attempt fails using a novel bit-flipping metric. My contributions were to help in the implementation of the ideal and in the preparation of the manuscript.

9. **N. Doan**, S. A. Hashemi, M. Mondelli, and W. J. Gross, "On the Decoding of Polar Codes on Permuted Factor Graphs", IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, UAE, 2018.

   This paper shows that the permutations on the factor graph of polar codes can be mapped into suitable permutations on the codeword positions, allowing the use of a single decoder architecture with different factor-graph permutations. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript. The contributions to this paper are presented in Chapter 5.

10. **N. Doan**, S. Ali Hashemi and W. J. Gross, "Neural Successive Cancellation Decoding of Polar Codes", IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Kalamata, Greece, 2018.

    This paper proposes a Neural SC (NSC) decoder to overcome the high decoding latency issue associated with the partitioned neural network decoder [41]. My contributions to this paper were to develop and implement the idea, produce the results, and write the manuscript.

11. S. A. Hashemi, **N. Doan**, M. Mondelli, and W. J. Gross, "Decoding Reed-Muller and Polar Codes by Successive Factor Graph Permutations", IEEE International Symposium on Turbo Codes & Iterative Information Processing (ISTC), Hong Kong, China, 2018.

    This paper proposes an SP scheme that finds the permutations on the fly, thus the decoding always progresses on a single factor graph permutation. My contributions were to help in the implementation of the ideal and in the preparation of the manuscript.

1.3 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we introduce some background knowledge of polar and RM codes, as well as their state-of-the-art decoders. Chapter 3 introduces bit-flipping algorithms of SC and FSC decoding, while Chapter 4 proposes a bit-flipping algorithm tailored to FSCL decoding. In Chapter 5, novel decoding techniques are introduced to significantly improve the error probability of BP decoding under the CRC-polar concatenated codes. Chapter 6 and Chapter 7 deal with the decoding of RM codes under SC-based decoding by utilizing the rich symmetry group of the codes and FHT. Finally, concluding remarks and some future research directions are drawn in Chapter 8.

Chapter 2

Background

In this chapter we provide a brief background on the encoding and decoding of polar and RM codes. We start this chapter by first introducing notations. Throughout this thesis boldface letters indicate vectors and matrices, while unless otherwise specified non-boldface letters indicate either binary, integer or real numbers. In addition, by $\boldsymbol{a}_{i_{\min}}^{i_{\max}} = \{a_{i_{\min}}, \ldots, a_{i_{\max}}\}$ we denote a vector of size $i_{\max} - i_{\min} + 1$ containing the a elements from index $i_{\min}$ to $i_{\max}$ ($i_{\min} < i_{\max}$). Sets are denoted by blackboard bold letters, e.g., $\mathbb{R}$ is the set containing real numbers. Finally, $\mathbb{I}_X$ is an indicator function where $\mathbb{I}_X = 1$ if the condition X is true, and $\mathbb{I}_X = 0$ otherwise.

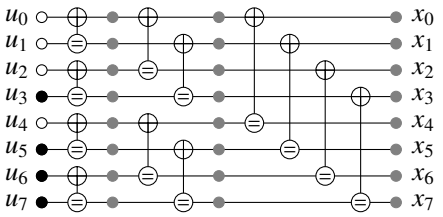
2.1 Encoding of Polar and RM codes

A polar code $\mathcal{P}(N, K)$ of length N with K information bits is encoded by applying a linear transformation to the binary message word $\boldsymbol{u} = \{u_0, u_1, \ldots, u_{N-1}\}$ as $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G}^{\otimes n}$, where $\boldsymbol{x} = \{x_0, x_1, \ldots, x_{N-1}\}$ is the codeword, $\boldsymbol{G}^{\otimes n}$ is the n-th Kronecker power of the polarizing matrix $\boldsymbol{G} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, and $n = \log_2 N$. The vector $\boldsymbol{u}$ contains a set $\mathcal{I}$ of K information bit indices and a set $\mathcal{I}^c$ of $N - K$ frozen bit indices. On the other hand, a RM code is specified by a pair of integers $0 \leq r \leq m$ and is denoted as $\mathcal{RM}(r, m)$, where r is the order of the code. $\mathcal{RM}(r, m)$ has a code length $N = 2^m$ with $K = \sum_{i=0}^{r} \binom{m}{i}$ information bits, and a minimum distance $d = 2^{m-r}$ [7, 8]. RM codes are similar to polar codes under the factor-graph representation of the codes, which allows them to share the same encoding algorithm.

The main difference between RM and polar codes is that the frozen-bit set $\mathcal{I}^c$ of RM codes is constructed to maximize the minimum distance among all the codewords [7, 8], while the set $\mathcal{I}^c$

$$\boldsymbol{G}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

(a)



(b)

Figure 2.1: (a) Generator matrix of $\mathcal{P}(8,4)$ and $\mathcal{RM}(1,3)$ and (b) its equivalent factor-graph representation with $\mathcal{I}^c = \{0,1,2,4\}$.

of polar codes is constructed to minimize the error probability under SC decoding [42, 43, 44, 45] or SCL decoding [46, 47]. Similar to polar codes, a RM code is encoded by applying a linear transformation to the binary message word $\boldsymbol{u}$ as $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G}^{\otimes m}$. The element u_i of $\boldsymbol{u}$ is fixed to 0 if the Hamming weight of the i-th row of $\boldsymbol{G}^{\otimes m}$, denoted as w_i, is smaller than d. Formally, $u_i = 0$ $\forall i \in \mathcal{I}^c$, where $\mathcal{I}^c = \{i | 0 \leq i < N, w_i < d\}$, and $\mathcal{I} = \{i | 0 \leq i < N, w_i \geq d\}$.

For both polar and RM codes, once constructed, the set $\mathcal{I}$ and $\mathcal{I}^c$ are known to both the encoder and the decoder. The codeword $\boldsymbol{x}$ is sent through the channel using a binary phase-shift keying (BPSK) modulation and an AWGN channel model is considered. Thus, the soft vector of the transmitted codeword received by the receiver is $\boldsymbol{y} = (\mathbf{1} - 2\boldsymbol{x}) + \boldsymbol{z}$, where $\mathbf{1}$ is an all-one vector of size N, and $\boldsymbol{z} \in \mathbb{R}^N$ is the noise vector with variance σ^2 and zero mean. In the Log-Likelihood Ratio (LLR) domain, the LLR vector of the transmitted codeword is

$$\boldsymbol{\alpha}_m = \frac{2\boldsymbol{y}}{\sigma^2}. \tag{2.1}$$

Given the generator matrix $\boldsymbol{G}^{\otimes m}$, the factor-graph representation of the code is constructed using the Forney's transformation introduced in [48]. In Fig. 2.1 we illustrate an example of the encoding of polar and RM codes of size 8 using their factor-graph representation [8, 15, 48, 49] given $\boldsymbol{G}^{\otimes 3}$. During the encoding, the $\bigoplus$ symbol indicates a binary XOR operation while the equality symbol $\circledequal$ indicates an equality operation.

The transmitted message word $\boldsymbol{u}$ is then estimated by a decoding algorithm given the LLR values $\boldsymbol{\alpha}_n$. In the following sections, we briefly review some of the mainstream decoding algorithms of polar and RM codes.

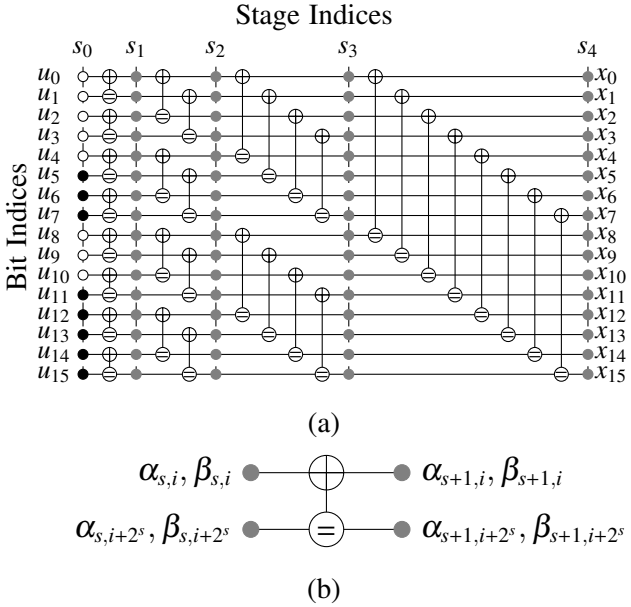


Figure 2.2: (a) Factor-graph representation of $\mathcal{P}(16, 8)$ with $\mathcal{I}^c = \{0, 1, 2, 3, 4, 8, 9, 10\}$, and (b) an SC PE.

2.2 Successive-Cancellation-Based Decoding

This section introduces preliminary knowledge and notations required by Chapter 3, Chapter 4, Chapter 6, and Chapter 7.

2.2.1 Successive-Cancellation and Successive-Cancellation List Decoding

SC decoding is executed on the factor-graph representation of the code [15]. Fig. 2.2(a) illustrates the factor-graph representation of $\mathcal{P}(16, 8)$ during the course of decoding. To obtain the message word, the soft LLR values and the hard bit estimations are propagated through all the SC processing elements (PEs) with respect to the parity-check and equality constraints, which are depicted in Fig. 2.2(b). Each SC PE performs the following computations: $\alpha_{s,i} = f(\alpha_{s+1,i}, \alpha_{s+1,i+2^s})$ and $\alpha_{s,i+2^s} = g(\alpha_{s+1,i}, \alpha_{s+1,i+2^s}, \beta_{s,i})$, where $\alpha_{s,i}$ and $\beta_{s,i}$ are the soft LLR value and the hard-bit estimation at the s-th stage and the i-th bit, respectively. The min-sum approximation formulations of f and g are $f(a, b) = \min(|a|, |b|)\operatorname{sgn}(a)\operatorname{sgn}(b)$, and $g(a, b, c) = b + (1 - 2c)a$. The soft LLR values at the m-th stage are initialized to α_m and the hard-bit estimation of an information bit at the 0-th stage is obtained as $\hat{u}_i = \beta_{0,i} = \frac{1-\operatorname{sgn}(\alpha_{0,i})}{2}$, $\forall i \in \mathcal{I}$. The hard-bit values of the SC PE are then

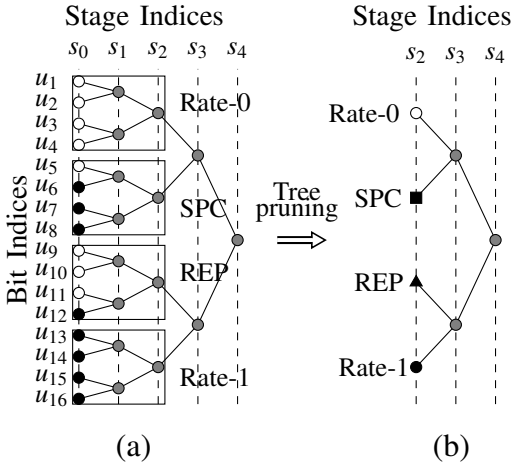


Figure 2.3: (a) Full binary tree representation of $\mathcal{P}(16, 8)$ and (b) its corresponding pruned tree using various special node types.

computed as $\beta_{s+1,i} = \beta_{s,i} \oplus \beta_{s,i+2^s}$ and $\beta_{s+1,i+2^s} = \beta_{s,i+2^s}$.

The SCL decoding algorithm was proposed in [1, 19, 21, 22] to decode polar and RM codes by maintaining L most probable SC decoding paths at the same time. It is noteworthy that the SC decoding algorithm in [15] is a special case of the recursive list algorithm proposed in [19], when the list size L is set to 1. Under SCL decoding, the estimation of a message bit $\hat{u}_i$ ($i \in \mathcal{I}$) is considered to be both 0 and 1, i.e., a path split. Thus, the number of candidate codewords (decoding paths) doubles after each information bit is estimated. To prevent the exponential growth of the number of decoding paths, a path metric is utilized to select the L most probable decoding paths after each information bit is decoded. In the LLR domain, the low-complexity path metric can be obtained as [1]

$$\mathrm{PM}_l = \begin{cases} \mathrm{PM}_l + \left|\alpha_{0,i_l}\right| & \text{if } \hat{u}_i \neq \frac{1-\mathrm{sgn}(\alpha_{0,i_l})}{2}, \\ \mathrm{PM}_l & \text{otherwise,} \end{cases} \tag{2.2}$$

where α_{0,i_l} denotes the soft value of the i-th bit at stage 0 of the l-th path, and initially $\mathrm{PM}_l = 0 \; \forall l$. At the end of the decoding process, only the path that has the smallest path metric is selected as the decoding output.

2.2.2 Fast Successive-Cancellation List Decoding

SC and SCL decoding can also be illustrated on a binary tree representation of the code [23,24,50]. Fig. 2.3(a) shows a full binary tree representation of $\mathcal{P}(16, 8)$, whose factor graph is depicted in Fig. 2.2(a). The authors in [3, 26, 27, 51] proposed fast decoding operations for various special nodes under SCL decoding, which preserve the error-correction performance of SCL decoding while preventing tree-traversal to the leaf nodes. Thus, the decoding latency of SCL decoding is significantly reduced. Similar to [3], we consider four types of special nodes, namely Rate-0, Rate-1, repetition (REP), and single parity check (SPC), for all the fast SCL-based decoding algorithms in this thesis. Note that only the REP and SPC nodes are encountered when decoding RM codes.

Consider a parent node v located at the s-th stage ($0 < s \leq n$) of the polar (RM) code binary tree. There are N_v LLR values and N_v hard decisions associated with this node, where $N_v = 2^s$. Let $\boldsymbol{\alpha}_{v_l}$ and $\boldsymbol{\beta}_{v_l}$ be the vectors containing the soft and hard values associated with a parent node v of the l-th decoding path, respectively. $\boldsymbol{\alpha}_{v_l}$ and $\boldsymbol{\beta}_{v_l}$ are defined as $\boldsymbol{\alpha}_{v_l} = \{\alpha_{s,i_{\min_{v_l}}}, \ldots, \alpha_{s,i_{\max_{v_l}}}\}$ and $\boldsymbol{\beta}_{v_l} = \{\beta_{s,i_{\min_{v_l}}}, \ldots, \beta_{s,i_{\max_{v_l}}}\}$, respectively, where $i_{\min_{v_l}}$ and $i_{\max_{v_l}}$ are the bit indices corresponding to v such that $1 \leq i_{\min_{v_l}} < i_{\max_{v_l}} \leq N$ and $i_{\max_{v_l}} - i_{\min_{v_l}} = N_v - 1$. For all the fast SCL-based decoders considered in this thesis, the elements of $\boldsymbol{\alpha}_{v_l}$ corresponding to the SPC and Rate-1 nodes are considered to be sorted in the following order [3]:

$$\left|\alpha_{s,\iota_{\min_{v_l}}}\right| \leq \ldots \leq \left|\alpha_{s,\iota_{\max_{v_l}}}\right|, \tag{2.3}$$

where $i_{\min} \leq \iota_{\min}, \iota_{\max} \leq i_{\max}$. In addition, let τ be the minimum number of path splittings occurred at an SPC or a Rate-1 node that allows FSCL decoding to preserve the error-correction performance of the conventional SCL decoding algorithm [3]. The definitions and decoding operations of each special node under FSCL decoding are given as follows.

Rate-0 node

All the leaf nodes of a Rate-0 node are frozen bits. Therefore, all the hard values associated with the parent node are set to 0 and the path metric of the l-th path is given as [3]

$$\mathrm{PM}_l = \mathrm{PM}_l + \sum_{i=i_{\min_{v_l}}}^{i_{\max_{v_l}}} \frac{\left|\alpha_{s,i}\right| - \alpha_{s,i}}{2}. \tag{2.4}$$

REP node

All the leaf nodes of a REP node are frozen bits, except for $\beta_{0,i_{\max_{v_l}}}$. The path metric of the l-th decoding path is calculated as [3]

$$\mathrm{PM}_l = \mathrm{PM}_l + \sum_{i=i_{\min_{v_l}}}^{i_{\max_{v_l}}} \frac{\left|\alpha_{s,i}\right| - (1-2\beta_{s,i_{\max_{v_l}}})\alpha_{s,i}}{2}, \tag{2.5}$$

where $\beta_{s,i_{\max_{v_l}}}$ denotes the bit estimate of the information bit of the REP node.

Rate-1 node

All the leaf nodes of a Rate-1 node are information bits. FSCL decoding performs τ path splittings, where $\tau = \min(L-1, N_v)$ [3]. The path metric of the l-th decoding path for a Rate-1 node is calculated as [3]

$$\mathrm{PM}_l = \mathrm{PM}_l + \sum_{i=\imath_{\min_{v_l}}}^{\imath_{\max_{v_l}}} \frac{\left|\alpha_{s,i}\right| - (1-2\beta_{s,i})\alpha_{s,i}}{2}, \tag{2.6}$$

where $\beta_{s,i}$ denotes the bit estimate of the i-th bit of v.

SPC node

All the leaf nodes of an SPC node are information bits, except for $\beta_{0,i_{\min_{v_l}}}$. The parity check sum of the l-th path is first obtained as [3]

$$p_l = \bigoplus_{i=\imath_{\min_{v_l}}}^{\imath_{\max_{v_l}}} \frac{1-\mathrm{sgn}(\alpha_{s,i})}{2}. \tag{2.7}$$

The path metric is then updated as [3]

$$\mathrm{PM}_l = \mathrm{PM}_l + p_l\left|\alpha_{s,\imath_{\min_l}}\right|. \tag{2.8}$$

The decoding continues with τ path splittings, where $\tau = \min(L, N_v)$ [3]. In each new path splitting at the i-th index, the path metric is updated as [3]

$$\mathrm{PM}_l = \begin{cases} \mathrm{PM}_l + \left|\alpha_{s,i}\right| + (1-2p_l)\left|\alpha_{s,\iota_{\min_l}}\right| \\ \qquad\qquad \text{if } 1-2\beta_{s,i} \neq \operatorname{sgn}(\alpha_{s,i}), \\ \mathrm{PM}_l \qquad\qquad \text{otherwise,} \end{cases} \tag{2.9}$$

then the parity check sum is updated as [26]

$$p_l = \begin{cases} 1 \oplus p_l & \text{if } 1-2\beta_{s,i} \neq \operatorname{sgn}(\alpha_{s,i}), \\ p_l & \text{otherwise.} \end{cases} \tag{2.10}$$

where i is selected by following the bit indices of the sorted absolute LLR values in (2.3) [3]. When all the bits are estimated, the hard decision of the least reliable bit is updated to maintain the parity check condition of the SPC node [3]

$$\beta_{s,\iota_{\min_{v_l}}} = \bigoplus_{\substack{\forall i_{\min_{v_l}} \leq i \leq i_{\max_{v_l}} \\ i \neq \iota_{\min_{v_l}}}} \beta_{s,i}. \tag{2.11}$$

Note that the FSC decoding algorithm introduced in [23] is a special case of the FSCL decoder with $L = 1$.

It was analytically shown in [3, 25] that the error-correction performance of the fast SCL decoding algorithm when considering Rate-0, Rate-1, and REP nodes is exactly the same as that of the conventional SCL decoder. However, for SPC nodes this property was only empirically observed in [26]. Here, we provide an analytical proof that shows that the SPC decoding operations under fast SCL decoding [3, 25, 26] also yield exactly the same error-correction performance when compared to SCL decoding.

Theorem 1. *The path metric calculated at the parent node level by following the FSCL decoding operations for SPC nodes as described in (2.7)-(2.11) [3, 26] is exactly the same as the path metric calculated at its leaf node level.*

Proof. The path metric of the l-th path calculated at the parent node level of the SPC nodes by

following (2.7)-(2.11) [3, 26] can be rewritten as[1]

$$\mathrm{PM}_{v_s} = \sum_{i=0}^{N_v-1} \frac{\left|\alpha_{s,i}\right| - \eta_{s,i}\alpha_{s,i}}{2} + p_{s,i_{\min}}\left|\alpha_{s,i_{\min}}\right| - \frac{\left|\alpha_{s,i_{\min}}\right| - \eta_{s,i_{\min}}\alpha_{s,i_{\min}}}{2} + \left[\bigoplus_{\substack{0\le i<N_v \\ i\neq i_{\min}}} \frac{1-\eta_{s,i}\,\mathrm{sgn}(\alpha_{s,i})}{2}\right](1-2p_{s,0})\left|\alpha_{s,i_{\min}}\right|, \tag{2.12}$$

where $\eta_{s,i} = 1 - 2\beta_{s,i}$.

As v is an SPC node, we have the following constraint

$$\begin{aligned} 0 = \bigoplus_{i=0}^{N_v-1} \beta_{s,i} &= \beta_{s,i_{\min}} \oplus \left[\bigoplus_{\substack{0\le i<N_v \\ i\neq i_{\min}}} \left(\frac{1-\eta_{s,i}\,\mathrm{sgn}(\alpha_{s,i})}{2} \oplus \frac{1-\mathrm{sgn}(\alpha_{s,i})}{2}\right)\right] \\ &= \beta_{s,i_{\min}} \oplus \left[\bigoplus_{\substack{0\le i<N_v \\ i\neq i_{\min}}} \frac{1-\mathrm{sgn}(\alpha_{s,i})}{2}\right] \oplus \left[\bigoplus_{\substack{0\le i<N_v \\ i\neq i_{\min}}} \frac{1-\eta_{s,i}\,\mathrm{sgn}(\alpha_{s,i})}{2}\right] \\ &= \beta_{s,i_{\min}} \oplus p_{s,0} \oplus \frac{1-\mathrm{sgn}(\alpha_{s,i_{\min}})}{2} \oplus \left[\bigoplus_{\substack{0\le i<N_v \\ i\neq i_{\min}}} \frac{1-\eta_{s,i}\,\mathrm{sgn}(\alpha_{s,i})}{2}\right], \end{aligned} \tag{2.13}$$

where $p_{s,i_{\min}}$ is the initial parity check sum of the l-th decoding path obtained in (2.7). Therefore,

$$\begin{aligned} \bigoplus_{\substack{0\le i<N_v \\ i\neq i_{\min}}} \frac{1-\eta_{s,i}\,\mathrm{sgn}(\alpha_{s,i})}{2} &= \beta_{s,i_{\min}} \oplus p_{s,0} \oplus \frac{1-\mathrm{sgn}(\alpha_{s,i_{\min}})}{2} \\ &= \frac{1-\eta_{s,i_{\min}}(1-2p_{s,0})\,\mathrm{sgn}(\alpha_{s,i_{\min}})}{2}. \end{aligned} \tag{2.14}$$

By substituting (2.14) into (2.12) and using [25, Theorem 2], we obtain:

$$\mathrm{PM}_{v_s} = \sum_{i=0}^{N_v-1} \frac{\left|\alpha_{s,i}\right| - \eta_{s,i}\alpha_{s,i}}{2} = \sum_{i=0}^{N_v-1} \frac{\left|\alpha_{0,i}\right| - \eta_{0,i}\alpha_{0,i}}{2} = \mathrm{PM}_{v_0}, \tag{2.15}$$

where PM_{v_0} indicates the path metric of the SPC node obtained at the leaf-node level. Therefore, Theorem 1 is proved. □

Note that FSCL decoding requires high computational complexity, decoding latency and mem-

[1] We drop the path index l in the proof for better clarity.

ory requirement to obtain reasonable error-correction performance of polar and RM codes when targeting the mMTC and URLLC use cases of the 5G standard. In Chapter 4, Chapter 6, and Chapter 7 of the thesis, various improvements are proposed to address the associated problems of the FSCL-based decoders, making polar and RM codes more attractive to the mMTC and URLLC scenarios.

2.3 Bit-Flipping-Based Decoding

This section introduces preliminary knowledge and notations required by Chapter 3 and Chapter 4.

2.3.1 Dynamic Successive Cancellation Flip Decoding

The error-correction performance of SC decoding for short to moderate block lengths is not satisfactory. To improve its error-correction performance, a CRC of length C is concatenated to the message word of polar codes to check whether SC decoding succeeded or not. If the estimated message word $\hat{\boldsymbol{u}}$ does not satisfy the CRC after the initial SC decoding attempt, a secondary SC decoding attempt is made by flipping the estimation of an information bit in $\hat{\boldsymbol{u}}$ that is most likely to be erroneous. This process can be performed multiple times by applying a predetermined number of SC decoding attempts, with each attempt flipping the estimation of a different information bit. If the resulting message word after one of the SC decoding attempts satisfies the CRC, the decoding is declared successful. This algorithm is referred to as SCF decoding [40]. The main problem associated with SCF decoding is that only the first erroneous bit after the initial SC decoding can be corrected. However, it is common that even after the first erroneous bit is corrected, the resulting message word still contains erroneous bits. Therefore, further flipping attempts for the additional erroneous bits are required. DSCF decoding was introduced in [29] to address this problem.

Let $\mathcal{E}_\omega = \{i_1, \ldots, i_\omega\}$, where $\{i_1, \ldots, i_\omega\} \subset \mathcal{I}$, be the set of bit-flipping positions of order ω such that $i_1 < \cdots < i_\omega$, $0 \leq \omega \leq K + C$, and $|\mathcal{E}_\omega| = \omega$. Note that $\mathcal{E}_0 = \emptyset$. In the course of DSCF decoding, the hard-bit estimations of all the bit indices in $\mathcal{E}_\omega$ are flipped. The set $\mathcal{E}_\omega$ is constructed progressively based on the set $\mathcal{E}_{\omega-1} = \{i_1, \ldots, i_{\omega-1}\}$. In fact, if SC decoding fails after flipping all the bit-flipping positions in $\mathcal{E}_{\omega-1}$, i_ω is added to $\mathcal{E}_{\omega-1}$ to form $\mathcal{E}_\omega$ and an additional SC decoding attempt is performed by flipping the bit estimation at all the bit-flipping positions in $\mathcal{E}_\omega$. Furthermore, a maximum number of decoding attempts m_ω is imposed on the decoder to limit the computational complexity in practice. The bit-flipping process of the ω-th error order under SC

decoding can be written as

$$\hat{u}[\mathcal{E}_\omega]_i = \begin{cases} 0 & \text{if } u_i \in \mathcal{I}^c, \\ \frac{1+\text{sgn}(\alpha[\mathcal{E}_\omega]_{0,i})}{2} & \text{if } u_i \in \mathcal{I}, i \in \mathcal{E}_\omega, \\ \frac{1-\text{sgn}(\alpha[\mathcal{E}_\omega]_{0,i})}{2} & \text{otherwise,} \end{cases} \tag{2.16}$$

where $\boldsymbol{\alpha}[\mathcal{E}_\omega]$ is the vector of LLR values obtained at the ω-th error order.

Let

$$p_i^*(\mathcal{E}_{\omega-1}) = \Pr(\hat{u}[\mathcal{E}_{\omega-1}]_i = u_i | \boldsymbol{y}, \hat{\boldsymbol{u}}[\mathcal{E}_{\omega-1}]_0^{i-1} = \boldsymbol{u}_0^{i-1}), \tag{2.17}$$

where

$$\begin{cases} \hat{\boldsymbol{u}}\,[\mathcal{E}_{\omega-1}]_0^{i-1} & = \{\hat{u}[\mathcal{E}_{\omega-1}]_0, \hat{u}[\mathcal{E}_{\omega-1}]_1, \ldots, \hat{u}[\mathcal{E}_{\omega-1}]_{i-1}\}, \\ \boldsymbol{u}_0^{i-1} & = \{u_0, u_1, \ldots, u_{i-1}\}. \end{cases}$$

The probability that SC decoding is successful after flipping all the bit-flipping positions in $\mathcal{E}_\omega$ is then defined as [29]

$$P_{i_\omega} = \prod_{\substack{\forall i \in \mathcal{I} \setminus \mathcal{E}_\omega \\ i < i_\omega}} p_i^*(\mathcal{E}_{\omega-1}) \times \prod_{\forall i \in \mathcal{E}_\omega} \left(1 - p_i^*(\mathcal{E}_{\omega-1})\right). \tag{2.18}$$

Therefore, the bit-flipping position ι_ω that maximizes the probability of $\hat{\boldsymbol{u}}[\mathcal{E}_{\omega-1}]$ being correctly decoded is

$$\iota_\omega = \underset{\substack{\forall i_\omega \in \mathcal{I}, i_{\omega-1} < i_\omega \leq N-1 \\ \mathcal{E}_\omega = \mathcal{E}_{\omega-1} \cup i_\omega}}{\arg\max} P_{i_\omega}. \tag{2.19}$$

Note that the probability $p_i^*(\mathcal{E}_{\omega-1})$ cannot be obtained during the course of decoding as the values of the elements of $\boldsymbol{u}$ are unknown to the decoder [29]. As a result, DSCF decoding uses a known probability $p_i(\mathcal{E}_{\omega-1})$ to estimate $p_i^*(\mathcal{E}_{\omega-1})$. The known probability $p_i(\mathcal{E}_{\omega-1})$ is defined as

$$\begin{aligned} p_i(\mathcal{E}_{\omega-1}) &= \max\left(\Pr(\hat{u}[\mathcal{E}_{\omega-1}]_i = 0 | \boldsymbol{y}, \hat{\boldsymbol{u}}[\mathcal{E}_{\omega-1}]_0^{i-1}),\right. \\ &\qquad \left.\Pr(\hat{u}[\mathcal{E}_{\omega-1}]_i = 1 | \boldsymbol{y}, \hat{\boldsymbol{u}}[\mathcal{E}_{\omega-1}]_0^{i-1})\right) \\ &= \frac{1}{1 + \exp\left(-|\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right)}, \end{aligned} \tag{2.20}$$

where $\alpha[\mathcal{E}_{\omega-1}]_{0,i}$ is the corresponding LLR value of $\hat{u}[\mathcal{E}_{\omega-1}]_i$. It was shown in [29] that the estimation in (2.20) is not accurate. Therefore, [29] introduced a perturbation parameter λ to have a

better estimation of $p_i^*(\mathcal{E}_{\omega-1})$ as

$$p_i^*(\mathcal{E}_{\omega-1}) \approx \frac{1}{1+\exp\left(-\lambda|\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right)}. \tag{2.21}$$

It should be noted that $\lambda \in \mathbb{R}^+$ is a scaling factor for the magnitude of the LLR values and is determined by a Monte-Carlo simulation. To enable a trade-off between decoding latency and error-correction performance, instead of only flipping the most probable bit-flipping position, DSCF decoding attempts to improve SC decoding with a list of most probable bit-flipping indices ι_ω at each error order ω [29].

In order to have numerically stable computations in the hardware implementation of the DSCF decoder, the bit-flipping metric in (2.18) can be written in the log-likelihood (LL) domain as [29]

$$\begin{aligned} Q_{i_\omega} &= -\frac{1}{\lambda}\ln(P_{i_\omega}) \\ &= \sum_{\substack{\forall i\in\mathcal{I} \\ i\le i_\omega}} \frac{1}{\lambda}\ln\left(1+\exp\left(-\lambda|\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right)\right) \\ &\quad + \sum_{\forall i\in\mathcal{E}_\omega} |\lambda[\mathcal{E}_{\omega-1}]_{0,i}|. \end{aligned} \tag{2.22}$$

Consequently, the most probable bit-flipping position ι_ω can be found in the LL domain as

$$\iota_\omega = \underset{\substack{\forall i_\omega\in\mathcal{I}, i_{\omega-1}<i_\omega\le N-1 \\ \mathcal{E}_\omega=\mathcal{E}_{\omega-1}\cup i_\omega}}{\arg\min} Q_{i_\omega}. \tag{2.23}$$

The DSCF decoder suffers from a high decoding complexity due to the logarithmic and exponential computations, which is addressed in Chapter 3 of the thesis.

2.3.2 Successive-Cancellation List Flip Decoding

Similar to DSCF, SCLF decoding also relies on a CRC verification to indicate whether the initial SCL decoding attempt is successful or not. If the first SCL decoding attempt does not satisfy the CRC verification, the SLCF decoding algorithm tries to identify the first information bit index ι, at which the correct path is discarded from the list of the L most probable decoding paths [30]. Given that the ι-th bit index is correctly identified, in the next decoding attempt and after the path

splitting occurs at the ι-th bit index, the path selection is reversed where the L decoding paths that have the highest (worst) path metrics are selected to continue the decoding [30]. This reversed path-selection scheme recovers the correct decoding path, which was discarded at the initial SCL decoding at the ι-th bit index, to the list of the active decoding paths. SCLF decoding then performs conventional SCL decoding operations for all the bit indices following ι.

As we only need to locate the error decision occurred at a path splitting of an information bit, in this section the bit indices are referred to information bits and are indexed from 1 to $K + C$. Given that at the i-th information bit under SCL decoding, there are L active decoding paths denoted as l, $l \in [1, 2L]$. After the path splitting of the current L active paths, the path metrics of the new $2L$ paths are computed and sorted. Let l' be the index of a discarded decoding path after the path metric sorting, i.e., the path metric corresponding to l' is among the L largest path metric values. The probability that the path with index l' is the correct decoding path is [29]

$$\begin{aligned} Pr(\hat{\boldsymbol{u}}_{1_{l'}}^{i_{l'}} = \boldsymbol{u}_1^i | \boldsymbol{\alpha}_n) = & \prod_{\substack{1 \le j \le i \\ \forall j \in \mathbb{A}_{l'}}} Pr(\hat{u}_{j_{l'}} = u_j | \boldsymbol{\alpha}_n, \hat{\boldsymbol{u}}_{1_{l'}}^{j_{l'}-1} = \boldsymbol{u}_1^{j-1}) \\ & \times \prod_{\substack{1 \le j \le i \\ \forall j \in \mathbb{A}_{l'}^c}} \left[1 - Pr(\hat{u}_{j_{l'}} = u_j | \boldsymbol{\alpha}_n, \hat{\boldsymbol{u}}_{1_{l'}}^{j_{l'}-1} = \boldsymbol{u}_1^{j-1}) \right], \end{aligned} \tag{2.24}$$

where $Pr(\hat{\boldsymbol{u}}_{1_{l'}}^{i_{l'}} = \boldsymbol{u}_1^i | \boldsymbol{\alpha}_n) = Pr(\hat{u}_{1_{l'}} = u_1, \ldots, \hat{u}_{i_{l'}} = u_i | \boldsymbol{\alpha}_n)$. $\mathbb{A}_{l'}$ is the set of information bit indices where their hard decisions follow the sign of the corresponding LLR values, while $\mathbb{A}_{l'}^c$ is the set of information bit indices whose hard decisions do not follow the sign of the LLR values [30].

Note that $Pr(\hat{u}_{j_{l'}} = u_j | \boldsymbol{\alpha}_n, \hat{\boldsymbol{u}}_{1_{l'}}^{j_{l'}-1} = \boldsymbol{u}_1^{j-1})$ is not available during the course of decoding as $\boldsymbol{u}$ is unknown, thus it is approximated as [29, 30]

$$Pr(\hat{u}_{j_{l'}} = u_j | \boldsymbol{\alpha}_n, \hat{\boldsymbol{u}}_{1_{l'}}^{j_{l'}-1} = \boldsymbol{u}_1^{j-1}) \approx \frac{1}{1 + \exp\left(-\lambda \left| \alpha_{0,j_{l'}} \right| \right)}, \tag{2.25}$$

where $\lambda \in \mathbb{R}^+$ is a perturbation parameter that is optimized offline to improve the approximation accuracy of (2.25).

The probability that the correct decoding path is discarded at the information bit with index i is [30]

$$P_i = \sum_{\forall l'} Pr(\hat{\boldsymbol{u}}_{1_{l'}}^{i_{l'}} = \boldsymbol{u}_1^i | \boldsymbol{\alpha}_n). \tag{2.26}$$

Therefore, the bit index at which the error decision is most likely to take place is [30]

$$\iota = \underset{\log_2 L<i\leq K+C}{\arg\max} \; P_i. \tag{2.27}$$

Directly computing (2.26) is not numerically stable [29, 30]. Thus, a flipping metric based on the max-log approximation is derived from (2.26) as [30]

$$\begin{aligned} Q_i = -\frac{1}{\lambda}\ln P_i &\approx -\max_{\forall l'}\left[\frac{1}{\lambda}\ln Pr\left(\hat{\boldsymbol{u}}_{1_{l'}}^{i_{l'}} = \boldsymbol{u}_1^i|\boldsymbol{\alpha}_n\right)\right] \\ &\approx \min_{\forall l'}\left[\sum_{\forall j\in\mathbb{A}_{l'}^c}\left|\alpha_{0,j_{l'}}\right| + \sum_{1\leq j\leq i}\frac{1}{\lambda}\ln\left[1+\exp\left(-\lambda\left|\alpha_{0,j_{l'}}\right|\right)\right]\right]. \end{aligned} \tag{2.28}$$

The computation of Q_i can be further simplified by using a hardware-friendly approximation introduced in [32]:

$$f_\lambda(x) = \frac{1}{\lambda}\ln\left[1+\exp\left(-\lambda|x|\right)\right] \approx \begin{cases} a_\lambda & \text{if } |x| \leq b_\lambda, \\ 0 & \text{otherwise,} \end{cases} \tag{2.29}$$

where $a_\lambda, b_\lambda \in \mathbb{R}^+$ are tunable parameters selected based on a predetermined value of λ. Consequently, the most probable information bit index where the correct path is discarded can be estimated as

$$\iota = \underset{\log_2 L<i\leq K+C}{\arg\min} \; Q_i. \tag{2.30}$$

The SCLF decoder suffers from a high decoding latency due to the full binary-tree traversal during the course of decoding. This problem is addressed in Chapter 4 of the thesis.

2.4 Belief-Propagation-Based Decoding

This section introduces preliminary knowledge and notations required by Chapter 5.

2.4.1 Scaled Belief Propagation Decoding

Fig. 2.4a illustrates BP decoding on a factor graph representation of $\mathcal{P}(8, 5)$. The messages are iteratively propagated through the BP processing elements (PEs) [52] located in each stage. An update iteration starts with a right-to-left message pass that propagates the LLR values from the

channel (rightmost) stage, to the information bit (leftmost) stage, and ends with the left-to-right message pass which occurs in the reverse order. Fig. 2.4b shows a BP PE with its corresponding messages, where $r_{t,s}$ denotes a left-to-right message, and $l_{t,s}$ denotes a right-to-left message of the t-th bit index at stage s. Equivalently, BP decoding of polar codes can be represented on an unrolled factor graph, in which BP iterations are performed sequentially [53]. Fig. 2.4c and Fig. 2.4d illustrate the input and output messages of a BP PE for the right-to-left and left-to-right message updates on an unrolled factor graph, where the superscript i denotes the iteration number. The update rule [52] for the right-to-left messages of a BP PE is

$$\begin{cases} l_{t,s}^{i} &= f_{\mathrm{BP}}(l_{t,k}^{i}, r_{j,s}^{i-1} + l_{j,k}^{i}), \\ l_{j,s}^{i} &= f_{\mathrm{BP}}(l_{t,k}^{i}, r_{t,s}^{i-1}) + l_{j,k}^{i}, \end{cases} \tag{2.31}$$

and for the left-to-right messages is

$$\begin{cases} r_{t,k}^{i} &= f_{\mathrm{BP}}(r_{t,s}^{i}, l_{j,k}^{i} + r_{j,s}^{i}), \\ r_{j,k}^{i} &= f_{\mathrm{BP}}(r_{t,s}^{i}, l_{t,k}^{i}) + r_{j,s}^{i}, \end{cases} \tag{2.32}$$

where $j = t + 2^s$, $k = s + 1$, and

$$f_{\mathrm{BP}}(x, y) = 2 \operatorname{arctanh}\left(\tanh\left(\frac{x}{2}\right)\tanh\left(\frac{y}{2}\right)\right), \tag{2.33}$$

for any $x, y \in \mathbb{R}$. Note that implementing (2.33) is costly in practice, instead the following approximation of (2.33) is used in this thesis [36]:

$$f_{\mathrm{BP}}(x, y) \approx \tilde{f}(x, y) = 0.9375 \operatorname{sgn}(x) \operatorname{sgn}(y) \min(|x|, |y|). \tag{2.34}$$

BP decoding performs a predetermined $I_{\max}$ update iterations where the messages are propagated through all BP PEs in accordance with (2.31) and (2.32). Initially, for $0 \leq t < N$ and $\forall i \leq I_{\max}$, $l_{t,n}^{i}$ are set to the received channel LLR values α_n, $r_{t,0}^{i}$ are set to the LLR values of the information and frozen bits as

$$\alpha_0 = \begin{cases} 0, & \text{if } t \in \mathcal{I}, \\ +\infty, & \text{if } t \in \mathcal{I}^c. \end{cases} \tag{2.35}$$

All the other left-to-right and right-to-left messages of the PEs at the first iteration are set to 0.

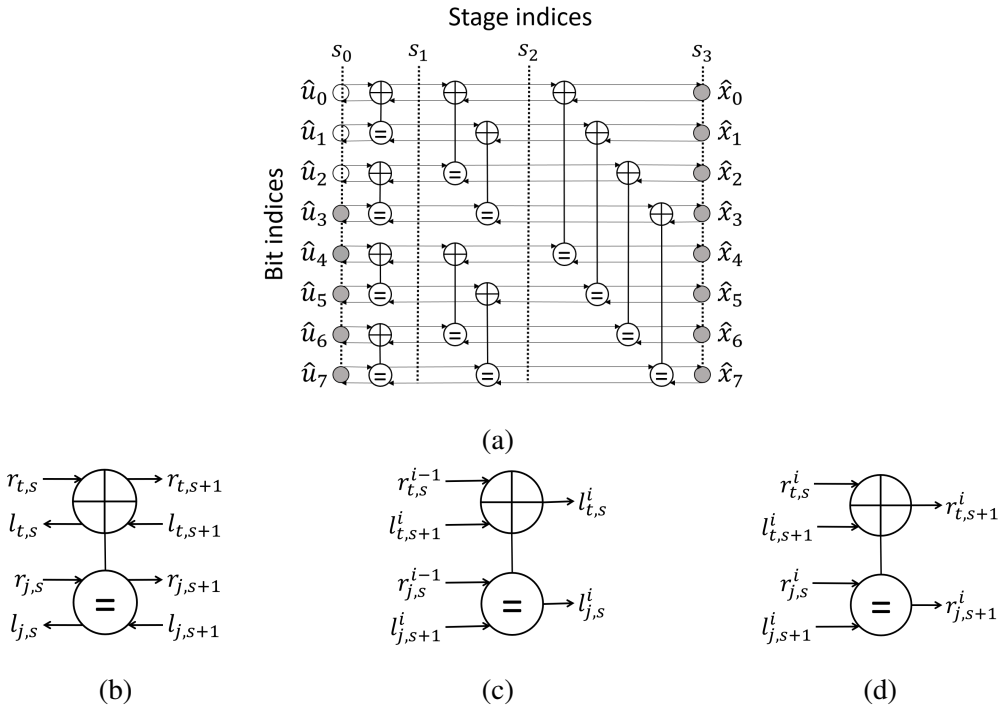


Figure 2.4: (a) BP decoding on the factor graph of $\mathcal{P}(8,5)$ with $\{u_0, u_1, u_2\} \in \mathcal{I}^c$, (b) a BP PE, (c) a right-to-left message update of a BP PE on an unrolled factor graph, and (d) a left-to-right message update of a BP PE on an unrolled factor graph.

After running $I_{\max}$ iterations, the decoder makes a hard decision on the LLR values of the t-th bit at the information bit stage to obtain the estimated message word as

$$\hat{u}_t = \begin{cases} 0, & \text{if } r_{t,0}^{I_{\max}} + l_{t,0}^{I_{\max}} \geq 0, \\ 1, & \text{otherwise.} \end{cases} \tag{2.36}$$

In the rest of the thesis, the vector forms of the left-to-right and right-to-left messages at the s-th stage and the i-th iteration are denoted as $\boldsymbol{l}_s^i$ and $\boldsymbol{r}_s^i$, respectively.

A CRC is used for BP decoding to either early terminate the BP process [36], or to help select the correct codeword among a list of candidates as considered in [54, 55]. However, these CRC utilizations do not take into account the factor graph realization of CRC, on which the BP decoder can be applied.

2.4.2 Neural Belief Propagation Decoding

Neural BP decoding was introduced in [56, 57] to improve the error-correction performance of BP decoding on Bose-Chaudhuri-Hocquenghem (BCH) codes by assigning trainable weights to the conventional BP decoding. Neural Normalized Min-Sum Recurrent Neural Network (NNMS-RNN) is a powerful variant of neural BP [56] with the following weight assignment scheme for the message update rule of a PE in (2.31) and (2.32):

$$\begin{cases} l^i_{t,s} &= w_0 \tilde{f}(l^i_{t,k}, w_1 r^{i-1}_{j,s} + w_2 l^i_{j,k}), \\ l^i_{j,s} &= w_4(w_3 \tilde{f}(l^i_{t,k}, r^{i-1}_{t,s})) + w_5 l^i_{j,k}, \end{cases} \tag{2.37}$$

$$\begin{cases} r^i_{t,k} &= w_6 \tilde{f}(r^i_{t,s}, w_7 l^i_{j,k} + w_8 r^i_{j,s}), \\ r^i_{j,k} &= w_{10}(w_9 \tilde{f}(r^i_{t,s}, l^i_{t,k})) + w_{11} r^i_{j,s}, \end{cases} \tag{2.38}$$

where w's $\in \mathbb{R}$ are the trainable weights.

The NNMS-RNN BP decoder suffers from a large number of weights which adversely affects its implementation cost. A Neural Normalized Min-Sum (NNMS) decoder was used to decode polar codes by only enabling the training for w_0, w_3, w_6 and w_9, while setting the other weights in (2.37) and (2.38) to 1 [53]. However, the error-correction performance improvement of [53] with respect to the conventional BP is not significant. Chapter 5 of the thesis introduces novel techniques that utilize the CRC and factor-graph permutations to further improve the error-correction performance of polar-CRC concatenated codes under BP and neral BP decoding.

Chapter 3

Machine-Learning-Aided Successive-Cancellation Flip Decoding of Polar Codes

In this chapter, we first provide an approximation scheme to greatly reduce the computational complexity of the state-of-the-art DSCF decoding algorithm while maintaining its error correction performance. In particular, the costly multiplications and transcendental computations used in the bit-flipping model of DSCF decoding are replaced by an approximation scheme that only requires additions. We then propose a novel bit-flipping model tailored to Fast SC (FSC) decoding to reduce the decoding latency of the DSCF decoder. The proposed bit-flipping model is parameterized by a correlation matrix, which is trained using an reinforcement learning optimization framework.

3.1 Neural Successive Cancellation Flip Decoding

In this section, a novel low-complexity bit-flipping metric computation scheme is presented to allow efficient hardware implementation. Then, a machine learning framework is introduced to optimize the parameter in the proposed bit-flipping metric computation scheme.

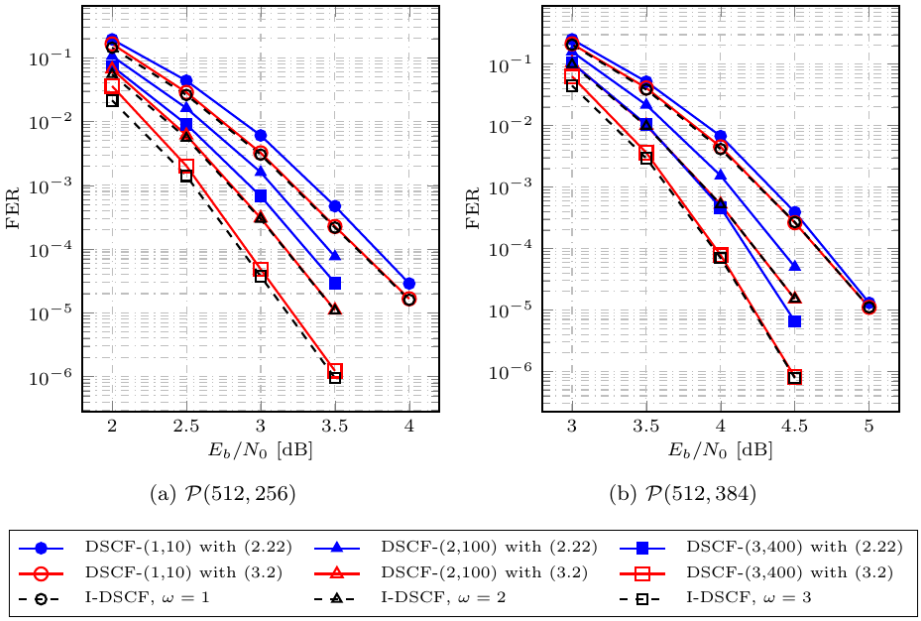


Figure 3.1: Effect of the simplification in (3.2) on the FER of DSCF decoding for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. The polar codes are concatenated with a 24-bit CRC used in 5G standard. The ideal DSCF decoder (I-DSCF) is also plotted as a reference.

3.1.1 Bit-flipping Metric Computation

Efficient hardware implementation of DSCF decoding is contingent on the efficient implementation of the bit-flipping metric in (2.22). However, (2.22) involves logarithmic and exponential functions that are not hardware friendly. A common approach to approximate the logarithmic and exponential function in (2.22) is to use the rectifier linear unit (ReLU) as [1]

$$\ln(1 + \exp(x)) \approx \mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

However, since $\lambda > 0$, $-\lambda|\alpha[\mathcal{E}_{\omega-1}]_{0,i}| < 0$. Therefore, (2.22) can be simplified as

$$Q_{i_\omega} \approx \sum_{\substack{\forall i \in \mathcal{I} \\ i \le i_\omega}} \frac{1}{\lambda} \mathrm{ReLU}\left(-\lambda|\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right) + \sum_{\forall i \in \mathcal{E}_\omega} |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|$$

$$= \sum_{\forall i \in \mathcal{E}_\omega} |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|, \tag{3.2}$$

which is independent of the perturbation parameter λ. Fig. 3.1 shows the effect of the simplification in (3.2) on the error-correction performance of DSCF decoding in terms of Frame Error Rate (FER) for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. The polar codes are concatenated with a 24-bit CRC used in the control channel of 5G standard. In this figure, the FER curve of the DSCF decoder at error order ω with m_ω decoding attempts is denoted as DSCF-(ω, m_ω). The value of α is set to 0.3 as it provides a good result across a wide range of SNR values [29][1]. The FER of the ideal DSCF decoder, denoted as I-DSCF, where the erroneous bits up to the ω-th error order are always accurately corrected, is also plotted for comparison. As seen from Fig. 3.1, the over-simplification of the bit-flipping metric calculation in (3.2) results in 0.15, 0.3, and 0.45 dB error-correction performance loss for $\mathcal{P}(512, 256)$ compared to the DSCF decoder when $\omega = \{1, 2, 3\}$ and $m_\omega = \{10, 100, 400\}$, respectively, at a target FER of 10^{-4}. For $\mathcal{P}(512, 384)$, the corresponding FER degradation caused by the over-simplification operations is 0.05, 0.16, and 0.23 dB for $\omega = \{1, 2, 3\}$ and $m_\omega = \{10, 100, 400\}$ at the same target FER of 10^{-4}, respectively.

To address this issue, we propose to use a perturbation parameter $\theta \in \mathbb{R}^+$ that unlike λ, is an additive positive parameter, and like λ, tries to improve the estimation of $p_i^*(\mathcal{E}_{\omega-1})$. We write the proposed estimation of $p_i^*(\mathcal{E}_{\omega-1})$ as

$$p_i^*(\mathcal{E}_{\omega-1}) \approx \frac{1}{1 + \exp\left(\theta - |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right)}. \tag{3.3}$$

In addition, we propose to use a bit-flipping metric in the LL domain, Q_{ι_ω}, which is tailored to the proposed $p_i^*(\mathcal{E}_{\omega-1})$ in (3.3) as

$$\begin{aligned} Q_{\iota_\omega} &= -\ln(P_{\iota_\omega}) + \omega\theta \\ &= \sum_{\substack{\forall i \in \mathcal{I} \\ i \le \iota_\omega}} \ln\left(1 + \exp\left(\theta - |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right)\right) \\ &\quad + \sum_{\forall i \in \mathcal{E}_\omega} |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|, \end{aligned} \tag{3.4}$$

where we used the fact that $\omega\theta$ is a constant and it will not affect the selection of ι_ω in (3.4).

[1] Since the channel output $\boldsymbol{y}$ is directly used in this thesis as the decoding input, we set $\alpha = \frac{0.6}{\sigma^2}$ to obtain the same FER performance of the DSCF decoder in [29].

Algorithm 1: NSCF Decoding Algorithm

Input : $\boldsymbol{y}, \boldsymbol{m} = \{m_1, m_2, \ldots, m_\omega\}, \theta$
Output: $\hat{\boldsymbol{u}}$

/* Perform NSCF decoding upto the ω-th error order */
for $\tau \leftarrow 0$ **to** ω **do**
  /* Initialize the bit-flipping data structure */
  if $\tau \leftarrow 0$ **then**
    $S_0 \leftarrow [\emptyset, \mathcal{E}_0]$
  $S_{\tau+1} \leftarrow \emptyset$
  /* SC decoding with bit-flipping operations */
  forall $\mathcal{E}_\tau$ *in* S_τ **do**
    Perform SC decoding to obtain $\hat{u}[\mathcal{E}_\tau]_i$
    if $i > \arg\max_{\forall j \in \mathcal{E}_\tau} \{\mathcal{E}_\tau\}$ *and* $i \in \mathcal{I}$ **then**
      Form the candidate bit-flipping set $\mathcal{E}_{\tau+1} \leftarrow \mathcal{E}_\tau \cup i$
      Obtain $Q_{i_{\tau+1}}$ using (3.5)
      InsertionSort $(S_{\tau+1}, [Q_{i_{\tau+1}}, \mathcal{E}_{\tau+1}])$
    if $\hat{\boldsymbol{u}}[\mathcal{E}_\tau]$ *satisfies CRC* **then**
      $\hat{\boldsymbol{u}} = \hat{\boldsymbol{u}}[\mathcal{E}_\tau]$
      return $\hat{\boldsymbol{u}}$

Let us now use the ReLU function in (3.1) to simplify the proposed bit-flipping metric in (3.4) as

$$\begin{aligned} Q_{i_\omega} \approx & \sum_{\substack{\forall i \in \mathcal{I} \\ i \le i_\omega}} \mathrm{ReLU}\left(\theta - |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right) \\ & + \sum_{\forall i \in \mathcal{E}_\omega} |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|, \end{aligned} \tag{3.5}$$

where we used the fact that if $\theta - |\alpha[\mathcal{E}_{\omega-1}]_{0,i}| > 0$, then

$$\mathrm{ReLU}\left(\theta - |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|\right) = \theta - |\alpha[\mathcal{E}_{\omega-1}]_{0,i}|.$$

It can be seen that the resulting bit-flipping metric is dependent on the value of θ, and it is hardware friendly since only additions are required for the metric computation. Note that in this thesis, a different value of θ is used to calculate the bit-flipping metric in (3.5) at each error order.

We summarize the proposed NSCF decoding algorithm which corrects up to the ω-th error order under SC decoding in Algorithm 1. We denote by S_τ ($0 \leq \tau \leq \omega$) a data structure whose elements contain a pair of $[Q_{i_\tau}, \mathcal{E}_\tau]$, where Q_{i_τ} is the bit-flipping metric associated with a bit-flipping set $\mathcal{E}_\tau$ at the τ-th error order. Note that

$$|S_\tau|_{\max} = \begin{cases} 0 & \text{if } \tau = 0 \\ m_\tau - \sum_{k=0}^{\tau-1} m_k & \text{otherwise,} \end{cases}$$

thus the maximum number of all the additional decoding attempts up to the τ-th error order is m_τ.

At the τ-th error order, the proposed decoder loops over all the candidate bit-flipping sets $\mathcal{E}_\tau$ stored in S_τ. SC decoding with bit-flipping operations is then carried out given a bit-flipping set $\mathcal{E}_\tau$. At the i-th information bit and if $i > \arg\max_{\forall j \in \mathcal{E}_\tau}\{\mathcal{E}_\tau\}$, a new candidate bit-flipping set is constructed for the $(\tau + 1)$-th error order by concatenating the current information bit position to the current bit-flipping set $\mathcal{E}_\tau$, i.e., $\mathcal{E}_{\tau+1} \leftarrow \mathcal{E}_\tau \cup i$. The bit-flipping metric $Q_{i_{\tau+1}}$, which is associated with the newly constructed bit-flipping set $\mathcal{E}_{\tau+1}$, is then calculated using the proposed bit-flipping metric computation in (3.5). The data structure $S_{\tau+1}$ is then updated with the newly constructed element $[Q_{i_{\tau+1}}, \mathcal{E}_{\tau+1}]$ by performing an insertion sort using the new bit-flipping metric $Q_{i_{\tau+1}}$. If $Q_{i_{\tau+1}}$ is among the $|S_{\tau+1}|_{\max}$ smallest bit-flipping metrics of $S_{\tau+1}$, the new element is inserted in $S_{\tau+1}$, and the element that has the largest bit-flipping metric is discarded. Otherwise, the newly constructed element is discarded. This process is carried out in the InsertionSort($\cdot$) function denoted in Algorithm 1. Note that if the resulting estimated message word given the bit-flipping set $\mathcal{E}_\tau$, i.e. $\hat{\boldsymbol{u}}[\mathcal{E}_\tau]$, satisfies the CRC verification, the decoding terminates and outputs $\hat{\boldsymbol{u}}[\mathcal{E}_\tau]$ as the estimated message word. Also note that the proposed NSCF decoder reverts to the conventional DSCF decoder by replacing (3.5) in Algorithm 1 with (2.22).

3.1.2 Parameter Optimization

In this thesis, we formalize the optimization problem of the additive parameter θ as a separate classification problem at each error order and use ML techniques to train θ offline. As observed from (3.5), the bit-flipping metric computation takes the absolute values of the soft messages given by SC decoding as the input. Therefore, the bit-flipping metric computation does not depend on the value of u_i. Thus, it allows the use of the all-zero codeword for the training of θ, which simplifies the data collection process, as also observed in [56, 58].

In [59], the decoding process is modeled as a deep neural network that consists of ω unfolded DSCF decoding attempts. The training data used to train the parameter at the ω-th error order in [59] includes both the samples that cannot be correctly decoded and those that can be correctly decoded up to the $(\omega-1)$-th error order. As a result, the size of the training dataset is excessively large. For example, the framework introduced in [59] requires 2.5×10^5 samples for $\omega = 2$. Unlike [59], in this thesis we consider the parameter optimization of each error order individually. To train the parameter at the ω-th error order, only the frames that do not satisfy the CRC verification of the ideal DSCF decoder at the $(\omega-1)$-th error order are used. In fact, the frames that are not decoded correctly contribute to the training and optimization of parameter θ.

Let $T_{\omega-1}$ be the set of the bit-flipping indices, and $t_{\omega-1}$ be the $(\omega-1)$-th bit-flipping index of the $(\omega-1)$-th ideal DSCF decoder. In addition, let $\boldsymbol{\alpha}[T_{\omega-1}]$ be the LLR values of the $(w-1)$-th ideal DSCF decoder given that the corresponding hard decisions of $\boldsymbol{\alpha}[T_{\omega-1}]_0$ do not satisfy the CRC verification. For the rest of this thesis, since we only consider non-frozen bits, all the bit indices only indicate non-frozen bit positions. Thus, they are in the range of $[0, K+C-1]$.

The bit-flipping metric rendered for the i_ω-th bit index, $t_{\omega-1} < i_\omega < K+C$, of the ideal DSCF decoder is written as

$$Q_{i_\omega} \approx \sum_{0\le i\le i_\omega} \mathrm{ReLU}\left(\theta - |\alpha[T_{\omega-1}]_{0,i}|\right) + \sum_{\forall i\in\{T_{\omega-1}\cup i_\omega\}} |\alpha[T_{\omega-1}]_{0,i}|. \tag{3.6}$$

The value of Q_{i_ω} is normalized using the soft-min function $\delta(\cdot)$ as

$$\tilde{O}_{i_\omega} = \delta(Q_{i_\omega}) = \frac{\exp(-Q_{i_\omega})}{\sum_{j=t_{\omega-1}+1}^{K+C-1}\exp\left(-Q_{j_\omega}\right)}. \tag{3.7}$$

It can be seen that for all values of i_ω, $0 < \tilde{O}_{i_\omega} < 1$, and

$$\iota_\omega = \underset{\substack{\forall i_\omega \\ t_{\omega-1}<i_\omega<K+C}}{\arg\min}\ Q_{i_\omega} = \underset{\substack{\forall i_\omega \\ t_{\omega-1}<i_\omega<K+C}}{\arg\max}\ \tilde{O}_{i_\omega}, \tag{3.8}$$

where ι_ω is the most probable bit-flipping position. Note that $\tilde{O}_{i_\omega}$ can be viewed as the predicted

bit-flipping probability. Let us define the training label O_{i_ω} as

$$O_{i_\omega} = \begin{cases} 1 & \text{if } i_\omega = t_\omega, \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

In this thesis, the binary cross-entropy loss function is used to quantify the differences of the estimated value $\tilde{O}_{i_\omega}$ and the exact training label O_{i_ω}. This can be written as

$$\mathcal{L} = -\sum_{i_\omega = t_{\omega-1}+1}^{K+C-1} \left[O_{i_\omega} \ln \tilde{O}_{i_\omega} + (1 - O_{i_\omega}) \ln\left(1 - \tilde{O}_{i_\omega}\right) \right]. \tag{3.10}$$

By using the stochastic gradient-descent optimization technique or its variants, the parameter θ can be optimized to minimize the loss $\mathcal{L}$. The gradient of the objective loss function with respect to the additive parameter θ can be obtained as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{i_\omega = t_{\omega-1}+1}^{K+C-1} \frac{\partial \mathcal{L}}{\partial \tilde{O}_{i_\omega}} \frac{\partial \tilde{O}_{i_\omega}}{\partial Q_{i_\omega}} \frac{\partial Q_{i_\omega}}{\partial \theta} \tag{3.11}$$

where

$$\frac{\partial \mathcal{L}}{\partial \tilde{O}_{i_\omega}} = \frac{\tilde{O}_{i_\omega} - O_{i_\omega}}{\tilde{O}_{i_\omega}(1 - \tilde{O}_{i_\omega})}, \tag{3.12}$$

$$\frac{\partial \tilde{O}_{i_\omega}}{\partial Q_{i_\omega}} = \tilde{O}_{i_\omega}(\tilde{O}_{i_\omega} - 1), \tag{3.13}$$

$$\frac{\partial Q_{i_\omega}}{\partial \theta} = \sum_{j=0}^{i_\omega} \mathbb{I}_{\theta > |\alpha[T_{\omega-1}]_{0,j}|}, \tag{3.14}$$

and $\mathbb{I}_{a>b}$ $(a, b \in \mathbb{R})$ is an indicator function such that

$$\mathbb{I}_{a>b} = \begin{cases} 1 & \text{if } a > b, \\ 0 & \text{otherwise.} \end{cases} \tag{3.15}$$

Substituting (3.12)-(3.14) into (3.11) gives

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{i_\omega = i_{\omega-1}+1}^{K+C-1} \left(O_{i_\omega} - \tilde{O}_{i_\omega} \right) \sum_{j=0}^{i_\omega} \mathbb{I}_{\theta > |\alpha[T_{\omega-1}]_{0,j}|}. \tag{3.16}$$

In this thesis, we use Root Mean Square Propagation (RMSProp), a variant of the SGD optimization technique, to update the parameter θ [60]. The additive parameter θ is then updated as [60]

$$\theta = \theta - \frac{\lambda}{\sqrt{\mu_\theta}} \frac{\partial \mathcal{L}}{\partial \theta}, \tag{3.17}$$

where μ_θ is a running average of the magnitudes of recent gradients for θ that is defined as [60]

$$\mu_\theta = \gamma \mu_\theta + (1-\gamma) \left(\frac{\partial \mathcal{L}}{\partial \theta} \right)^2, \tag{3.18}$$

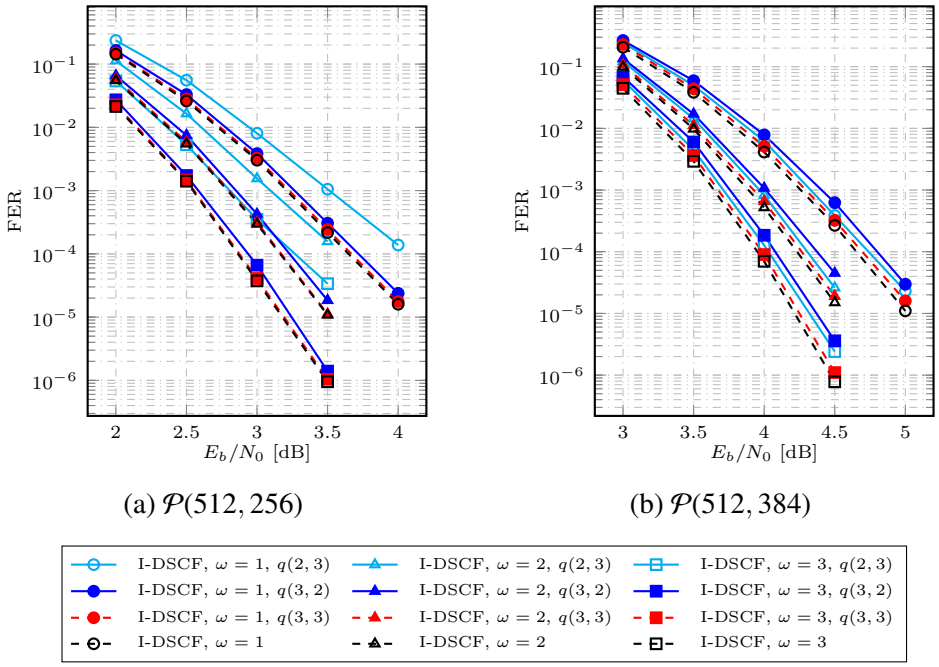
and λ and γ are the learning rate and the forgetting factor, respectively. The value of μ_θ is initialized for the first update as

$$\mu_\theta = (1-\gamma) \left(\frac{\partial \mathcal{L}}{\partial \theta} \right)^2. \tag{3.19}$$

It is worth mentioning that by manually deriving $\frac{\partial \mathcal{L}}{\partial \theta}$ as denoted in (3.16), the additive parameter θ can be optimized using a SGD-based optimization technique at the decoder side without the need of a sophisticated machine learning library, which paves the way for a dedicated hardware implementation that optimizes θ online using the all-zero codeword pilot signals.

3.1.3 Quantization Scheme

If training with quantization is considered, the additive parameter θ is optimized by taking into account the quantization effect caused by the SC decoding operations. At a given error order ω, quantization operations are first applied to the ideal DSCF decoders to obtain the quantized values of $\boldsymbol{\alpha}[T_{\omega-1}]$. In addition, the forward pass computations of the bit-flipping metric in (3.6) are also quantized. On the other hand, all other computations in (3.7)-(3.19) required for the backward pass to obtain the partial derivative of θ are carried out in the full-precision representation. After each parameter update in (3.17), the value of θ is quantized for the next forward pass computation carried out in (3.6). Note that this technique is widely used for the training of quantized deep neural



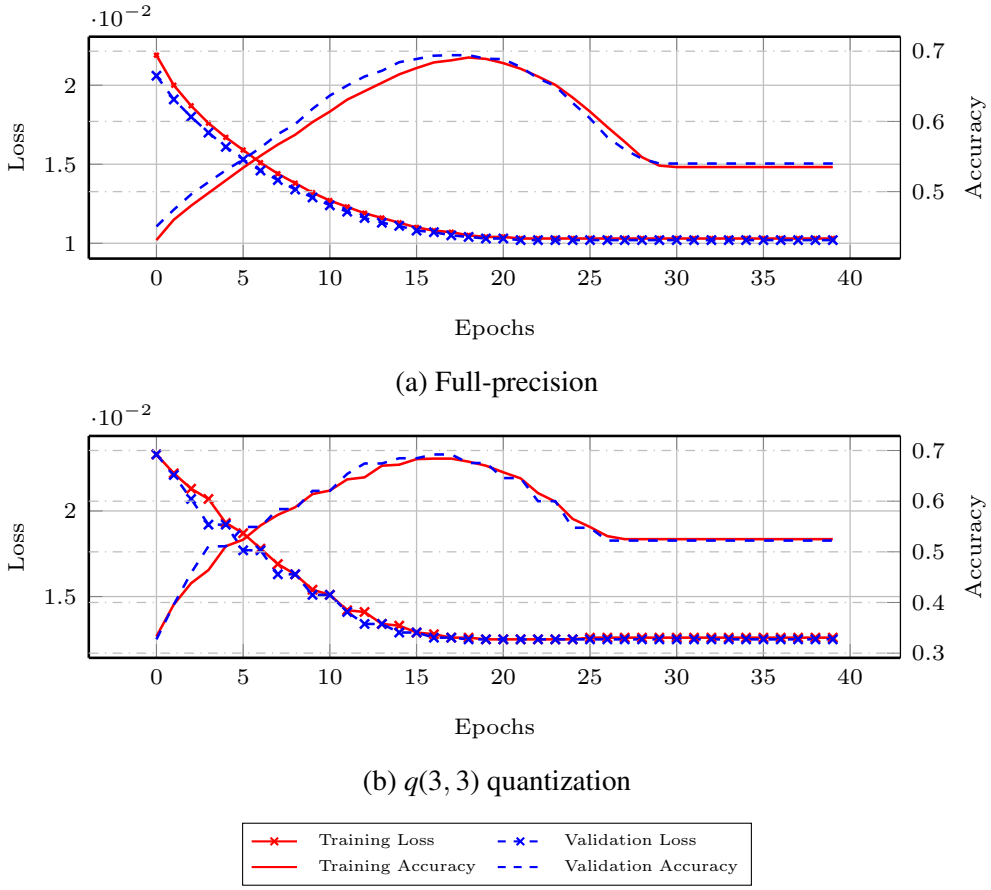
(a) $\mathcal{P}(512, 256)$ (b) $\mathcal{P}(512, 384)$

Figure 3.2: Effect of quantization on the FER of ideal DSCF decoding for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. The polar codes are concatenated with a 24-bit CRC.

networks [61]. Given the quantized value of θ, in the decoding phase, NSCF decoding performs all of the SC decoding operations as well as the bit-flipping metric computations in (3.5) using a single quantization scheme, which is characterized by a set of quantization parameters.

To find the quantization parameters for the proposed NSCF decoder, we evaluate the quantization parameters on the ideal DSCF decoder and use those quantization parameters in the proposed NSCF decoder. Let $q(n, m)$ denote a quantization configuration where n and m indicate the number of binary bits used to represent the integral and fractional parts of a floating-point number, respectively. Fig. 3.2 compares the FER of the ideal DSCF decoder when the soft messages used by SC decoding are quantized using $q(2, 3)$, $q(3, 2)$, and $q(3, 3)$ formats. As observed from Fig. 3.2, the $q(3, 3)$ format introduces almost no FER performance degradation compared to the full-precision ideal DSCF decoder for both $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. Therefore, we select $q(3, 3)$ as the quantization scheme for the proposed NSCF decoder.

In the following sections, we first provide the training results for both full-precision and quantized scenarios. We then evaluate the error-correction performance and decoding latency of the proposed decoders.



(a) Full-precision

(b) $q(3,3)$ quantization

Figure 3.3: Plot of training (validation) accuracy and loss of the full-precision and quantized models when $\omega = 3$ for $\mathcal{P}(512, 256)$. The value of θ is selected at the epoch that has the highest validation accuracy.

3.1.4 Parameter Optimization Results

In this thesis, we use Pytorch [62] to implement the parameter optimization framework[2]. We use 5000 samples to optimize θ at each error order $\omega \in \{1, 2, 3\}$ individually, with 4000 samples used for training and 1000 samples used for validation. In addition, the training samples are obtained at $E_b/N_0 = 3.0$ dB and $E_b/N_0 = 4.0$ dB for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$, respectively. For both full-precision and quantized scenarios, the learning rate λ and the forgetting factor γ used in (3.17) and (3.18) are set to 5×10^{-4} and 0.9, respectively. The mini-batch size is 200 and the number of

[2]We manually implement the computations in (3.6)-(3.19) instead of using the built-in automatic differentiation mechanism and SGD-based optimizers supported by Pytorch. The main purpose of using Pytorch is to make use of its GPU support to reduce the training time.

training epochs is 40. Initially, the value of θ at each error order is drawn from an i.i.d distribution in the range of $(0, 5)$. When training with quantization, θ is in $q(2, 3)$ format and the bit-flipping metric Q_{i_ω} is in $q(3, 3)$ format. We do not use a sign bit for the quantized values of θ and Q_{i_ω}.

Fig. 3.3 illustrates the training (validation) loss in accordance with (3.10) and accuracy, i.e., the probability that the estimated error positions match the training labels, when the θ parameter is optimized for $\omega = 3$ with $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. As the optimization of θ is formalized as a classification process, the training (validation) accuracy depicted in Fig. 3.3 indicates the probability that the estimated error bit ι_ω is the correct bit-flipping index t_ω. In the ML literature, the accuracy considered in Fig. 3.3 is also referred as the top-1 accuracy in a classification task. It can be observed that for both full-precision and quantized scenarios, the training loss and the validation loss values are almost similar, indicating that the θ parameter is generalized well for unseen samples. The value of θ is then selected at the training epoch that has the highest validation accuracy. It can also be observed that the full-precision model provides a smoother learning curves compared to those of the quantized model. This is because the quantized model often requires more training epochs than the full-precision model to update the parameter.

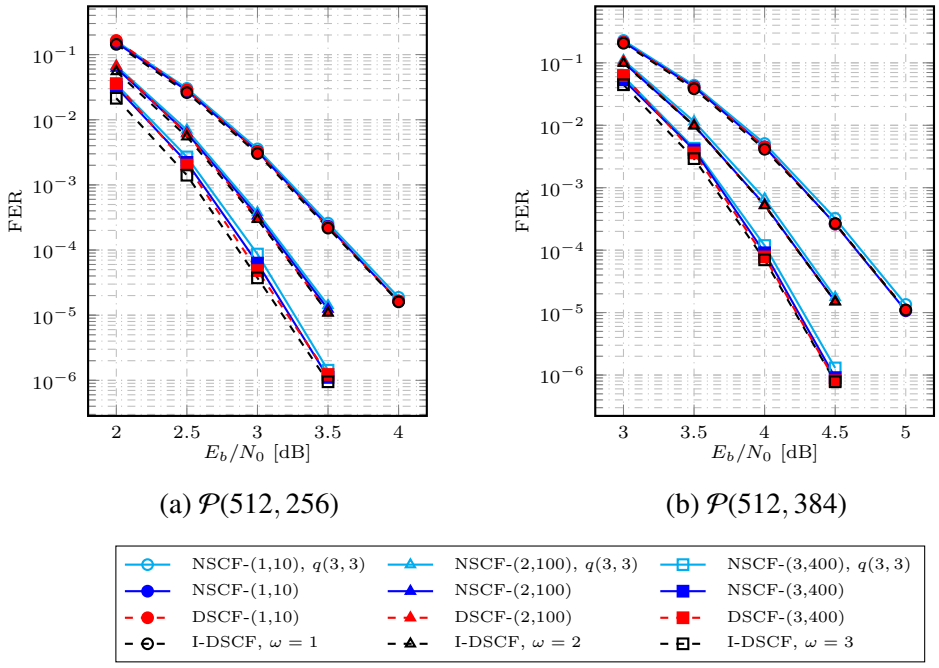
Table 3.1: Optimized parameter θ at each error order of the proposed NSCF decoders.

	ω		1	2	3
θ	$\mathcal{P}(512, 256)$	Full-precision	0.9772	0.8166	0.7046
		$q(0, 3)$	0.875	0.75	0.625
	$\mathcal{P}(512, 384)$	Full-precision	0.7993	0.3671	0.3243
		$q(0, 3)$	0.5	0.375	0.375

Table 3.1 provides the optimized values of θ for both full-precision and quantized formats for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. As the values of θ at all error orders are within the range of $(0, 1)$, during the decoding phase, θ is quantized using the $q(0, 3)$ format.

3.1.5 Error-Correction Performance

The error-correction performance of the proposed decoders in terms of FER are evaluated using the optimized values of θ. We use the same polar codes $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$ as in Fig. 3.1 and Fig. 3.2 to evaluate the error-correction performance of the proposed decoders. The FERs of the DSCF and NSCF at error order ω, with a maximum of m_ω attempts, are denoted as DSCF-



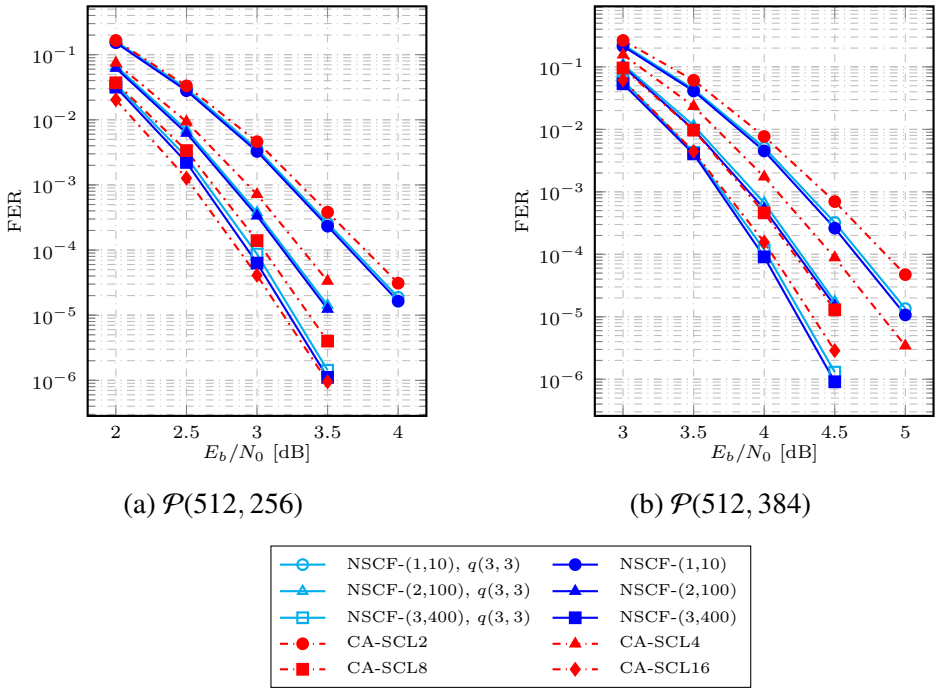
(a) $\mathcal{P}(512, 256)$ (b) $\mathcal{P}(512, 384)$

Figure 3.4: FER performance of the proposed decoders for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. The polar codes are concatenated with a 24-bit CRC. The FERs of the full-precision DSCF and ideal DSCF (I-DSCF) decoders are also plotted for comparison.

(ω, m_ω) and NSCF-(ω, m_ω), respectively, and are shown in Fig. 3.4 and Fig. 3.5. In addition, the FERs of the ideal DSCF decoder, denoted as I-DSCF, are plotted for comparison. In this thesis, we set $\omega \in \{1, 2, 3\}$ and the number of maximum decoding attempts for all the DSCF and NSCF decoders are $m_\omega \in \{10, 100, 400\}$, respectively. The bit-flipping metric used for DSCF decoding for all the simulations is calculated as in (2.22).

It can be seen in Fig. 3.4 that for $\omega = \{1, 2\}$, the proposed decoder in full-precision and in $q(3, 3)$ formats experiences almost no error-correction performance degradation compared to the ideal DSCF decoder. At $\omega = 3$, the error-correction performance of the NSCF decoder in full-precision and in quantized schemes only has a degradation of less than 0.1 dB at the target FER of 10^{-4} when compared to that of the ideal DSCF decoder for both $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$. On the other hand, when compared with that of the DSCF decoder, the FER performance loss of the proposed decoder is negligible at all considered E_b/N_0 values.

Fig. 3.5 shows the FER performances of the proposed NSCF decoders and those of the CRC-aided SCL (CA-SCL) decoders [1] with list size m_L, denoted as CA-SCLm_L, where $m_L \in \{2, 4, 8, 16\}$.



(a) $\mathcal{P}(512, 256)$ (b) $\mathcal{P}(512, 384)$

Figure 3.5: FER comparison of the proposed NSCF decoders and CA-SCL decoders in [1].

It can be seen that for $\mathcal{P}(512, 256)$, at the target FER of 10^{-4}, compared to CA-SCL with $m_L \in \{2, 4, 8\}$, the NSCF decoders at $m_\omega \in \{1, 2, 3\}$ obtain the FER performance gains of up to 0.1 dB. Moreover, for $\mathcal{P}(512, 256)$, the proposed NSCF decoder at $m_\omega = 3$ only experiences an error-correction performance loss of less than 0.1 dB compared to CA-SCL16, at the same target FER. In the case of $\mathcal{P}(512, 384)$, at the target FER of 10^{-4} the NSCF decoder at $m_\omega \in \{1, 2, 3\}$ obtains the FER gains of at least 0.2 dB when compared with the CA-SCL decoder with list size $m_L \in \{2, 4, 8\}$, respectively. In addition, for $\mathcal{P}(512, 384)$ the NSCF decoder at $\omega = 3$ has a slightly better error correction performance when compared with that of CA-SCL16 at the same target FER.

3.1.6 Complexity Reduction and Decoding Latency

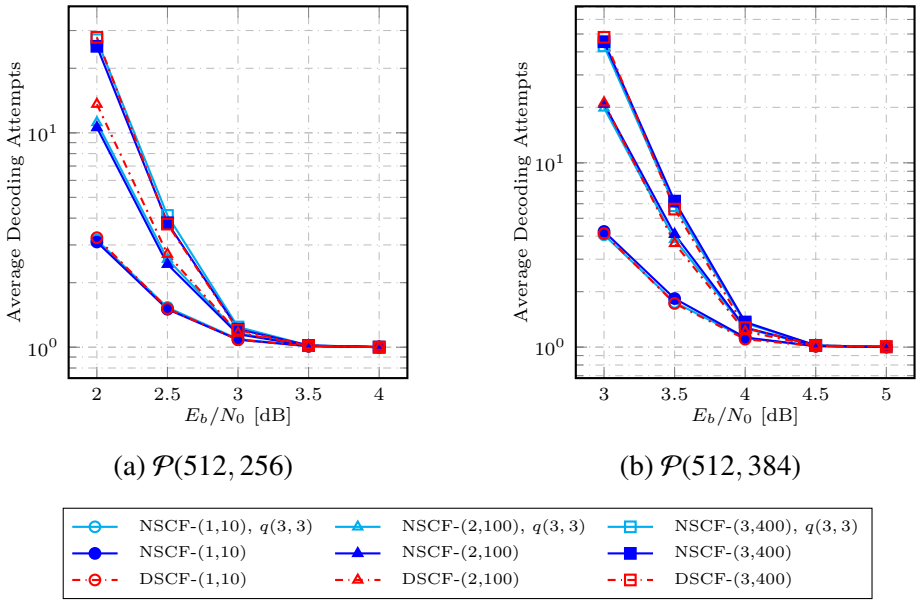
Since DSCF and NSCF decoding algorithms both rely on SC decoding algorithm, the only difference in terms of computational complexity comes from the bit-flipping metric computation. Table 3.2 shows the average number of computations performed at $E_b/N_0 = 3$ dB for $\mathcal{P}(512, 256)$ and $E_b/N_0 = 4$ dB for $\mathcal{P}(512, 384)$, which are required by the bit-flipping metric calculation of the decoders in Fig. 3.4. Note that the bit-flipping metric computations of the DSCF decoder and that

of the proposed NSCF decoder are specified in (2.22) and (3.5), respectively.

Table 3.2: Computational complexity of the bit-flipping metric in terms of the average number of operations performed for different polar codes, which are concatenated with a 24-bit CRC used in 5G.

	ω	m_ω	Decoder	ln / exp	×	+
$\mathcal{P}(512, 256)$	1	10	DSCF	560	560	840
			NSCF	0	0	560
			NSCF-$q(3, 3)$	0	0	560
	2	100	DSCF	620.77	620.77	923.36
			NSCF	0	0	621.55
			NSCF-$q(3, 3)$	0	0	626.59
	3	400	DSCF	663.17	663.17	981.05
			NSCF	0	0	666.55
			NSCF-$q(3, 3)$	0	0	677.68
$\mathcal{P}(512, 384)$	1	10	DSCF	816	816	1224
			NSCF	0	0	816
			NSCF-$q(3, 3)$	0	0	816
	2	100	DSCF	933.67	933.67	1390.8
			NSCF	0	0	949.3
			NSCF-$q(3, 3)$	0	0	956.8
	3	400	DSCF	1020.32	1020.32	1512.7
			NSCF	0	0	1058.8
			NSCF-$q(3, 3)$	0	0	1059.9

It can be seen in Table 3.2 that for both full-precision and quantized schemes, the proposed NSCF decoder requires around 31% fewer total number of additions compared to the DSCF decoder at all error orders. In addition, for $\omega = \{2, 3\}$, the prediction of the quantized bit-flipping model of NSCF is less accurate compared with that of the full-precision model, thus it results in a slightly larger number of additions compared to the full-precision model. On the other hand, at $\omega = 1$, the average number additions required by the bit-flipping metric computation of the NSCF decoder is the same for both quantized and full-precision schemes. It can also be observed that the proposed bit-flipping metric computation completely removes the need to perform multiplications and costly transcendental computations, while only experiencing negligible error-correction



(a) $\mathcal{P}(512, 256)$ (b) $\mathcal{P}(512, 384)$

Figure 3.6: Average number of decoding attempts.

performance loss when compared to DSCF as observed in Fig. 3.4.

Fig. 3.6 depicts the average number of decoding attempts for the DSCF decoder and the proposed NSCF decoder. It can be seen that when $E_b/N_0 > 2.5$ dB for $\mathcal{P}(512, 256)$ and $E_b/N_0 > 3.5$ dB for $\mathcal{P}(512, 384)$, the average number of decoding attempts of the proposed NSCF decoder is similar to that of the DSCF decoder, under the same decoding configurations. Note that the average number of decoding attempts of all the decoders depicted in Fig. 3.6 approaches 1 at high E_b/N_0 values. This also indicates that at high SNR regime, the average complexity of all the DSCF-based decoders approaches the complexity of a single SC decoder, while their error-correction performance is comparable to that of CA-SCL decoder as observed from Fig. 3.4 and Fig. 3.5.

3.2 Reinforcement-Learning-Aided Fast-SCF Decoding

Several attempts have integrated fast decoding operations to SCF decoding to improve its decoding latency [26, 32, 63, 64]. It was shown in [32] that using the DSCF-based bit flipping model for FSC decoding results in a better error-correction performance than the decoders in [26, 63, 64]. In this chapter, we propose a novel bit-flipping algorithm for FSC decoding that significantly improves

the error-correction performance of the decoder in [32]. In particular, a new bit-flipping strategy tailored to single parity-check (SPC) constituent codes is proposed. Furthermore, a new parameterized bit-flipping model based on [65] is developed to exploit the inherent correlations of the decoded bits under FSC decoding. We formalize the parameter optimization of the proposed bit-flipping model as an on-policy reinforcement learning (RL) problem and use RL techniques [66] to optimize the parameters during the course of decoding. Simulation results show that for a 5G polar code of length 512 with 256 information bits and concatenated with a 24-bit CRC, the proposed decoder has a better or similar error-correction performance compared to the state-of-the-art fast DSCF (FDSCF) decoding algorithm in [32], when the same number of maximum decoding attempts is considered.

3.2.1 Bit-Flipping Scheme for FSC Decoding

The proposed decoding algorithm is based on the generation of a specific vector of LLR values. Consider the first FSC decoding attempt does not satisfy the CRC verification and let v be a node located at the s-th stage ($s \geq 0$) of the polar decoding tree that is visited by FSC decoding. We construct a vector $\boldsymbol{\gamma} = \{\gamma_0, \gamma_1, \ldots\}$ by the following procedure:

- If v is a leaf node that contains an information bit:

$$\boldsymbol{\gamma} = \text{concat}(\boldsymbol{\gamma}, \boldsymbol{\alpha}_{0,i_{\min_v}}). \tag{3.20}$$

- If v is a REP node:

$$\boldsymbol{\gamma} = \text{concat}(\boldsymbol{\gamma}, \sum_{i=i_{\min_v}}^{i_{\max_v}} \boldsymbol{\alpha}_{s,i}). \tag{3.21}$$

- If v is a Rate-1 node:

$$\boldsymbol{\gamma} = \text{concat}(\boldsymbol{\gamma}, \boldsymbol{\alpha}_{s,i}), \tag{3.22}$$

  for all $i_{\min_v} \leq i \leq i_{\max_v}$.

- If v is an SPC node:

$$\boldsymbol{\gamma} = \text{concat}(\boldsymbol{\gamma}, \boldsymbol{\alpha}_{s,i}), \tag{3.23}$$

  for all $i_{\min_v} \leq i \leq i_{\max_v}$ and $i \neq i_v$, where $i_v = \arg\min_{i_{\min_v} \leq i \leq i_{\max_v}} |\alpha_{s,i}|$.

Note that concat($\boldsymbol{\gamma}, a$) indicates a concatenation of $a \in \mathbb{R}$ to the end of $\boldsymbol{\gamma}$ and $\boldsymbol{\gamma} = \emptyset$ initially. Also note that the size of $\boldsymbol{\gamma}$ at the end of the first FSC decoding attempt is $K + C$. In addition, $\boldsymbol{\gamma}$ remains unchanged if ν does not satisfy any of the above conditions.

The generation of $\boldsymbol{\gamma}$ allows us to directly predict the index of the first erroneous bit in $\boldsymbol{\gamma}$, denoted as ι_e. In the next FSC decoding attempt, the proposed decoder flips the hard decision associated with the ι_e-th LLR value in $\boldsymbol{\gamma}$ and continues the FSC decoding operations. One important consequence of using $\boldsymbol{\gamma}$ to predict the bit-flipping positions is for SPC nodes. In fact, there is no need to perform a bit-flipping for the least reliable bit of an SPC node. The value of this bit is calculated from all the other bits in the SPC node to satisfy the parity-check constraint (see (2.11)). Therefore, the hard decision value of the least reliable bit is automatically adjusted when another bit in an SPC node is flipped. As a result, the proposed algorithm only considers $N_\nu - 1$ possibilities to identify a bit flip that occurs in an SPC node. This is significantly smaller than the maximum search space of size $\binom{N_\nu}{2}$ required to flip a pair of indices, especially as N_ν increases.

To estimate ι_e, we propose a method that learns the correlations of the LLR values in $\boldsymbol{\gamma}$. Let η_i be the hard decision value of γ_i and let l_i^* be the likelihood ratio that η_i is correctly decoded given $\boldsymbol{y}$ and $\boldsymbol{u}$. l_i^* is calculated as

$$l_i^* = \max\left\{\frac{\Pr(\eta_i = 0|\boldsymbol{y}, \boldsymbol{u})}{\Pr(\eta_i = 1|\boldsymbol{y}, \boldsymbol{u})}, \frac{\Pr(\eta_i = 1|\boldsymbol{y}, \boldsymbol{u})}{\Pr(\eta_i = 0|\boldsymbol{y}, \boldsymbol{u})}\right\}. \tag{3.24}$$

ι_e is then obtained as

$$\iota_e = \underset{\forall i,\, 0 \le i < K+C}{\arg\min}\ l_i^*. \tag{3.25}$$

Since $\boldsymbol{u}$ is unknown, it is practically impossible to calculate l_i^* during the course of decoding. Thus, we estimate l_i^* as [65]

$$l_i^* \approx \prod_{\forall j,\, 0 \le j < K} l_j^{\Theta_{i,j}}, \tag{3.26}$$

where

$$l_j = \max\left\{\frac{\Pr(\eta_j = 0|\boldsymbol{y})}{\Pr(\eta_j = 1|\boldsymbol{y})}, \frac{\Pr(\eta_j = 1|\boldsymbol{y})}{\Pr(\eta_j = 0|\boldsymbol{y})}\right\} = \exp\big(|\gamma_j|\big) \tag{3.27}$$

and $\Theta_{i,j} \in \mathbb{R}$ are perturbation parameters such that $\Theta_{i,j} = \Theta_{j,i}$ and $\Theta_{i,i} = 1$, for $0 \le i, j < K + C$. The perturbation parameters are such that if there is a correlation between the i-th and j-th decoded bits, $\Theta_{i,j}$ and $\Theta_{j,i}$ are nonzero values, otherwise $\Theta_{i,j} = \Theta_{j,i} = 0$.

To enable numerically stable computations, the likelihood ratio l_i^* can be transformed to the

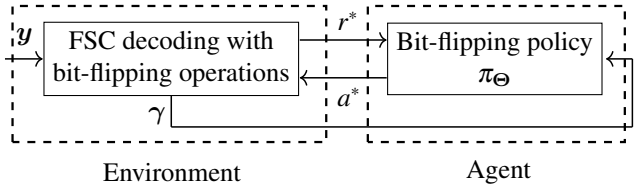


Figure 3.7: The training setup of the proposed bit-flipping policy when formalized as a RL problem.

LLR domain as

$$\begin{aligned} M_i = \ln(l_i^*) &\approx \ln\left(\prod_{\forall j, 0\le j<K} \exp\left(\Theta_{i,j}|\gamma_j|\right)\right) \\ &= \sum_{\forall j, 0\le j<K+C} \Theta_{i,j}|\gamma_j|. \end{aligned} \tag{3.28}$$

The most probable bit-flipping index ι_e is then selected as

$$\iota_e = \underset{\forall i, 0\le i<K+C}{\arg\min}\ M_i. \tag{3.29}$$

3.2.2 Parameter Optimization

In this section, we formalize the optimization of the matrix of parameters Θ, with $\Theta_{i,j}$ as the element in its i-th row and j-th column, as an on-policy RL problem. We use policy gradient techniques to train Θ [67, Chapter 13]. Let π_Θ be a bit-flipping policy characterized by Θ. The input of π_Θ is γ and the output of π_Θ is a probability vector $\boldsymbol{p} = \{p_0, p_1, \ldots, p_{K+C-1}\}$, where $\boldsymbol{p} = \pi_\Theta(\gamma)$ and p_i indicates the probability that $\iota_e = i$. The value of p_i can be obtained from the bit-flipping metric in (3.28) as

$$p_i = \frac{\exp(-M_i)}{\sum_{j=0}^{K+C-1} \exp\left(-M_j\right)}. \tag{3.30}$$

It can be seen from (3.28) and (3.30) that the bit index that has the smallest bit-flipping metric is also the bit index that has the highest probability to be flipped.

Fig. 3.7 illustrates the parameter optimization framework used in this chapter in an RL setup, in which the bit-flipping policy π_Θ acts as an online learning agent and the FSC decoder with the proposed bit-flipping operations is categorized as part of the environment. Given that the first FSC

decoding attempt is not successful, a bit index a^* ($0 \leq a^* < K + C$) is first sampled from the categorical distribution of the bit-flipping policy $\pi_{\boldsymbol{\Theta}}$, where the selection probability of the i-th bit is p_i. In the RL terminology, a^* is referred to as the action selected by the agent (with the bit-flipping policy $\pi_{\boldsymbol{\Theta}}$). We denote by $r^* \in \{0, 1\}$ a reward value associated with the bit-flipping index a^*. If the resulting message word satisfies the CRC verification after the a^*-th bit of $\boldsymbol{\gamma}$ is flipped in the secondary FSC decoding attempt, a reward of 1 ($r^* = 1$) is given to the a^*-th output of $\pi_{\boldsymbol{\Theta}}$, otherwise the reward is set to 0 ($r^* = 0$).

In practice, a bit-flipping set $\mathbb{A}$ that contains $T_{\max}$ ($1 \leq T_{\max} \leq K + C$) different bit-flipping indices is considered at the secondary decoding attempts. The bit-flipping set $\mathbb{A}$ is first constructed to contain the most probable erroneous indices as

$$\mathbb{A} = \{a_0, a_1, \ldots, a_{T_{\max}-1}\}, \tag{3.31}$$

where $p_{a_0} \geq p_{a_1} \geq \ldots \geq p_{a_{T_{\max}-1}}$ and $a_0 = \arg\max_{\forall i, 0 \leq i < K+C} p_i$. If the sampled bit index a^* is not in $\mathbb{A}$, the bit index $a_{T_{\max}-1}$ is replaced by a^*. The proposed decoder then performs consecutive FSC decoding attempts, each time flipping a different bit index given in $\mathbb{A}$.

Given a sample of the LLR vector $\boldsymbol{\gamma}$, the objective of the bit-flipping policy $\pi_{\boldsymbol{\Theta}}$ is to derive an action a^* that maximizes the expected reward value of r^*, which is given as [66]

$$J(\boldsymbol{\Theta}) = E_{a^* \sim \pi_{\boldsymbol{\Theta}}}[r^*] \approx \frac{1}{|\mathbb{D}|} \sum_{\forall \boldsymbol{\gamma} \in \mathbb{D}} r^*, \tag{3.32}$$

where $\mathbb{D}$ is a mini-batch of the dataset that contains B different instances of $\boldsymbol{\gamma}$, and $B = |\mathbb{D}|$. The derivative of the objective function $J(\boldsymbol{\Theta})$ with respect to the parameter set $\boldsymbol{\Theta}$ can be derived as [66]

$$\boldsymbol{\nabla}_{\boldsymbol{\Theta}} = \frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} = \frac{1}{|\mathbb{D}|} \sum_{\forall \boldsymbol{\gamma} \in \mathbb{D}} \frac{\partial \ln p_{a^*}}{\partial \boldsymbol{\Theta}} (r^* - \bar{r}), \tag{3.33}$$

where $\bar{r} \in \mathbb{R}$ is a baseline reward [67, Section 13.4]. The parameters in $\boldsymbol{\Theta}$ are then updated using a variant of the stochastic-gradient ascent technique, where the parameters are updated as

$$\boldsymbol{\Theta} = \boldsymbol{\Theta} + \lambda \boldsymbol{\nabla}_{\boldsymbol{\Theta}} \tag{3.34}$$

and $\lambda > 0$ is the learning rate.

Algorithm 2: RL-Aided FSCF Decoding

Input : $T_{\max}$, B, λ
Output: $\hat{\boldsymbol{u}}$

$t \leftarrow 1$, $\bar{r} \leftarrow 0$, $\boldsymbol{\nabla}_{\text{tmp}} \leftarrow \mathbf{0}$ /* Initialization */
while *True* **do**
  Obtain $\boldsymbol{\alpha_n}$ from the channel output $\boldsymbol{y}$
  $\hat{\boldsymbol{u}}, \boldsymbol{\gamma} \leftarrow \text{FSC}(\boldsymbol{\alpha_n})$ /* Initial FSC Decoding */
  if $\hat{\boldsymbol{u}}$ *fails the CRC test* **then**
    /* Action Selection */
    Obtain $\boldsymbol{M}$, $\boldsymbol{p}$, $\mathbb{A}$ using (3.28), (3.30), and (3.31)
    $a^* \sim \boldsymbol{\pi}_{\boldsymbol{\Theta}}$, $r^* \leftarrow 0$
    if $a^* \notin \mathbb{A}$ **then**
      $a_{T_{\max}-1} \leftarrow a^*$
    /* Proposed FSCF with Bit Flipping */
    for $j \leftarrow 0$ **to** $T_{\max} - 1$ **do**
      $\hat{\boldsymbol{u}}_{\text{tmp}} \leftarrow \text{FSCF}(\boldsymbol{\alpha_n}, a_j)$ /* FSCF Decoding */
      if $\hat{\boldsymbol{u}}_{tmp}$ *passes the CRC test* **then**
        $a^* \leftarrow a_j$, $r^* \leftarrow 1$, $\hat{\boldsymbol{u}} \leftarrow \hat{\boldsymbol{u}}_{\text{tmp}}$
        break
    /* Parameter Optimization */
    $\boldsymbol{\nabla}_{\text{tmp}} \leftarrow \boldsymbol{\nabla}_{\text{tmp}} + \frac{\partial \ln p_{a^*}}{\partial \boldsymbol{\Theta}}(r^* - \bar{r})$
    $\bar{r} \leftarrow \bar{r} + \frac{r^* - \bar{r}}{t}$ /* Update baseline reward */
    if $(t \mod B) == 0$ **then**
      $\boldsymbol{\nabla}_{\boldsymbol{\Theta}} \leftarrow \boldsymbol{\nabla}_{\text{tmp}}/B$
      $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} + \lambda \boldsymbol{\nabla}_{\boldsymbol{\Theta}}$
      $\boldsymbol{\nabla}_{\text{tmp}} \leftarrow \mathbf{0}$
    $t \leftarrow t + 1$ /* Increase the time step */
  Output $\hat{\boldsymbol{u}}$

We provide the details of the proposed decoder with the optimization of $\boldsymbol{\Theta}$ in Algorithm 2. In Algorithm 2, t is the time step, $\boldsymbol{\nabla}_{\text{tmp}}$ is the matrix of size $(K+C)\times(K+C)$ that stores the gradients of $\boldsymbol{\Theta}$, and $\mathbf{0}$ is an all-zero matrix of size $(K+C)\times(K+C)$. In addition, the cumulative average reward is used as the reward baseline [67, Section 13.4], which is periodically updated at each time step.

3.2.3 Simulation Results

In this section, we first provide the training results of the proposed bit-flipping model in Algorithm 2. Throughout this section, we use $\mathcal{P}(512, 256)$ concatenated to a 24-bit CRC as stated in the 5G standard. We use the bit-flipping models in [40] and [29] as a benchmark of estimating the first error bit in $\boldsymbol{\gamma}$. Based on [29, 40] the first error bit of $\boldsymbol{\gamma}$ can be obtained as follows.

- SCF-based model [40]: $\iota_e = \arg\min_{\forall i,\, 0\leq i<K+C} |\gamma_i|$.
- DSCF-based model [29]: $\iota_e = \arg\min_{\forall i,\, 0\leq i<K+C} Q_i$, where

$$Q_i = |\gamma_i| + \sum_{\forall j\leq i} \frac{1}{0.3} \ln\Big[1 + \exp\Big(-0.3|\gamma_j|\Big)\Big].$$

In addition, we also consider an ideal bit-flipping model which can identify ι_e correctly given $\boldsymbol{\gamma}$ and $\boldsymbol{u}$.

Fig. 3.8 compares the performance of different bit-flipping models (policies) in terms of the cumulative average reward, denoted as $\bar{r}$, when applied to the proposed bit-flipping algorithm. We also consider the conventional supervised learning (SL) technique as an optimization scheme for the parameter set $\boldsymbol{\Theta}$ and compare it with the proposed RL-based optimization scheme. 10^5 training samples are obtained with all-zero codewords to train the parameter set $\boldsymbol{\Theta}$ when the SL techniques are used. In fact, increasing the training samples of the SL approach does not improve $\bar{r}$ in our problem setting. For both the RL-based and SL-based approaches, we set $B = 100$, $\lambda = 2 \times 10^{-5}$ and use PyTorch with Adam optimizer [68] as the training framework.

It can be observed from Fig. 3.8 that the cumulative average reward of the proposed bit-flipping model when trained with RL or SL techniques outperforms that of the DSCF-based [29] and SCF-based [40] models. This is because the proposed algorithm utilizes a more powerful predictive model characterized by $\boldsymbol{\Theta}$, where $|\boldsymbol{\Theta}| = (K+C)^2$, while the models used in [29,40] only contain up to a single trainable parameter. Moreover, the RL-based optimization approach provides a slight improvement in the cumulative average reward when compared with the SL-based approach, as the objective functions of the RL-based and SL-based approaches are different. The SL-based approach trains the model to make an ML decision given the training dataset. On the other hand, the RL-based approach trains the model to directly maximize the numerical reward $\bar{r}$. Fig. 3.9 shows the FER of the proposed bit-flipping algorithm with the bit-flipping models shown in Fig. 3.8 for

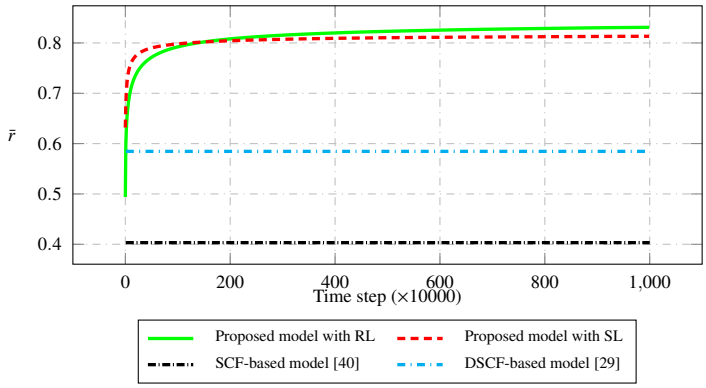


Figure 3.8: The cumulative average rewards of various bit-flipping models when applied to the proposed bit-flipping algorithm. The simulation is carried out at $E_b/N_0 = 3$ dB for $\mathcal{P}(512, 256)$ with a 24-bit CRC, and $T_{\max} = 1$.

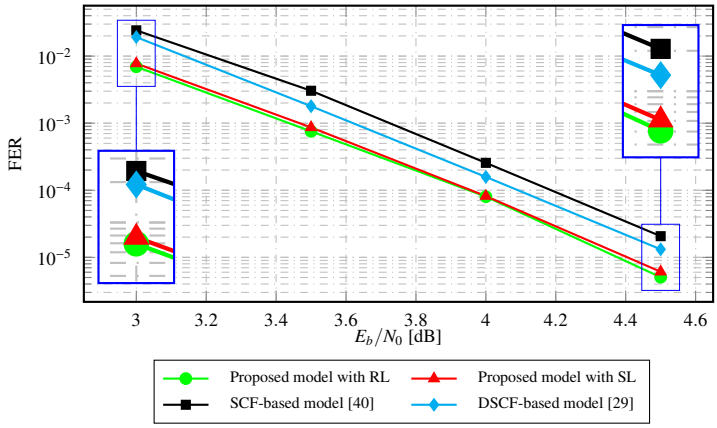


Figure 3.9: The error-correction performance of the proposed bit-flipping algorithm with various bit-flipping models in Fig. 3.8.

$T_{\max} = 1$. The experimental results in Fig. 3.9 show that a direct optimization of $\boldsymbol{\Theta}$ to maximize $\bar{r}$ also results in the best error-correction performance of the proposed decoder, when compared with other approaches in Fig. 3.8. Note that the parameter set $\boldsymbol{\Theta}$ is optimized at each SNR value for the SL-based approach and the FERs are simulated using 10^7 frames at each SNR value.

It is worth mentioning that by formalizing the optimization of $\boldsymbol{\Theta}$ as an RL problem, the training can be carried out at the decoder side without the need of pilot signals, which is suitable for a pilot-less communication system. Furthermore, unlike the SL approach, the training of $\boldsymbol{\Theta}$ does not require a large memory to store the training dataset as observed from Algorithm 2.

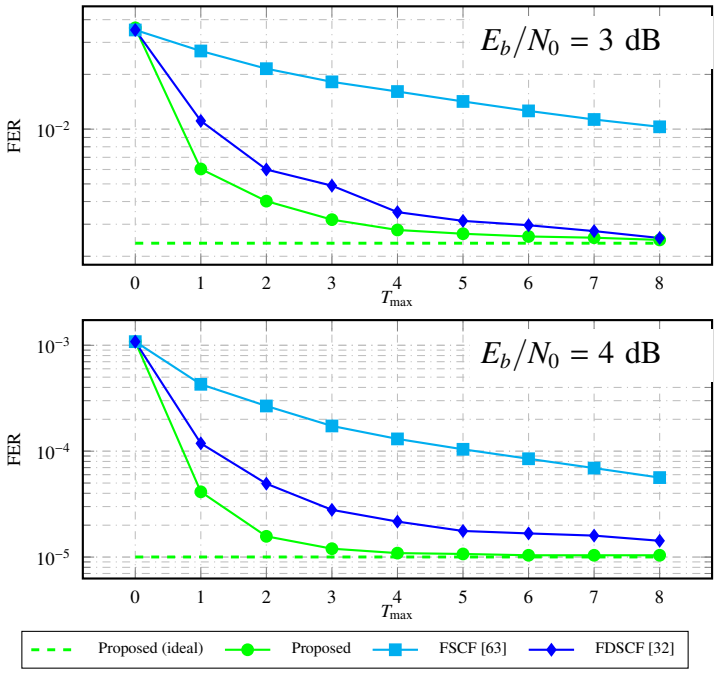


Figure 3.10: The FER of various fast SCF decoding algorithms as a function of $T_{\max}$ at $E_b/N_0 = \{3, 4\}$ dB.

Fig. 3.10 illustrates the FER curves of various fast SCF decoders with different values of $T_{\max}$ at $E_b/N_0 = \{3, 4\}$ dB. With $T_{\max} = 0$, all the decoders in Fig. 3.10 revert to the FSC decoder [24]. It can be seen from Fig. 3.10 that at the same value of $T_{\max}$ when $1 \leq T_{\max} \leq 8$, the proposed decoder has the best error-correction performance when compared to the decoders in [32, 63]. At $T_{\max} = 8$, the proposed decoder only experiences a negligible error-correction performance degradation compared to the ideal bit-flipping algorithm.

Fig. 3.11 shows the error probabilities of various bit-flipping algorithms of polar codes when $T_{\max} = 8$. The FERs of the state-of-the-art SCL-L decoding algorithm [1] are also plotted for comparison, where $L \in \{2, 4\}$ is the list size. It can be observed that the proposed decoder has a similar error-correction performance when compared with that of the DSCF [29] and FDSCF [32] decoders, respectively. At the target FER of 10^{-4}, the error probability of the proposed decoder is 0.25 dB and 0.2 dB better than that of the FSCF [63] and SCL-2 [1] decoders. At the same target FER, when compared with SCL-4, the proposed algorithm experiences an error-correction performance loss of 0.3 dB. In Fig. 3.12, the decoding latency, in terms of the average number of decoding attempts (T_{avg}) of various fast SCF decoders, is plotted for comparison. For all the

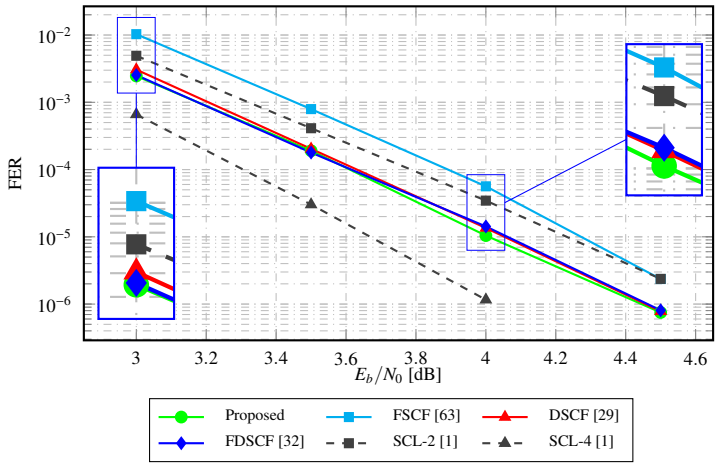


Figure 3.11: The error-correction performance of various decoding algorithms. $T_{\max}$ is set to 8 for all the bit-flipping algorithms.

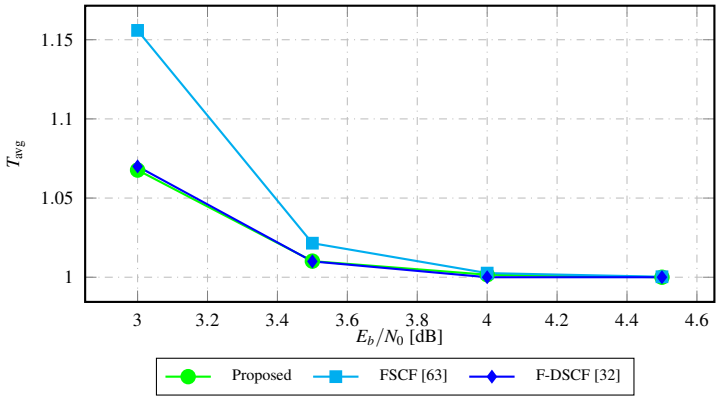


Figure 3.12: Average number of decoding iterations of various fast SCF algorithms with $T_{\max} = 8$.

decoders in Fig. 3.12, $T_{\max} = 8$. Note that the average number of decoding attempts of all the decoders depicted in Fig. 3.12 approaches 1 at high E_b/N_0 values. This indicates that at high E_b/N_0 values, the complexity of the decoders approaches the complexity of a single FSC decoder. In addition, the decoding latency of the proposed decoder in terms of T_{avg} is similar to that of the FDSCF decoder [32] at all the SNR values.

3.3 Chapter Conclusion

This chapter aims to reduce the computational complexity and latency of the dynamic SC-Flip (DSCF) decoding algorithm by replacing the costly multiplications and transcendental computations used in its bit-flipping model by an approximation that requires only additions. The proposed bit-flipping model is parameterized by a correlation matrix, which can provide an accurate estimation of the error indices and can be trained by a supervised learning approach. Furthermore, a novel reinforcement-learning-aided fast SC-Flip (FSCF) decoder is then proposed which provides a superior error-bit estimation accuracy when compared to the state-of-the-art FSCF decoder while relatively preserve the decoding latency of the FSCF decoder. In the next chapter, the worst-case latency of the decoders proposed in this chapter is further improved by extending the fast bit-flipping strategies to the state-of-the-art FSCL decoding algorithm of polar codes.

Chapter 4

Fast Successive-Cancellation List Flip Decoding of Polar Codes

In this chapter, the Fast-SCLF decoding algorithm is proposed to tackle the underlying high-decoding latency of the SCLF decoder introduced in [30]. In particular, a bit-flipping strategy tailored to FSCL decoding of polar codes is first introduced. Then, a path-selection error metric is derived for the proposed bit-flipping strategy. The proposed path-selection error metric utilizes a trainable parameter to improve the estimation accuracy of the error position, which is optimized online using an efficient supervised learning framework. By utilizing online training, the proposed path-selection error-model does not require the parameter to be optimized offline at various SNRs. Instead, the parameter is automatically optimized at the operating SNR of the decoder, which obviates the need for pilot signals.

4.1 Bit-flipping Scheme for FSCL Decoding

We first introduce the bit-flipping scheme tailored to FSCL decoding by illustrating the proposed scheme under various examples. We consider the case where an all-zero codeword of $\mathcal{P}(16, 8)$ is transmitted through the channel. Similar to SCL-based decoding, under FSCL-based decoding, we denote by l the path index corresponding to the current L active decoding paths, while $\tilde{l}$ is used to indicate the indices of the paths that are forked from l. Finally, l' indicates the path indices of the decoding paths that are discarded due to their high path metric values. Note that $l, \tilde{l}, l' \in [1, 2L]$.

Bit-Flipping Scheme for SPC Nodes

Table 4.1 shows an example of FSCL decoding when applied to the SPC node of $\mathcal{P}(16, 8)$ with $L = 4$. The decoding order is first determined by sorting the magnitude of the LLR values associated with the SPC node in the increasing order. In this example, the following decoding order is considered: $\{\beta_{2,8_l}, \beta_{2,5_l}, \beta_{2,7_l}, \beta_{2,6_l}\}$. Thus, $\beta_{2,8_l}$ is selected as the parity bit of the SPC node for all the active decoding paths. The path splittings at $\beta_{2,6_l}$ are considered in this example, and the paths with indices $\tilde{l} \in \{5, 6, 7, 8\}$ are forked from the paths with indices $l \in \{1, 2, 3, 4\}$ at $\beta_{2,6_l}$, respectively, followed by the path metric sorting operations. The most likely decoding paths with indices $l = \{1, 2, 3, 4\}$ are then selected to continue the decoding, while the paths with indices $l' \in \{5, 6, 7, 8\}$ are discarded as shown in Table 4.1.

At this stage, the parity bit $\beta_{2,8_l}$ of the SPC node is not yet decoded. As an all-zero codeword is considered, the correct decoding path is $l' = 5$, which is discarded after $\beta_{2,6_l}$ is decoded, i.e., after the third path-splitting index. Given that this erroneous decision in the initial FSCL decoding is detected by a CRC verification, this erroneous path selection is reversed in the next decoding attempt by swapping the path indices of l' and l after the path splittings at $\beta_{2,6_l}$ for all the decoding paths. The decoding continues by setting the values of the parity bit $\beta_{2,8_l}$ to maintain the parity constraint for all the corrected paths. Similar to the bit-flipping schemes introduced in [69, 70], in the proposed scheme, the bit-flipping operation is not applicable to the parity bits of the SPC nodes. This is due to the fact that the parity bits are determined after the all the other bits are calculated to ensure the parity check is satisfied. Therefore, if all the other bits of the SPC node are correctly decoded, the parity bit of this decoding path is also correctly decoded. As a result, the proposed algorithm only considers a maximum of $N_v - 1$ possibilities to identify a bit flip that occurs in an SPC node. This is significantly smaller than the maximum search space of size $\binom{N_v}{2}$ required to flip

Table 4.1: An example of FSCL decoding applied to an SPC node of size 4 with $L = 4$, where the decoding is at the third path splitting. $l' \in \{5, 6, 7, 8\}$ are the indices of the discarded paths.

Node type	Path splitting index	$l' = 5$	$l' = 6$	$l' = 7$	$l' = 8$
SPC	-	$\beta_{2,8_5} = \text{n/a}$	$\beta_{2,8_6} = \text{n/a}$	$\beta_{2,8_7} = \text{n/a}$	$\beta_{2,8_8} = \text{n/a}$
	1	$\beta_{2,5_5} = 0$	$\beta_{2,5_6} = 0$	$\beta_{2,5_7} = 1$	$\beta_{2,5_8} = 1$
	2	$\beta_{2,7_5} = 0$	$\beta_{2,7_6} = 1$	$\beta_{2,5_7} = 0$	$\beta_{2,5_8} = 1$
	3	$\beta_{2,6_5} = 0$	$\beta_{2,6_6} = 1$	$\beta_{2,6_7} = 0$	$\beta_{2,6_8} = 1$

a pair of bits to maintain the parity check constraint, especially as N_v increases [32].

Note that in this example, the minimum number of path splittings required by the SPC node to preserve the SCL decoding performance is $\tau = \min\{L-1, N_v-1\} = \min\{3,3\} = 3$, where v indicates the SPC node of size 4. Under the proposed bit-flipping scheme for SPC nodes, if a decision error occurs at the path-splitting index after the minimum number of τ path splittings are obtained, in the next decoding attempt, the hard values at the estimated error index are flipped for all the active paths. The path metrics PM_l of the surviving paths are then updated by using the LLR value corresponding to the flipped position and the current parity checksum p_l.

Bit-Flipping Scheme for REP Nodes

Since the soft and hard estimate of the information bit associated with a REP node can be directly obtained at the parent node level under FSCL decoding, the path splitting operation under FSCL decoding applied to the information bit of a REP node is similar to that of SCL decoding when applied to an information bit at the leaf node level. Therefore, in this chapter, the reversed path selection scheme used in SCLF decoding is directly applied to the information bit associated with a REP node or to an information bit at the leaf-node level under FSCL decoding [69, 70].

Bit-Flipping Scheme for Rate-1 Nodes

Table 4.2 shows an example of FSCL decoding on the Rate-1 node of $\mathcal{P}(16,8)$ at the fifth path-splitting index with $L = 2$. In Table 4.2, the hard estimates of the discarded paths with indices $l' \in \{2,4\}$ are indicated, while the hard estimates of the surviving paths with indices $l' \in \{1,3\}$ are omitted. It can be observed that the decoding path with index $l' = 2$ is the correct path as all the estimated bits are 0, which is discarded after bit $\beta_{2,13_2}$ is decoded. Therefore, in the next decoding attempt, the decoding paths with indices $l' \in \{2,4\}$ will be selected to continue the decoding instead of the paths with indices $l' \in \{1,3\}$ [69,70]. Similar to the case of SPC nodes, after τ path splittings, if the hard decision of a bit of the Rate-1 node results in the elimination of the correct path, this erroneous decision is reversed in the next FSCL decoding attempt by flipping the hard estimates of all the active paths at that erroneous index. The path metrics of the active paths are then added with the corresponding absolute LLR values of the flipping indices. In this example, the minimum number of path splittings is $\tau = \min\{L-1, N_v\} = \min\{1,4\} = 1$, which is obtained at the fifth path-splitting index. Therefore, under FSCL decoding, the hard values of all the active decoding

Table 4.2: An example of FSCL decoding applied to a Rate-1 node of size 4 with $L = 2$, where the decoding is at the 6-th path splitting. $l' \in \{2, 4\}$ are the indices of the discarded paths.

Node type	Path splitting index	$l' = 2$	$l' = 4$
SPC	-	$\beta_{2,8_2} = 0$	$\beta_{2,8_4} = 0$
	1	$\beta_{2,5_2} = 0$	$\beta_{2,5_4} = 0$
	2	$\beta_{2,7_2} = 0$	$\beta_{2,7_4} = 0$
	3	$\beta_{2,6_2} = 0$	$\beta_{2,6_4} = 0$
REP	4	$\beta_{0,12_2} = 0$	$\beta_{0,12_4} = 0$
Rate-1	5	$\beta_{2,13_2} = 0$	$\beta_{2,15_4} = 1$
	6	$\beta_{2,15_2} =$ n/a	$\beta_{2,16_4} =$ n/a
	7	$\beta_{2,16_2} =$ n/a	$\beta_{2,14_4} =$ n/a
	8	$\beta_{2,14_2} =$ n/a	$\beta_{2,13_4} =$ n/a

paths following the fifth path-splitting index are set to follow the signs of their LLR values.

Similar to SCLF decoding, the proposed scheme only aims at correcting the first erroneous decision in the initial FSCL decoding attempt. Fig 4.1 shows the ideal FER of the proposed bit-flipping scheme where the first erroneous path selection is always accurately corrected. In Fig 4.1, we use the 5G polar codes $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$ concatenated with a 24-bit CRC[1]. Note that the positions of the first erroneous decoding decision can be obtained by comparing the discarded paths with the correct path after each path splitting. The FER performance of the ideal SCLF [30] and SSCLF [69] decoders and the FSCL decoder with list size 32 [3] are also plotted for comparison. In Fig 4.1, the ideal SCLF, SSCLF, and Fast-SCLF decoders with list size L are denoted as I-SCLF-L, I-SSCLF-L and I-Fast-SCLF-L, respectively, with $L \in \{2, 4, 8, 16, 32\}$.

As seen from Fig 4.1, I-Fast-SCLF-L obtains a slight FER performance gain over I-SCLF-L. In addition, as the reversed path-selection scheme of [69] is not applied to the decoding steps that occur after the minimum number of τ path-splittings is obtained for the Rate-1 and SPC nodes, this simplified bit-flipping scheme of [69] introduces FER performance degradation when compared with the ideal SCLF and Fast-SCLF decoders, especially when the list size is small, ($L \in \{2, 4\}$). For $L = 4$ and at the target FER of 10^{-4}, the error-correction performance degradations of 0.2 dB and 0.3 dB are recorded for the ideal SSCLF decoder when compared to the ideal Fast-SCLF decoder for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$, respectively. Also note that the FER performance of the

[1]We use the 24-bit CRC specified as 24C in the 5G standard.

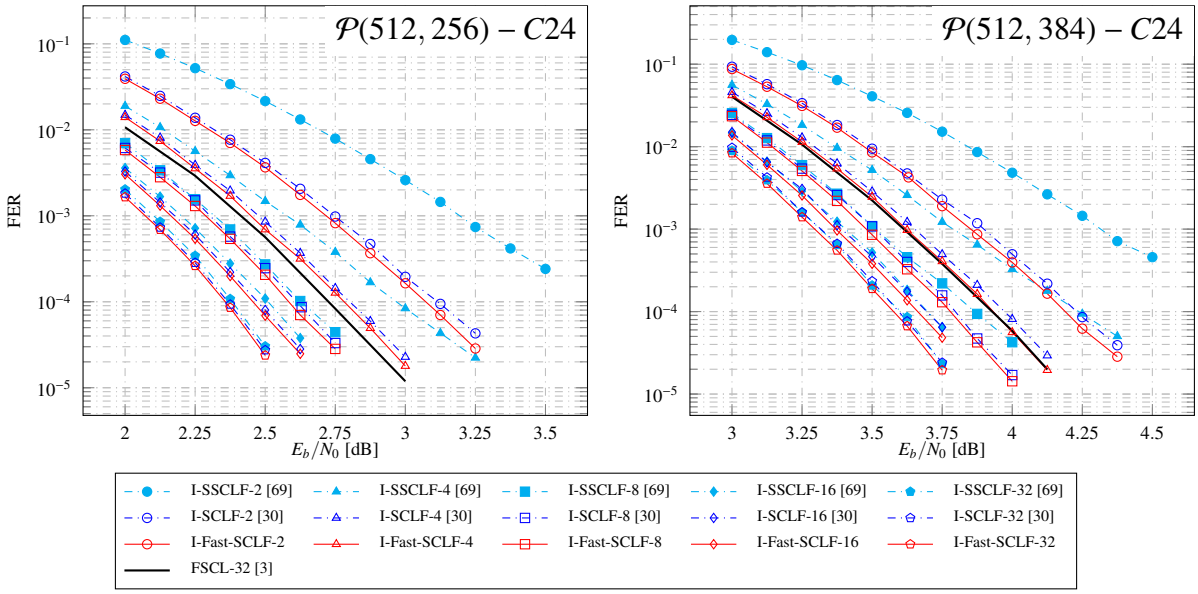
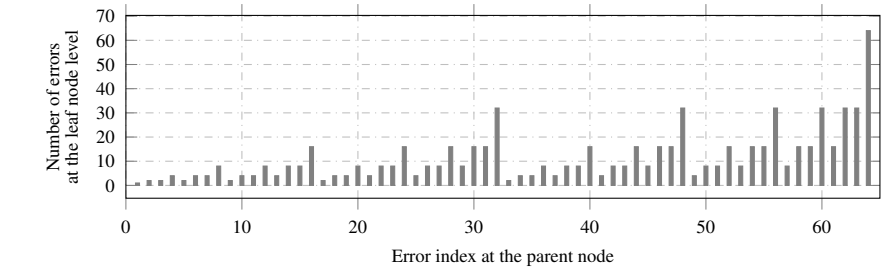


Figure 4.1: Ideal error-correction performance in terms of FER of various SCLF-based decoders. The FER values of the FSCL decoder with list size 32 are also plotted for comparison.

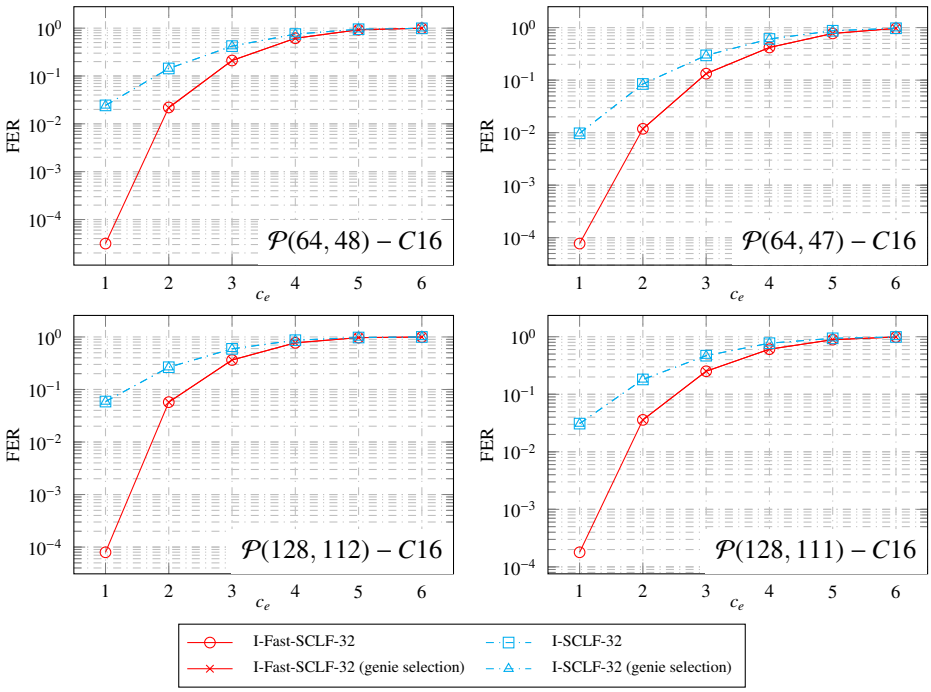
ideal SSCLF decoder with $L \in \{2, 4\}$ degrades quickly as the SNR increases.

We now explain the slight improvement in the error-correction performance of I-Fast-SCLF-L over that of I-SCLF-L as observed in Fig. 4.1. The error-correction performance of I-Fast-SCLF-L and I-SCLF-L are identical for REP nodes. Therefore, we empirically show that I-Fast-SCLF-L outperforms I-SCLF-L when applied to the same channel LLR vectors for Rate-1 and SPC nodes, where the LLR vectors contain an exact number of c_e ($c_e > 0$) channel errors. A similar study was conducted in [71] for the case of fast SCF decoding.

We first consider the error event where only a single error is present in the LLR vector of the Rate-1 and SPC nodes ($c_e = 1$), which causes an unsuccessful CRC verification in the first FSCL and SCL decoding attempt. After the correct decoding path is recovered in the second FSCL decoding attempt of I-Fast-SCLF-L, FSCL decoding operations, e.g., path forking and path metric sorting, are applied to the L recovered paths. Then, all the hard decisions of the correct path are set to follow the signs of the corresponding LLR values to maintain its path metric. Therefore, the path metric is equal to the absolute LLR value of the flipped bit. Recall that the LLR values of the Rate-1 and SPC nodes are sorted in accordance with (2.3). Thus, at the subsequent decoding steps after the flipped position, any new candidate path with at least a hard decision not following its corresponding LLR value will contain a higher path metric compared to that of the correct



(a)



(b)

Figure 4.2: (a) The number of translated errors at the leaf node level given a single error at a specific bit index at the parent node level for a sized-64 polar code and (b) the error-correction performance of I-Fast-SCLF-32 and I-SCLF-32 for the Rate-1 and SPC codes of lengths $N \in \{64, 128\}$ with various values of c_e.

decoding path. Therefore, with $c_e = 1$, the correct decoding path is always found in the list of the best paths after the second FSCL decoding attempt of I-Fast-SCLF-L.

Note that a single error at the parent node level can translate into multiple errors at the leaf node level. This phenomenon is illustrated in Fig. 4.2a for an all-zero polar code of length $N = 64$, where the number of error bits at the leaf node level is provided with respect to the position of the single error bit at the parent node level. Consequently, there are cases that I-SCLF-L has to perform the reserved path selection schemes multiple times to maintain the correct codeword in the list of the best paths, while I-Fast-SCLF-L only requires a single reserved path selection. Thus with $c_e = 1$, the error correction performance of I-Fast-SCLF-L is improved when compared to I-SCLF-L. As c_e increases ($c_e \geq 2$), due to the complicated error patterns caused by multiple error bits, we expect that both I-Fast-SCLF-L and I-SCLF-L almost equally likely discard the correct path in the second decoding attempt, causing a wrong estimation of the transmitted codeword. Also note that when the channel reliability is improved at the high SNR regimes and given that $c_e > 0$, the performance gain of I-Fast-SCLF-L over I-SCLF-L is mainly obtained from the case of $c_e = 1$, as the LLR vectors are more likely to contain a single error than multiple ones ($c_e > 1$). On the other hand, at the low SNR regimes, it is more likely to have multiple errors at the parent node level, thus the error-correction performance gain of I-Fast-SCLF-L is incremental when compared to that of I-SCLF-L. This phenomenon can also be observed from Fig. 4.1.

In Fig. 4.2b, we plot the FER curves of I-Fast-SCLF-32 and I-SCLF-32 for the polar codes of lengths 64 and 128 concatenated with a 16-bit CRC used in the 5G standard, the values of K are selected to form the Rate-1 and SPC nodes, respectively. The simulations are carried out at $E_b/N_0 = 3$ dB and the FER values of the decoders in Fig. 4.2b are only obtained for the channel LLR vectors that contain exactly $c_e \in \{1, 2, 3, 4, 5, 6\}$ errors. It can be confirmed from Fig. 4.2b that with $c_e = 1$, I-Fast-SCLF-32 has a significant FER performance improvement when compared to I-SCLF-32. In addition, the error-correction performance gains of I-Fast-SCLF-32 with respect to I-SCLF-32 quickly reduce as c_e increases, with the error probabilities of both decoders approaching 1 when $c_e \geq 4$.

Note that the error-correction performance degradation of I-Fast-SCLF-32 is also caused by the imperfect error detection of the CRC, where a wrong estimate of the correct codeword that satisfies the CRC is selected as the decoding output. As shown in Fig. 4.2b, if the CRC verification is replaced by a genie selection scheme, the FER values of I-SCLF-32 and I-Fast-SCLF-32 are relatively unchanged, except for the case of I-Fast-SCLF-32 with $c_e = 1$, where an FER of 0 is

obtained[2]. This confirms that after the second FSCL decoding attempt of I-Fast-SCLF-32, the correct codeword is always present in the list of the best decoding paths for $c_e = 1$.

In the next section, a path selection error model is derived to accurately estimate the index of the path splitting that causes the elimination of the correct path at the initial FSCL decoding. Therefore, the error-correction performance of the I-Fast-SCLF-L decoder provided in Fig. 4.1 serves as the empirical lower bound of the proposed decoding algorithm.

4.1.1 Path Selection Error Model for FSCL Decoding

We use the methods introduced in [29,30] to estimate the erroneous path-splitting index, which predicts the error position using the LLR values associated with each discarded decoding path. Thus, the proposed path selection error model relies on the construction of the LLR vectors obtained at each path splitting under FSCL decoding.

Consider that the first FSCL decoding attempt does not pass the CRC verification and ν is a node located at the s-th stage of the binary tree, that is visited by FSCL decoding. Let $k \in [1, K+C]$ be the path-splitting index that occurs during the decoding of ν. Let $\gamma_{l'} = \gamma_{1_{l'}}^{k_{l'}}$ be an LLR vector of a discarded decoding path l', which contains k LLR values corresponding to the hard estimates of the discarded path l'. After each information bit is decoded, $\gamma_{l'}$ is constructed progressively up to the k-th path splitting. Formally, $\gamma_{l'}$ is obtained using the following procedure:

- If ν is a leaf node that contains an information bit:

$$\gamma_{l'} = \text{concat}(\gamma_{l'}, \alpha_{0,i_{\min_{\nu_{l'}}}}). \tag{4.1}$$

- If ν is a REP node:

$$\gamma_{l'} = \text{concat}(\gamma_{l'}, \sum_{j=i_{\min_{\nu_{l'}}}}^{i_{\max_{\nu_{l'}}}} \alpha_{s,j}). \tag{4.2}$$

- If ν is a Rate-1 node: Updating $\gamma_{l'}$ using the following function after each path splitting at the j-th bit:

$$\gamma_{l'} = \text{concat}(\gamma_{l'}, \alpha_{s,j}), \tag{4.3}$$

[2]We exclude the FER of I-Fast-SCLF-32 with $c_e = 1$ from Fig. 4.2b as an FER of 0 cannot be plotted in the logarithmic scale.

where $j \in \{i^*_{\min_{v_{l'}}}, \ldots, i^*_{\max_{v_{l'}}}\}$ is selected by following the indices of the sorted absolute LLR values.

- If v is an SPC node: Updating $\gamma_{l'}$ using the following function after each path splitting at the j-th bit:

$$\gamma_{l'} = \text{concat}(\gamma_{l'}, \alpha_{s,j}), \tag{4.4}$$

where $j \in \{i^*_{\min_{v_{l'}}}, \ldots, i^*_{\max_{v_{l'}}}\} \setminus i^*_{\min_{v_{l'}}}$ is selected by following the order of the sorted absolute LLR values. Note that $i^*_{\min_{v_{l'}}}$ is the bit index of the parity bit whose LLR value is ignored when constructing $\gamma_{l'}$.

concat$(\gamma_{l'}, a)$ is a function that concatenates $a \in \mathbb{R}$ to the end of $\gamma_{l'}$ and initially $\gamma_{l'} = \emptyset$. In addition, $\gamma_{l'}$ is not altered if v does not satisfy any of the above conditions. For example, the LLR vector $\gamma_{l'}$ obtained after the fifth path-splitting index in Table 4.2 for $l' = 2$ is $\gamma_2 = \gamma^{5_2}_{1_2} = \{\alpha_{2,5_2}, \alpha_{2,7_2}, \alpha_{2,6_2}, \alpha_{0,12_2}, \alpha_{2,13_2}\}$. We now define the hard estimates of $\gamma_{l'}$ as $\hat{\eta}_{l'} = \hat{\eta}^{k_{l'}}_{1_{l'}}$, and the correct hard values associated with $\gamma_{l'}$ as $\eta_{l'} = \eta^{k_{l'}}_{1_{l'}}$. For instance, $\hat{\eta}_2 = \hat{\eta}^{5_2}_{1_2} = \{\beta_{2,5_2}, \beta_{2,7_2}, \beta_{2,6_2}, \beta_{0,12_2}, \beta_{2,13_2}\}$ is the discarded decoding path obtained after the fifth path-splitting index in Table 4.2 with $l' = 2$, and $\eta_2 = \eta^{5_2}_{1_2} = \{0, 0, 0, 0, 0\}$. It is worth to note that by not considering the bit-flipping operations for the parity bits of the SPC nodes, the search space of the first error path selection for FSCL decoding contains $K + C$ possible positions, which is equal to that of the SCLF decoder.

Unlike SCLF decoding, at the same path splitting index the hard estimates and LLR values of $\hat{\eta}_{l'}$ and $\gamma_{l'}$ of different path indices l' can correspond to different bit indices of the polar binary tree. However, similar to SCLF decoding, each instance of the hard estimates and LLR values of $\hat{\eta}_{l'}$ and $\gamma_{l'}$ are obtained sequentially by following the course of FSCL decoding. Therefore, in this chapter we utilize the conditional error probability model considered in [29, 30, 72] to estimate the erroneous decision occurred at the k-th path splitting index of FSCL decoding. Specifically, the probability that the discarded path l' at the k-th path splitting index under FSCL decoding is the correct path is

$$\begin{aligned} &Pr(\hat{\eta}^{k_{l'}}_{1_{l'}} = \eta^{k_{l'}}_{1_{l'}} | \alpha_n) \\ &\quad = \prod_{\substack{1 \le j \le k \\ \forall j \in \mathbb{A}_{l'}}} Pr(\hat{\eta}_{j_{l'}} = \eta_{j_{l'}} | \alpha_n, \hat{\eta}^{j_{l'}-1}_{1_{l'}} = \eta^{j_{l'}-1}_{1_{l'}}) \\ &\quad \times \prod_{\substack{1 \le j \le k \\ \forall j \in \mathbb{A}^c_{l'}}} \left[1 - Pr(\hat{\eta}_{j_{l'}} = \eta_{j_{l'}} | \alpha_n, \hat{\eta}^{j_{l'}-1}_{1_{l'}} = \eta^{j_{l'}-1}_{1_{l'}})\right], \end{aligned} \tag{4.5}$$

where $\mathbb{A}_{l'}$ is the set of bit indices j in which the hard estimates $\hat{\eta}_{j_{l'}}$ follow the sign of $\gamma_{j_{l'}}$, and $\mathbb{A}_{l'}^{c}$ is the set of bit indices j in which the hard estimates $\hat{\eta}_{j_{l'}}$ do not follow the sign of $\gamma_{j_{l'}}$.

Similar to $\boldsymbol{u}$, $\boldsymbol{\eta}_{l'}$ is also not available during the decoding process, thus we use the approximation introduced in [72] to calculate $Pr(\hat{\eta}_{j_{l'}} = \eta_{j_{l'}} | \boldsymbol{\alpha}_n, \hat{\boldsymbol{\eta}}_{1_{l'}}^{j_{l'}-1} = \boldsymbol{\eta}_{1_{l'}}^{j_{l'}-1})$ as

$$Pr(\hat{\eta}_{j_{l'}} = \eta_{j_{l'}} | \boldsymbol{\alpha}_n, \hat{\boldsymbol{\eta}}_{1_{l'}}^{j_{l'}-1} = \boldsymbol{\eta}_{1_{l'}}^{j_{l'}-1}) \approx \frac{1}{1 + \exp\left(\theta - \left|\gamma_{j_{l'}}\right|\right)}. \tag{4.6}$$

The path selection error metric obtained at the k-th path splitting based on (4.5) and (4.6) can be obtained as

$$\begin{aligned} Q_k &= -\ln\left[\sum_{\forall l'} Pr(\hat{\boldsymbol{\eta}}_{1_{l'}}^{k_{l'}} = \boldsymbol{\eta}_{1_{l'}}^{k_{l'}} | \boldsymbol{\alpha}_n)\right] \\ &\approx -\ln\left[\max_{\forall l'} Pr(\hat{\boldsymbol{\eta}}_{1_{l'}}^{k_{l'}} = \boldsymbol{\eta}_{1_{l'}}^{k_{l'}} | \boldsymbol{\alpha}_n)\right] \\ &\approx \min_{\forall l'}\left[\sum_{\substack{1\le j\le k \\ \forall j\in\mathbb{A}_{l'}^{c}}} \left(\left|\gamma_{j_{l'}}\right| - \theta\right) + \sum_{1\le j\le k} \mathrm{ReLU}\left(\theta - \left|\gamma_{j_{l'}}\right|\right)\right]. \end{aligned} \tag{4.7}$$

Consequently, the most probable erroneous position ι is obtained as

$$\iota = \underset{\log_2 L<k\le K+C}{\arg\min}\, Q_k. \tag{4.8}$$

The error metric described in (4.7) can be progressively calculated during the course of decoding, allowing for an efficient implementation of the proposed decoder. In particular, for each active decoding path l we denote by q_{k-1_l} the path-error metric at the $(k-1)$-th path splitting index of l, which is given as

$$q_{k-1_l} = \sum_{\substack{1\le j\le k-1 \\ \forall j\in\mathbb{A}_{l}^{c}}} \left(\left|\gamma_{j_l}\right| - \theta\right) + \sum_{1\le j\le k-1} \mathrm{ReLU}\left(\theta - \left|\gamma_{j_l}\right|\right) \tag{4.9}$$

if $k > 1$ and $q_{0_l} = 0\ \forall l$. Thus, the path-error metric of the path l at the k-th path splitting index can be calculated from q_{k-1_l} as

$$q_{k_l} = q_{k-1_l} + \mathrm{ReLU}\left(\theta - \left|\gamma_{k_l}\right|\right). \tag{4.10}$$

The path-error metric of the forked path with index $\tilde{l}$ originated from l, whose hard value at the

k-th path splitting index does not follow the sign of its LLR value, is calculated as

$$q_{k_{\bar{l}}} = q_{k_l} + \left|\gamma_{k_l}\right| - \theta. \tag{4.11}$$

(4.10) and (4.11) are used to compute the path-error metrics of all the $2L$ paths associated with the current L active paths and the L forked paths progressively. Next, the path metric sorting is carried out and a list of discarded paths with indices l' is determined. The flipping metric in (4.7) is obtained as

$$Q_k = q_{k_{l'_{\min}}}, \tag{4.12}$$

where $l'_{\min} = \arg\min_{l'} q_{k_{l'}}$. Therefore, under a practical implementation one only needs to maintain the path-error metrics q corresponding to the $2L$ decoding paths to progressively calculate the path selection error metric Q_k.

In this chapter, we tackle the disadvantage of Monte-Carlo simulation which optimizes the single parameter θ offline [29, 30, 32]. This is because in practice, e.g., in the 5G standard, there is a vast number of polar code configurations with different code lengths and rates, and the parameter also requires to be optimized at various SNR values. Thus, optimizing the parameter for each specific configuration is a time-consuming task as adequate training data samples need to be collected for each code configuration. Therefore, we propose an efficient online supervised learning approach to directly optimize the parameter at the operating SNR of the decoder, while obviating the need of pilot signals.

In particular, let $\mathbb{D}$ be a data batch that contains $B = |\mathbb{D}|$ instances of the path selection error metrics $\boldsymbol{Q} = \boldsymbol{Q}_1^{K+C}$, where the corresponding message word estimated by the initial FSCL decoding algorithm does not satisfy the CRC test. Under supervised learning, we need to obtain the erroneous path-splitting index ι_e to train θ. Note that in a practical scenario, the proposed decoder often requires a maximum number of m additional FSCL decoding attempts where a different estimated error index is associated with each additional decoding attempt. By assuming that a correct codeword is obtained if the CRC verification is successful, the error index ι_e can be obtained when a secondary FSCL decoding attempt passes the CRC verification. Let $\boldsymbol{o}$ be a one-hot encoded vector of size $K + C$ that indicates the error bit index ι_e as

$$o_k = \begin{cases} 1 & \text{if } k = \iota_e, \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

A data sample $d \in \mathbb{D}$ contains a pair of the input $\boldsymbol{Q}$ and its corresponding encoded output $\boldsymbol{o}$, i.e., $d \triangleq \{\boldsymbol{Q}, \boldsymbol{o}\}$.

Given a data sample d, the path selection error metric introduced in Section 4.1.1 provides an estimate of ι_e as ι by selecting the index corresponding to the smallest element of $\boldsymbol{Q}$ (see (4.8)). To enable training, the error metrics are converted to the probability domain using the following softmin conversion:

$$\hat{o}_k = \frac{\exp(-Q_k)}{\sum_{j=1}^{K+C} \exp\left(-Q_j\right)}, \tag{4.14}$$

where Q_k is manually set to ∞ for $k \in [1, \log_2 L]$ as the correct decoding path is always present in the first $\log_2 L$ path splittings. It can be seen from (4.7) and (4.14) that the bit index that has the smallest error metric is also the bit index that has the highest probability to be in error. In this chapter, we use the binary cross entropy (BCE) loss function to quantify the dissimilarity between the target output $\boldsymbol{o}$ and the estimated output $\hat{\boldsymbol{o}}$ as

$$\text{Loss} = -\sum_{k=1}^{K+C} \left[o_k \ln \hat{o}_k + (1 - o_k) \ln(1 - \hat{o}_k)\right]. \tag{4.15}$$

The parameter θ can then be trained to minimize the loss function by using the stochastic gradient descent (SGD) technique or one of its variants. An update step is given as

$$\theta = \theta - \frac{\lambda}{B} \sum_{\forall d \in \mathbb{D}} \frac{\partial \text{Loss}}{\partial \theta}, \tag{4.16}$$

where $\lambda \in \mathbb{R}^+$ is the learning rate and $\frac{1}{B} \sum_{\forall d \in \mathbb{D}} \frac{\partial \text{Loss}}{\partial \theta}$ is the estimation of the true gradient obtained from a data set that contains an infinite number of data samples. By using the chain rule and simple algebraic manipulations, given an instance $\boldsymbol{Q}$ of a data sample d, $\frac{\partial \text{Loss}}{\partial \theta}$ can be calculated as

$$\frac{\partial \text{Loss}}{\partial \theta} = \sum_{k=1}^{K+C} \frac{\hat{o}_k - o_k}{(1 - \hat{o}_k) \exp(-Q_k)} \left[\frac{\partial \phi_k}{\partial \theta} - \hat{o}_k \sum_{j=1}^{K+C} \frac{\partial \phi_j}{\partial \theta}\right], \tag{4.17}$$

where $\phi_k = \exp(-Q_k)$ and $\frac{\partial \phi_k}{\partial \theta} = -\exp(-Q_k) \frac{\partial Q_k}{\partial \theta}$.

It can be observed that the computation of $\frac{\partial \text{Loss}}{\partial \theta}$ requires the computation of $\frac{\partial Q_k}{\partial \theta}$. Similar to Q_k, $\frac{\partial Q_k}{\partial \theta}$ can also be progressively calculated during the course of decoding. In particular, from (4.10)

and (4.11) we obtain

$$\frac{\partial q_{k_l}}{\partial \theta} = \frac{\partial q_{k-1_l}}{\partial \theta} + \frac{\partial \operatorname{ReLU}(\theta - \gamma_{k_l})}{\partial \theta} = \frac{\partial q_{k-1_l}}{\partial \theta} + \mathtt{Ind}_{\theta > \gamma_{k_l}}, \tag{4.18}$$

and

$$\frac{\partial q_{k_{\tilde{l}}}}{\partial \theta} = \frac{\partial q_{k_l}}{\partial \theta} - 1, \tag{4.19}$$

respectively, and $\frac{\partial q_{0_l}}{\partial \theta} = 0 \; \forall l$. Since the values of $\frac{\partial q_{k_l}}{\partial \theta}$ and $\frac{\partial q_{k_{\tilde{l}}}}{\partial \theta}$ are available for all the current active decoding paths with indices l and the forked paths with indices $\tilde{l}$, after the path-metric sorting, $\frac{\partial Q_k}{\partial \theta}$ can be obtained as

$$\frac{\partial Q_k}{\partial \theta} = \frac{\partial q_{k_{l'_{\min}}}}{\partial \theta}. \tag{4.20}$$

Note that $\frac{\partial Q_k}{\partial \theta}$ contains integer values and $\frac{\partial Q_k}{\partial \theta} \in [-(K + C), K + C]$. To reduce the computational complexity of the training process, we use the method in [73] to implement the $\exp(\cdot)$ function as required in (4.14) and (4.17). Specifically, the Taylor series are utilized to approximate the $\exp(\cdot)$ function, which is given as [73]

$$\exp(x) \approx \max\{0, \sum_{t=0}^{T} \frac{x^t}{t!}\}, \tag{4.21}$$

where $x \in \mathbb{R}$, $T \geq 0$ is an integer number, and the approximation is exact if $T = \infty$ [74].

In Algorithm 3, we outline the proposed Fast-SCLF decoding algorithm integrated with the online training framework. The inputs of Algorithm 3 contain the channel vector $\boldsymbol{y}$, the list size L, the maximum number of additional FSCL decoding attempts m, and the size of the data batch $\mathbb{D}$, denoted as B. The parameter θ is first randomly initialized from $(0, 1)$. Given a channel output vector $\boldsymbol{y}$, the initial FSCL decoding is carried out in the `InitialFSCL`$(\cdot)$ function described in Algorithm 4, which performs the conventional FSCL decoding operations to obtain the estimated message word $\hat{\boldsymbol{u}}_{\text{init}}$. In addition, at each path splitting with index k of the initial FSCL decoding attempt, the path-error metrics $\{q_{k_l}, q_{k_{\tilde{l}}}\}$ and the derivatives $\{\frac{\partial q_{k_l}}{\partial \theta}, \frac{\partial q_{k_{\tilde{l}}}}{\partial \theta}\}$ of all the paths with indices l and $\tilde{l}$ are progressively calculated (line 3-4, Algorithm 2), followed by the computations of Q_k and $\frac{\partial Q_k}{\partial \theta}$ (line 5-6, Algorithm 2). Note that $\frac{\partial Q_k}{\partial \theta}$ is set to 0 and Q_k is set to ∞ for all the path splittings with index $k \in [1, \log_2 L]$. At the end of the `InitialFSCL`$(\cdot)$ function, the first estimate of the message word $\hat{\boldsymbol{u}}_{\text{init}}$, the path selection error metrics $\boldsymbol{Q}$, and their derivatives $\frac{\partial \boldsymbol{Q}}{\partial \theta}$ are returned to the main decoding algorithm.

Algorithm 3: Fast-SCLF Decoding Algorithm

Input : $\boldsymbol{y}, L, m, B$
Output: $\hat{\boldsymbol{u}}$

1 $\theta \sim (0, 1)$ // Initialize θ
2 $\hat{\boldsymbol{u}}_{\text{init}}, \boldsymbol{Q}, \frac{\partial \boldsymbol{Q}}{\partial \theta} \leftarrow \texttt{InitialFSCL}(\boldsymbol{y}, \theta, L)$
/* Perform FSCL decoding with the reserved path selection scheme */
3 **if** $\hat{\boldsymbol{u}}_{\text{init}}$ passes CRC **then**
4 **return** $\hat{\boldsymbol{u}}_{\text{init}}$
5 **else**
6 $\{\iota_1^*, \ldots, \iota_m^*\} \leftarrow \texttt{Sort}(\boldsymbol{Q})$
7 **for** $i \leftarrow 1$ **to** m **do**
8 $\hat{\boldsymbol{u}}_{\text{flip}} \leftarrow \texttt{FSCL}(\boldsymbol{y}, \iota_i^*, L)$
9 **if** $\hat{\boldsymbol{u}}$ passes CRC **then**
10 Construct $\boldsymbol{o}$ using (4.13) given $\iota_e = \iota_i^*$
11 $\theta \leftarrow \texttt{OptimizeTheta}(\theta, \boldsymbol{o}, \boldsymbol{Q}, \frac{\partial \boldsymbol{Q}}{\partial \theta})$
12 **return** $\hat{\boldsymbol{u}}_{\text{flip}}$
13 **return** $\hat{\boldsymbol{u}}_{\text{init}}$

In the next step, if $\hat{\boldsymbol{u}}_{\text{init}}$ satisfies the CRC test, the Fast-SCLF decoder then outputs $\hat{\boldsymbol{u}}_{\text{init}}$ and terminates. Otherwise, the path selection error metrics $\boldsymbol{Q}$ are sorted in the increasing order such that $Q_{i_1^*} \leq \ldots \leq Q_{i_{K+C}^*}$, and the path-splitting indices corresponding to the m smallest elements of $\boldsymbol{Q}$ are selected for the secondary FSCL decoding attempts, i.e., $\{\iota_1^*, \ldots, \iota_m^*\}$. The Fast-SCLF decoder then performs a maximum number of m additional FSCL decoding attempts (line 8, Algorithm 3) with each attempt performs the reversed path selection scheme at a different path-flipping index ι_i^*. If one of the secondary FSCL decoding attempts results in a successful CRC verification, the optimization process of θ implemented in the `OptimizeTheta`$(\cdot)$ function is queried, which performs the proposed optimization process based on supervised learning. The details of the function `OptimizeTheta`$(\cdot)$ are provided in Algorithm 5. To reduce the memory consumption required to store the data batch $\mathbb{D}$ for each parameter update, we use a variable Δ in Algorithm 5 to store the accumulated gradients $\sum_{\forall d \in \mathbb{D}} \frac{\partial \text{Loss}}{\partial \theta}$ as shown in (4.16). In addition, each data sample d is completely different from the others due to the presence of channel noise. Therefore, the proposed training framework can prevent overfitting without the need of a separate validation set, which also reduces the memory consumption of the parameter optimization.

Algorithm 4: Initial FSCL Decoding Algorithm

Input : $\boldsymbol{y}$,θ,L
Output: $\hat{\boldsymbol{u}}_{\text{init}}, \boldsymbol{Q}, \frac{\partial \boldsymbol{Q}}{\partial \theta}$

Function `InitialFSCL(`$\boldsymbol{y}$,θ,L`)`**:**
  for each path-splitting with index $k \in [1, K + C]$ **do**
    Compute q_{k_l} and $q_{k_{\tilde{l}}}$ based on (4.10) and (4.11) for all the paths l and $\tilde{l}$
    Compute $\frac{\partial q_{k_l}}{\partial \theta}$ and $\frac{\partial q_{k_{\tilde{l}}}}{\partial \theta}$ based on (4.18) and (4.19) for all the paths l and $\tilde{l}$
    Compute Q_k based on (4.12)
    Compute $\frac{\partial Q_k}{\partial \theta}$ based on (4.20)
  for $k \leftarrow 1$ **to** $\log_2 L$ **do**
    $Q_k \leftarrow \infty$
    $\frac{\partial Q_k}{\partial \theta} \leftarrow 0$
  Obtain $\hat{\boldsymbol{u}}_{\text{init}}$ from the first FSCL decoding attempt
  return $\hat{\boldsymbol{u}}_{\text{init}}, \boldsymbol{Q}, \frac{\partial \boldsymbol{Q}}{\partial \theta}$

Finally, if the resulting estimated message word $\hat{\boldsymbol{u}}_{\text{flip}}$ obtained from one of the addtional FSCL decoding attempts satisfies the CRC test, $\hat{\boldsymbol{u}}_{\text{flip}}$ is returned as the final decoding output. On the other hand, if none of the addtional FSCL decoding attempt can provide a message word that passes the CRC verification, the estimated message word $\hat{\boldsymbol{u}}_{\text{init}}$ of the initial FSCL decoding is returned as the final output of the decoding process.

4.1.2 Quantitative Complexity Analysis

To quantify the computational complexity of the decoders considered in this chapter, we compute a weighted complexity of the performed floating-point additions/subtractions, comparisons, multiplications, and divisions. The complexity of a floating-point addition/subtraction or a floating point comparison is considered to be one unit of complexity, a multiplication requires 3 units of complexity and a division requires 24 units of complexity [75]. In this chapter, we use the merge sort algorithm to sort a vector with N elements, which requires a worst case of $N\lceil \log_2 N \rceil - 2^{\lceil \log_2 N \rceil} + 1$ floating-point comparisons if N is not a power of 2, otherwise the number of comparisons needed is $N \log_2 N$ [76, Chapter 2]. We compute the decoding latency of the SCL-based decoders by using the method considered in [3, 26]. In particular, we count the number of time steps for various decoding operations with the following assumptions. First, the hard decisions obtained from the

Algorithm 5: Parameter optimization

Input : $\theta, \boldsymbol{o}, \boldsymbol{Q}, \frac{\partial \boldsymbol{Q}}{\partial \theta}$
Output: θ

$c \leftarrow 0$ // The number of data samples
$\Delta \leftarrow 0$ // The accumulated garadient
Function `OptimizeTheta` $(\theta, \boldsymbol{o}, \boldsymbol{Q}, \frac{\partial \boldsymbol{Q}}{\partial \theta})$**:**
  $c \leftarrow c + 1$
  Compute $\frac{\partial \text{Loss}}{\partial \theta}$ using (4.17)
  $\Delta \leftarrow \Delta + \frac{\partial \text{Loss}}{\partial \theta}$
  /* Update θ and reset the accumulated gradient */
  if $c \mod B == 0$ **then**
    $\theta \leftarrow \theta - \frac{\lambda}{B}\Delta$
    $\Delta \leftarrow 0$
  return θ

LLR values and binary operations are computed instantaneously [1, 3, 26]. Second, we consider the time steps required by a merge sort algorithm to sort a vector of size N is $\lceil \log_2 N \rceil$ [76, Chapter 2]. In addition, we also measure the average runtime in seconds required to decode a frame of all the decoders considered in this chapter. The runtime is measured based on a single-core C++ implementation of the considered decoders on a similar Linux system, with an AMD Ryzen 5 CPU and a DRAM memory of 16 GBytes.

Note that the `OptimizeTheta`$(\cdot)$ function can be executed in parallel with the decoding process presented in Algorithm 3 and the decoding latency in time steps of the `OptimizeTheta`$(\cdot)$ function is significantly smaller than the time steps required by an FSCL decoding attempt. Therefore, we do not include the number of time steps needed by the `OptimizeTheta`$(\cdot)$ function in the time steps of the proposed algorithm. However, to enable a fair comparison with other decoders that do not require parameter optimization during the course of decoding, we include the runtime of the `OptimizeTheta`$(\cdot)$ function when computing the runtime of the proposed decoder. Furthermore, the computational complexity and memory requirement of the `OptimizeTheta`$(\cdot)$ function are also considered when computing those of the proposed decoder. The memory consumption of the proposed decoder with list size L can be calculated as

$$\begin{aligned}\mathcal{M}_{\text{Fast-SCLF}} &= \mathcal{M}_{\text{FSCL}} + \underbrace{b_f}_{\theta\text{-memory}} + \underbrace{Lb_f}_{q\text{-memory}} + \underbrace{Lb_i}_{\frac{\partial q}{\partial \theta}\text{-memory}} \\ &+ \underbrace{(K+C)b_f}_{\boldsymbol{Q}\text{-memory}} + \underbrace{(K+C)b_i}_{\frac{\partial \boldsymbol{Q}}{\partial \theta}\text{-memory}} + \underbrace{(K+C)}_{\boldsymbol{o}\text{-memory}} \\ &+ \underbrace{5b_f}_{\frac{\partial \text{Loss}}{\partial \theta}\text{-related memory}} + \underbrace{b_f}_{\Delta\text{-memory}} \\ &= \mathcal{M}_{\text{FSCL}} + [K+C+L+7]\, b_f + (K+C+L)b_i \\ &+ K + C,\end{aligned} \tag{4.22}$$

where b_i is the number of memory bits used to quantize the integer values of $\frac{\partial \boldsymbol{Q}}{\partial \theta}$ and $\frac{\partial q}{\partial \theta}$. We consider that $\frac{\partial \text{Loss}}{\partial \theta}$ is progressively calculated, thus $4b_f$ memory bits are used to store the temporal values of $\sum_{j=1}^{K+C} \exp\left(-Q_j\right)$, $\exp(-Q_k)$, $\hat{o}_k$, and $\sum_{j=1}^{K+C} \frac{\partial \phi_j}{\partial \theta}$, and b_f memory bits are used to store $\frac{\partial \text{Loss}}{\partial \theta}$, whose value is progressively summed over $K + C$ indices.

4.2 Evaluation

4.2.1 Optimized Parameter and Error-Correction Performance

We measure the accuracy of the proposed training framework by calculating the probability that the most probable error index ι derived from (4.8) is the actual error index, denoted as ι_e^*, given that the initial FSCL decoding attempt does not satisfy the CRC test. Note that the error index ι_e used as the training label can be different from ι_e^*. This is because satisfying the CRC test after performing the reserved path selection scheme at the ι_e-th path-splitting index does not warranty that the estimated codeword is the sent codeword. Therefore, the training accuracy is quantified as

$$\mathrm{E}\left[Pr(\iota = \iota_e^* | \boldsymbol{y})\right] = \frac{\sum_{\text{training samples}} \mathtt{Ind}_{\iota=\iota_e^*}}{\text{Number of training samples}}. \tag{4.23}$$

In this chapter, we use the conventional SGD algorithm to optimize θ with $\lambda = 2^{-4}$ and $B = 32$, thus $\frac{\lambda}{B}$ is fixed to 2^{-9} and a multiplication with $\frac{\lambda}{B}$ can be implemented as a shift operation. We set $b_f = 32$ for both the training and decoding processes as single-precision floating-point format is

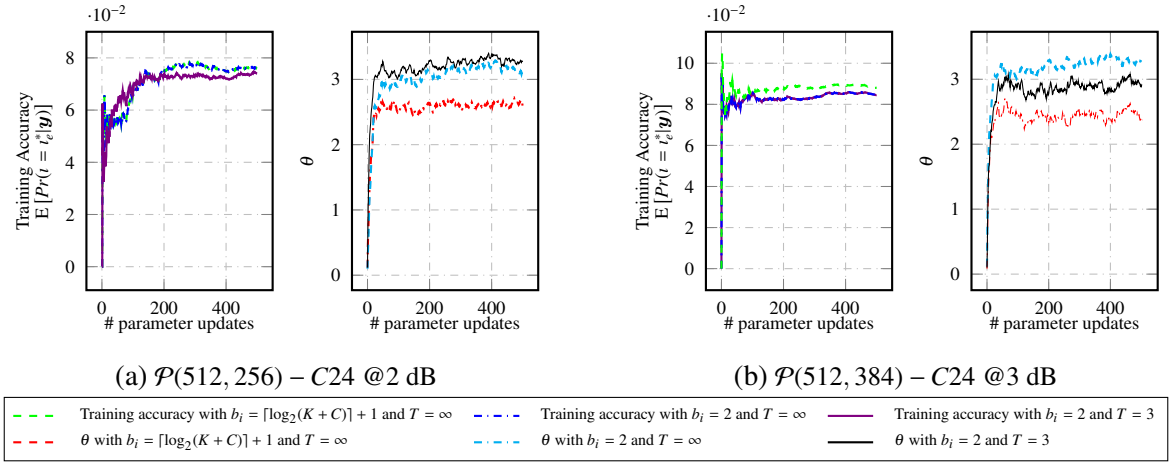


Figure 4.3: Training curves of the parameter θ for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$ with $L = 32$ and $m = 80$. A 24-bit CRC used in 5G is concatenated with the polar codes.

used to quantize a floating-point number. In addition, an integer number a is quantized using the sign-magnitude representation, which requires $\lceil \log_2(|a|) \rceil + 1$ memory bits.

Fig. 4.3 illustrates the learning curves of θ for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$ with $m = 80$, $L = 32$, $b_i \in \{\lceil \log_2(K + C) \rceil + 1, 2\}$, and $T \in \{\infty, 3\}$. With $b_i = \lceil \log_2(K + C) \rceil + 1$ the maximum and minimum values of the derivatives $\frac{\partial \boldsymbol{Q}}{\partial \theta}$ and $\frac{\partial q}{\partial \theta}$ are exactly represented under the sign-magnitude quantization scheme. On the other hand with $b_i = 2$, $\frac{\partial \boldsymbol{Q}}{\partial \theta}$ and $\frac{\partial q}{\partial \theta}$ are constrained to $\{-1, 0, 1\}$. As observed from Fig. 4.3, for $T = 3$, constraining $\frac{\partial \boldsymbol{Q}}{\partial \theta}$ and $\frac{\partial q}{\partial \theta}$ with the ternary values of $\{-1, 0, 1\}$ does not significantly degrade the estimation accuracy of the proposed error model compared to the configuration using $T = \infty$ and $b_i = \lceil \log_2(K + C) \rceil + 1$. Therefore, in the rest of this chapter, we set $b_i = 2$ and $T = 3$ for the proposed decoder as the computational complexity and memory consumption are significantly reduced by using a small value of T and b_i, which can be observed from (4.21) and (4.22), respectively. Note that the spikes in the early part of the training accuracy are caused by the small number of the training samples, which makes the calculation of the training accuracy unreliable at the initial phases of the parameter optimization. As also observed from Fig. 4.3, the value of θ becomes relatively stable as the number of parameter updates increases. Thus, in practice the function `OptimizeTheta`$(\cdot)$ can be skipped after a predefined number of parameter updates to further reduce the computational complexity and memory accesses of the proposed framework. In this chapter, we stop querying the `OptimizeTheta`$(\cdot)$ function after 50 parameter updates to further reduce the computational complexity of the proposed decoder.

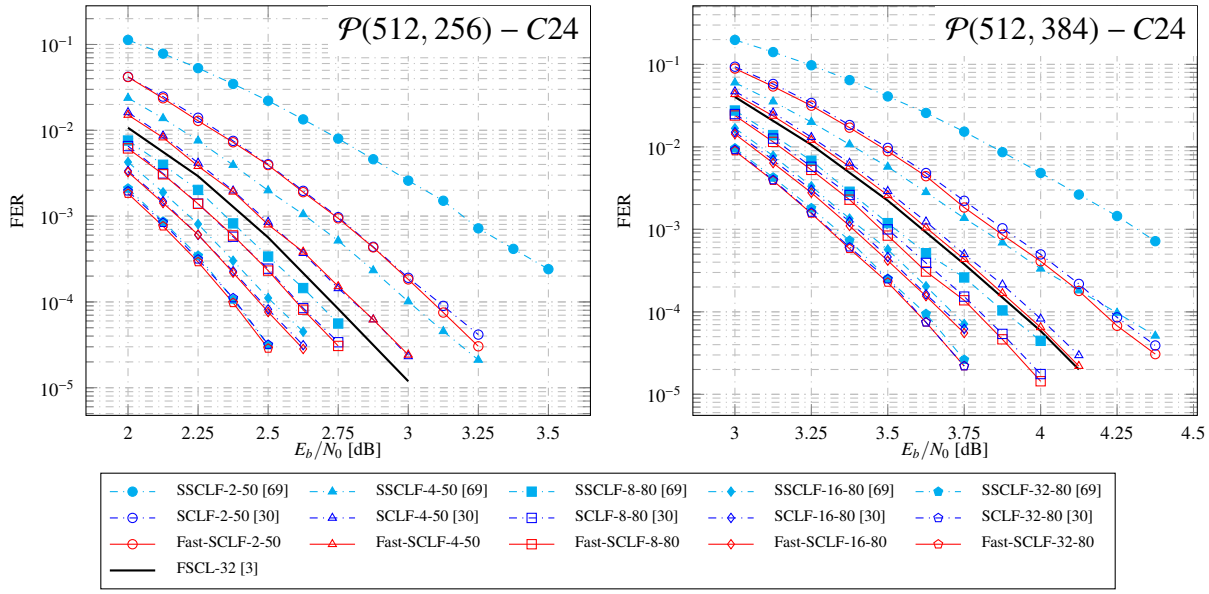


Figure 4.4: Error-correction performance of all the SCLF-based decoders considered in this chapter. The FER values of the FSCL decoder with list size $L = 32$ is also plotted for comparison.

Fig. 4.4 shows the error correction performance in terms of FER of various SCLF-based decoders where $m = 50$ for $L \in \{2, 4\}$ and $m = 80$ for $L \in \{8, 16, 32\}$. The proposed decoder is denoted as Fast-SCLF-L-m while the SCLF and SSCLF decoders are denoted as SCLF-L-m and SSCLF-L-m, respectively. In addition, the FER values of the FSCL decoder with list size 32 are also plotted for comparison. The parameter θ of the SCLF decoder is optimized offline for each value of L with the Monte-Carlo approach [30]. The E_b/N_0 values of the Monte-Carlo simulations are chosen to have an FER of approximately 10^{-4} with the selected values of L and m. From Fig. 4.4, it can be observed that under all considered polar codes and list sizes, the SCLF decoder has a relatively similar error-correction performance compared to that of the proposed Fast-SCLF decoder. In some configurations of the polar codes, the Fast-SCLF decoder obtains a slight FER gain over the SCLF decoder with the same list size. This behavior is similar to that of the ideal Fast-SCLF decoder presented in Section 4.1 when compared with the ideal SCLF decoder. It can also be seen from Fig. 4.4 that with $L \in \{2, 4\}$ the simplified path-selection scheme proposed in [69] results in a significant error-correction performance degradation in comparison with those of the Fast-SCLF and SCLF decoders at the target FER of 10^{-4}, which also degrades quickly as the SNR increases.

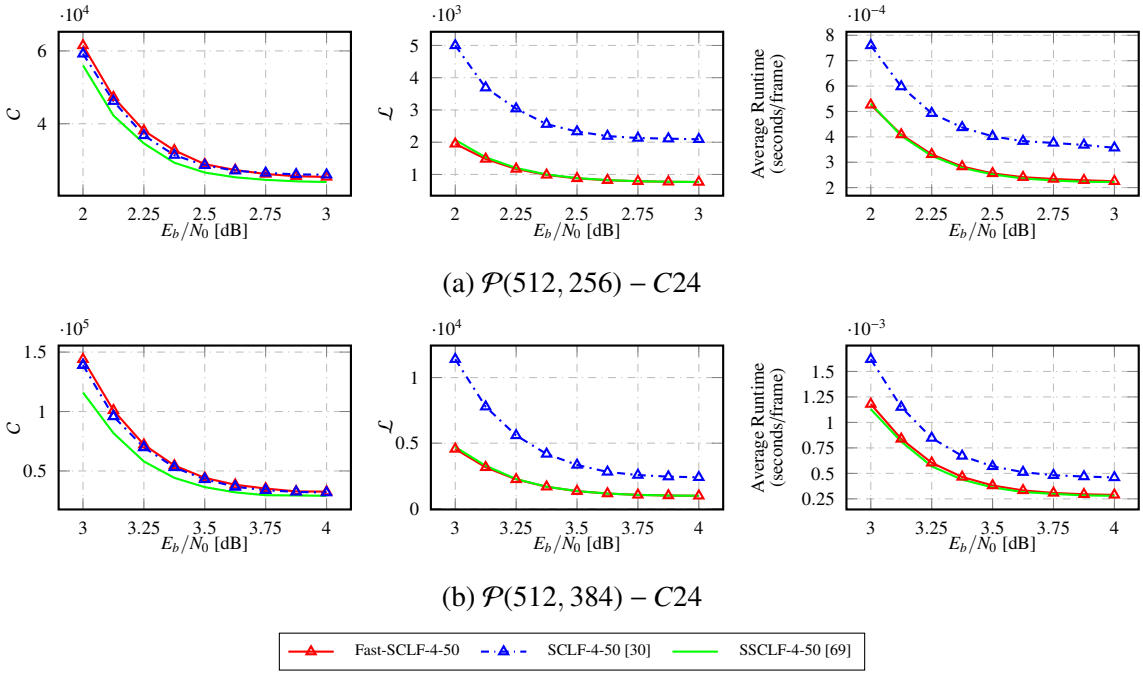


Figure 4.5: Average computational complexity and latency in terms of time steps and runtime of the SCLF-based decoders with list size 4.

4.2.2 Computational Complexity, Decoding Latency, and Memory Requirement

In Table 4.3, we summarize the average computational complexities (C) and the average decoding latency in time steps ($\mathcal{L}$) of all the SCLF-based decoders considered in Fig. 4.4. The E_b/N_0 values in Table 4.3 are selected from Fig. 4.4 where the simulated FER values of the proposed decoder are closest to the target FER of 10^{-4}.

The effectiveness of the proposed decoder is confirmed in Table 4.3 as the decoding latency of the SCLF decoding algorithm is significantly higher than that of the Fast-SCLF decoder with the same list size. However, with the list size increases the Fast-SCLF decoder imposes a more significant computational complexity overhead when compared to that of the SCLF decoder. This is due to the complexity devoted for sorting the LLR values associated with the special SPC and Rate-1 nodes, which significantly increases with the increase of the list size under FSCL decoding. From Table 4.3, it is observed that the proposed Fast-SCLF decoder reduces up to 73.4% of the average decoding latency of the SCLF decoder with the same list size at the FER of 10^{-4}, while incurring a maximum computational overhead of 27.3%. As also seen from Table 4.3 and Fig. 4.4,

Table 4.3: Summary of the average computational complexity in terms of weighted complexity of all floating-point operations performed (C) and the average decoding latency in time steps ($\mathcal{L}$) of the SCLF-based decoders considered in Fig. 4.4.

$\mathcal{P}(512, 256)$	$L = 2, m = 50$ @3.125 dB		$L = 4, m = 50$ @2.75 dB		$L = 8, m = 80$ @2.625 dB		$L = 16, m = 80$ @2.5 dB		$L = 32, m = 80$ @2.375 dB	
	C	$\mathcal{L}$	C	$\mathcal{L}$	C	$\mathcal{L}$	C	$\mathcal{L}$	C	$\mathcal{L}$
SCLF [30]	1.46E+4	1.84E+3	2.69E+4	2.17E+3	5.21E+4	2.44E+3	1.13E+5	2.72E+3	2.21E+5	3.02E+3
SSCLF [69]	1.32E+4	5.21E+2	2.48E+4	7.87E+2	4.97E+4	1.06E+3	1.06E+5	1.35E+3	2.64E+5	1.62E+3
Fast-SCLF	1.37E+4	5.01E+2	2.60E+4	7.88E+2	5.27E+4	1.06E+3	1.18E+5	1.34E+3	2.82E+5	1.63E+3
$\mathcal{P}(512, 384)$	$L = 2, m = 50$ @4.25 dB		$L = 4, m = 50$ @4.0 dB		$L = 8, m = 80$ @3.75 dB		$L = 16, m = 80$ @3.75 dB		$L = 32, m = 80$ @3.625 dB	
	C	$\mathcal{L}$	C	$\mathcal{L}$	C	$\mathcal{L}$	C	$\mathcal{L}$	C	$\mathcal{L}$
SCLF [30]	1.71E+4	1.97E+3	3.18E+4	2.40E+3	6.36E+4	2.87E+3	1.28E+5	3.23E+3	2.68E+5	3.65E+3
SSCLF [69]	1.60E+4	5.89E+2	2.94E+4	1.11E+3	5.96E+4	1.43E+3	1.15E+5	1.80E+3	2.60E+5	2.22E+3
Fast-SCLF	1.68E+4	5.73E+2	3.26E+4	9.90E+2	6.48E+4	1.43E+3	1.33E+5	1.80E+3	2.95E+5	2.25E+3

when compared with the SSCLF decoder with $L \in \{2, 4\}$, the proposed decoder with the same list size only incurs negligible overheads in the computational complexity while achieving significantly error-correction performance improvements and maintaining relatively similar decoding latency in time steps. On the other hand, with $L \in \{8, 16, 32\}$, a maximum complexity overhead of 13.5% is recorded for the proposed decoder when compared with SSCLF decoding with the same list size, while obtaining relatively similar error-correction performance and decoding latency.

Note that the path selection error metric of SCLF decoding can be progressively calculated using a similar approach as described in (4.10) and (4.11). Therefore, the memory consumption of the SCLF decoder with list size L is calculated as

$$\begin{aligned} \mathcal{M}_{\text{SCLF}} &= \mathcal{M}_{\text{SCL}} + \underbrace{b_f}_{\theta\text{-memory}} + \underbrace{Lb_f}_{q\text{-memory}} + \underbrace{(K+C)b_f}_{\boldsymbol{Q}\text{-memory}} \\ &= \mathcal{M}_{\text{SCL}} + (K + C + L + 1)b_f. \end{aligned} \tag{4.24}$$

In addition, the memory consumption of the SSCLF decoder only requires an addition of $(K+C)b_f$ memory bits to store the path-selection error metric when compared with that of the FSCL decoder with the same list size [69]. The memory consumption of the SSCLF decoder with list size L is

given as [69]

$$\mathcal{M}_{\text{SSCLF}} = \mathcal{M}_{\text{FSCL}} + \underbrace{(K + C)b_f}_{\boldsymbol{Q}\text{-memory}}. \tag{4.25}$$

In Table 4.4, we summary the memory consumption in KBits of all the SCL-based decoders considered in this chapter.

Table 4.4: Memory requirement in KBits of all the SCL-based decoders considered in this chapter.

$\mathcal{P}(512, 256)$	L				
	2	4	8	16	32
FSCL [3]	50.0	84.0	152.0	288.0	560.0
SCLF [30]	58.8	92.9	161.0	297.3	569.8
SSCLF [69]	58.7	92.7	160.7	295.7	568.7
Fast-SCLF	60.1	94.2	162.3	298.6	571.1
$\mathcal{P}(512, 384)$	L				
	2	4	8	16	32
FSCL [3]	50.0	84.0	152.0	288.0	560.0
SCLF [30]	62.8	97.0	165.0	301.3	573.8
SSCLF [69]	62.7	96.7	164.7	300.7	572.7
Fast-SCLF	64.6	98.7	166.8	303.1	575.6

Table 4.5: The average computational complexity, average decoding latency, memory consumption, and error-correction performance degradation of the Fast-SCLF, SCLF, and SSCLF decoders with $L = 4$ and $m = 50$ in comparison with those of the FSCL-32 decoder.

	$\mathcal{P}(512, 256) - C24$				$\mathcal{P}(512, 384) - C24$			
	Fast-SCLF-4 @2.75 dB	SCLF-4 @2.75 dB	SSCLF-4 @3.0 dB	FSCL-32	Fast-SCLF-4 @4.0 dB	SCLF-4 @4.0 dB	SSCLF-4 @4.25 dB	FSCL-32
C (weighted complexity)	2.60E+4	2.69E+4	2.40E+04	2.42E+5	3.26E+4	3.18E+4	2.91E+04	2.12E+5
$\mathcal{L}$ (time steps)	7.88E+2	2.17E+3	7.66E+02	1.40E+3	9.90E+2	2.40E+3	9.78E+02	1.36E+3
$\mathcal{M}$ (KBits)	94.2	92.9	92.7	560.0	98.7	97.0	96.7	560.0
Avg. Runtime (seconds/frame)	2.34E-4	3.76E-4	2.22E-4	1.92E-3	2.95E-4	4.69E-4	2.85E-4	2.05E-3
FER Degradation (dB)	0.07	0.07	0.27	-	0.02	0.05	0.31	-

We illustrate the average complexity, average decoding latency in time steps, and average runtime of the Fast-SCLF-4-50, SCLF-4-50, and SSCLF-4-50 decoders in Fig. 4.5. As seen from Fig. 4.5, the proposed decoder requires a relatively similar decoding complexity when compared with the SCLF, while the SSCLF decoder has the lowest average computational complexity among all the SCLF-based decoders. In addition, the SSCLF and Fast-SCLF decoding algorithms re-

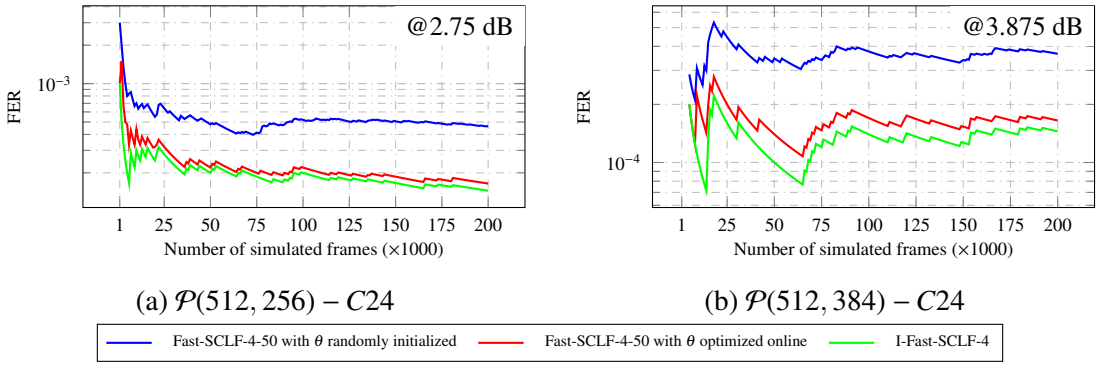


Figure 4.6: Effects of online training on the error-correction performance of the Fast-SCLF-4-50 decoder.

quire significantly smaller average decoding latency both in terms of time steps and runtime when compared with those of the SCLF decoder.

In Table 4.5, we summarize the average computational complexity, memory requirement, and average decoding latency in terms of time steps and runtime of the SCLF-based decoders with $L = 4, m = 50$, and those of the FSCL-32 decoder. The error-correction performance degradation of the SCLF-based decoders when compared to FSCL-32 is also provided in Table 4.5. The E_b/N_0 values are selected to allow an FER performance close to the target FER of 10^{-4} for all the considered decoders. In particular, for $\mathcal{P}(512, 256)$, the average complexity and average latency in time steps of Fast-SCLF-4-50 account for approximately 10.7% and 56.3% of the complexity and time steps of FSCL-32, respectively. For $\mathcal{P}(512, 384)$, Fast-SCLF-4-50 reduces 84.6% of the average complexity and 27.2% of the average time steps in comparison with FSCL-32. In addition, the proposed decoder with list size 4 requires around 17% of the memory requirement of FSCL-32, while having an FER degradation of less than 0.07 dB. When compared with the SSCLF decoder, the proposed decoder obtains the FER performance gains of 0.2 dB and 0.3 dB at the cost of 8.3% and 12.0% computational complexity overhead for $\mathcal{P}(512, 256)$ and $\mathcal{P}(512, 384)$, respectively, while the average decoding latency and memory consumption are relatively preserved at the target FER of 10^{-4}. Note that due to its high complexity, the average runtime of FSCL-32 is significantly higher than those of all the SCLF-based decoders with list size 4.

In Fig. 4.6 we study the effects of the θ parameter on the error-correction performance of the proposed decoder when online training is considered. Specifically, we illustrate the FER values obtained at the first 200000 frames of Fast-SCLF-4-50 with and without online learning. In ad-

dition, the FER values of the ideal Fast-SCLF-4 decoder are also plotted for reference. It can be observed that the proposed online learning scheme effectively optimizes the θ parameter, allowing the FER of the proposed decoder to quickly approach its ideal FER performance. On the other hand, when online training is not considered, using the proposed decoder with the initialized value of θ results in a poor error-correction performance.

4.3 Chapter Conclusion

In this chapter, we proposed a bit-flipping scheme tailored to the state-of-the-art fast successive-cancellation list (FSCL) decoding, forming the fast successive-cancellation list flip decoder (Fast-SCLF). We then derived a parameterized path selection error metric that estimates the erroneous path-splitting index at which the correct decoding path is eliminated from the initial FSCL decoding. The trainable parameter of the proposed error model is optimized using online supervised learning, which directly trains the parameter at the operating signal-to-noise ratio of the decoder without the need of pilot signals. We numerically evaluated the proposed decoding algorithm and compared its error-correction performance, average computational complexity, average decoding latency, and memory requirement with those of the state-of-the-art FSCL decoder, the successive-cancellation list flip (SCLF) decoder, and the simplified SCLF (SSCLF) decoder. The simulation results confirm the effectiveness of the proposed decoder when compared with the FSCL and the SCLF decoders for different polar codes and various list sizes. As also observed from the simulation results, the error-correction performance of the Fast-SCLF decoder significantly outperforms that of the SSCLF decoder with small list sizes (2 and 4), at the cost of negligible computational complexity overhead, while maintaining relatively similar memory consumption and decoding latency also compared to SSCLF decoding.

In the next chapter, we extend the use of polar-CRC concatenated codes to a new communication system that requires soft-input soft-output BP decoding algorithm as opposed to the soft-input hard-output SC-based decoders as considered in this chapter.

Chapter 5

Improved Belief Propagation Decoding of CRC-Polar Concatenated Codes

In this chapter, we provide efficient decoding techniques to greatly improve the error-correction performance of BP decoding for the CRC-polar concatenated codes. In particular, we first introduce a CRC-aided decoding algorithm that utilizes the CRC factor graph to aid BP decoding of the polar factor graph. Trainable weights are then assigned to the edges of the unrolled CRC-polar factor graphs to reduce the decoding latency. Finally, we introduce novel decoding techniques to further improve the error-correction performance of the CRC-aided BP decoder by utilizing the code permutations. In particular, we show that there is a one-to-one mapping between the factor-graph permutations and the Codeword permutations, which allows the use of a single decoder architecture when multiple code permutations are considered. Furthermore, we propose a method to select a set of good permutations on the fly based on reinforcement learning.

5.1 CRC-Polar BP Decoding

In this section, we present the CPBP decoding algorithm which exploits the concatenated factor graph of a polar code and a CRC. Fig. 5.1 shows the concatenated factor graph of $\mathcal{P}(8,3)$ and a CRC of length 2. We run BP decoding algorithm on the concatenated factor graph to exploit the extrinsic information of the two constituent factor graphs. A similar approach was performed in [77] for a LDPC-polar concatenated code by passing the BP messages between the factor graphs of LDPC and polar code at each iteration. A direct application of the BP decoder in [77] to the

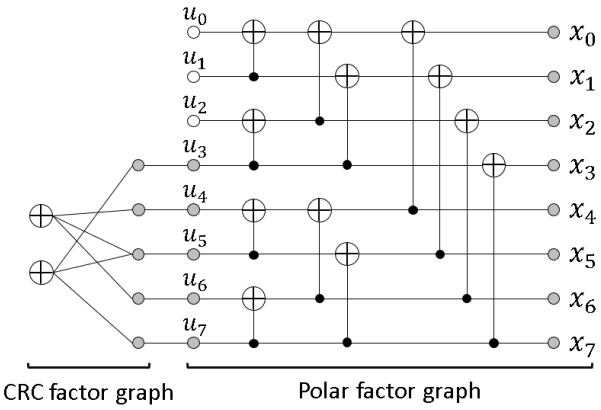


Figure 5.1: Factor graph representation of a CRC-polar concatenated code. The polar code is $\mathcal{P}(8,3)$ and a 2-bit CRC is used.

CRC-polar concatenated code is not beneficial. This is due to the fact that the LDPC code is only connected to a few pre-selected information bits of polar codes which ensures the extrinsic information received by polar codes is reliable enough, even in the initial iterations of BP decoding where the LLR values are not evolved yet. This is not the case for CRC-polar concatenated codes since the CRC is connected to all the information bits of polar code, some of which are highly unreliable during the early iterations of BP decoding.

In order to address the above issue, we first run BP decoding on the polar code factor graph for a maximum of I_{thr} iterations and if the BP decoding is not successful after I_{thr} iterations, we then continue the BP decoder on the CRC-polar concatenated factor graph. In order to determine if the decoder has succeeded, we use the CRC at each iteration as an early stopping criterion. The proposed decoding algorithm is summarized in Algorithm 6. The LLR vectors $\boldsymbol{\alpha}_s^i$ and $\boldsymbol{r}_s^i$ at all stages and iterations are initialized as explained in Section 2.4. The `BP_PolarLeft` and `BP_PolarRight` functions compute (2.31) and (2.32) at all the bit indices to perform the polar right-to-left and left-to-right LLR updates, respectively. The estimated message word $\hat{\boldsymbol{u}}$ is obtained at every iteration by making a hard decision based on $\boldsymbol{\alpha}_0^i$ and $\boldsymbol{r}_0^i$, which is done by executing (2.36) in the `HardDecision` function. A CRC is then applied on $\hat{\boldsymbol{u}}$ and the decoding can be early terminated if the CRC is satisfied. After I_{thr} iterations, if the decoding is not terminated, BP decoding on the CRC-polar factor graph is carried out in the `BP_CRC` function. It is worth mentioning that BP decoding after I_{thr} iterations runs on the concatenated CRC-polar factor graph at every iteration.

Algorithm 6: CPBP Decoding Algorithm

Input : $I_{\max}, I_{\text{thr}}, n$
Output: $\hat{\boldsymbol{u}}$

Initialize $\boldsymbol{\alpha}_s^i, \boldsymbol{r}_s^i$ $(1 \le i \le I_{max}, 0 \le s \le n)$
for $i \leftarrow 1$ **to** $I_{\max}$ **do**
  for $s \leftarrow n-1$ **to** 0 **do**
    $\boldsymbol{\alpha}_s^i \leftarrow$ `BP_PolarLeft`$(\boldsymbol{\alpha}_{s+1}^i, \boldsymbol{r}_s^{i-1})$
  if $i > I_{\text{thr}}$ **then**
    $\boldsymbol{r}_0^i \leftarrow$ `BP_CRC`$(\boldsymbol{\alpha}_0^i)$
  $\hat{\boldsymbol{u}} \leftarrow$ `HardDecision`$(\boldsymbol{r}_0^i + \boldsymbol{\alpha}_0^i)$
  if $\hat{\boldsymbol{u}}$ *satisfies CRC* **then**
    Terminate
  if $i \le I_{\max} - 1$ **then**
    for $s \leftarrow 1$ **to** $n-1$ **do**
      $\boldsymbol{r}_s^i \leftarrow$ `BP_PolarRight`$(\boldsymbol{\alpha}_s^i, \boldsymbol{r}_{s-1}^i)$
return $\hat{\boldsymbol{u}}$

Fig. 5.2 shows the FER performance of the proposed CPBP algorithm in comparison with the CRC-aided BP decoder of [36], for the $\mathcal{P}(128, 80)$ concatenated with the 16-bit CRC, which is selected for 5G. In this figure, we set $I_{\max} \in \{30, 200\}$ and we set $I_{\text{thr}} \in \{15, 30\}$ when $I_{\max} = 30$ and $I_{\text{thr}} \in \{0, 50, 100, 150, 200\}$ when $I_{\max} = 200$. We denote CPBP decoding with parameters $I_{\max}$ and I_{thr} as CPBP-($I_{\max}$,I_{thr}). Note that CPBP-($I_{\max}$,$I_{\max}$) is equivalent to the decoder in [36] and CPBP-($I_{\max}$,0) is the direct application of the approach in [77]. It can be seen that, CPBP-(30,15) provides a gain of almost 0.25 dB in comparison with the CRC-aided BP decoder of [36] at the target FER of 10^{-5}. In addition, among the selected I_{thr} for $I_{\max} = 200$, CPBP-(200,50) provides the best error-correction performance at the target FER of 10^{-5}. Furthermore, CPBP-$(200, 50)$ has an error-correction performance gain of about 0.75 dB at FER = 10^{-5} in comparison with the CRC-aided BP decoder of [36]. It is worth mentioning that increasing $I_{\max}$ does not improve the error probabilities of the conventional BP decoder in [36] at high E_b/N_0 regime. On the contrary, the FER of the proposed CPBP decoder is greatly benefited from a high value of $I_{\max}$ as observed from Fig. 5.2.

We now evaluate the latency of the proposed CPBP decoding scheme and compare it with state-of-the-art. The latency of a BP-based decoder can be measured using the number of time

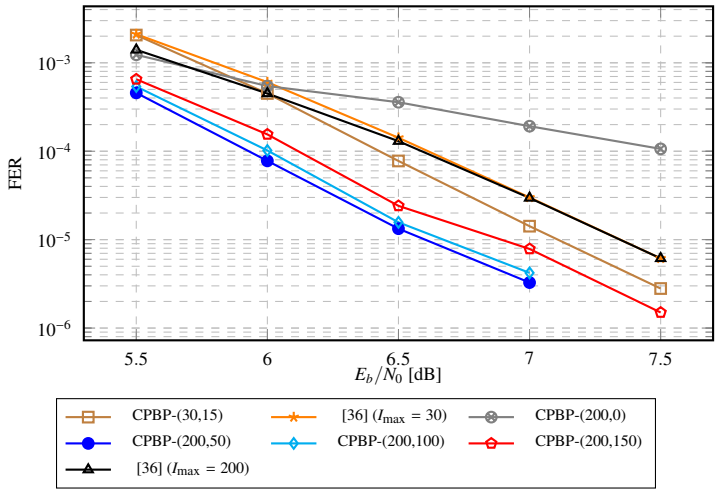


Figure 5.2: FER performance of CPBP decoding for $\mathcal{P}(128, 80)$ and a 16-bit CRC used in 5G.

steps required to finish the decoding process [52]. Let us consider the decoding process terminates at iteration I_{ET} ($1 \leq I_{\mathrm{ET}} \leq I_{\max}$). Then the decoding latency of a conventional BP decoder with early stopping criterion can be represented as

$$\mathcal{T}_{\mathrm{BP}} = (2n-1)(I_{\mathrm{ET}}-1) + n. \tag{5.1}$$

The latency of the proposed CPBP decoder depends on when the decoding process terminates and can be represented as

$$\mathcal{T}_{\mathrm{CPBP}} = \begin{cases} (2n-1)(I_{\mathrm{ET}}-1)+n, & \text{if } I_{\mathrm{ET}} \leq I_{\mathrm{thr}}, \\ (2n-1)(I_{\mathrm{ET}}-1)+n+2(I_{\mathrm{ET}}-I_{\mathrm{thr}}), & \text{otherwise.} \end{cases} \tag{5.2}$$

In fact, if $I_{\mathrm{ET}} \leq I_{\mathrm{thr}}$, (5.2) reverts to (5.1) since the CPBP decoder terminates without traversing the CRC factor graph. It should be noted that the worst case latency of the BP decoder and the proposed CPBP decoder can be calculated using (5.1) and (5.2) respectively, by setting $I_{\mathrm{ET}} = I_{\max}$.

Fig. 5.3 illustrates the average latency of the proposed CPBP decoding algorithm in comparison with a conventional CRC-aided BP decoder of [36] for the same code as in Fig. 5.2. For the proposed decoders, we set $I_{\mathrm{thr}} \in \{15, 30\}$ for $I_{\max} = 30$, and $I_{\mathrm{thr}} \in \{50, 100, 150, 200\}$ for $I_{\max} = 200$. It can be seen that the proposed CPBP algorithm incurs negligible latency overhead in comparison with [36], while providing significant performance gain. Moreover, the average latency of the

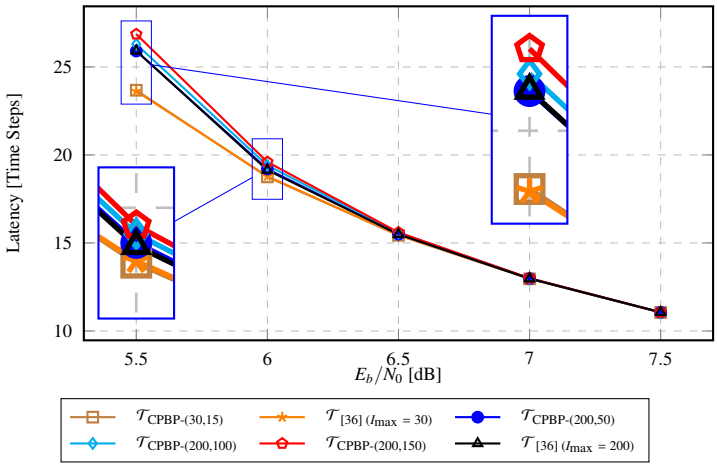


Figure 5.3: Average decoding latency of CPBP decoding for $\mathcal{P}(128, 80)$ and a 16-bit CRC used in 5G.

CPBP decoder when $I_{\max} = 30$ is always smaller than that of the CPBP decoder when $I_{\max} = 200$. This average latency saving is more significant for lower E_b/N_0 values. Furthermore, the worst case latency of CPBP(200,50) is 2887 time steps, and that of CPBP(30,15) is 407 time steps which is only 14% of the worst case latency of CPBP(200,50). For applications with stringent latency requirements, a small $I_{\max}$ is needed. However, the latency saving as a result of a small $I_{\max}$ comes at the cost of error-correction performance loss as shown in Fig. 5.2. In the next section, we propose a method to improve the error-correction performance of CPBP decoding for small values of $I_{\max}$, by using trainable weights.

5.2 Neural CRC-Polar BP Decoding

In this section, we propose the NCPBP decoder to improve the error-correction performance of CPBP decoding. The NCPBP decoder assigns trainable weights to the edges of the CRC-polar concatenated factor graph. Therefore, the NCPBP decoder resembles a neural network architecture by mapping the message updates of CPBP decoding to different computational layers in the neural network. In other words, each computational layer of the neural network is represented either as a set of PEs for BP decoding on the factor graph of polar codes, or as a set of operations required to perform BP decoding on the CRC factor graph. This network architecture greatly simplifies the training process since it can be adapted to recent deep learning frameworks, e.g. Tensorflow [78].

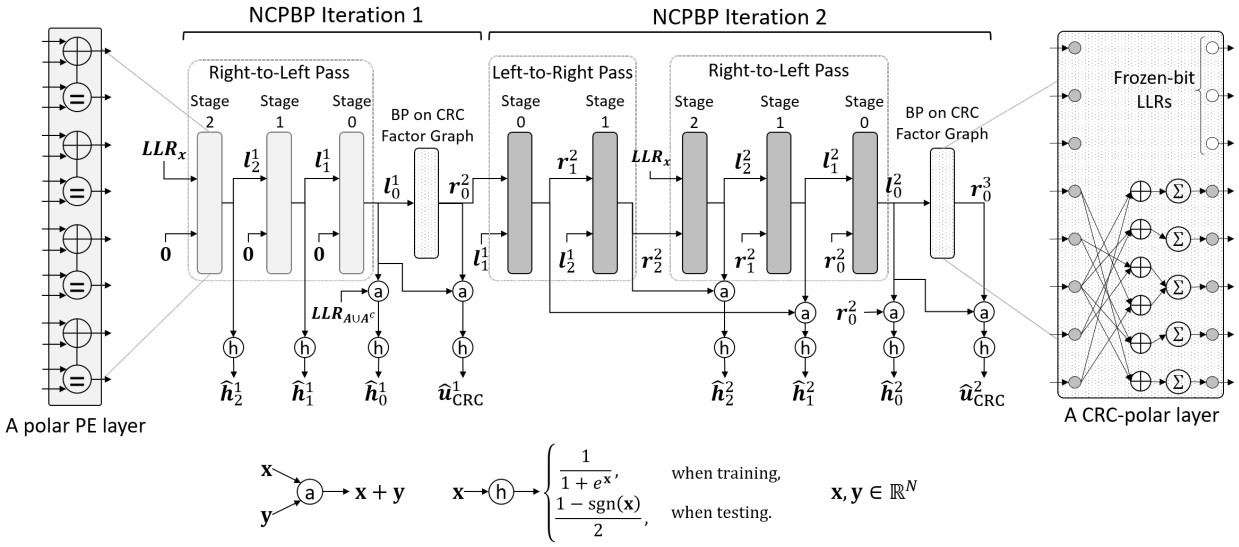


Figure 5.4: NCPBP architecture with $I_{\max} = 2$ and $I_{\text{thr}} = 0$ for $\mathcal{P}(8, 3)$ concatenated with a 2-bit CRC.

Fig. 5.4 depicts the architecture of the proposed NCPBP decoder for $\mathcal{P}(8, 3)$, with $I_{\max} = 2$ and $I_{\text{thr}} = 0$. The architecture contains the unrolled CRC-polar concatenated factor graph. Therefore, the message updates of BP decoding on polar codes at a computational layer is represented as the ones in Fig. 2.4c and Fig. 2.4d. In order to assign the weights to the $\frac{N}{2}$ parallel PEs at polar code computational layers, we represent the product of the weights w_3 and w_4 in (2.37), and the product of the weights w_9 and w_{10} in (2.38), as single trainable weights $w_{3,4}$ and $w_{9,10}$, respectively. This is due to the fact that the product of two trainable weights can be merged into one as the new weight can also be optimized during training. In addition, we merge the weights w_1 and w_2 in (2.37) into $w_{1,2}$, and the weights w_7 and w_8 in (2.38) into $w_{7,8}$, to further reduce the number of trainable weights. As a result, we define the weight assignment scheme of a PE in NCPBP decoding as

$$\begin{cases} l_{t,s}^{i} &= w_0 \tilde{f}(l_{t,k}^{i}, w_{1,2}(r_{j,s}^{i} + l_{j,k}^{i})), \\ l_{j,s}^{i} &= w_{3,4} \tilde{f}(l_{t,k}^{i}, r_{t,s}^{i}) + w_5 l_{j,k}^{i}, \end{cases} \tag{5.3}$$

$$\begin{cases} r_{t,k}^{i} &= w_6 \tilde{f}(r_{t,s}^{i}, w_{7,8}(l_{j,k}^{i-1} + r_{j,s}^{i})), \\ r_{j,k}^{i} &= w_{9,10} \tilde{f}(r_{t,s}^{i}, l_{t,k}^{i-1}) + w_{11} r_{j,s}^{i}. \end{cases} \tag{5.4}$$

For the BP decoding on the CRC factor graph of the proposed NCPBP decoder, we adopt the weight assignment scheme of the NNMS-RNN decoder in [56]. It should be noted that the polar code computational layers share the same set of weights in each iteration of the proposed NCPBP decoding, while this set of weights is different for different iterations. This is illustrated in Fig. 5.4, in which the layers depicted in the same color indicate that they use the same set of weights. On the contrary, the weights used in all the CRC layers are shared among all the decoding iterations of NCPBP. This is particularly useful in order to limit the number of required weights for NCPBP.

The NCPBP decoding algorithm starts by a right-to-left message update at iteration 1. At the i-th iteration and the s-th stage of the NCPBP decoder, $\boldsymbol{\alpha}_s^i$ and $\boldsymbol{r}_s^i$ denote the LLR vectors of the right-to-left and left-to-right message updates computed by a polar code PE layer, respectively. Furthermore, the output LLR vector of the CRC layer is denoted as $\boldsymbol{r}_0^{i+1}$. The hard estimated values of all the stages in the polar code factor graph are obtained at the right-to-left message updates, denoted as $\hat{\boldsymbol{h}}_s^i$, while the hard estimated values derived from the CRC layer is denoted as $\hat{\boldsymbol{u}}_{\text{CRC}}^i$.

The weights of all the polar code and CRC computational layers are trained using a multiloss function defined as

$$\text{Loss} = \sum_{i=1}^{I_{\max}} \sum_{s=0}^{n-1} H_{\text{CE}}(\hat{\boldsymbol{h}}_s^i, \boldsymbol{h}_s) + \sum_{i'=I_{\text{thr}}+1}^{I_{\max}} H_{\text{CE}}(\hat{\boldsymbol{u}}_{\text{CRC}}^{i'}, \boldsymbol{u}), \tag{5.5}$$

where H_{CE} is the cross-entropy function, and $\boldsymbol{h}_s$ is the correct hard value vector at stage s of the polar code factor graph which is obtained from the training samples. Note that in the testing phase, only the hard estimated values at stage 0 of the polar code factor graph, i.e. $\hat{\boldsymbol{h}}_0^i$ $(1 \leq i \leq I_{\max})$, and the hard estimated values at the CRC layer, i.e. $\hat{\boldsymbol{u}}_{\text{CRC}}^i$, $(I_{\text{thr}} < i \leq I_{\max})$, are required to obtain the decoded message bits.

We evaluate the proposed NCPBP decoder for $\mathcal{P}(128, 80)$ concatenated with a 16-bit CRC which is also used in Section 5.1, and we compare the error-correction performance and latency of NCPBP with those of [36, 53, 56]. All the neural BP-based decoders in this section are trained using stochastic gradient descent with RMSPROP optimizer [60] and the learning rate is set to 0.001. We use Tensorflow [78] as our deep learning framework. Since all the considered neural BP-based decoders satisfy the symmetry conditions [79], we collect $100,000$ zero codewords at each E_b/N_0 value for training, where $E_b/N_0 \in \{4, 4.5, 5, 5.5\}$ dB. All the weights of all the neural BP-based decoders are initialized to one and all the LLR values are clipped to be in the interval of $[-20, 20]$. The mini-batch size is set to 64 and each neural decoder is trained for 40 epochs.

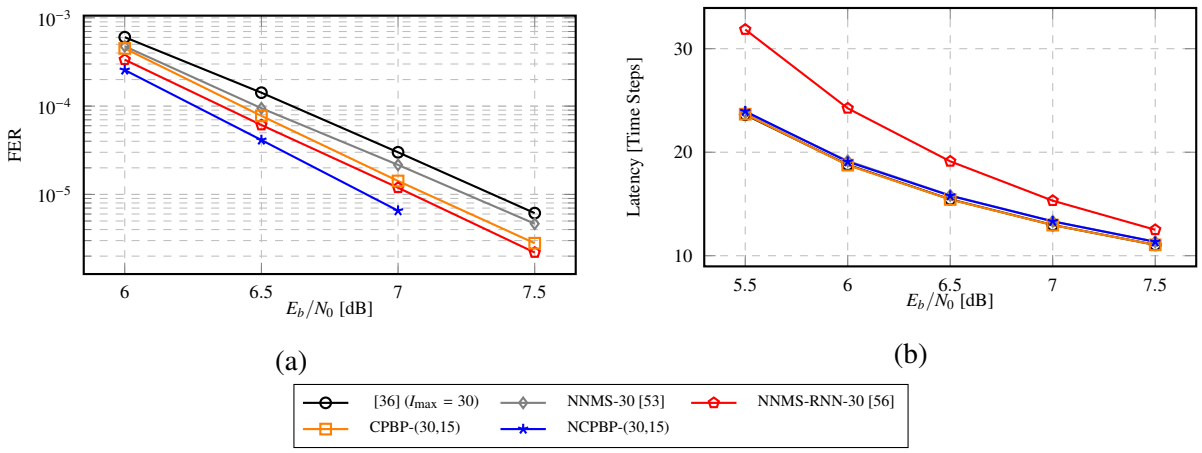


Figure 5.5: (a) FER performance and (b) average decoding latency of various BP decoding algorithms for $\mathcal{P}(128, 80)$ and a 16-bit CRC used in 5G.

To evaluate the error-correction performance, randomly generated codewords are used during the testing phase and each decoder simulates at least $10,000$ codewords until it obtains at least 50 frames in error.

Fig. 5.5a compares the error-correction performance of the proposed NCPBP decoder with state-of-the-art BP-based decoders in [36, 53, 56]. We use the NNMS-$I_{\max}$ decoder of [53] and the NNMS-RNN-$I_{\max}$ decoder of [56] for our comparisons. In all the decoders, we set $I_{\max} = 30$. At a target FER of 10^{-5}, the proposed NCPBP decoder provides about 0.5 dB gain with respect to [36], 0.4 dB gain with respect to [53], and 0.2 dB gain with respect to [56]. Compared to the CPBP decoder of Section 5.1, the proposed NCPBP provides 0.25 dB FER performance improvement.

Table 5.1: Number of weights required by different neural BP decoders.

Decoder	Number of weights
NNMS-30 [53]	3840
NNMS-RNN-30 [56]	11520
NCPBP-(30,15)	8288

Fig. 5.5b illustrates the average latency requirements of the NCPBP decoder compared to the state-of-the-art decoders in [36,53,56]. It can be seen that while the average latency of the NCPBP decoder is similar to that of the decoders in [36, 53], it is always better than that of [56]. In addition, NCPBP incurs almost no latency overhead with respect to the proposed CPBP decoder

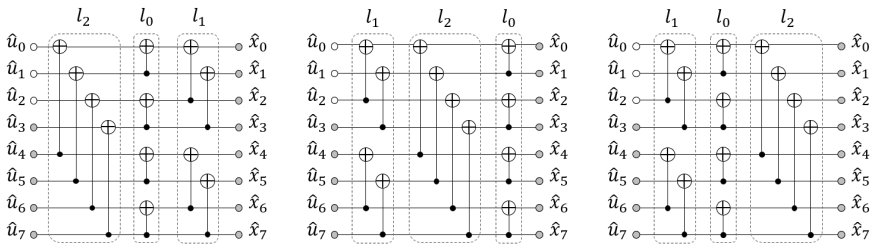


Figure 5.6: Permuted factor graph representations for $\mathcal{P}(8, 5)$.

while having a notably smaller error probability.

Table 5.1 shows the number of weights required for the proposed NCPBP decoder in comparison with the neural BP decoders of [53, 56]. The proposed NCPBP decoder requires 28% fewer weights with respect to the decoder in [56]. The decoder in [53] requires 46% of the weights that is required by the proposed NCPBP decoder. However, the smaller number of weights in [53] results in significant error-correction performance loss as shown in Fig. 5.5a.

5.3 Improved CRC-Polar BP Decoding with Codeword Permutations

In this section, we first introduce a technique to transform a factor-graph permutation to a codeword permutation of polar codes. We then briefly summarize the multi-armed bandit problem and formalize the selection of factor-graph permutations for polar decoding as a k-armed bandit problem. Next, we introduce the proposed decoding method that utilizes the multi-armed bandit algorithms to select the factor-graph permutations under CPBP decoding.

5.3.1 From Factor-Graph Permutations to Codeword Permutations

Factor graph permutations are a way to provide multiple representations of a single code. It was observed in [80, 81] that there exists $n!$ different ways to represent a polar code by permuting the layers in its factor graph. Fig. 5.6 illustrates such permutations for $\mathcal{P}(8, 5)$, where 3 out of 3! = 6 permutations are shown. Note that the two leftmost factor graphs in Fig. 5.6 are formed by applying cyclic shifts to the original factor graph depicted in Fig. 2.4a. In [55], parallel BP decoders are applied on a set of randomly selected factor graphs of a polar code concatenated with a CRC. Although this decoding scheme shows improvement in error probability when compared to a non CRC-aided SCL decoder, permuting layers results in different BP scheduling which consequently

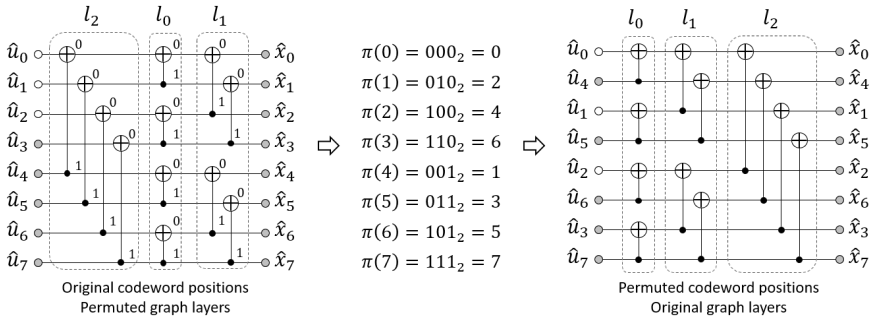


Figure 5.7: The proposed mapping from factor graph permutation to codeword permutation for $\mathcal{P}(8, 5)$.

requires the design of a different BP decoder for each permutation. Moreover, a large number of random permutations are required to achieve a reasonable error-correction performance, which makes this decoding scheme too complex for practical applications.

We denote by $\{l_{n-1}, \ldots, l_0\}$ the layers of the original factor graph of polar codes (the one represented in the right part of Fig. 5.7) and by $\{b_{n-1}^{(i)}, \ldots, b_0^{(i)}\}$ the binary expansion of the integer i.

Theorem 2. *Let $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ be a permutation. Then, the synthetic channel associated to the position with binary expansion $\{b_{n-1}^{(i)}, \ldots, b_0^{(i)}\}$ on the factor graph with layers $\{l_{n-1}, \ldots, l_0\}$ is the same as the synthetic channel associated to the position with binary expansion $\{b_{\pi(n-1)}^{(i)}, \ldots, b_{\pi(0)}^{(i)}\}$ on the factor graph with layers $\{l_{\pi(n-1)}, \ldots, l_{\pi(0)}\}$.*

Proof. Consider the original factor graph of polar codes. Then, the synthetic channel associated to the position with binary expansion $\{b_{n-1}^{(i)}, \ldots, b_0^{(i)}\}$ is given by

$$(((W^{(b_{n-1}^{(i)})})^{(b_{n-2}^{(i)})})^{\cdots})^{(b_0^{(i)})},$$

where W is the transmission channel and the transformations $W \to W^{(0)}$ and $W \to W^{(1)}$ are the "minus" and "plus" polar transforms formally defined in (2.3) and (2.4) of [80]. Note that the i-th component of the message word u_i is connected to the i-th component of the codeword x_i by associating a "XOR" to a 0 and a "dot" to a 1 in the binary expansion of the integer i. Note also that, by permuting the layers of the factor graph, we simply permute the order of those "XOR"s and "dot"s operations. Hence, the effect of applying a permutation π to the binary expansion of i is the same as the effect of applying the same permutation on the layers of the factor graph. □

Fig. 5.7 shows an example of the proposed mapping applied to a permuted factor graph of $\mathcal{P}(8,5)$. It can be seen that, by using the permuted bit indices, the structure of the factor graph is unchanged. Therefore, the original decoder can be used to perform decoding on all the required permutations. In addition, the proposed mapping allows to use other decoding algorithms, such as SC and SCL, on the permuted factor graphs without changing the decoder structure. This is particularly useful for hardware implementation.

5.3.2 Multi-Armed Bandit Problem

A multi-armed bandit problem, or a k-armed bandit problem ($k > 1$), is an RL problem where an agent has to repeatedly make a choice among k different actions (options). After each action is performed, the agent receives a numerical reward that is drawn from a distribution that depends on the selected action. The agent's objective is to maximize the expected cumulative rewards over a time period [67]. Let $\mathbb{A} = \{a_1, a_2, \ldots, a_k\}$ be the set of actions and $q^*(a_j)$ ($1 \leq j \leq k$) be the corresponding expected reward of an action a_j. $q^*(a_j)$ is called the value function and its value is unknown to the agent. In this chapter, we consider three state-of-the-art algorithms designed for the multi-armed bandit problem, namely, ε-greedy, upper confidence bound (UCB), and Thompson sampling (TS).

ε-Greedy and UCB Algorithms

Let n_{a_j} be the number of times that an action a_j is selected up to the t-th time step. If a_j is selected at the t-th time step, n_{a_j} is updated as $n_{a_j} = n_{a_j} + 1$ [67]. Then, the value function $q^*(a_j)$ is estimated as Q_{a_j} in accordance with $Q_{a_j} = Q_{a_j} + \frac{1}{n_{a_j}}\left[R_t - Q_{a_j}\right]$, where R_t is the reward received by selecting action a_j at the t-th time step [67]. Initially, Q_{a_j} and a_j are set to 0 ($\forall j, 1 \leq j \leq k$). Given the estimated expected rewards Q_{a_j}, an exploitation occurs when the agent selects an action that has the largest expected reward value [67]. On the other hand, an exploration occurs when the agent selects any action that does not have the largest expected reward value [67].

Let a_{j^*} be the action selected by the agent at the t-th time step. Under the ε-greedy algorithm a_{j^*} is selected as [67]

$$a_{j^*} = \begin{cases} \underset{\forall a_j}{\arg\max}\, Q_{a_j} & \text{with probability } 1-\varepsilon, \\ a_{\text{random}} & \text{with probability } \varepsilon, \end{cases} \tag{5.6}$$

where a_{random} is a random action drawn i.i.d. from $\mathbb{A}$. On the other hand, under the UCB algorithm a_{j^*} is selected as

$$a_{j^*} = \arg\max_{\forall a_j} \left[Q_{a_j} + c\sqrt{\frac{\ln t}{n_{a_j}}} \right], \tag{5.7}$$

where $n_{a_j} \neq 0$ and $c \in \mathbb{R}^+$. If $n_{a_j} = 0$, a_j is considered as an exploitation action. Note that ε and c control the degree of exploration of the ε-greedy and UCB algorithms, respectively.

Thompson Sampling

Instead of estimating the expected reward value $q^*(a_j)$ as in the ε-greedy and UCB algorithms, the TS algorithm directly estimates the distribution of the reward value associated with each action. In this chapter, as $R_t \in \{0, 1\}$ a Beta distribution is used to estimate the reward's distribution [82]. A Beta distribution has two shape parameters: $\alpha, \beta \in \mathbb{R}^+$, and a different set of shape parameters is used for each action. We denote a random sampling from the estimated reward distribution of the j-th action as $\upsilon_{a_j} = \text{Beta}(\alpha_{a_j}, \beta_{a_j})$. At the t-th time step, the TS algorithm first draws a random sample from each of the estimated reward distributions. The agent then selects the action a_{j^*} as $a_{j^*} = \arg\max_{\forall a_j} \upsilon_{a_j}$. The shape parameters corresponding to the selected action a_{j^*} are then updated as $\alpha_{a_{j^*}} = \alpha_{a_{j^*}} + R_t$ and $\beta_{a_{j^*}} = \beta_{a_{j^*}} + R_t$ [82]. Initially, $\alpha_{a_j} = \beta_{a_j} = 1$ $(\forall j, 1 \leq j \leq k)$ [82].

5.3.3 Problem Formulation

Under BP decoding of polar codes, the original factor-graph permutation π_0 is empirically observed to have the best error-correction performance compared to other factor-graph permutations [81]. However, there are cases that a specific channel output realization, which cannot be decoded using the original factor-graph permutation, can be decoded using another factor-graph permutation [81]. As the number of permutations, $n!$, is large, running BP decoding on all of the permutations is not possible in real applications. Instead, the decoding is performed on a small set of M factor-graph permutations, including the original factor-graph permutation [54, 55, 81, 83].

Let an action $a_j \in \mathbb{A}$ $(1 \leq j \leq k)$ be a random selection of $M - 1$ $(M > 1)$ factor-graph permutations that do not include the original factor-graph permutation. Consider the CRC verification is not successful when CPBP decoding is performed on the original factor-graph permutation π_0. The proposed decoder then selects an action a_j from the set $\mathbb{A}$. If one of the factor-graph permutations in a_j results in a successful CRC verification, a reward of 1 is given to the decoder. Otherwise, if

Algorithm 7: Forming the action set

Input : n, k, M
Output: $\mathbb{A}$

```
/* Define the original permutation                                  */
π_0 ← {s_0, s_1, ··· , s_{n−1}}
/* Select M − 1 random permutations for each action                 */
𝔸 ← ∅
for j ← 1 to k do
    a_j ← ∅
    for t ← 1 to M − 1 do
        π_{j,t} ← RandShuffle(π_0)
        a_j ← a_j ∪ π_{j,t}
    𝔸 ← 𝔸 ∪ a_j
return 𝔸
```

none of the permutations in a_j results in a successful CRC verification under CPBP decoding, a reward of 0 is given to the decoder. Therefore, among k sets of predefined factor-graph permutations, i.e., k different actions, the proposed decoding algorithm decides which set of factor-graph permutations maximizes the reward during the course of decoding. The selection of factor-graph permutations for CPBP decoding can thus be formalized as a k-armed bandit problem as defined in Section 5.3.2.

5.3.4 Reinforcement Learning-Aided CPBP Decoding

The proposed decoding algorithm starts with the construction of $\mathbb{A}$, the set of k different actions, which is outlined in Algorithm 7. Each action $a_j \in \mathbb{A}$ contains $M-1$ random factor-graph permutations. Formally, $a_j = \{\pi_{j,1}, \pi_{j,2}, \cdots, \pi_{j,M-1}\}$, $\pi_{j,t} \neq \pi_0 \; \forall j, t$, where $1 \leq j \leq k$, and $1 \leq t \leq M-1$. A random factor-graph permutation is formed by randomly permuting the PE stages of the original factor graph π_0, which is obtained by the RandShuffle function in Algorithm 7. The number of all possible actions is

$$k_{\max} = \binom{n! - 1}{M - 1} = \frac{(n! - 1)!}{(M-1)!(n! - M)!}, \tag{5.8}$$

which is generally intractable for practical values of n and M. Therefore, only the subset $\mathbb{A}$ of all the possible actions is considered. In fact, $\mathbb{A}$ is constructed by randomly sampling from the

complete set of actions as shown in Algorithm 7. Note that after $\mathbb{A}$ is formed, the set of actions in $\mathbb{A}$ remains unchanged during the course of decoding.

Algorithm 8 outlines the proposed RL-CPBP decoding algorithm, given the predefined set of actions $\mathbb{A}$ constructed in Algorithm 7. The proposed RL-CPBP decoder first initializes the parameters of the multi-armed bandit algorithm depending on its type, which is defined by the parameter Algo in Algorithm 8. If Algo indicates the ε-greedy or UCB algorithms, the parameters of the multi-armed bandit algorithm are initialized as $Q_{a_j} = n_{a_j} = 0 \; \forall j, \; 1 \leq j \leq k$. If the TS algorithm is used, the set of parameters is initialized as $\alpha_{a_j} = \beta_{a_j} = 1 \; \forall j, \; 1 \leq j \leq k$. Note that the initialization process is only carried out once in the course of decoding.

Then, the proposed RL-CPBP decoding applies CPBP decoding over the original factor-graph permutation π_0. If the CRC verification, which is obtained by the VerifyCRC function in Algorithm 8 is successful, the proposed decoder outputs the estimated message word $\hat{\boldsymbol{u}}$ and the decoding process is terminated. Otherwise, the RL-CPBP decoder selects an action a_{j^*} from $\mathbb{A}$, which contains a set of $M-1$ random factor-graph permutations as described in Algorithm 7. Depending on the type of the algorithm, the function SelectAction implements the selection criteria of the considered multi-armed bandit algorithms as introduced in Section 5.3.2. Note that the SelectAction function can be performed in parallel with the first CPBP decoding attempt as there is no dependency between them. Therefore, the selected action a_{j^*} can be obtained in advance without adding a latency overhead to the proposed decoding algorithm. Moreover, if the first CPBP decoding attempt over π_0 is successful, the selected action a_{j^*} is discarded.

If the first CPBP decoding attempt fails in the proposed RL-CPBP decoding algorithm, additional CPBP decoding attempts are sequentially carried over the factor-graph permutations specified by a_{j^*}. As soon as the CRC verification is successful after CPBP decoding on one of the factor-graph permutations in a_{j^*}, a reward of 1 is given to a_{j^*}, and the decoding outputs the estimated message word that satisfies the CRC verification. On the other hand, if running CPBP on all of the permutations in a_{j^*} does not result in a successful CRC test, a reward of 0 is given to a_{j^*} and the decoding is declared unsuccessful. Finally, after each action selection, the parameters associated with the selected action a_{j^*} are updated using the UpdateBandit function. Note that the parameter update process is based on the received reward and the type of the mutli-armed bandit algorithm as provided in Section 5.3.2.

Algorithm 8: RL-CPBP Decoding

Input : $\boldsymbol{\alpha}, \mathbb{A}, k, M$, Algo
Output: $\hat{\boldsymbol{u}}$

/* Initialize the bandit parameters */
InitBandit(k, Algo)
/* Apply CPBP decoding on π_0 */
$\hat{\boldsymbol{u}} \leftarrow \text{CPBP}(\boldsymbol{\alpha}, \pi_0)$
$isCorrect_{\pi_0} \leftarrow \text{VerifyCRC}(\hat{\boldsymbol{u}})$
/* Select an action in advance */
$a_{j^*} \leftarrow$ SelectAction($\mathbb{A}$, Algo)
/* If applicable, apply CPBP decoding on the permutations specified by a_{j^*} */
if $(isCorrect_{\pi_0} = 0)$ **then**
    $isCorrect_{a_{j^*}} \leftarrow 0$
    for $t \leftarrow 1$ **to** $M - 1$ **do**
        $\hat{\boldsymbol{u}} \leftarrow \text{CPBP}(\boldsymbol{\alpha}, \pi_{j^*,t})$
        $isCorrect_{a_{j^*}} \leftarrow$ VerifyCRC($\hat{\boldsymbol{u}}$)
        if $(isCorrect_{a_{j^*}} = 1)$ **then**
            break
    /* Update the bandit parameters associated with a_{j^*} */
    $R_t \leftarrow isCorrect_{a_{j^*}}$
    UpdateBandit(R_t, a_{j^*}, Algo)
return $\hat{\boldsymbol{u}}$

5.3.5 Simulation Results

In this section, the performance of various multi-armed bandit algorithms used by the proposed RL-CPBP decoding is numerically evaluated. In addition, the error-correction performance of the proposed RL-CPBP decoding in terms of FER is compared with that of other polar decoding techniques. A complexity comparison of different multi-armed bandit algorithms in the proposed RL-CPBP decoding is also given. We use $\mathcal{P}(128, 64)$ selected for the eMBB control channel of the 5G standard [16]. Furthermore, the polar code is concatenated with a CRC of length 16, which is also used in 5G [16]. The total number of factor-graph permutations used by all BP-based decoders is set to 7. We set $I_{\max} = 100$ and $I_{\min} = 50$ for all BP-based decoding algorithms.

Fig. 5.8 illustrates the dependence of the average reward on the parameters in ε-greedy and UCB algorithms for $\mathcal{P}(128, 64)$. The simulation is carried out at $E_b/N_0 = 3.0$ dB and we set

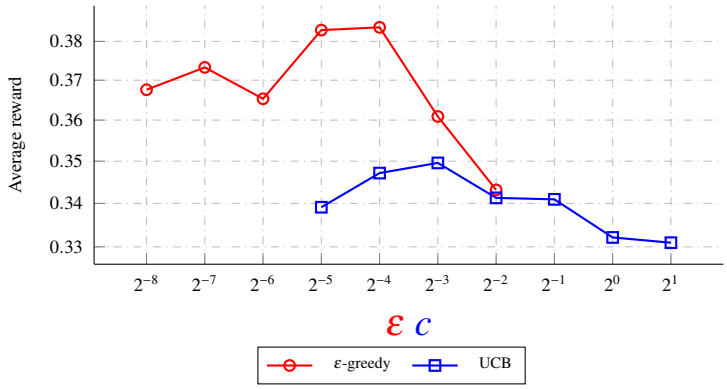


Figure 5.8: A parameter study of the ε-greedy and UCB algorithms. The average reward is obtained for the first 10000 time steps with $k = 500$ at $E_b/N_0 = 3.0$ dB.

$k = 500$ for all multi-armed bandit algorithms. In this figure, the average reward of the first 10000 time steps received by the RL-CPBP decoder is plotted against the parameter value. Note that a time step is increased by 1 when the multi-armed bandit algorithm is required for the action selection, i.e., when CPBP decoding has failed on the original factor-graph permutation π_0. As seen from Fig. 5.8, at $\varepsilon = 2^{-4}$ and $c = 2^{-3}$, RL-CPBP decoding has the highest average reward value for ε-greedy and UCB algorithms, respectively. The TS algorithm does not require parameter tuning since α and β parameters associated with each action are optimized during the decoding process.

Fig. 5.9 illustrates the performance of multi-armed bandit algorithms used by RL-CPBP decoding with different values of k. This simulation is also carried out at $E_b/N_0 = 3.0$ dB. We set $\varepsilon = 2^{-4}$ for the ε-greedy algorithm and $c = 2^{-3}$ for the UCB algorithm as those configurations provide the best performance in Fig. 5.8. It can be observed that for all the bandit algorithms, $k = 500$ provides the largest cumulative reward after the first 10000 time steps. Thus, we set $k = 500$ for the rest of the chapter.

Fig. 5.10 illustrates the average cumulative reward over the first 10000 time steps for all the multi-armed bandit algorithms. The simulation is performed at $E_b/N_0 = 3.0$ dB with $k = 500$, $\varepsilon = 2^{-4}$, and $c = 2^{-3}$. It can be seen that the ε-greedy algorithm has the best performance in terms of the average cumulative reward. In addition, the UCB algorithm performs slightly better than the TS algorithm. Note that the spikes in the early part of the curves are caused by the small value of the time step, which makes the calculation of the average cumulative reward unreliable at the initial phases of the algorithm.

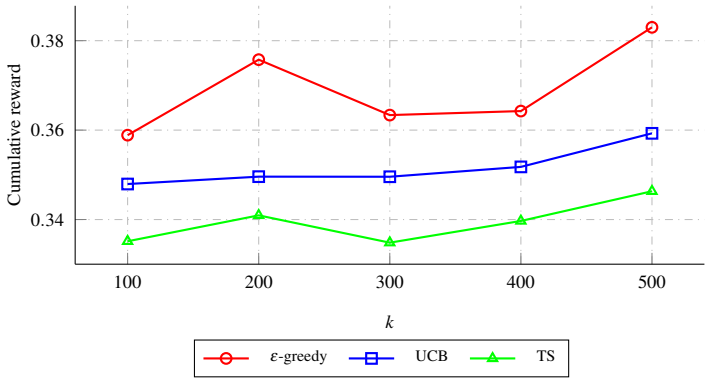


Figure 5.9: The impact of k on the performance of different multi-armed bandit algorithms used by RL-CPBP decoding for $\mathcal{P}(128, 64)$, obtained for the first 10000 time steps.

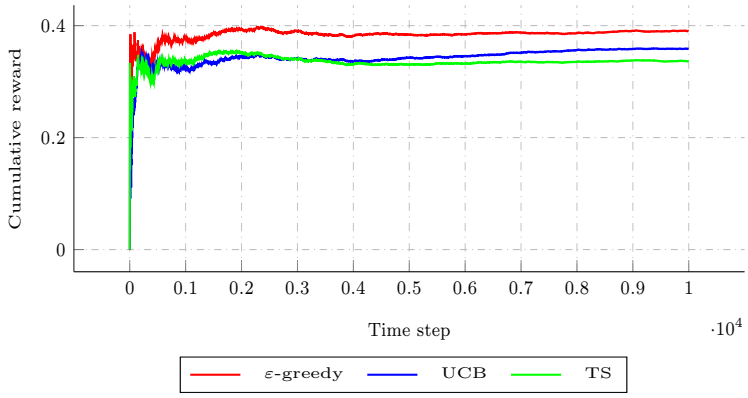


Figure 5.10: Performance comparison of various multi-armed bandit algorithms used by RL-CPBP decoding. The simulation is obtained at $E_b/N_0 = 3.0$ dB with $k = 500$, $\varepsilon = 2^{-4}$, and $c = 2^{-3}$.

Fig. 5.11 compares the FER of different factor-graph permutation selection schemes under the CPBP decoding algorithm. In this figure, CPBP denotes the CPBP decoding algorithm performed only on the original factor-graph permutation. CP-CPBP and RP-CPBP denote the cyclically-shifted and random factor-graph permutations selection schemes proposed in [81] and [55], respectively. Note that as there are $n = 7$ cyclically-shifted permutations for $\mathcal{P}(128, 64)$, we set the number of additional random permutations used by RP-CPBP to 6, and $M = 7$ for the proposed RL-CPBP decoder for a fair comparison. It can be seen that the proposed RL-CPBP decoder under various multi-armed bandit algorithms has a similar FER performance. When compared with CP-CPBP and RP-CPBP, an error-correction performance gain of at least 0.125 dB is obtained at

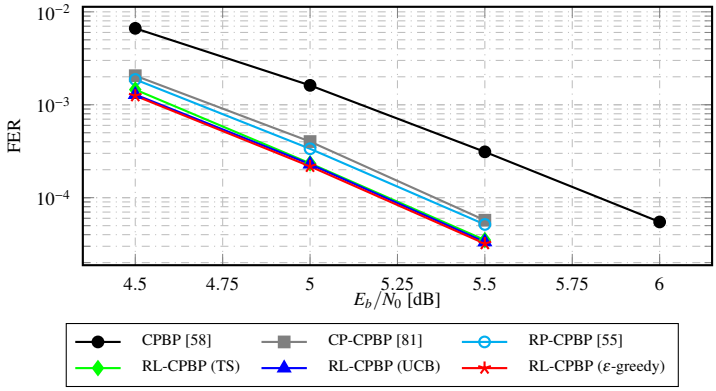


Figure 5.11: Error-correction performance of different factor-graph permutation selection schemes for $\mathcal{P}(128, 64)$.

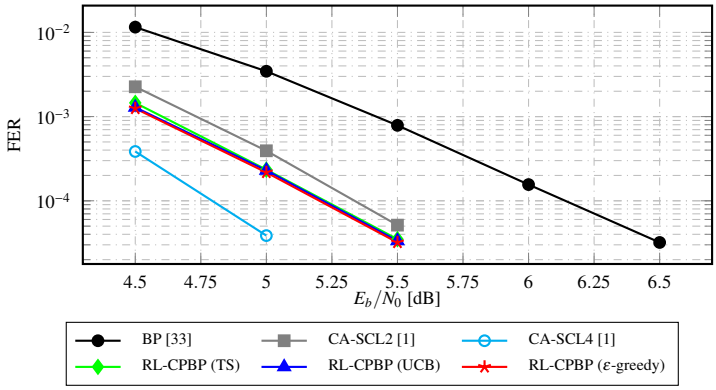


Figure 5.12: Error-correction performance of RL-CPBP decoding and other decoding algorithms of polar codes.

the target FER of 10^{-4}. In addition, an FER gain of around 0.62 dB is obtained when the proposed RL-CPBP decoding algorithm is compared with the baseline CPBP decoder at the FER of 10^{-4}.

Fig. 5.12 compares the error-correction performance of the proposed RL-CPBP decoding with BP decoding and CA-SCL decoding of polar codes. In Fig. 5.12, CA-SCLL indicates the CA-SCL decoder with a list size of L. It can be observed that at the target FER of 10^{-4}, the FER performance of the proposed RL-CPBP decoder is around 0.92 dB better than that of the BP decoding algorithm in [33]. At the same target FER, CA-SCL4 provides a better error-correction performance in comparison with the proposed RL-CPBP decoder. However, compared with CA-SCL2 at the same target FER, the proposed decoder has a performance gain of around 0.12 dB, under different multi-armed bandit algorithms.

Table 5.2: Computational complexity of different permutation selection schemes in terms of the maximum number of operations performed

Operations	[81]	[55]	ε-greedy	UCB	TS
$+$	0	0	2	$2+k$	2
$-$	0	0	1	1	0
$\times$	0	0	1	$1+k$	0
$\div$	0	0	0	k	0
$\sqrt{}$	0	0	0	k	0
ln	0	0	0	k	0
Random sampling	0	$M-1$	1	0	k
Sorting	0	0	k	k	k

Table 5.2 shows the maximum number of computations required by various permutation selection schemes used in Fig. 5.11. Among all the multi-armed bandit algorithms, the ε-greedy algorithm in general has the lowest computational complexity. This is because the TS algorithm requires a sampling process for k different Beta distributions, which in general requires higher computational complexity than applying an i.i.d. sampling from the interval of $(0, 1)$ and doing a multiplication as required by the ε-greedy algorithm. In addition, although using the cyclically-shifted factor-graph permutations does not consume any additional complexity for the factor-graph permutation selection, this technique is not applicable when more than n different permutations are required. It can also be observed that the main drawback of the multi-armed bandit algorithms is the sorting operations required to identify the exploitation action. However, as described in Section 5.3.4, the action selection process can be performed in parallel with the first CPBP decoding attempt. Therefore, there is no additional latency overhead. Furthermore, the approaches in [81] and [55] come with the cost of error-correction performance degradation when compared with the proposed RL-CPBP decoder as illustrated in Fig. 5.11.

5.4 Chapter Conclusion

This chapter aims to improve the error-correction performance of CRC-Polar concatenated codes under BP decoding in two steps. First, we explore the CRC factor-graph to provide extrinsic information to the polar factor-graph so that trainable weights are assigned to the edges of the unrolled CRC-polar factor graphs to reduce the decoding latency. Second, we use the code permutations to further improve the error-correction performance. In particular, the factor-graph selections of polar

codes under CRC-aided BP decoding are formalized as a multi-armed bandit problem to select the set of good permutations on the fly.

In the next chapters, we consider a closely related coding scheme when compared to polar codes by considering communications systems using RM codes. Note that RM codes are highly applicable for the URLLC use case of 5G thanks to their excellent error-correction performance under highly-parallel near ML decoders.

Chapter 6

Decoding Reed-Muller Codes with Fast Hadamard Transforms

In this chapter, we introduce a novel permutation decoding algorithm of RM codes tailored to the existing FHT-FSCL decoder. In particular, the proposed Permuted FHT-FSCL (p-FHT-FSCL) decoder performs the path extension in the codeword permutation domain to select the L best decoding paths until the first constituent RM code of order 1 is decoded. Furthermore, as p-FHT-FSCL utilizes different subsets of the codeword permutations sampled from the full symmetry group of the codes, the error-correction performance of RM codes can be significantly improved by running M p-FHT-FSCL decoders with list size L in parallel. We then perform a detailed numerical performance analysis of the computational complexity, decoding latency, and memory requirement of the proposed decoders and compare with those of sparse recursive-projection aggregation, FHT-FSCL, and the state-of-the-art permuted successive-cancellation (Aut-SSC) decoders.

6.1 Permuted FHT-FSCL Decoding

The FHT-FSCL decoder introduced in [4] provides a better error-correction performance in comparison with the FSCL decoder for RM codes of low orders when small list sizes ($L \leq 8$) are used. However, for RM codes of orders greater than 2 and with a relatively large list size ($L > 8$), FHT-FSCL decoding provides a negligible error-correction performance gain compared to FSCL decoding. Inspired by the previous RM decoders introduced in [19, 84] and [38], in this section, we propose a permuted FHT-FSCL decoding algorithm that significantly improves the FER

performance of FHT-FSCL decoding, while relatively maintaining the computational complexity, decoding latency, and memory requirement of FHT-FSCL decoding when the same list size is used. The details of the proposed p-FHT-FSCL decoder with list size $L \geq 1$ are provided in Algorithm 9.

The proposed decoder is initialized with L active decoding paths whose LLR vectors are set to the received channel LLRs $\boldsymbol{y}$, and the path metric is set to 0. The proposed decoder then permutes the LLR vectors of the L decoding paths using L random codeword permutations sampled from the full symmetry group of the RM codes (lines 2-4 in Algorithm 9), where π_l^{init} indicates the initial codeword permutation applied to the l-th path. This initialization process is only performed once for each received channel LLR vector $\boldsymbol{y}$. The decoding continues with the path extension performed in the permutation domain until the first constituent RM code of order 1 is visited. Lines 6-12 of Algorithm 9 specify the proposed permutation decoding. In particular, we sample two random codeword permutations, π_l, for each active decoding path to obtain the permutations of $\boldsymbol{\alpha}_{v_l}$, denoted as $\boldsymbol{\alpha}_v^{\text{tmp}}$. The permuted LLR vector $\boldsymbol{\alpha}_v^{\text{tmp}}$ is used to compute the LLR values of the left-child node $\boldsymbol{\alpha}_\lambda^{\text{tmp}}$ using the $f(\cdot)$ function. Then, the reliability metric proposed in [84] is computed to select the L permutations that have the maximum channel reliabilities LM_l of the left-child node λ (lines 8-11 of Algorithm 9). In line 12 of Algorithm 9, the selected permutations are applied to the input LLR vectors to form the L best decoding paths. Here, by l_{org}^* we denote the index of the input LLR vector whose permutation π_l^* provides the channel reliability that is among the L largest channel reliabilities.

Note that the proposed permutation decoding selects the best permutations originated from all the current active decoding paths, which is different from the decoders proposed in [19, 38, 84] where permutation decoding is utilized separately for each decoding path. Furthermore, the left and right child node of v are recursively decoded using the proposed decoder as specified in lines 13-16 of Algorithm 9. Also note that a permutation sampled from the full symmetry group transforms a RM code to another RM code of similar length and order, whose frozen bit indicies are in general different from the frozen-bit indices of the original RM code. Therefore, one needs to re-permute the estimated codeword of the permuted LLR vector $\boldsymbol{\alpha}_{v_l}$ to reconstruct the original codeword [38]. These operations are described in lines 18-21 of Algorithm 9. Finally, the `FHTL`$(\cdot)$ and `SPCL`$(\cdot)$ functions are queried to decode the first-order RM subcodes and the SPC subcodes, respectively. The `SPCL`$(\cdot)$ function carries out the FSCL decoding operations, while the details of the `FHTL`$(\cdot)$ function are provided in Algorithm 10.

In Algorithm 10, for each input path with index l, we apply a modified FHT decoding algo-

Algorithm 9: p-FHT-FSCL$(\cdot)$ Decoding

Input : $\{\boldsymbol{\alpha}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L}$
Output: $\{\hat{\boldsymbol{x}}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L}$

Function p-FHT-FSCL $\left(\{\boldsymbol{\alpha}_{\gamma_l}, \mathrm{PM}_l\}_{0\le l<L}\right)$**:**
  /* Initialize L decoding paths with L random codeword permutaitons once */
  if $\mathcal{RM}(r_\nu, m_\nu)$ is the root node **then**
    isPermutation $\leftarrow$ True
    $\{\pi_l^{\text{init}} : \boldsymbol{\alpha}_{\nu_l} \xrightarrow{\pi_l^{\text{init}}} \boldsymbol{\alpha}_{\nu_l}\}_{0\le l<L}$
  if $1 < r_\nu < m_\nu - 1$ **then**
    /* Permutation decoding if applicable */
    if isPermutation = True **then**
      isPer_ν = True
      for $l \leftarrow 0$ **to** $2L$ **do**
        $l_{\text{org}} \leftarrow l \mod L$; $\pi_l : \boldsymbol{\alpha}_{\nu_{l_{\text{org}}}} \xrightarrow{\pi_l} \boldsymbol{\alpha}_\nu^{\text{tmp}}$
        $\boldsymbol{\alpha}_\lambda^{\text{tmp}} \leftarrow f\left(\boldsymbol{\alpha}_\nu^{\text{tmp}}\right)$; $\mathrm{LM}_l \leftarrow \sum_{\forall i} \left|\alpha_\lambda^{\text{tmp}}[i]\right|$
      $\{\pi_l^*, l_{\text{org}}^*\}_{0\le l<L} \leftarrow \mathtt{Sort}\left(\mathrm{LM}_0, \ldots, \mathrm{LM}_{2L-1}\right)$
      $\{\pi_l^* : \boldsymbol{\alpha}_{\nu_{l_{\text{org}}^*}} \xrightarrow{\pi_l^*} \boldsymbol{\alpha}_{\nu_l}\}_{0\le l<L}$
    /* Recursively decode the left-child node $\mathcal{RM}(r_\nu - 1, m_\nu - 1)$ */
    $\{\boldsymbol{\alpha}_{\lambda_l} \leftarrow f\left(\boldsymbol{\alpha}_{\nu_l}\right)\}_{0\le l<L}$
    $\{\hat{\boldsymbol{x}}_{\lambda_l}, \mathrm{PM}_l\}_{0\le l<L} \leftarrow$ p-FHT-FSCL $\left(\{\boldsymbol{\alpha}_{\lambda_l}, \mathrm{PM}_l\}_{0\le l<L}\right)$
    /* Recursively decode the right-child node $\mathcal{RM}(r_\nu, m_\nu - 1)$ */
    $\{\boldsymbol{\alpha}_{\gamma_l} \leftarrow g(\boldsymbol{\alpha}_{\nu_{l_{\text{org}}}}, \hat{\boldsymbol{x}}_{\lambda_l})\}_{0\le l<L}$
    $\{\hat{\boldsymbol{x}}_{\gamma_l}, \mathrm{PM}_l\}_{0\le l<L} \leftarrow$ p-FHT-FSCL $\left(\{\boldsymbol{\alpha}_{\gamma_l}, \mathrm{PM}_l\}_{0\le l<L}\right)$
    /* Form the estimation of $\mathcal{RM}(r_\nu, m_\nu)$ and repermute if applicable */
    $\{\hat{\boldsymbol{x}}_{\nu_l} \leftarrow \mathtt{Concate}(\hat{\boldsymbol{x}}_{\gamma_l} \oplus \hat{\boldsymbol{x}}_{\lambda_{l_{\text{org}}}}, \hat{\boldsymbol{x}}_{\gamma_l})\}_{0\le l<L}$
    if isPer_ν = True **then**
      $\{(\pi_{l_{\text{org}}}^*)^{-1} : \hat{\boldsymbol{x}}_{\nu_l} \xrightarrow{(\pi_{l_{\text{org}}}^*)^{-1}} \hat{\boldsymbol{x}}_{\nu_l}\}_{0\le l<L}$
    if $\mathcal{RM}(r_\nu, m_\nu)$ is the root node **then**
      $\{(\pi_{l_{\text{org}}}^{\text{init}})^{-1} : \hat{\boldsymbol{x}}_{\nu_l} \xrightarrow{(\pi_{l_{\text{org}}}^{\text{init}})^{-1}} \hat{\boldsymbol{x}}_{\nu_l}\}_{0\le l<L}$
  else if $r_\nu = 1$ **then**
    if isPermutation = True **then**
      isPermutation $\leftarrow$ False
    $\{\hat{\boldsymbol{x}}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L} \leftarrow$ FHTL $\left(\{\boldsymbol{\alpha}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L}\right)$
  else if $r_\nu = m - 1$ **then**
    $\{\boldsymbol{\alpha}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L} \leftarrow$ SPCL $\left(\{\hat{\boldsymbol{x}}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L}\right)$
  return $\{\hat{\boldsymbol{x}}_{\nu_l}, \mathrm{PM}_l\}_{0\le l<L}$

Algorithm 10: FHTL$(\cdot)$ Decoding

Input : $\{\boldsymbol{\alpha}_{v_l}, \mathrm{PM}_l\}_{0\le l<L}$
Output: $\{\hat{\boldsymbol{x}}_{v_l}, \mathrm{PM}_l\}_{0\le l<L}$

1 **Function** FHTL$(\{\boldsymbol{\alpha}_{v_l}, \mathrm{PM}_l\}_{0\le l<L})$**:**
  /* Perform FHT decoding for each active decoding path */
2 $\mathcal{E} \leftarrow \emptyset$
3 **for** $l \leftarrow 0$ **to** $L-1$ **do**
4 $\{\boldsymbol{Q}_0, \ldots, \boldsymbol{Q}_{\min(L,2^s)-1}\} \leftarrow \mathrm{FHT}(\boldsymbol{\alpha}_{v_l}, \mathrm{PM}_l)$
5 $\mathcal{E} \leftarrow \mathcal{E} \cup \{\boldsymbol{Q}_0, \ldots, \boldsymbol{Q}_{\min(L,2^s)-1}\}$
6 $\{\boldsymbol{Q}^*_0, \ldots, \boldsymbol{Q}^*_{L-1}\} \leftarrow \mathrm{Sort}(\mathcal{E})$
  /* Return the L best decoding paths */
7 **for** $l \leftarrow 0$ **to** $L-1$ **do**
8 $\hat{\boldsymbol{x}}_{v_l} \leftarrow \boldsymbol{Q}^*\{\hat{\boldsymbol{x}}_v\}$; $\mathrm{PM}_l \leftarrow \boldsymbol{Q}^*\{\mathrm{PM}_v\}$
9 **return** $\{\hat{\boldsymbol{x}}_{v_l}, \mathrm{PM}_l\}_{0\le l<L}$

rithm on $\boldsymbol{\alpha}_{v_l}$ and generate the $\min(L, 2^s)$ most probable decoding paths and their associated path metrics originated from $\boldsymbol{\alpha}_{v_l}$. The modified FHT decoding algorithm, FHT$(\cdot)$, that utilizes a low-complexity path metric computation scheme is provided in Algorithm 11. Specifically, after the FHT operations are applied to $\boldsymbol{\alpha}_{v_l}$ in Algorithm 11, the indices of the largest absolute values of the transformed LLR vector $\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}$ are obtained using a sorting algorithm (line 10 of Algorithm 11). We only need to construct a maximum of L best decoding paths generated from the current active path l under FHT decoding. Therefore, it is not necessary to sort all the elements of the transformed LLR values $\left|\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}[i]\right|$ $(0 \le i < N_v)$ given a small list size L. Consequently, the complexity of the sorting algorithm used in line 10 of Algorithm 11 is $\min(L2^s, s2^s)$. Note that $L2^s$ is the number of comparisons required by a straight-forward sorting algorithm that loops through the vector $\left|\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}\right|$ L times to identify the indices of L maximum elements, while a maximum of $s2^s$ comparisons are required by the merge sort algorithm, which is efficient for a large value of L [76]. The sorting algorithm in line 10 of Algorithm 11 outputs the sorted indices $\{i^*_0, \ldots, i^*_{\min(L,2^s)-1}\}$, where $\left|\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}[i^*_0]\right| \ge \ldots \ge \left|\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}[i^*_{\min(L,2^s)-1}]\right|$, and $\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}[i]$ indicates the i-th element of $\boldsymbol{\alpha}^{\mathrm{FHT}}_{v_l}$. The indices $\{i^*_0, \ldots, i^*_{\min(L,2^s)-1}\}$ are then used to estimate the message word $\hat{\boldsymbol{u}}_v$ associated with $\boldsymbol{\alpha}_{v_l}$ (lines 12-17 of Algorithm 11). dec2bin(i) is a function that converts the decimal value of a bit index i to its binary expansion represented by s binary numbers.

In line 18 of Algorithm 11, $\hat{\boldsymbol{x}}_v$ is the estimated codeword corresponding to the i^*_j-th element of

Algorithm 11: FHT$(\cdot)$ Decoding

Input : $\boldsymbol{\alpha}_v, \mathrm{PM}_l$
Output: $\{\boldsymbol{Q}_0, \ldots, \boldsymbol{Q}_{\min(L,2^s)-1}\}$

Function $\mathtt{FHT}(\boldsymbol{\alpha}_v, \mathrm{PM}_l, l)$**:**
  /* Initialization */
  $\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}} \leftarrow \boldsymbol{\alpha}_{v_l}$; $\mathrm{LLR}_{\mathrm{abs}} \leftarrow \sum_{k=0}^{N_v-1} \left|\boldsymbol{\alpha}_{v_l}[i]\right|$
  /* Fast Hadamard Transform of $\boldsymbol{\alpha}_{v_l}$ */
  for $t \leftarrow 0$ **to** $s-1$ **do**
    for $j \leftarrow 0$ **to** $2^{t+1}-1$ **do**
      for $i \leftarrow j2^{s-t}$ **to** $j2^{s-t}+2^{s-t-1}-1$ **do**
        $a \leftarrow \boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i+2^{s-t-1}] - \boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i]$
        $b \leftarrow \boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i+2^{s-t-1}] + \boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i]$
        $\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i] \leftarrow a$
        $\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i+2^{s-t-1}] \leftarrow b$
  /* Obtain up to L best decoding paths */
  $\{i_0^*, \ldots, i_{\min(L,2^s)-1}^*\} \leftarrow \mathtt{Sort}\left(\left|\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}\right|\right)$
  for $j \leftarrow 0$ **to** $\min(L, 2^s)-1$ **do**
    /* Form the estimated messageword $\hat{\boldsymbol{u}}_v$ */
    $m_s = \frac{1-\mathrm{sgn}\,\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i_j^*]}{2}$
    $\{m_{s-1}, \ldots, m_0\} = \mathrm{dec2bin}(2^s - 1 - i_j^*)$
    $t \leftarrow 0$; $\hat{\boldsymbol{u}}_v \leftarrow \boldsymbol{0}$
    for $k \leftarrow 0$ **to** 2^s-1 **do**
      if k is an information bit index **then**
        $\hat{\boldsymbol{u}}_v[k] \leftarrow m_t$; $t \leftarrow t+1$
    $\hat{\boldsymbol{x}}_v \leftarrow \hat{\boldsymbol{u}}_v \boldsymbol{G}^{\otimes s}$
    $\mathrm{PM}_v \leftarrow \mathrm{PM}_l + \frac{1}{2}\left(\mathrm{LLR}_{\mathrm{abs}} - \left|\alpha_{v_l}^{\mathrm{FHT}}[i_j^*]\right|\right)$
    /* Form the output data structure */
    $\boldsymbol{Q}_j \leftarrow \{\hat{\boldsymbol{x}}_v, \mathrm{PM}_v\}$
  return $\{\boldsymbol{Q}_0, \ldots, \boldsymbol{Q}_{\min(L,2^s)-1}\}$

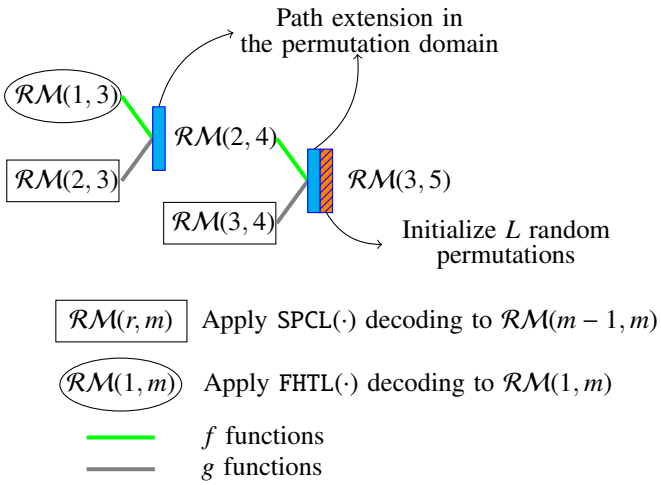


Figure 6.1: An example of the p-FHT-FSCL decoder with list size $L \geq 1$ when applied to $\mathcal{RM}(3,5)$.

$\left|\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}\right|$. The path metric associated with $\hat{\boldsymbol{x}}_v$ is calculated as [3]

$$\mathrm{PM}_v = \mathrm{PM}_l + \frac{1}{2}\sum_{k=0}^{N_v-1}\left(\left|\boldsymbol{\alpha}_{v_l}[k]\right| - (1-2\hat{\boldsymbol{x}}_v[k])\,\boldsymbol{\alpha}_{v_l}[k]\right), \tag{6.1}$$

which can be rewritten as

$$\begin{aligned}\mathrm{PM}_v &= \mathrm{PM}_l + \frac{1}{2}\left[\mathrm{LLR}_{\mathrm{abs}} - \sum_{k=0}^{N_v-1}(1-2\hat{\boldsymbol{x}}_v[k])\,\boldsymbol{\alpha}_{v_l}[k]\right] \\ &= \mathrm{PM}_l + \frac{1}{2}\left(\mathrm{LLR}_{\mathrm{abs}} - \left|\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}[i_j^*]\right|\right).\end{aligned} \tag{6.2}$$

Line 19 of Algorithm 11 computes the path metric associated with $\hat{\boldsymbol{x}}_v$ using (6.2), which reuses the transformed LLR vector $\boldsymbol{\alpha}_{v_l}^{\mathrm{FHT}}$ to reduce the number of additions compared to (6.1). Algorithm 11 outputs the most probable decoding paths under FHT decoding as a set of the data structure $\boldsymbol{Q}_j$, i.e., $\{\boldsymbol{Q}_0, \ldots, \boldsymbol{Q}_{\min(L,2^s)-1}\}$, where $\boldsymbol{Q}_j$ consists of an estimated codeword $\hat{\boldsymbol{x}}_v$ and its corresponding path metric PM_v. Note that as a maximum of L best decoding paths are selected to continue the decoding after v is visited, it is sufficient for Algorithm 11 to generate a maximum of L candidate paths associated with each LLR vector $\boldsymbol{\alpha}_{v_l}$ $(0 \leq l < L)$.

In Algorithm 10, the outputs of the FHT$(\cdot)$ function that is applied to all the current active

Algorithm 12: p-FHT-FSCL-L-$M(\cdot)$ Decoding

Input : $\boldsymbol{y}, M, L$
Output: $\hat{\boldsymbol{x}}$

1 $\text{PM}^* \leftarrow \infty$
2 **for** $i \leftarrow 0$ **to** $M-1$ **do**
  /* Initialization and decoding of the target RM code for each decoding attempt */
3 $\{\boldsymbol{\alpha}_{\nu_l} \leftarrow \boldsymbol{y}, \text{PM}_l \leftarrow 0\}_{0 \le l < L}$
4 $\{\hat{\boldsymbol{x}}_{\nu_l}, \text{PM}_l\}_{0 \le l < L} \leftarrow \text{p-FHT-FSCL}\left(\{\boldsymbol{\alpha}_{\nu_l}, \text{PM}_l\}_{0 \le l < L}\right)$
5 $l^* \leftarrow \arg\min\{\text{PM}_0, \ldots, \text{PM}_{L-1}\}$
  /* Select the best estimated codeword from M decoding attempts */
6 **if** $\text{PM}_{l^*} < \text{PM}^*$ **then**
7 $\text{PM}^* \leftarrow \text{PM}_{l^*}$
8 $\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}}_{\nu_{l^*}}$
9 **return** $\hat{\boldsymbol{x}}$

paths are stored in a set $\mathcal{E}$. The sorting function applied to $\mathcal{E}$ (line 6 of Algorithm 10) generates a set of sorted data structures $\{\boldsymbol{Q}_0^*, \ldots, \boldsymbol{Q}_{L-1}^*\}$ such that $\boldsymbol{Q}_0^*\{\text{PM}_\nu\} \le \ldots \le \boldsymbol{Q}_{L-1}^*\{\text{PM}_\nu\}$, where $\boldsymbol{Q}_i^*\{\text{PM}_\nu\}$ indicates the path metric associated with the i-th data structure $\boldsymbol{Q}_i^*$. We use the merge sort algorithm to output the sorted data structure $\boldsymbol{Q}_i^*$. Since the maximum size of $\mathcal{E}$ is L^2, the maximum number of floating-point comparisons required by the sorting operation in line 6 of Algorithm 10 is $2L^2 \log_2 L$ [76]. The remainder of Algorithm 10 outputs the estimated codewords $\hat{\boldsymbol{x}}_{\nu_l}$ and the associated path metrics PM_l of all the best L decoding paths. Fig. 6.1 depicts an example of the proposed p-FHT-FSCL decoder on $\mathcal{RM}(3,5)$, where the proposed permutation decoding is only applied to $\mathcal{RM}(3,5)$ and its descendant $\mathcal{RM}(2,4)$. On the other hand, $\mathcal{RM}(3,4)$ and $\mathcal{RM}(2,3)$ are decoded using the FSCL decoding operations specified for the SPC nodes, while $\mathcal{RM}(1,3)$ is decoded using the FHTL decoding algorithm specified in Algorithm 10.

It can be observed in Algorithm 9 that the proposed p-FHT-FSCL decoder utilizes different subsets of the codeword permutations during the course of decoding. Therefore, to further utilize the rich symmetry group of RM codes, we run M ($M > 1$) p-FHT-FSCL decoders with list size L ($L \ge 1$) in parallel. Then, we select the output codeword that has the smallest path metric as the final estimated codeword. This improved decoder is denoted at p-FHT-FSCL-L-M. In Algorithm 12, we summarize the p-FHT-FSCL-L-M decoder that utilizes path splitting in both codeword permutation and information bit domains.

6.2 Performance Evaluation

6.2.1 Quantitative Complexity Analysis

We calculate the computational complexity of all the decoders presented in this chapter by counting the number of floating-point additions and comparisons performed during the course of decoding for a received channel LLR vector $\boldsymbol{y}$. We summarize the computational complexities of all the decoding functions applied to a RM subcode of the proposed decoders in Table 6.1 and Table 6.2. Furthermore, we compute the decoding latency of all the decoders presented in this chapter by using the assumptions considered in [3,26]. Specifically, the hard decisions obtained from the LLR values and binary operations are computed instantaneously, and all the independent computations are calculated in parallel. Finally, we consider the number of time steps required by a merge sort algorithm to sort a vector of size N to be $\log_2 N$ [76].

Table 6.1: Normalized computational complexities of different decoding functions required by the p-FHT-FSCL-L decoder with $L > 1$. The decoding functions are applied to a RM sub-code $\mathcal{RM}(r,m)$ visited by the decoding algorithm.

Function	Computation		Sorting		Total
	LLR	Path Metric	LLR	Path Metric	
$f(\cdot)$	$L2^{m-1}$	-	-	-	$L2^{m-1}$
$g(\cdot)$	$L2^{m-1}$	-	-	-	$L2^{m-1}$
$\mathcal{RM}(1,m)$	$Lm2^m$	$L^2 2^m$	$\min(Lm2^m, L^2 2^m)$	$2L^2 \log_2 L$	$Lm2^m + \min(Lm2^m, L^2 2^m) + 2L^2(2^{m-1} + \log_2 L)$
$\mathcal{RM}(m-1,m)$	-	$L(1+2\tau)$	$\min(Lm2^m, L^2 2^m)$	$2\tau L(1+\log_2 L)$	$\min(Lm2^m, L^2 2^m) + L\,[2\tau(2+\log_2 L)+1]$

Table 6.2: Normalized computational complexities of different decoding functions of the p-FHT-FSCL-1 decoder. The decoding functions are applied to a RM sub-code $\mathcal{RM}(r,m)$ visited by the decoding algorithm.

Function	LLR Computation	LLR Sorting	Total
$f(\cdot)$	2^{m-1}	-	2^{m-1}
$g(\cdot)$	2^{m-1}	-	2^{m-1}
$\mathcal{RM}(1,m)$	$m2^m$	2^m	$2^m(m+1)$
$\mathcal{RM}(m-1,m)$	-	2^m	2^m

The FHT used in Algorithm 11 only uses in-place computations that do not require extra memory for the LLR values [85]. In addition, the path extension in the permutation domain of the

Table 6.3: Summary of the memory requirements of the proposed decoders.

Algorithm	Memory Requirement in Bits
p-FHT-FSC-1	$(2N + 1)Q + N$
p-FHT-FSC-1-M $(M > 0)$	$(N + M(N + 1))Q + MN$
p-FHT-FSC-L $(L > 1)$	$N(L + 1)Q + 2LQ + 2NL$
p-FHT-FSC-L-M $(L > 1, M > 1)$	$N(LM + 1)Q + 2MLQ + 2MNL$

proposed decoders is carried out sequentially for each decoding path. This allows the proposed decoders to maintain a similar memory requirement to store the LLR values compared to FHT-FSCL decoding with the same list size. The memory requirements in terms of the number of bits for the proposed decoders are summarized in Table 6.3, where $Q = 32$ is the number of bits used to store a floating-point value.

6.2.2 Comparison with FSCL and FHT-FSCL Decoding

Fig. 6.2 illustrates the FER performance of FHT-FSCL and p-FHT-FSCL decoders with various list sizes $L \geq 1$. The FER performance of FSCL decoding with list size 32 (FSCL-32) is also plotted for comparison. Furthermore, Fig. 6.3 plots the computational complexity $\mathcal{C}$, decoding latency in time steps $\mathcal{T}$, and memory requirement $\mathcal{M}$ in Kilobytes (KBs) of FHT-FSCL and p-FHT-FSCL decoders considered in Fig. 6.2. Note that FHT-FSCL-1 indicates the FHT-FSC decoder and p-FHT-FSCL-1 indicates the proposed decoder that performs the path extension in the permutation domain until the first constituent RM code of order 1 is visited, at which point the decoding operations are performed exactly similar to those of FHT-FSC decoding.

It can be observed from Fig. 6.2 and Fig. 6.3 that by utilizing the proposed permutation decoding scheme, p-FHT-FSCL decoding significantly outperforms FHT-FSCL decoding with a similar list size $L > 1$ for all the considered RM codes, while relatively maintaining all the complexity metrics. In particular, for $\mathcal{RM}(4, 9)$, p-FHT-FSCL-32 provides an error-correction performance gain of 1 dB at the target FER of 10^{-4} in comparison with FHT-FSCL-32, while having overheads of 9% in the computation complexity and 2.4% in the decoding latency. Note that p-FHT-FSCL-32 preserves the memory requirement of FHT-FSCL-32.

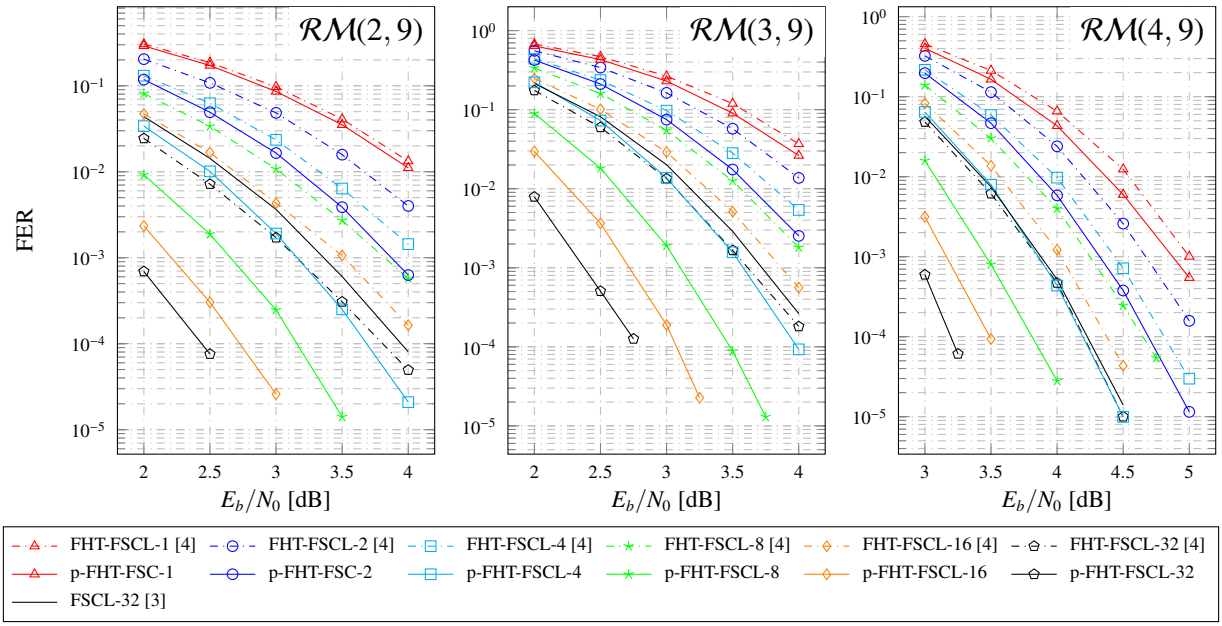


Figure 6.2: FER performance of the FHT-FSCL and p-FHT-FSC decoders for various RM codes. The FER values of the FSCL decoder with list size 32 (FSCL-32) are also plotted for comparison.

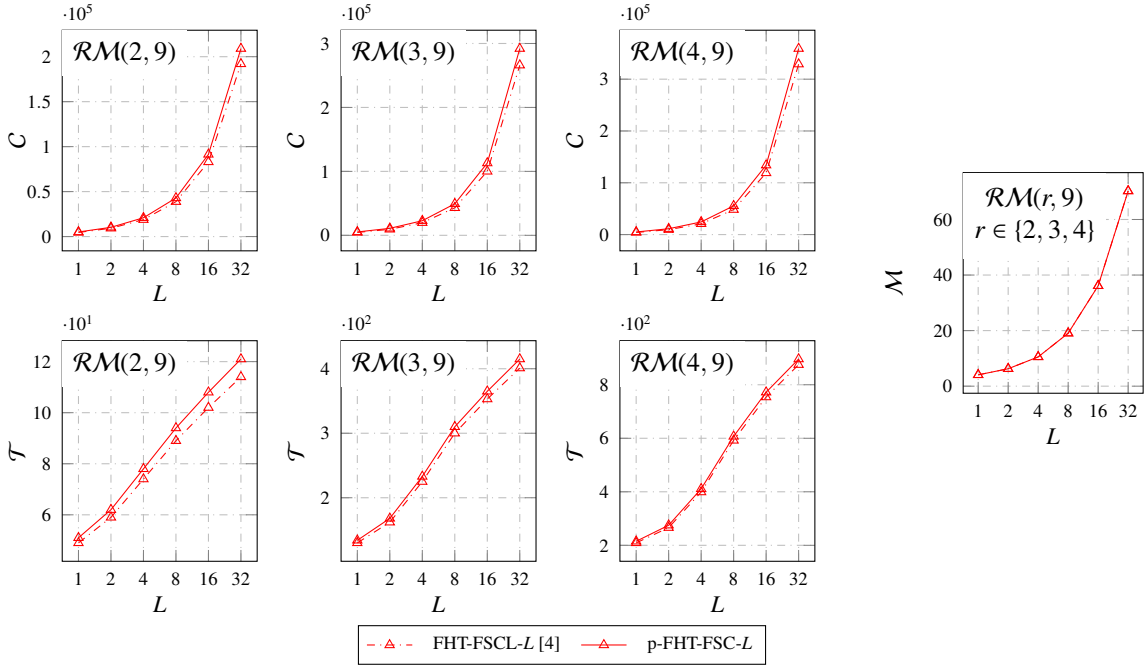


Figure 6.3: Computational complexity ($\mathcal{C}$), decoding latency in time steps ($\mathcal{T}$), and memory requirement in KBs ($\mathcal{M}$) of FHT-FSCL-L and p-FHT-FSCL-L considered in Fig. 6.2.

Table 6.4 summarizes the computational complexity, decoding latency, and memory requirement of FSCL-32, FHT-FSCL-32, and p-FHT-FSCL-4, whose FER values are relatively similar at the target FER of 10^{-3} as shown in Fig. 6.2. It can be observed from Table 6.4 that the negligible error-correction performance improvement of FHT-FSCL-32 with respect to FSCL-32 comes at the cost of significant computational complexity overhead, which is mainly caused by the sorting operations required by the FHT-based decoding algorithm. Note that the computational complexity required by the sorting operations under FHT-FSCL-based decoding increases significantly as the list size increases. On the other hand, by utilizing the proposed permutation decoding algorithm, the permuted FHT-FSCL decoder only requires a list size of 4 to obtain a similar or better error-correction performance compared to FSCL-32 and FHT-FSCL-32 at the target FER of 10^{-3}. The use of a much smaller list size (4 instead of 32) also enables p-FHT-FSCL-4 to obtain significantly lower complexity metrics compared to FSCL-32 and FHT-FSCL-32 as observed from Table 6.4. For example, in comparison with FHT-FSCL-32 for $\mathcal{RM}(4,9)$, p-FHT-FSCL-4 reduces 93% of the computational complexity, 53% of the decoding latency, and 85% of the memory requirement.

Table 6.4: Comparison of normalized computational complexity ($\mathcal{C}$), decoding latency in time steps ($\mathcal{T}$), and memory requirement in KBs ($\mathcal{M}$) of FSCL-32 [3], FHT-FSCL-32 [4], and p-FHT-FSCL-4, whose FER values are shown in Fig. 6.2.

	FSCL-32 [3]			FHT-FSCL-32 [4]			p-FHT-FSCL-4		
	$\mathcal{C}$	$\mathcal{T}$	$\mathcal{M}$	$\mathcal{C}$	$\mathcal{T}$	$\mathcal{M}$	$\mathcal{C}$	$\mathcal{T}$	$\mathcal{M}$
$\mathcal{RM}(2,9)$	9.59E+04	373	70.25	1.92E+05	114	70.25	2.09E+04	78	10.53
$\mathcal{RM}(3,9)$	1.55E+05	1039	70.25	2.66E+05	401	70.25	2.28E+04	233	10.53
$\mathcal{RM}(4,9)$	2.24E+05	1991	70.25	3.29E+05	875	70.25	2.44E+04	411	10.53

6.2.3 Comparison with Permuted SC-Based Decoding and RPA-Based Decoding

Fig. 6.4 illustrates the error-correction performance of the simplified SC (SSC) [24] decoder when utilizing P random codeword permutations sampled from the full symmetry group of the codes (Aut-SSC-P) and that of the RPA [39] and Sparse RPA (SRPA) [86] decoders. In addition, we consider the following configurations of the proposed decoders in Fig. 6.4: p-FHT-FSCL-L, p-FHT-FSCL-1-M_1, and p-FHT-FSCL-4-M_4. Note that p-FHT-FSCL-1-M_1 runs M_1 p-FHT-FSCL-1 decoders in parallel while p-FHT-FSCL-4-M_4 runs M_4 p-FHT-FSCL-4 decoders in parallel. Also note that under p-FHT-FSCL-1, only the path extension in the permutation domain is carried out.

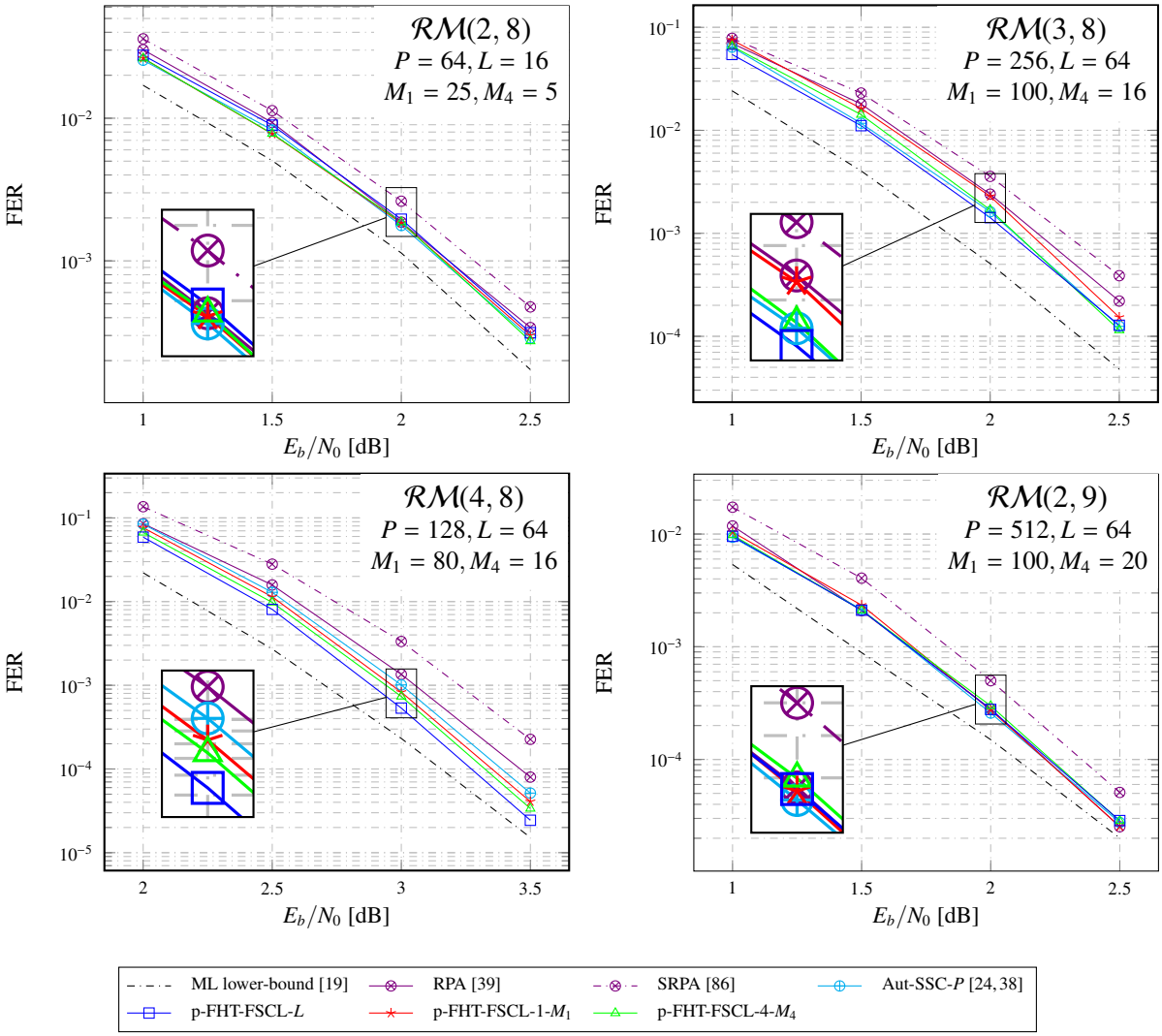


Figure 6.4: Error-correction performance of various RM decoders.

Table 6.5: Comparison of normalized computational complexity (C), decoding latency in time steps ($\mathcal{T}$), and memory requirement in KBs ($\mathcal{M}$) of various RM decoders considered in Fig 6.4.

	RPA [39]			SRPA [86]			Aut-SSC-P [24, 38]				p-FHT-FSCL-L				p-FHT-FSCL-1-M_1				p-FHT-FSCL-4-M_4			
	C	$\mathcal{T}$	$\mathcal{M}$	C	$\mathcal{T}$	$\mathcal{M}$	P	C	$\mathcal{T}$	$\mathcal{M}$	L	C	$\mathcal{T}$	$\mathcal{M}$	M_1	C	$\mathcal{T}$	$\mathcal{M}$	M_4	C	$\mathcal{T}$	$\mathcal{M}$
$\mathcal{RM}(2,8)$	1.8E+6	3592	135.5	6.5E+5	3592	69.2	64	9.4E+4	80	67.0	16	4.5E+4	92	18.1	25	5.7E+4	46	26.9	5	4.8E+4	68	22.4
$\mathcal{RM}(3,8)$	4.3E+8	6184	556.8	7.9E+7	6184	281.5	256	4.5E+5	147	265.0	64	5.2E+5	356	69.5	100	2.3E+5	100	104.5	16	1.7E+5	173	69.5
$\mathcal{RM}(4,8)$	3.8E+10	7816	922.2	3.6E+9	7816	465.2	128	2.2E+5	165	133.0	64	5.7E+5	673	69.5	80	1.7E+5	131	83.8	16	1.8E+5	251	69.5
$\mathcal{RM}(2,9)$	9.8E+6	10250	535.1	3.4E+6	10250	271.6	512	1.5E+6	106	1058.0	64	5.4E+5	134	138.5	100	5.1E+5	58	208.6	20	4.2E+5	83	172.6

The values of L, M_1, and M_4 are selected to provide a similar error-correction performance of the proposed decoders in comparison with RPA and Aut-SSC decoding at the target FER of 10^{-3}. We also plot the empirical ML lower bounds of the FER values for all the RM codes considered in Fig. 6.4 [19]. In Table 6.5, we summarize the computational complexity, decoding latency in time steps, and memory requirement in KBs of all the decoders considered in Fig. 6.4.

It can be observed from Fig. 6.4 and Table 6.5 that all the permutation decoding algorithms outperform the RPA and SRPA decoders in various complexity metrics while having a similar or better error-correction performance compared to the RPA decoder. Note that p-FHT-FSCL-L is the most memory-efficient decoding algorithm, while p-FHT-FSCL-1-M_1 provides the lowest decoding latency in time steps among all the decoders. On the other hand, the p-FHT-FSCL-4-M_4 configuration enables a better decoding latency and memory requirement trade-off compared to p-FHT-FSCL-L and p-FHT-FSCL-1-M_1 settings and obtains the smallest computational complexity for $\mathcal{RM}(3,8)$ and $\mathcal{RM}(2,9)$. In addition, for $\mathcal{RM}(2,8)$ and $\mathcal{RM}(3,8)$, p-FHT-FSCL-4-M_4 has a similar memory requirement to p-FHT-FSCL-L, while having significantly smaller computational complexity and decoding latency. Compared to Aut-SSC-512 for $\mathcal{RM}(2,9)$ and at a target FER of 10^{-4}, p-FHT-FSCL-4-M_4 reduces 72% of the computational complexity, 22% of the decoding latency, and 84% of the memory requirement. Compared to SRPA decoding for $\mathcal{RM}(2,9)$ and at the target FER of 10^{-4}, p-FHT-FSCL-4-M_4 provides 36% reduction in the memory consumption and 88% reduction in the computational complexity, while achieving several order-of-magnitude lower decoding latency and 0.14 dB gain in error-correction performance.

6.3 Chapter Conclusion

In this chapter, we introduced a novel permutation decoding algorithm for Reed-Muller (RM) codes tailored to the existing fast successive-cancellation list decoder with fast Hadamard transform (FHT-FSCL). The proposed permuted FHT-FSCL (p-FHT-FSCL) decoder performs the path extension in the codeword permutation domain to select the L best decoding paths until the first constituent RM code of order 1 is decoded. As the p-FHT-FSCL decoder utilizes different subsets of the codeword permutations sampled from the full symmetry group of the codes, the error-correction performance of RM codes can be significantly improved by running M p-FHT-FSCL decoders with list size L in parallel. We performed a detailed numerical performance analysis of the computational complexity, decoding latency, and memory requirement of the proposed decoders

and compared with those of sparse recursive-projection aggregation, FHT-FSCL, and the state-of-the-art permuted successive-cancellation (Aut-SSC) decoders. The simulation results show that for the RM code of length 512 with order 2, the proposed decoder with $L = 4$ and $M = 20$ reduces 72% of the computational complexity, 22% of the decoding latency, and 84% of the memory requirement with respect to the state-of-the-art Aut-SSC decoder with 512 random codeword permutations, while obtaining a similar error-correction performance at the target frame error rate of 10^{-4}.

Chapter 7

Decoding Reed-Muller Codes with Successive Codeword Permutations

In this chapter, a generalized Successive Permutations (SP) scheme for the RLD-based algorithms of RM codes is first proposed. We then provide details on the integration of the proposed SP scheme into an improved RLD-based algorithm, forming the SP-aided RLD (SP-RLD) and the simplified SP-RLD (SSP-RLD) decoding algorithms. Finally, we numerically analyze the error-correction performance, computational complexity, decoding latency, and memory requirement of the proposed decoder and compare them with those of the state-of-the-art RM decoders.

7.1 Improved Successive Permutation Scheme

The SP selection criteria used in [84] is an oversimplification that does not take into account the existing parity constraints in the code. In fact, it treats all the constituent RM codes λ as Rate-1 codes. This oversimplification becomes inaccurate, especially for low-order RM codes, as the number of information bits is significantly smaller than $N_\lambda = 2^{s-1}$. Consequently, the error probability of the SP scheme in [84] for SCL decoding on low-order RM codes is not satisfactory, especially when a small to moderate list size is used.

To tackle this issue, we propose an accurate SP scheme that selects the best codeword permutation π^* by performing ML decoding on the symmetry group of RM codes. The proposed selection

criteria is given as

$$\pi^* = \arg\max_{\pi \in \mathcal{P}_s} M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right), \tag{7.1}$$

where $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ is the permutation metric of π when π is applied to the parent node ν that is calculated as

$$M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right) = \max_{\forall \boldsymbol{\eta}^{(\lambda)}} \sum_{i=i_{\min_\lambda}}^{i_{\max_\lambda}} \eta_{s-1,i}^{(\lambda)} \alpha_{s-1,i}^{(\lambda)}, \tag{7.2}$$

with $\boldsymbol{\eta}^{(\lambda)}$ being the hard decisions of a valid codeword corresponding to λ. It can be observed that if λ is a Rate-1 code, (7.1) reverts to the scheme used in [84] as $\eta_{s-1,i}^{(\lambda)}$ is set to $\operatorname{sgn}(\alpha_{s-1,i}^{(\lambda)})$ to maximize the likelihood of $\boldsymbol{\eta}^{(\lambda)}$ and $\boldsymbol{\alpha}^{(\lambda)}$. The elements of $\boldsymbol{\eta}^{(\lambda)}$ can be calculated by performing ML decoding on λ. However, ML decoding is generally of high complexity. Therefore, in this chapter we derive $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ for special cases of λ for which the ML decoding operations can be realized with low complexity. Unlike [84], the set $\mathcal{P}_s$ considered in this chapter contains the general codeword permutations sampled from the full symmetry group of the codes [38]. Nevertheless, we limit the maximum number of permutations stored in $\mathcal{P}_s$ to s, which is equal to the number of cyclic factor-graph permutations as considered in [84].

Since first-order constituent RM codes can be decoded efficiently using ML decoding [87, Chapter 14], the permutation metric $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ can be efficiently calculated if λ is a first-order RM code. When λ is of order 2 or higher, we propose to simplify the computation of $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ by using the metric proposed in [84]. The calculation of $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ for the considered special nodes in this chapter is summarized as follows.

- $\mathcal{RM}(1, s-1)$: The metric $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ is the likelihood of the best decoding path of λ given $\boldsymbol{\alpha}^{(\lambda)}$, which can be calculated efficiently using FHT decoding [87, Chapter 14].

- $\mathcal{RM}(r_\lambda, s-1)$ $(r_\lambda \geq 2)$: We calculate $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ for constituent RM codes of order $r_\lambda \geq 2$ as

$$M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right) = \sum_{i=i_{\min_\lambda}}^{i_{\max_\lambda}} \left|\alpha_{s-1,i}^{(\lambda)}\right|. \tag{7.3}$$

Algorithm 13: Improved RLD with SP of RM Codes

Input : $\boldsymbol{y}$
Output: $\hat{\boldsymbol{x}}$

/* Initialize L random codeword permutations associated with L decoding paths */
for $l \leftarrow 0$ **to** $L-1$ **do**
  $\pi_{\text{tmp}} : \boldsymbol{y} \xrightarrow{\pi_{\text{tmp}}} \boldsymbol{y}_{\pi_l}$
  $\mathcal{Q}_{(r,m)}[l].\pi_{\text{init}} \leftarrow \pi_{\text{tmp}}$; $\mathcal{Q}_{(r,m)}[l].\boldsymbol{\alpha} \leftarrow \boldsymbol{y}_{\pi_l}$
  $\mathcal{Q}_{(r,m)}[l].\text{PM} \leftarrow 0$; $\mathcal{Q}_{(r,m)}[l].\hat{\boldsymbol{x}} \leftarrow \mathbf{0}$

/* Improved RLD with SP */
$\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1] \leftarrow \texttt{SP-RLD}\left(\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]\right)$

/* Selection of the best decoding path */
$l^* \leftarrow \arg\min_{0 \le l \le L-1} \{\mathcal{Q}_{(r,m)}[l].\text{PM}\}$
$\pi^*_{\text{init}} \leftarrow \mathcal{Q}_{(r,m)}[l^*].\pi_{\text{init}}$
$\left(\pi^*_{\text{init}}\right)^{-1} : \mathcal{Q}_{(r,m)}[l^*].\hat{\boldsymbol{x}} \xrightarrow{\left(\pi^*_{\text{init}}\right)^{-1}} \hat{\boldsymbol{x}}$
return $\hat{\boldsymbol{x}}$

7.2 Improved Recursive List Decoding with Successive Permutation

We now propose an improved RLD algorithm that utilizes the SP scheme introduced in Section 7.1. The details of the proposed algorithm are provided in Algorithm 13. In the beginning of Algorithm 13, the proposed algorithm with list size L initializes all the L decoding paths with L random codeword permutations sampled from the symmetry group of RM codes. Each decoding path with index l ($0 \le l < L$) associated with $\mathcal{RM}(r,m)$ is characterized by a data structure $\mathcal{Q}_{(r,m)}$ that stores the channel LLR vector $\boldsymbol{\alpha}$, the path metric PM, the estimated codeword $\hat{\boldsymbol{x}}$, and the initial codeword permutation π_{init}. The permutation π_{tmp}, assigned to π_{init} for each decoding path, is a random codeword permutation sampled from the full symmetry group of RM codes. The recursive decoding algorithm utilizing the SP scheme, denoted as the $\texttt{SP-RLD}(\cdot)$ function, is then applied to the initialized data structures $\{\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]\}$, and returns the updated data structures of the L best decoding paths. Next, the updated path metrics of all the estimated decoding paths given by the $\texttt{SP-RLD}(\cdot)$ function are used to identify the best decoding path with index l^* that has the smallest path metric. The initial codeword permutation π^*_{init} associated with the best decoding path l^* is then obtained. Finally, an inverted permutation $\left(\pi^*_{\text{init}}\right)^{-1}$ is applied to the best candidate codeword $\mathcal{Q}_{(r,m)}[l^*].\hat{\boldsymbol{x}}$ to obtain the final estimated codeword $\hat{\boldsymbol{x}}$.

Algorithm 14: SP-RLD($\cdot$)

Input : $\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]$
Output: $\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]$

if $r = 1$ **then**
  $\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1] \leftarrow$ FHT-List$\left(\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]\right)$
else if $r = m - 1$ **then**
  $\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1] \leftarrow$ SPC-List$\left(\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]\right)$
else
  /* Decode the left-child node with SP */
  for $l \leftarrow 0$ **to** $L-1$ **do**
    $\mathcal{Q}_{(r-1,m-1)}[l].\pi_{\text{init}} \leftarrow \mathcal{Q}_{(r,m)}[l].\pi_{\text{init}}$
    $\mathcal{Q}_{(r-1,m-1)}[l].\text{PM} \leftarrow \mathcal{Q}_{(r,m)}[l].\text{PM}$
    $M^* \leftarrow -\infty$
    for $p \leftarrow 0$ **to** $m-1$ **do**
      $\pi_{\text{tmp}} : \mathcal{Q}_{(r,m)}[l].\boldsymbol{\alpha} \xrightarrow{\pi_{\text{tmp}}} \boldsymbol{\alpha}_{\text{tmp}}$
      $\boldsymbol{\alpha}^{(\lambda)} \leftarrow f(\boldsymbol{\alpha}_{\text{tmp}})$
      if $r = 2$ **then**
        Compute $M_{\pi_{\text{tmp}}}\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ using FHT
      else if $r > 2$ **then**
        Compute $M_{\pi_{\text{tmp}}}\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ using (7.3)
      if $M_{\pi_{tmp}}\left(\boldsymbol{\alpha}^{(\lambda)}\right) > M^*$ **then**
        $\mathcal{Q}_{(r-1,m-1)}[l].\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}^{(\lambda)}$
        $\mathcal{Q}_{(r,m)}[l].\pi_{\text{SP}} \leftarrow \pi_{\text{tmp}}$
        $M^* \leftarrow M_{\pi_{\text{tmp}}}\left(\boldsymbol{\alpha}^{(\lambda)}\right)$
  $\mathcal{Q}_{(r-1,m-1)}[0], \ldots, \mathcal{Q}_{(r-1,m-1)}[L-1] \leftarrow$ SP-RLD$\left(\mathcal{Q}_{(r-1,m-1)}[0], \ldots, \mathcal{Q}_{(r-1,m-1)}[L-1]\right)$
  /* Decode the right-child node */
  for $l \leftarrow 0$ **to** $L-1$ **do**
    $\mathcal{Q}_{(r,m-1)}[l].\pi_{\text{init}} \leftarrow \mathcal{Q}_{(r-1,m-1)}[l].\pi_{\text{init}}$
    $\mathcal{Q}_{(r,m-1)}[l].\text{PM} \leftarrow \mathcal{Q}_{(r-1,m-1)}[l].\text{PM}$
    $\mathcal{Q}_{(r,m-1)}[l].\boldsymbol{\alpha} \leftarrow g\left(\mathcal{Q}_{(r-1,m-1)}[l].\hat{\boldsymbol{x}}, \mathcal{Q}_{(r,m)}[l_{\text{org}}^{(r,m)}].\boldsymbol{\alpha}\right)$
  $\mathcal{Q}_{(r,m-1)}[0], \ldots, \mathcal{Q}_{(r,m-1)}[L-1] \leftarrow$ SP-RLD$\left(\mathcal{Q}_{(r,m-1)}[0], \ldots, \mathcal{Q}_{(r,m-1)}[L-1]\right)$
  /* Repermute the decoded codewords */
  for $l \leftarrow 0$ **to** $L-1$ **do**
    $\mathcal{Q}_{(r,m)}^{\text{tmp}}[l].\pi_{\text{init}} \leftarrow \mathcal{Q}_{(r,m-1)}[l].\pi_{\text{init}}$
    $\mathcal{Q}_{(r,m)}^{\text{tmp}}[l].\text{PM} \leftarrow \mathcal{Q}_{(r,m-1)}[l].\text{PM}$
    $\mathcal{Q}_{(r,m)}^{\text{tmp}}[l].\hat{\boldsymbol{x}} \leftarrow \text{Concat}(\mathcal{Q}_{(r-1,m-1)}[l_{\text{org}}^{(r-1,m-1)}].\hat{\boldsymbol{x}},$
      $\mathcal{Q}_{(r-1,m-1)}[l_{\text{org}}^{(r-1,m-1)}].\hat{\boldsymbol{x}} \oplus \mathcal{Q}_{(r,m-1)}[l].\hat{\boldsymbol{x}})$
    $\pi_{\text{SP}} \leftarrow \mathcal{Q}_{(r,m)}[l_{\text{org}}^{(r,m)}].\pi_{\text{SP}}$
    $(\pi_{\text{SP}})^{-1} : \mathcal{Q}_{(r,m)}^{\text{tmp}}[l].\hat{\boldsymbol{x}} \xrightarrow{(\pi_{\text{SP}})^{-1}} \mathcal{Q}_{(r,m)}^{\text{tmp}}[l].\hat{\boldsymbol{x}}$
  for $l \leftarrow 0$ **to** $L-1$ **do**
    $\mathcal{Q}_{(r,m)}[l] \leftarrow \mathcal{Q}_{(r,m)}^{\text{tmp}}[l]$
return $\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]$

In Algorithm 14, we provide the details of the SP-RLD(·) function. If the constituent RM codes are of order 1 or $m-1$, the ML decoders of the first-order RM codes (FHT-List(·) [20]) or that of the SPC codes (SPC-List(·) [3]) is queried to obtain the estimated codewords of the best L decoding paths and their path metrics, respectively. Note that as the FHT-List(·) [20] and SPC-List(·) functions perform ML decoding at the parent node level, no codeword permutation is required to obtain the optimal decoding outputs of the L best decoding paths. On the other hand, if the constituent code $\mathcal{RM}(r,m)$ satisfies $1 < r < m-1$, the SP-RLD(·) function is recursively queried to decode the left and right child nodes $\mathcal{RM}(r-1,m-1)$ and $\mathcal{RM}(r,m-1)$ of $\mathcal{RM}(r,m)$, respectively, whose decoding results are used to construct the decoding output of $\mathcal{RM}(r,m)$. Specifically, from line 6 to line 20 of Algorithm 14, the best permutations of $\mathcal{RM}(r,m)$ are obtained independently for each decoding path with index l, using the proposed SP scheme. By π_{tmp}, we indicate a random permutation sampled from the full symmetry group of $\mathcal{RM}(r,m)$, which is used to obtain the LLR values associated with the right child $\mathcal{RM}(r-1,m-1)$, i.e., $\boldsymbol{\alpha}^{(\lambda)}$, followed by the permutation metric computation specified in Section 7.1. If a better permutation π_{tmp} is found for the l-th decoding path, the selected permutation π_{SP} is updated in the data structure $\mathcal{Q}_{(r,m)}[l]$, which is required to perform the inverted permutation after the right-child node $\mathcal{RM}(r,m-1)$ is decoded. In addition, the data structures of the left-child nodes $\mathcal{Q}_{(r-1,m-1)}$ are also initialized during the permutation selection of $\mathcal{RM}(r,m)$. Given the initialized data structures of the left-child node $\mathcal{Q}_{(r-1,m-1)}$, the SP-RLD(·) function is then queried to obtain the updated data structures $\mathcal{Q}_{(r-1,m-1)}$ corresponding to the L best decoding paths of the left-child node.

The decoding of the right-child nodes $\mathcal{RM}(r,m-1)$ is specified in line 22 to line 26 of Algorithm 14. The $g(\cdot)$ functions are used to obtain the LLR values of the right-child nodes, given the hard estimations of the left-child node, i.e., $\mathcal{Q}_{(r-1,m-1)}[l].\hat{\boldsymbol{x}}$, and the corresponding LLR values of the parent node where the left-child node is originated from, i.e., $\mathcal{Q}_{(r,m)}[l_{\text{org}}^{(r,m)}].\boldsymbol{\alpha}$. Here, by $l_{\text{org}}^{(r,m)}$ we indicate the path index of the parent node from which the surviving left-child node is derived. After the decoding of the right-child nodes is finished, the estimated codewords of the L best paths associated with the parent node $\mathcal{RM}(r,m)$ are obtained based on the estimated hard values of the left-child and the right-child nodes (see lines 30 and 31), where $\texttt{Concat}(\boldsymbol{a},\boldsymbol{b})$ indicates the concatenation of the binary vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. In addition, $\mathcal{Q}_{(r-1,m-1)}[l_{\text{org}}^{(r-1,m-1)}].\hat{\boldsymbol{x}}$ indicates the hard values of the left-child node that corresponds to the l-th active decoding path $\mathcal{Q}_{(r,m-1)}[l].\hat{\boldsymbol{x}}$ of the right-child node. Next, $\mathcal{Q}_{(r,m)}[l_{\text{org}}^{(r,m)}].\pi_{\text{SP}}$, the codeword permutation previously selected for the parent node from which the l-th active decoding path of the right-child node is originated from, is used to re-permute the es-

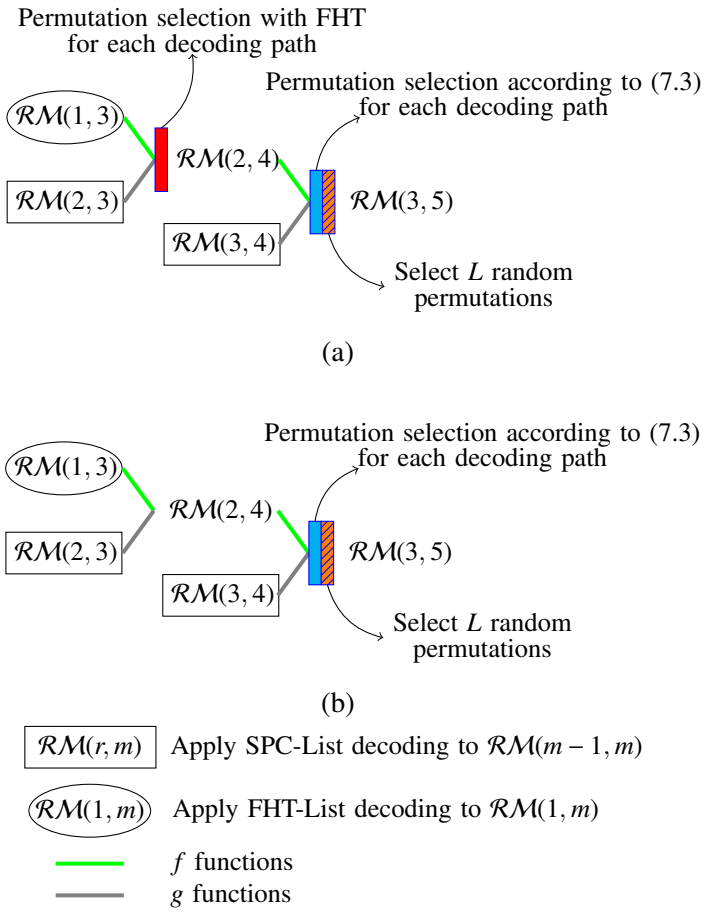


Figure 7.1: Examples of the proposed decoder when applied to $\mathcal{RM}(3,5)$ with (a) $S = 2$ and (b) $S = 1$.

timated codeword of the parent node. Finally, the data structures $\{\mathcal{Q}_{(r,m)}[0], \ldots, \mathcal{Q}_{(r,m)}[L-1]\}$ of the L best decoding paths for the parent node $\mathcal{RM}(r,m)$ are returned as the outputs of the recursive SP-RLD$(\cdot)$ function. Note that we keep track of the initial permutation π_{init} applied to the parent code for each active decoding path during the course of decoding.

As the constituent RM codes are decoded successively under the proposed decoder, we propose a complexity and decoding latency reduction scheme that only applies the SP operations for the first S ($S > 0$) left-child nodes. We refer to this simplified decoding algorithm as SSP-RLD. Fig. 7.1 illustrates an example of SSP-RLD decoding when applied to $\mathcal{RM}(3,5)$ using $S \in \{2,1\}$. In Fig. 7.1(a), since the left-child node of $\mathcal{RM}(3,5)$ is a RM code of order 2, the codeword permutation of $\mathcal{RM}(3,5)$ is selected in accordance with (7.3). Then, the LLR values associated with

$\mathcal{RM}(2,4)$ are obtained with the f functions. Since the left-child node of $\mathcal{RM}(2,4)$ is a first-order RM code, FHT decoding is used to select the best permutation for $\mathcal{RM}(2,4)$, followed by the FHT-List decoder applied on the list of L decoding paths with the selected codeword permutations for $\mathcal{RM}(2,4)$. Finally, as all the right-child RM codes are SPC codes, SPC-List decoding is used to decode them. The similar decoding operations are carried out for $\mathcal{RM}(2,5)$ in Fig. 7.1(b) except that the SP operations are only applied to the parent node of the first left-child node, while the original permutations are used for the parent node $\mathcal{RM}(2,4)$ of the second left-child node.

The computational complexity and the decoding latency of SP-RLD and SSP-RLD decoders significantly increase as the list size L increases. This is mainly caused by path metric and LLR sorting operations in the constituent `SPC-List`$(\cdot)$ and `FHT-List`$(\cdot)$ functions. Since random subsets of the full symmetry group of RM codes are utilized for the SSP-RLD decoder, we propose to further reduce the computational complexity and the decoding latency of the SSP-RLD decoder by using a variation of the ensemble decoding technique in [38]. In particular, we run T ($T \geq 1$) independent SSP-RLD decoders with a small list size L in parallel and select the output codeword that has the smallest path metric among the T resulting codewords from the T constituent SSP-RLD decoders. This variation of the proposed decoder is referred to as the ensemble SSP-RLD (Ens-SSP-RLD) decoding. Note that Ens-SSP-RLD decoding enables flexible design choices where the error-correction performance and complexity trade-offs can be explored with different choices of S, L, and T.

7.3 Performance Evaluation

7.3.1 Quantitative Complexity Analysis

In this chapter, we consider sequential and parallel implementations of the permutation selection scheme in Section 7.1. Under the sequential implementation of the proposed SP scheme, a similar memory consumption as SC-based decoders is required to store the internal LLR values [84]. On the other hand, under the parallel implementation, a memory of $mLNQ$ bits is required to store the internal LLR values, where m is the maximum number of the candidate permutations for the SP scheme, and Q is the number of quantization bits. In addition, the SP scheme in both the sequential and parallel implementations requires mQ memory bits to store the permutation metric $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$. Throughout this chapter, we use $Q = 32$ for all the considered decoders. Note that the FHT operations compute and store the new LLR values directly to $\boldsymbol{\alpha}^{(\lambda)}$, thus no extra memory is

Table 7.1: Memory requirement in terms of the number of bits required by the SP-RLD and SSP-RLD decoders.

Decoding Algorithm		Memory Requirement
SP-RLD-1 SSP-RLD-S-1	($L = 1$, sequential SP)	$2NQ + mQ + N$
SP-RLD-1 SSP-RLD-S-1	($L = 1$, parallel SP)	$(m + 1)NQ + mQ + N$
SP-RLD-L SSP-RLD-S-L	($L > 1$, sequential SP)	$N(L + 1)Q + mQ + 2NL$
SP-RLD-L SSP-RLD-S-L	($L > 1$, parallel SP)	$N(mL + 1)Q + mQ + 2NL$

Table 7.2: Memory requirement in terms of the number of bits required by the Ens-SSP-RLD decoder.

Decoding Algorithm		Memory Requirement
Ens-SSP-RLD-S-1-T	($L = 1$, sequential SP)	$(NQ + mQ + N)T + NQ$
Ens-SSP-RLD-S-1-T	($L = 1$, parallel SP)	$(mNQ + mQ + N)T + NQ$
Ens-SSP-RLD-S-L-T	($L > 1$, sequential SP)	$(NLQ + mQ + 2NL)T + NQ$
Ens-SSP-RLD-S-L-T	($L > 1$, parallel SP)	$(mNLQ + mQ + 2NL)T + NQ$

needed under FHT decoding. Table 7.1 summarizes the memory requirements of the SP-RLD-L ($L \geq 1$) and SSP-RLD-S-L ($S > 0$) decoders, while Table 7.2 provides the memory consumption of the Ens-SSP-RLD-S-L-T ($T \geq 1$) decoder. Note that with $T = 1$, Ens-SSP-RLD-S-L-T reverts to SSP-RLD-S-L. In addition, with S being the number of left-child nodes visited following the course of decoding and with $T = 1$, Ens-SSP-RLD-S-L-T reverts to SP-RLD-L.

The decoding latency of the proposed decoders is computed by counting the number of time steps required by all the floating point operations. We assume that there is no resource constraint. Thus, all concurrent operations in $f(\cdot)$ and $g(\cdot)$ functions, the path metric computations, and the computations of $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ in Section 7.1 require one time step [3, 26]. The permutation metric $M_\pi\left(\boldsymbol{\alpha}^{(\lambda)}\right)$ obtained from FHT requires s time steps for a first-order RM code located at the s-th stage [87, Chapter 14]. Furthermore, for the sequential implementation, the proposed SP scheme requires s time steps if $r_\lambda \geq 2$ and s^2 time steps if $r_\lambda = 1$, since each permutation is evaluated

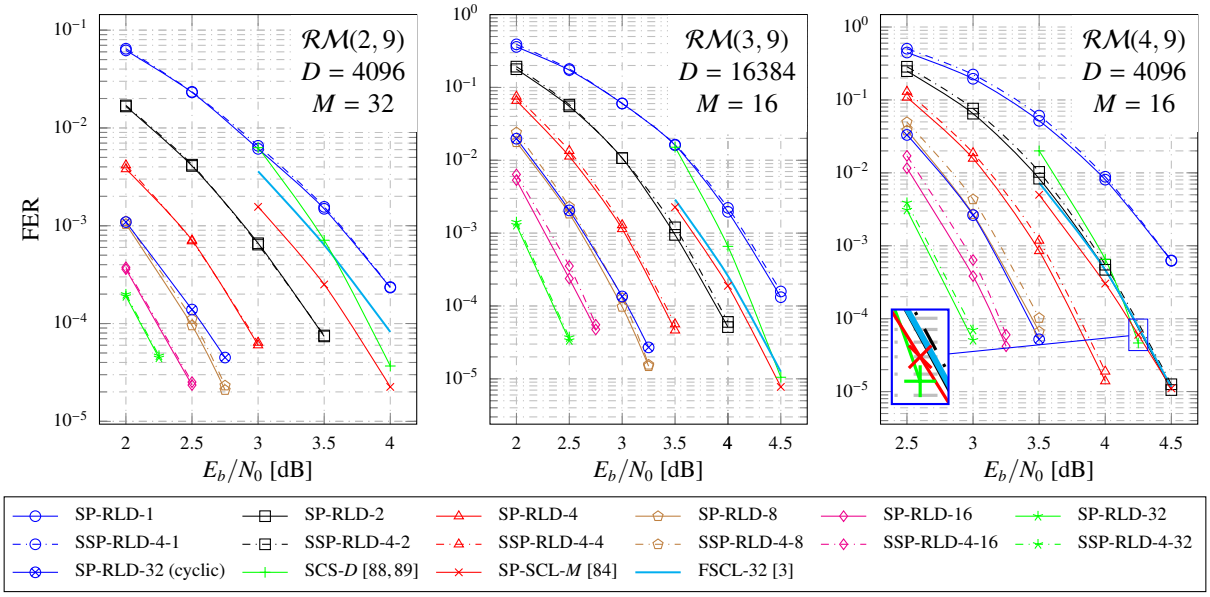


Figure 7.2: Error-correction performance of the proposed decoders and that of the SCS, SP-SCL, and FSCL decoders.

sequentially. In the parallel implementation the proposed SP scheme, a single time step is required if $r_\lambda \geq 2$ and s time steps are required if $r_\lambda = 1$. In addition, the hard decisions obtained from the LLR values and binary operations are computed instantaneously [1, 3, 26]. Finally, we assume that the number of time steps required by a merge sort algorithm to sort an array of N elements is $\log_2(N)$ [76, Chapter 2]. We also use similar assumptions to compute the decoding latency of all the other decoders considered in this chapter.

To calculate the computational complexity of the decoders considered in this chapter, we count the number of floating point operations, namely, the number of additions, subtractions, and comparisons, required during the course of decoding. Note that the merge sort algorithm requires $N \log_2 N$ comparisons to sort an array of length N [76, Chapter 2].

7.3.2 Comparison with FSCL, SC-Stack and SP-SCL Decoding Algorithms

Fig. 7.2 provides the error-correction performance in terms of FER of the proposed SP-RLD-L and SSP-RLD-S-L decoders, and that of the FSCL, SC-Stack (SCS), and SP-SCL decoders for $\mathcal{RM}(r, 9)$, $r \in \{2, 3, 4\}$. The SCS decoder considered in this chapter utilizes the enhanced score function introduced in [89] to reduce the stack size when compared with the conventional SCS

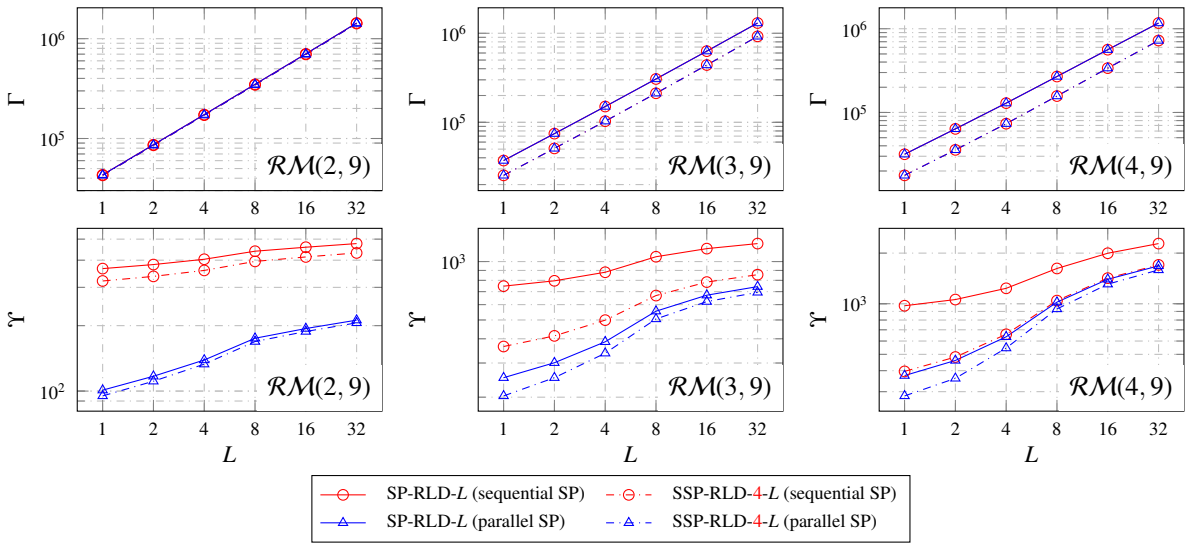


Figure 7.3: Computational complexity and decoding latency of the proposed decoders under the sequential and parallel implementations of the SP scheme.

decoder introduced in [88]. In Fig. 7.2, the SCS decoder with stack size D is denoted as SCS-D, while the SP-SCL decoder with list size M is denoted as SP-SCL-M. The values of D and M are selected to allow an FER performance comparable to that of the FSCL decoder with list size 32 (FSCL-32). With $S = 4$, the SSP-RLD decoder has a negligible error-correction performance degradation when compared to the SP-RLD decoder with the same list size. Furthermore, we also provide the FER performance of the proposed SP-RLD-32 decoder where only cyclic factor-graph permutations are considered.

It can be observed from Fig. 7.2 that the FER of the SP-RLD-32 decoder with cyclic factor-graph permutations is relatively similar to that of the SP-RLD-8 decoder with the codeword permutations sampled from the full symmetry group. In addition, at no additional cost, the SP-RLD-32 decoder that utilizes the general codeword permutations obtains a maximum gain of 0.7 dB at the target FER of 10^{-4}, when compared to the SP-RLD-32 decoder that only uses cyclic factor-graph permutations.

Fig. 7.3 illustrates the computational complexity (Γ) and the decoding latency (Υ) of SP-RLD-L and SSP-RLD-4-L under the sequential and parallel implementations of the proposed SP scheme. In addition, the memory requirement (Φ) in kilobytes (kB) of SP-RLD-L and SSP-RLD-4-L is provided in Fig. 7.4. It can be observed from Fig. 7.3 that the SSP-RLD-4-L decoder relatively maintains the computational complexity when compared with SP-RLD-L. However, SSP-RLD-4-

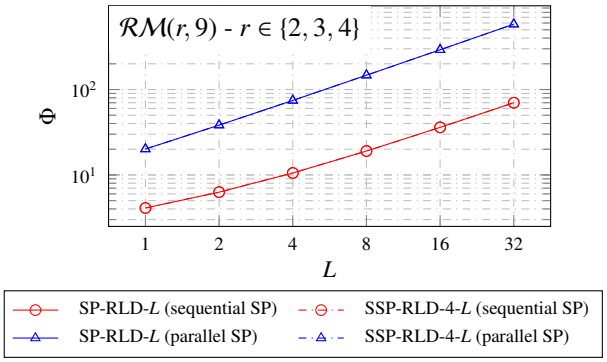


Figure 7.4: Memory consumption in kB (Φ) of the proposed decoders whose FER curves are provided in Fig. 7.2.

L significantly reduces the decoding latency of SP-RLD-L while only incurring negligible error-correction performance degradation as seen from Fig. 7.2. Furthermore, Fig. 7.3 and Fig. 7.4 reveal the trade-offs between the decoding latency and memory requirement of the proposed decoders under the sequential and parallel implementations of the SP scheme. In particular, the improvements in the decoding latency of the parallel implementation over the sequential implementation come at the cost of memory consumption overheads.

Table 7.3: Comparison of computational complexity (Γ), decoding latency in time steps (Υ), and memory requirement in kB (Φ) of SCS, SP-SCL, FSCL, and proposed SSP-RLD decoders considered in Fig. 7.2.

	SCS-D [88, 89]				SP-SCL-M [84]				FSCL-32 [3]			SSP-RLD-4-2				
	D	Γ	Υ	Φ	M	Γ	Υ	Φ	Γ	Υ	Φ	Γ	Υ_s	Φ_s	Υ_p	Φ_p
$\mathcal{RM}(2, 9)$	4096	4.21×10^7	3.0×10^5	8208	32	8.52×10^5	1.8×10^3	70	9.62×10^4	373	70	8.33×10^4	337	6.3	111	38.3
$\mathcal{RM}(3, 9)$	16384	4.83×10^6	1.0×10^4	32832	16	4.37×10^5	2.3×10^3	36	1.64×10^5	1039	70	4.86×10^4	414	6.3	252	38.3
$\mathcal{RM}(4, 9)$	8192	1.13×10^7	4.2×10^4	16416	16	4.58×10^5	3.1×10^3	36	2.25×10^5	1991	70	3.41×10^4	482	6.3	369	38.3

Table 7.3 summarizes the computational complexity (Γ), the decoding latency in time steps (Υ), and the memory requirement in kB (Φ) of the FSCL, SCS, and SP-SCL decoders, and those of the SSP-RLD decoder with $L = 2$ and $S = 4$, whose FER values are plotted in Fig. 7.2. For the SSP-RLD-4-2 decoder, Υ_s and Φ_s indicate the decoding latency and the memory requirement of the sequential SP implementation, while Υ_p and Φ_p indicate the decoding latency and the memory requirement of the parallel implementation of the SP scheme, respectively. It can be seen in Fig. 7.2 that the FER performance of SSP-RLD-4-2 is similar to or better than that of FSCL, SCS, and SP-

SCL decoders at the target FER of 10^{-4} for all the considered RM codes. In addition, under both sequential and parallel implementations of the SP scheme, SSP-RLD-4-2 significantly outperforms the FSCL, SCS, and SP-SCL decoders in various complexity metrics as shown in Table 7.3.

7.3.3 Comparison with State-of-the-Art RM Decoders

Fig. 7.5 compares the FER performance of SSP-RLD-S-L, Ens-SSP-RLD-S-L'-T, and that of the state-of-the-art decoders for various RM codes. Note that the list size L' and the number of decoding attempts T used by the Ens-SSP-RLD decoder satisfy the constraint $L = L'T$, where L is the list size used by the SSP-RLD decoder. L' is selected as the smallest list size that allows the Ens-SSP-RLD decoder to have an error-correction performance that is within 0.1 dB of that of the SSP-RLD decoder at the target FER of 10^{-3}. We consider the RLDP [19] and the RLDA [19, 38] algorithms with list size M, the SSC-FHT decoder [20, 24] when applied to P factor-graph permutations (Per-SSC-FHT-P), and P general permutations (Aut-SSC-FHT-P) sampled from the full symmetry group of RM codes. The empirical lower bounds of the error-correction performance of ML decoding [19] are also provided for all the RM configurations in Fig. 7.5. In addition, the FER performance curves of the sparse-RPA (SRPA) decoder introduced in [86] are shown for $\mathcal{RM}(2, 8)$, $\mathcal{RM}(3, 8)$, $\mathcal{RM}(4, 8)$, and $\mathcal{RM}(2, 9)$.

Table 7.4 summarizes the computational complexity, the decoding latency, and the memory requirement of SSP-RLD and Ens-SSP-RLD decoding, while Table 7.5 provides the complexity metrics of SRPA, Aut-SSC-FHT, and RLDA decoding. Note that the decoders provided in Table 7.4 and Table 7.5 have similar error-correction performance at the target FER of 10^{-3} as shown in Fig. 7.5. In this chapter, a fully-parallel implementation of the SRPA decoder, in which all the operations that can be carried out concurrently are executed at the same time, is considered. The SRPA decoding algorithm runs two fully-parallel RPA decoders with each decoder using a quarter of the code projections at each recursion step [86]. Thus, the SRPA decoder effectively reduces 50% of the total number of projections used by the conventional RPA algorithm [39]. This configuration incurs negligible error-correction performance loss with respect to the conventional RPA decoder in [39] for the second and third order RM codes of size 256.

In Table 7.5, we consider fully-parallel and semi-parallel implementations of the Aut-SSC-FHT decoder. Under the semi-parallel implementation, the number of parallel SSC-FHT decoders is set to the list size L used by SSP-RLD decoding for the same RM code. This configuration enables the Aut-SSC-FHT-P decoder to have a relatively similar memory consumption in comparison with

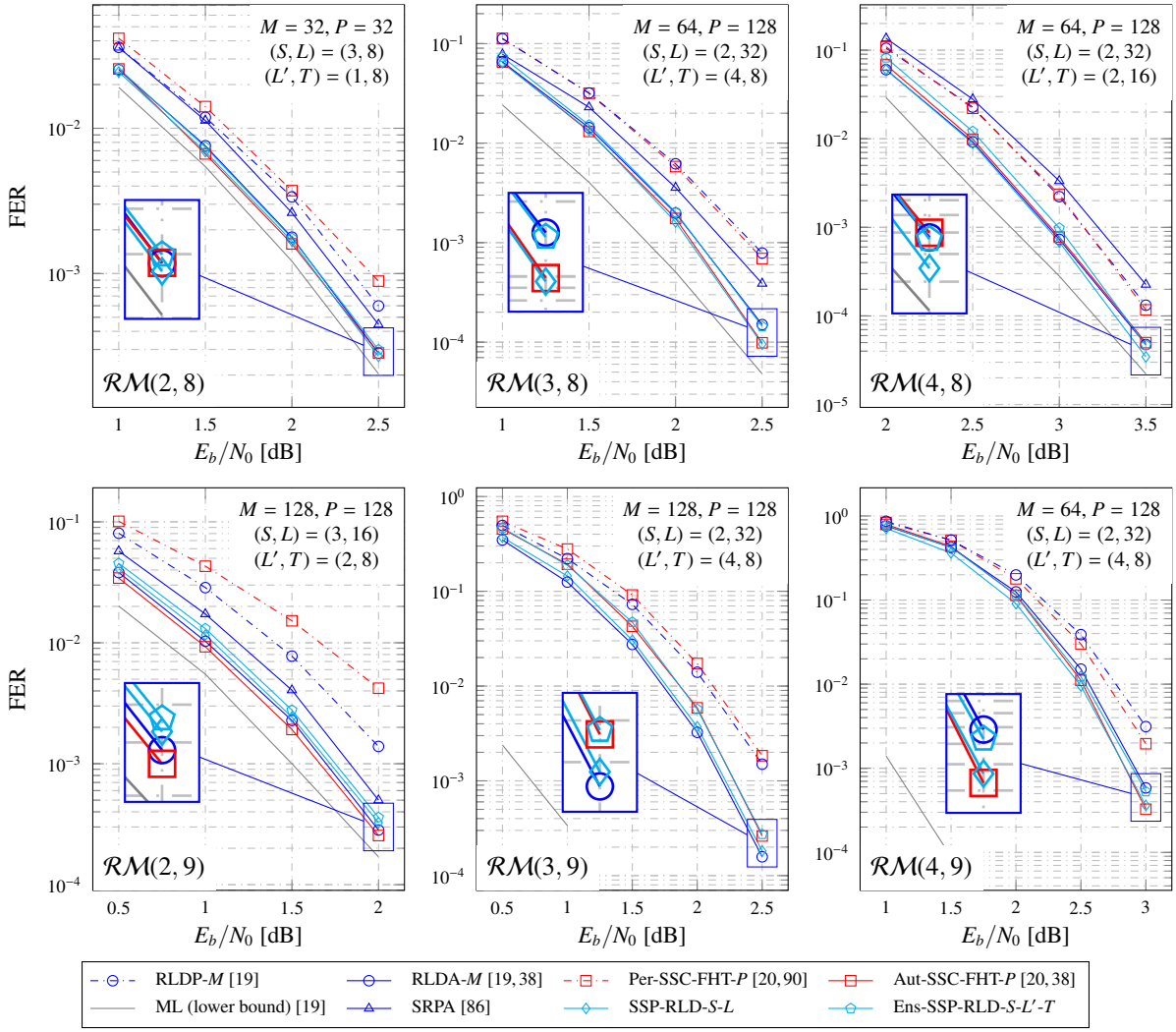


Figure 7.5: Error-correction performance of various permutation decoding algorithms of RM codes. The FER of the SRPA decoder and the lower bound of ML decoding are also plotted for comparison.

SSP-RLD-S-L and Ens-SSP-RLD-S-L'-T decoding when the sequential SP scheme is used. On the other hand, in the fully-parallel implementation of Aut-SSC-FHT decoding, P concurrent SSC-FHT decoders are used. For Aut-SSC-FHT decoding, Υ_{sp} and Φ_{sp} indicate the latency and memory requirement of the semi-parallel implementation, while Υ_{p} and Φ_{p} indicate the decoding latency and memory requirement of the fully-parallel implementation, respectively.

Table 7.4: Computational complexity (Γ), decoding latency in time steps (Υ), and memory requirement in kB (Φ) of the SSP-RLD and Ens-SSP-RLD decoders considered in Fig. 7.5.

	SSP-RLD-S-L							Ens-SSP-RLD-S-L'-T							
	S	L	Γ	Υ_{s}	Φ_{s}	Υ_{p}	Φ_{p}	S	L'	T	Γ	Υ_{s}	Φ_{s}	Υ_{p}	Φ_{p}
$\mathcal{RM}(2,8)$	3	8	1.32×10^5	287	9.5	141	65.5	3	1	8	1.21×10^5	222	9.5	76	65.5
$\mathcal{RM}(3,8)$	2	32	3.39×10^5	588	35.0	526	259.0	2	4	8	2.72×10^5	298	35.3	236	259.3
$\mathcal{RM}(4,8)$	2	32	2.63×10^5	1092	35.0	1066	259.0	2	2	16	1.77×10^5	231	35.5	205	259.5
$\mathcal{RM}(2,9)$	3	16	6.60×10^5	376	36.0	185	292.0	3	2	8	6.48×10^5	302	36.3	111	292.3
$\mathcal{RM}(3,9)$	2	32	7.55×10^5	767	70.0	688	582.0	2	4	8	6.48×10^5	413	70.3	334	582.3
$\mathcal{RM}(4,9)$	2	32	5.66×10^5	1613	70.0	1583	582.0	2	4	8	4.23×10^5	572	70.3	542	582.3

Table 7.5: Computational complexity (Γ), decoding latency in time steps (Υ), and memory requirement in kB (Φ) of the SPRA, RLDA, and Aut-SSC-FHT decoders considered in Fig. 7.5.

	SRPA [86]			RLDA-M [19, 38]				Aut-SSC-FHT-P [20, 38]					
	Γ	Υ	Φ	M	Γ	Υ	Φ	P	Γ	Υ_{sp}	Φ_{sp}	Υ_{p}	Φ_{p}
$\mathcal{RM}(2,8)$	6.55×10^5	3592	69.2	32	6.20×10^4	317	35.0	32	8.12×10^4	261	9.4	69	34.1
$\mathcal{RM}(3,8)$	7.92×10^7	6184	281.5	64	2.04×10^5	854	69.0	128	3.07×10^5	507	34.5	132	133.5
$\mathcal{RM}(4,8)$	3.63×10^9	7816	465.2	64	2.83×10^5	1433	69.0	128	2.64×10^5	583	34.5	151	133.5
$\mathcal{RM}(2,9)$	3.44×10^6	10250	271.6	128	4.82×10^5	490	274.0	128	7.18×10^5	663	35.5	89	266.5
$\mathcal{RM}(3,9)$	-	-	-	128	7.64×10^5	1336	274.0	128	6.97×10^5	767	68.5	197	266.5
$\mathcal{RM}(4,9)$	-	-	-	64	5.20×10^5	2283	138.0	128	6.37×10^5	1087	68.5	277	266.5

It can be observed in Table 7.4 that Ens-SSP-RLD significantly reduces the computational complexity and decoding latency of SSP-RLD, especially for $r \in \{3, 4\}$, while relatively preserving the error-correction performance and memory requirement of SSP-RLD decoding. In comparison with the semi-parallel implementation of Aut-SSC-FHT decoding, except for the case of $\mathcal{RM}(2, 8)$, significant improvements in the computational complexity and latency of Ens-SSP-RLD under the

sequential SP scheme are recorded, at the cost of negligible error-correction performance loss and memory consumption overheads. For instance, with relatively similar FER performance and memory consumption, Ens-SSP-RLD-2-4-8 reduces 34% of the computational complexity and 47% of the decoding latency of Aut-SSC-FHT-128 for $\mathcal{RM}(4, 9)$.

Under the fully-parallel implementation, Aut-SSC-FHT provides the best decoding latency in comparison with the RLDA, SP-RLD, and Ens-SSP-RLD decoders. Specifically, under the parallel implementation of the proposed SP scheme and in comparison with the fully-parallel implementation of Aut-SSC-FHT decoding for $\mathcal{RM}(2, 9)$, Ens-SSP-RLD-3-2-8 reduces 10% of the decoding complexity of Aut-SSC-FHT-128 at the cost of 10% memory requirement overhead and 1.25× increase in the number of decoding time steps. In addition, RLDA suffers from high computational complexity and high decoding latency that are mainly caused by sorting operations, especially with large values of M and r. In particular, for $\mathcal{RM}(4, 9)$ and with relatively similar FER performance, the Ens-SSP-RLD-2-4-8 decoder with sequential SP scheme reduces 19% of the computational complexity, 75% of the number of time steps, and 49% of the memory consumption compared to RLDA-64.

It can be seen in Fig. 7.5 that with the same list size M or the same number of permutations P, using the permutations randomly sampled from the full symmetry group of the codes provides significant error-correction performance improvement for RLDA-M and Aut-SSC-FHT-P decoders at no additional cost, compared to the RLDP-M and Per-SSC-FHT-P decoders, respectively. In addition, as observed from Fig. 7.5, Table 7.4, and Table 7.5, all permutation decoding algorithms, RLDA, Aut-SSC-FHT, SSP-RLD, and Ens-SSP-RLD, provide significantly better error-correction performance with significantly lower computational complexity and decoding latency compared to the SRPA decoder for various RM code configurations.

7.4 Chapter Conclusion

In this chapter, a novel successive permutation (SP) scheme is proposed to significantly improve the error-correction performance of Reed-Muller (RM) codes under an improved recursive list decoding (RLD) algorithm. We performed low-complexity decoding operations on the rich symmetry group of RM codes to select a good codeword permutation of the code on the fly. Efficient decoding latency and complexity reduction schemes were introduced that relatively maintain the error-correction performance. We performed a numerical analysis of the proposed decoders in

terms of error-correction performance, computational complexity, decoding latency, and memory requirement and compared them with those of the state-of-the-art RM decoders. The simulation results confirmed the effectiveness of the proposed decoder under various configurations of RM codes. Specifically, for the RM codes of lengths 256 and 512 and with code orders 3 and 4, the proposed decoder significantly reduces the computational complexity and the decoding latency of the state-of-the-art permuted successive-cancellation decoder with fast Hadamard transform (Aut-SSC-FHT), while relatively preserving the error-correction performance and memory requirement of Aut-SSC-FHT decoding.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

The deployment of 5G technologies has significantly stimulated the research topics that deal with high-performance and low-complexity decoding algorithms of short linear block codes. Recently, polar codes were selected the coding scheme in the eMBB and URLLC scenarios of the 5G standard, and they are being considered for the mMTC use case. In this thesis, we first introduced high-performance and low-complexity bit-flipping decoding algorithms that can achieve a similar error-correction performance of the state-of-the-art SCL decoding algorithm with list size 32, while negligibly increasing the average decoding latency, average computational complexity, and memory requirement of the SCL decoder with a small list size of 4. The advantages of the proposed bit-flipping decoders allow for an energy-efficient and high-reliability decoding algorithm of polar codes, making them suitable for the mMTC use case of the 5G standard. We then addressed the poor error-correction performance of BP decoding when applied to CRC-polar concatenated codes by introducing novel decoding techniques that utilize the CRC factor graph and the polar code permutations. The proposed decoding techniques tailored to BP decoding of CRC-polar concatenated codes are beneficial for communication systems that require soft-input soft-output decoder as part of a Turbo channel equalizer. Finally, when considering RM codes of short lengths and low rates, we proposed novel permutation decoding algorithms which yield near ML decoding performance, while providing more efficient error-correction performance and computational complexities trade-offs compared to state-of-the-art RM decoders.

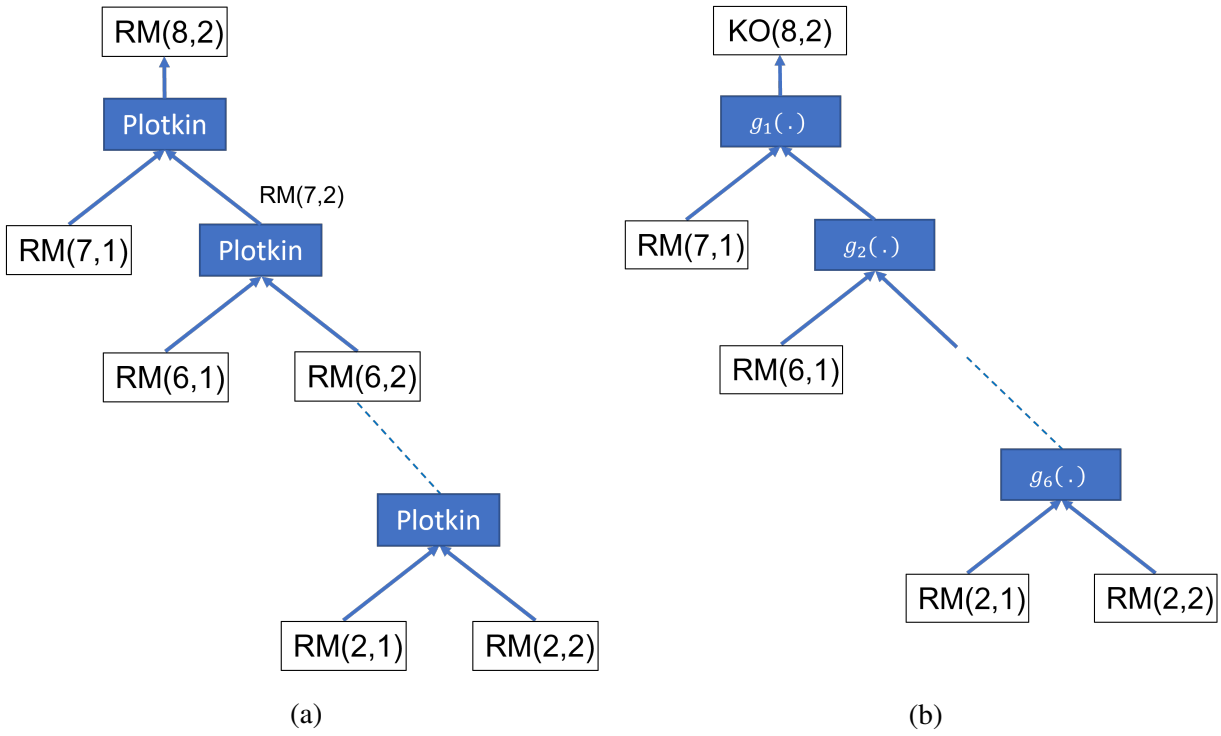


Figure 8.1: Encoding of (a) RM and (b) KO codes [2].

8.2 Future Work

Future research directions considering the decoding algorithms of polar and RM codes introduced in this thesis include designing hardware architectures and evaluating their performances, which are an important step to clearly examine the practicability of the proposed algorithms. In addition, analytical approaches that define theoretical bounds on the error-correction performance and decoding latency of the proposed decoders are also a potential research direction, which further complete the research initialized by the thesis.

It is worth to note that next-generation wireless systems such as 6G and beyond require more stringent requirements than 5G in terms of reliability and decoding latency. The remainder of this chapter discusses potential research directions for a new line of future work, focusing on a recently proposed coding scheme based on Deep Learning (DL), named Kronecker Operations (KO) codes, which was recently shown to outperform the conventional RM coding scheme.

KO codes are recently introduced in [2] which contain a pair of neural encoder and neural decoder optimized simultaneously under the autoencoder system modeling. Similar to RM codes, a

KO code is characterized by the code parameters (m, r), where $N = 2^m$ and $K = \sum_{i=0}^{r} \binom{r}{i}$. Given the code parameters (m, r), the RM and KO codes are denoted as RM(m, r) and KO(m, r), respectively, and $\boldsymbol{m}_{m,r}$ indicates a valid messageword of RM(m, r). To overcome the curse of dimensionality, i.e., with large value of K, KO codes are constructed using the recursive Plotkin representation similar to RM and polar codes [2]. Fig. 8.1 compares the encoding function of RM and KO codes for $m = 8$ and $r = 2$, respectively. It can be seen from Fig. 8.1 that KO codes use the same messages $\boldsymbol{m}_{m,r}$ similar to RM codes and directly learn a waveform representation $\boldsymbol{x}$ through the neural encoding functions $g(\cdot)$'s. Each different $g(\cdot)$ function is specified by a different set of trainable parameters. In terms of decoding, both RM and KO codes employ a successive cancellation decoding strategy, which is depicted in Fig. 8.2. Note that the log-sum-exponential (LSE) transform functions used in RM decoders are often approximated by the low complexity min-sum operations with negligible error-correction performance loss, whereas under KO codes, the LSE functions are replaced by the neural decoder functions $f(\cdot)$'s. It is worth to note that both RM and KO codes utilize the first-order RM codes RM$(m, 1)$, thus maximum-a-posteriori (MAP) or equivalently ML decoding algorithms are used to decode the first-order RM codes for both RM and KO codes. It was shown in [2] that KO codes outperform RM codes under the conventional fast Hadamard transform (FHT) and successive cancellation (SC) decoding thanks to the direct mapping from the message word $\boldsymbol{m}$ to the waveform representation $\boldsymbol{x}$ of the codes and the nonlinearities introduced by the neural encoder and decoder functions. However, KO codes suffer from various issues that prevent them to be practical. In this section, we state some research problems associated with KO codes and their potential solutions.

High computational complexity and memory consumption

The current soft-MAP decoding function used by KO codes requires the likelihood evaluation of all the possible codewords for RM$(m, 1)$, which contain 2^{m+1} codewords. Therefore, with a large value of m, the soft-MAP functions require a significant amount of computational complexity, i.e., $O\left(2^{(2m+1)}\right)$. On the other hand, FHT decoding is a low-complexity MAP (ML) decoding of the RM$(m, 1)$ codes with a complexity of $O(m2^m)$, however, it only outputs the hard estimate $\hat{\boldsymbol{m}}$ of $\boldsymbol{m}$. Therefore, we propose to use FHT decoding and belief propagation (BP) decoding to provide the soft estimate of $\boldsymbol{m}$. In particular, FHT decoding is first utilized to obtain the most likely codeword $\hat{\boldsymbol{m}}_{(\mathrm{m},1)}$ given $\boldsymbol{y}_{(\mathrm{m},1)}$, which is then used to initialized the posterior belief messages of BP decoding at the leaf-node level of the RM$(m, 1)$ code. The message passing operations of BP decoding are

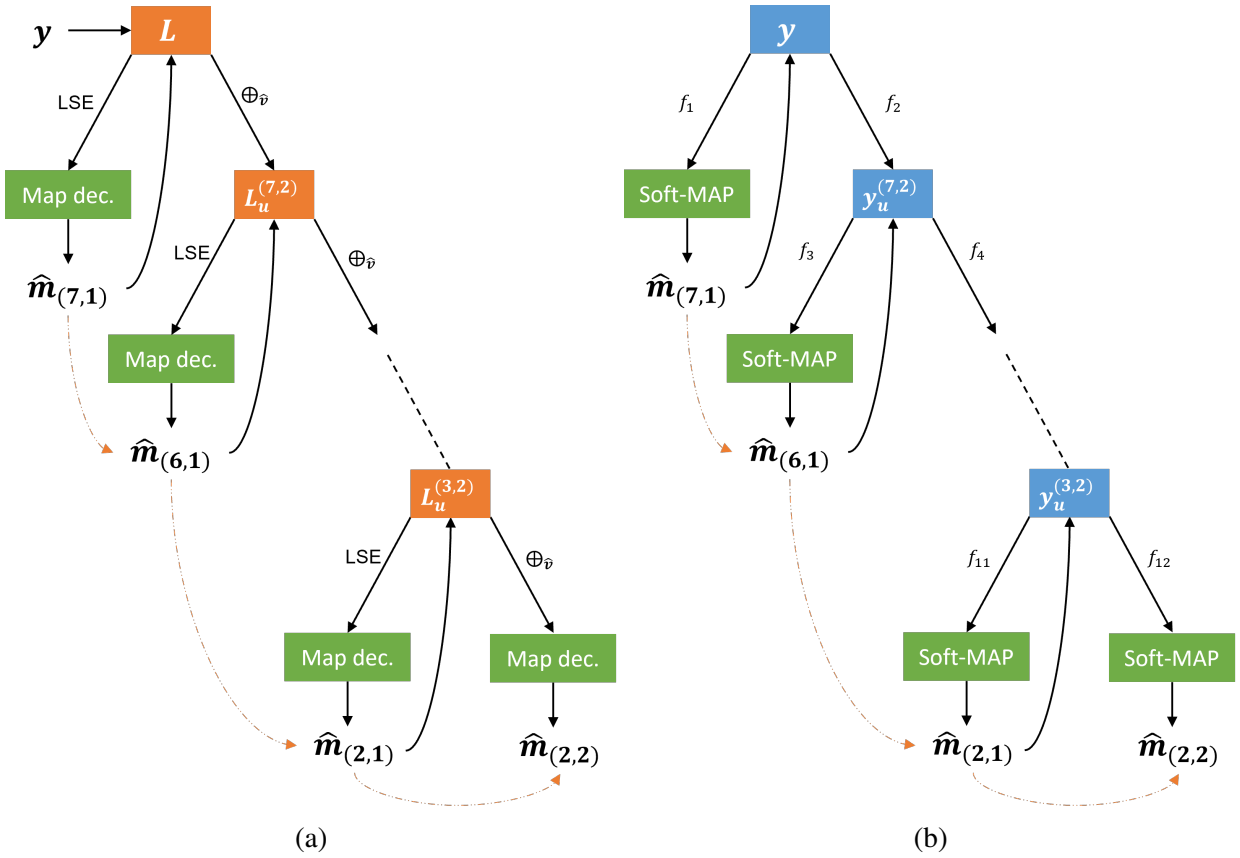


Figure 8.2: Decoding of (a) RM and (b) KO codes [2].

then applied to the factor-graph representation of RM(m, 1) for a fixed number of iterations, and a soft estimate of $\hat{\boldsymbol{m}}_{(m,1)}$ is obtained at the end of the BP decoding.

In addition, it can also be observed from Fig. 8.1 and Fig. 8.2 that KO codes require a significant amount of memory to store the weights of all the neural mapping functions, i.e., $g(\cdot)$'s and $f(\cdot)$'s functions, which can be addressed by using techniques such as weight sharing and weight quantization during the course of parameter optimization.

Inflexible code rates and low-order modulation

In [2], a KO code is optimized independently for a specific code length and rate. When a new KO code with a different length and rate is considered, both the encoder and the decoder are required to update all the trained parameters from memory to perform encoding and decoding. This incurs

significant decoding latency and memory overheads. To address this problem, the neural mapping functions used in KO codes can be optimized for a set of code rates to provide a reasonable trade-off between the error-correction performance degradation and the memory consumption overheads. In addition, the network architectures of KO codes can be modified to support higher order modulation schemes. In particular, instead of learning the mapping function from $\boldsymbol{m}$ to $\boldsymbol{x} \in \mathbb{C}^N$, the $f(\cdot)$ and $g(\cdot)$ functions can be directly trained to perform the mapping of $\boldsymbol{m}$ to $\boldsymbol{x}_t \in \mathbb{C}^{\frac{N}{2^t}}$ ($t \in \mathbb{Z}^+$) with some power constraint. This is particularly useful as the system bandwidth can be improved by using a higher modulation scheme at a high operating SNR.

Extension to list decoding

The optimization of the network parameters strictly requires differentiable encoding/decoding operations, which is only available for the most probable estimated codeword $\hat{\boldsymbol{m}}$ obtained through the bit-wise MAP decoding of the KO decoder. However, in practical scenarios, a list of the most probable decoding paths is often considered to significantly improve the error-correction performance of the codes. This extension is relatively similar for the conventional decoding algorithms such as SC decoding for polar and RM codes. On the other hand, the path extension operations are non-differentiable for KO codes as a new decoding path is forked from a current active decoding path by flipping the hard decision of a certain information bit, followed by the path metric sorting and pruning, which are also non-differentiable operations. As a consequent, the conventional supervised learning based approach used in [2] may not be directly suitable when KO codes are optimized targeting a list decoding algorithm. Therefore, indirect learning techniques that can circumvent this problem such as reinforcement learning should be investigated.

End-to-end training

KO codes are optimized with the assumption that there is a reliable feedback channel that allows the gradient flows from the receiver to the encoder to optimize all the trainable parameters. Therefore, we address the end-to-end training problem of KO codes where reliable communication channels between the receiver and the transmitter are not available, e.g., with unknown or partially available channel statistics. This problem can be addressed by using existing gradient approximation methods. However, novel low-complexity techniques to approximate the gradient should be investigated as the existing works mostly rely on the complicated Generative Adversarial Networks

(GANs) to generate the gradients, which impose another learning problem of the system.

Towards end-to-end multi-user communications with KO codes

In [2], KO codes are only designed for the simplified point-to-point communication scenarios. However, in practice, a transmission scheme considering Multiple-Input Multiple Output (MIMO) and Orthogonal Frequency Division Multiplexing (OFDM) is often considered. Therefore, MIMO-OFDM modeling should be included as additional inference layers of the end-to-end communication system using KO codes. The KO codes' parameters are then optimized to minimize the performance metric given the practical MIMO-OFDM scheme.

Bibliography

[1] A. Balatsoukas-Stimming, M. B. 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.

[2] A. V. Makkuva, X. Liu, M. V. Jamali, H. Mahdavifar, S. Oh, and P. Viswanath, "KO codes: inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7368–7378.

[3] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive-cancellation list decoders for polar codes," *IEEE Trans. on Sig. Proc.*, vol. 65, no. 21, pp. 5756–5769, Nov 2017.

[4] N. Ghaddar, H. Saber, H.-P. Lin, J. H. Bae, and J. Lee, "Simplified decoding of polar codes by identifying Reed-Muller constituent codes," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[5] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge University Press, 2009.

[6] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[7] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, 1954.

[8] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection," *Transactions of the I.R.E. Professional Group on Electronic Computers*, vol. EC-3, no. 3, pp. 6–12, 1954.

[9] S. Kudekar, S. Kumar, M. Mondelli, H. D. Pfister, E. Şaşoğlu, and R. L. Urbanke, "Reed–muller codes achieve capacity on erasure channels," *IEEE Trans. Inf. Theory*, vol. 63, no. 7, pp. 4298–4316, 2017.

[10] G. Reeves and H. D. Pfister, "Reed-Muller codes achieve capacity on BMS channels," 2021.

[11] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[12] D. J. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *IMA International Conference on Cryptography and Coding*. Springer, 1995, pp. 100–111.

[13] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

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

[15] 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, July 2009.

[16] 3GPP, "Multiplexing and channel coding 3GPP TS 21.101 v10.4.0. Release 10," Oct. 2018. [Online]. Available: http://www.3gpp.org/ftp/Specs/2018-09/Rel-10/21_series/21101-a40.zip

[17] ——, "System architecture for the 5G system (5GS) 3GPP TS 23.501 v16.6.0. Release 16," Oct. 2020. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/

[18] Ericsson, "5G wireless access: an overview," *White Papers*. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/5g-wireless-access-an-overview

[19] I. Dumer and K. Shabunov, "Soft-decision decoding of Reed-Muller codes: recursive lists," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1260–1266, 2006.

[20] I. Dumer, "Recursive decoding and its performance for low-rate Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 811–823, 2004.

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

[22] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, 2012.

[23] 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, October 2011.

[24] 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, April 2014.

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

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

[27] M. Hanif, M. H. Ardakani, and M. Ardakani, "Fast list decoding of polar codes: Decoders for additional nodes," in *IEEE Wire. Comm. and Net. Conf. Work.*, April 2018, pp. 37–42.

[28] S. A. Hashemi, C. Condo, F. Ercan, and W. J. Gross, "Memory-efficient polar decoders," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 7, no. 4, pp. 604–615, Dec. 2017.

[29] L. Chandesris, V. Savin, and D. Declercq, "Dynamic-SCFlip decoding of polar codes," *IEEE Trans. Commun.*, vol. 66, no. 6, pp. 2333–2345, June 2018.

[30] F. Cheng, A. Liu, Y. Zhang, and J. Ren, "Bit-flip algorithm for successive cancellation list decoder of polar codes," *IEEE Access*, vol. 7, pp. 58 346–58 352, 2019.

[31] Y.-H. Pan, C.-H. Wang, and Y.-L. Ueng, "Generalized SCL-Flip decoding of polar codes," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[32] F. Ercan, T. Tonnellier, N. Doan, and W. J. Gross, "Practical dynamic SC-flip polar decoders: Algorithm and implementation," *IEEE Trans. Signal Process.*, vol. 68, pp. 5441–5456, 2020.

[33] B. Yuan and K. K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Transactions on Signal Processing*, vol. 62, no. 24, pp. 6496–6506, Dec. 2014.

[34] S. M. Abbas, Y. Fan, J. Chen, and C. Tsui, "Concatenated LDPC-polar codes decoding through belief propagation," in *IEEE Int. Symp. on Circuits and Systems*, May 2017, pp. 1–4.

[35] C. Douillard, M. Jézéquel, C. Berrou, D. Electronique, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of intersymbol interference: turbo-equalization," *European transactions on telecommunications*, vol. 6, no. 5, pp. 507–511, 1995.

[36] Y. Ren, C. Zhang, X. Liu, and X. You, "Efficient early termination schemes for belief-propagation decoding of polar codes," in *IEEE 11th Int. Conf. on ASIC*, Nov 2015, pp. 1–4.

[37] S. Sun, S. Cho, and Z. Zhang, "Post-processing methods for improving coding gain in belief propagation decoding of polar codes," in *2017 IEEE Glob. Commun. Conf.*, Dec 2017, pp. 1–6.

[38] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. Ten Brink, "Automorphism ensemble decoding of reed—muller codes," *IEEE Trans. Commun.*, pp. 1–1, 2021.

[39] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 4948–4965, 2020.

[40] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *48th Asilomar Conf. on Sig., Sys. and Comp.*, Nov 2014, pp. 2116–2120.

[41] S. Cammerer, T. Gruber, J. Hoydis, and S. ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," pp. 1–6, December 2017.

[42] R. Pedarsani, S. H. Hassani, I. Tal, and E. Telatar, "On the construction of polar codes," in *IEEE Int. Symp. on Inf. Theory*, 2011, pp. 11–15.

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

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

[45] M. Mondelli, S. H. Hassani, and R. Urbanke, "Construction of polar codes with sublinear complexity," in *IEEE Int. Symp. on Inf. Theory*, 2017, pp. 1853–1857.

[46] L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "Reinforcement learning for nested polar code construction," *IEEE Global Commun. Conf.*, pp. 1–6, 2019.

[47] Y. Liao, S. A. Hashemi, J. Cioffi, and A. Goldsmith, "Construction of polar codes with reinforcement learning," *IEEE Global Commun. Conf.*, pp. 1–6, 2020.

[48] G. D. Forney, "Codes on graphs: normal realizations," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 520–548, 2001.

[49] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," Massachusetts Inst of Tech Lexington Lincoln Lab, Tech. Rep., 1953.

[50] G. Schnabl and M. Bossert, "Soft-decision decoding of Reed-Muller codes as generalized multiple concatenated codes," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 304–308, 1995.

[51] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast list decoders for polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 318–328, 2016.

[52] E. Arıkan, "Polar codes: A pipelined implementation," in *Proc. 4th Int. Symp. on Broad. Commun.*, 2010, pp. 11–14.

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

[54] N. Doan, S. A. Hashemi, M. Mondelli, and W. J. Gross, "On the decoding of polar codes on permuted factor graphs," *IEEE Global Commun. Conf.*, pp. 1–6, Dec 2018.

[55] A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Belief propagation decoding of polar codes on permuted factor graphs," in *IEEE Wireless Commun. and Net. Conf.*, April 2018, pp. 1–6.

[56] 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. of Sel. Topics in Signal Process.*, vol. 12, no. 1, pp. 119–131, February 2018.

[57] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *IEEE Int Symp. on Inf. Theory*, August 2017, pp. 1361–1365.

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

[59] N. Doan, S. A. Hashemi, F. Ercan, T. Tonnellier, and W. Gross, "Neural dynamic successive cancellation flip decoding of polar codes," *IEEE Int. Work. on Sig. Proc. Sys.*, 2019. [Online]. Available: https://arxiv.org/abs/1907.11563

[60] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent." [Online]. Available: https://cs.toronto.edu/csc321/slides/lecture_slides_lec6.pdf

[61] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *Int. Conf. on Learn. Rep.*, 2016. [Online]. Available: https://arxiv.org/abs/1510.00149

[62] A. Paszke, S. Gross, S. Chintala *et al.*, "Automatic differentiation in PyTorch," 2017.

[63] P. Giard and A. Burg, "Fast-SSC-flip decoding of polar codes," in *2018 IEEE Wireless Comm. and Net. Conf. Work.*, 2018, pp. 73–77.

[64] F. Ercan, T. Tonnellier, and W. J. Gross, "Energy-efficient hardware architectures for fast polar decoders," *IEEE Trans. Circuits Syst. I*, vol. 67, no. 1, pp. 322–335, 2020.

[65] S. A. Hashemi, N. Doan, T. Tonnellier, and W. J. Gross, "Deep-learning-aided successive-cancellation decoding of polar codes," in *53rd Asilomar Conf. on Sig., Sys., and Comp.*, 2019, pp. 532–536.

[66] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[67] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[68] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[69] H.-Y. Lee, Y.-H. Pan, and Y.-L. Ueng, "A node-reliability based CRC-aided successive cancellation list polar decoder architecture combined with post-processing," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5954–5967, 2020.

[70] N. Doan, S. A. Hashemi, F. Ercan, and W. J. Gross, "Fast SC-Flip decoding of polar codes with reinforcement learning," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[71] F. Ercan, T. Tonnellier, and W. J. Gross, "Energy-efficient hardware architectures for fast polar decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 322–335, 2020.

[72] N. Doan, S. A. Hashemi, F. Ercan, T. Tonnellier, and W. J. Gross, "Neural successive cancellation flip decoding of polar codes," *J. Sig. Proc. Sys.*, pp. 1–12, 2020.

[73] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using taylor series," in *2014 NORCHIP*, 2014, pp. 1–4.

[74] B. Taylor, *Methodus incrementorum directa et inversa*. Innys, 1717.

[75] P. Guide, "Intel 64 and ia-32 architectures software developer's manual," *Volume 3B: System programming Guide, Part*, vol. 2, no. 11, 2011.

[76] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[77] J. Guo, M. Qin, A. G. i Fàbregas, and P. H. Siegel, "Enhanced belief propagation decoding of polar codes through concatenation," in *IEEE Int. Symp. on Inf. Theory*, June 2014, pp. 2987–2991.

[78] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Conf. on Operating Systems Design and Impl.*, ser. OSDI'16. USENIX Association, 2016, pp. 265–283.

[79] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb 2001.

[80] S. B. Korada, "Polar codes for channel and source coding," Ph.D. dissertation, EPFL, Lausanne, Switzerland, 2009.

[81] N. Hussami, S. B. Korada, and R. Urbanke, "Performance of polar codes for channel and source coding," in *IEEE Int. Symp. on Inf. Theory*, 2009, pp. 1488–1492.

[82] S. Agrawal and N. Goyal, "Analysis of thompson sampling for the multi-armed bandit problem," in *Conf. on Learning Theory*, 2012, pp. 39–1.

[83] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "CRC-aided belief propagation list decoding of polar codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 395–400.

[84] S. A. Hashemi, N. Doan, M. Mondelli, and W. J. Gross, "Decoding Reed-Muller and polar codes by successive factor graph permutations," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, 2018, pp. 1–5.

[85] Y. Be'ery and J. Snyders, "Optimal soft decision block decoders based on fast hadamard transform," *IEEE Trans. Inf. Theory*, vol. 32, no. 3, pp. 355–364, 1986.

[86] D. Fathollahi, N. Farsad, S. A. Hashemi, and M. Mondelli, "Sparse multi-decoder recursive projection aggregation for Reed-Muller codes," in *IEEE Int. Symp. on Inf. Theory*, 2021. [Online]. Available: https://arxiv.org/abs/2011.12882

[87] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.

[88] K. Niu and K. Chen, "Stack decoding of polar codes," *Electronics letters*, vol. 48, no. 12, pp. 695–697, 2012.

[89] P. Trifonov, "A score function for sequential decoding of polar codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1470–1474.

[90] M. Kamenev, Y. Kameneva, O. Kurmaev, and A. Maevskiy, "A new permutation decoding method for Reed-Muller codes," *IEEE Int. Symp. on Inf. Theory*, pp. 26–30, 2019.