Energy-Proportional Polar Decoders

by

Ilshat SAGITOV

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, NOVEMBER 29, 2024

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Pascal Giard, thesis supervisor
Département de génie électrique à l'École de technologie supérieure

Mr. Kaiwen Zhang, president of the board of examiners
Département de génie électrique à l'École de technologie supérieure

Mr. Georges Kaddoum, member of the jury
Département de génie électrique à l'École de technologie supérieure

Ms. Elsa Dupraz, external independent examiner
Lab-STICC à IMT Atlantique

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON 21, NOVEMBER, 2024

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my research director Professor Pascal Giard for his guidance, support, and motivation throughout my PhD journey. I would like to thank Dr. Charles Pillet for his great friendship and assistance. I would like to express my gratitude to Professor Alexios Balatsoukas-Stimming for his support and expertise. I further extend my thanks to my dear colleagues at the LaCIME – communications and microelectronics laboratory, for their friendship and support. Finally, I express my heartfelt thanks to both my dearest mother as well as my beloved wife for their countless prayers, love, and support in helping me achieve my goals.

Décodeurs polaires proportionnels à l'énergie

Ilshat SAGITOV

RÉSUMÉ

Les codes polaires sont une classe de codes correcteur d'erreurs ayant reçus beaucoup d'attention depuis leur découverte. Cela s'explique en partie par le fait qu'il a été prouvé que les codes polaires atteignent asymptotiquement la capacité d'un canal binaire symmétrique sans mémoire (binary discrete memoryless channel (B-DMC)) via un algorithme de décodage à faible complexité nommé annulation successive (successive-cancellation (SC)). Cependant, les performances de correction d'erreurs restent limitées pour SC pour des longueurs de codes finis. Ainsi, un algorithme plus performant, le décodeur à annulation successive par liste (successive-cancellation list (SCL)) est considéré comme l'algorithme de référence dans de nombreuses applications pratiques, comme lors de la standardisation de la 5G. Le décodeur SCL génère une liste de candidats pour décoder la représentation bruitée d'un message transmis. En conséquence, les performances de correction d'erreurs de SCL sont considérablement améliorées par rapport à SC. Par contre, l'implémentation matérielle d'un décodeur SCL consomme plus d'énergie et requiert une plus grande surface en comparaison à SC.

Le décodage SC à inversion (successive-cancellation flip (SCF)) a été proposé pour améliorer les performances de correction d'erreurs de SC en générant les candidats à travers plusieurs essais de décodage, c'est-à-dire de manière séquentielle. À chaque essai supplémentaire, le bit de décision estimé comme le moins fiable est inversé. Le décodeur à annulation successive par liste à inversion (successive-cancellation list flip (SCLF)) a été proposé avec l'idée de combiner les stratégies de décodage SCL et SCF. Les variantes dynamiques de ces décodeurs, SCF dynamique (dynamic successive-cancellation flip (DSCF)) et SCLF dynamique (dynamic successive-cancellation list flip (DSCLF)), améliorent les performances de correction d'erreurs. Cette stratégie de décodage à inversion fait que SCF et SCLF présentent des temps d'exécution variables, rendant le temps d'exécution moyen et la latence potentiellement très élevés. Néanmoins, les architectures existantes montrent que les décodeurs à inversion sont plus efficaces en termes de surface et de consommation énergétique en comparaison au décodeur SCL.

Les contributions de ce doctorat s'articulent autour de la conception de décodeurs de codes polaires à faible consommation d'énergie. Les algorithmes de décodage par inversion basés sur SCF et leurs variations peuvent atteindre les performances de correction d'erreurs des décodeurs SCL tout en ayant des implémentations matérielles plus efficaces. Cependant, le temps d'exécution variable pose un problème pour la réalisation des récepteurs. Les mécanismes proposés dans cette étude doctorale améliorent le temps d'exécution des décodeurs à inversion avec un impact minimal ou nul sur les performances de correction d'erreurs et les ressources matérielles. Concernant l'implémentation matérielle, l'accent est mis sur la mémoire requise, majoritaire à la surface du décodeur.

La première contribution de ce mémoire est le mécanisme de redémarrage généralisé (generalized restart mechanism (GRM)) pour le décodeur par inversion basé sur SCF. Lors de l'application du GRM à chaque essai supplémentaire, la partie de l'arbre de décodage précédant l'inversion du bit non fiable est évitée. Le chemin de redémarrage parcourt l'arbre de décodage depuis la racine jusqu'au bit de redémarrage afin de pouvoir l'estimer tout en évitant l'estimation des bits précédents. Le GRM réduit le temps d'exécution moyen du décodeur DSCF-3 de 26% à 60% sans aucun effet négatif sur les performances de correction d'erreurs. Le GRM requiert environ 4% de mémoire supplémentaire pour ce décodeur.

La deuxième contribution est l'ajout d'un GRM modifié pour des décodeurs à inversion possédant des techniques de décodage rapide. Ces techniques de décodage rapide améliorent le temps d'exécution des décodeurs à inversion. Notre proposition GRM est conçue pour être adaptable à ces techniques de décodage rapide. Grâce à cela, le temps d'exécution moyen des décodeurs à inversion est encore plus réduit.

La troisième contribution est le mécanisme de redémarrage à localisation limitée (limited-locations restart mechanism (LLRM)) pour le décodeur SCLF. L'ajout du GRM aux décodeurs à inversion par liste requiert une quantité trop importante de mémoire. Pour résoudre ce problème, nous proposons le LLRM, une modification du

GRM. Une méthode basée sur les probabilités pour sélectionner les emplacements de redémarrage est proposée. Elle vise à maximiser la réduction du temps d'exécution tout en minimisant la mémoire requise. Le LLRM réduit le temps d'exécution moyen de 10% à 40% lorsqu'il est appliqué au décodeur DSCLF-3. Le LLRM nécessite environ 2% de mémoire supplémentaire.

La quatrième contribution est le mécanisme de terminaison anticipée pour les décodeurs à inversion. Cette contribution consiste en deux mécanismes distincts. Tout d'abord, un mécanisme d'arrêt précoce est introduit pour différencier les mots de code indécodables des mots de code décodables en utilisant notre métrique. Si la métrique suggère qu'un mot codé est probablement indécodable, le décodeur tente un nombre maximal réduit d'essais. Le mécanisme d'arrêt précoce réduit le temps d'exécution moyen du décodeur DSCF-1 de 22% au prix d'une perte de correction d'erreur mineure de 0,05 dB. Deuxièmement, le mécanisme à seuils multiples proposé limite la latence d'un décodeur à inversion en fonction de l'état du buffer afin d'éviter toute perte de données. Ce mécanisme est mis en œuvre dans un système où le canal produit des données à un taux fixe. Appliqué au décodeur DSCF-1, le mécanisme à seuils multiples permet d'opérer dans un système avec un taux de production de canaux fixe 1,13 fois inférieur au taux associé à un seul essai de décodage. Il en résulte une perte de correction d'erreurs mineure de 0,06 dB.

Mots-clés: Codes polaires, Codes correcteur d'erreurs, Décodage, Gestion de la mémoire, Temps d'exécution, Complexité, Efficacité énergétique

Energy-Proportional Polar Decoders

Ilshat SAGITOV

ABSTRACT

Polar codes are a class of linear error-correction codes that have received a lot of attention due to their ability to achieve channel capacity in an arbitrary binary discrete memoryless channel (B-DMC) with low-complexity successive-cancellation (SC) decoding. However, practical implementations often require better error-correction performance than what SC decoding provides, particularly at shorter code lengths. As a result, the successive-cancellation list (SCL) decoder has become the reference algorithm for many practical applications, such as 3GPP's next-generation mobile-communication standard (5G). The SCL decoder improves error-correction performance by generating a list of candidate codewords from the noisy received message. However, the hardware implementation of SCL decoder tends to be less energy- and area-efficient than that of a SC decoder.

Successive-cancellation flip (SCF) decoder involves multiple decoding trials, where at each additional trial, the decision bit estimated as the least reliable is flipped before resuming the standard SC decoding. The successive-cancellation list flip (SCLF) decoder combines the SCL and SCF decoding strategies. Variations of these decoders, such as dynamic successive-cancellation flip (DSCF) and dynamic successive-cancellation list flip (DSCLF), further improve the error-correction performance. Despite their advantages in error correction, the flip decoders (SCF and SCLF) have variable execution times, which can lead to high average execution time and latency. Nevertheless, existing architectural designs demonstrate that flip decoders are more efficient in area and energy requirements compared to the SCL decoder.

The contributions of this doctoral study are focused on the design of energy-efficient decoders for polar codes. The flip decoding algorithms based on SCF and their variations can achieve the error-correction performance of the state-of-the-art SCL decoders while resulting in more efficient hardware implementations. However, variable execution time poses a challenge for realization in receivers. This doctoral study proposes mechanisms to improve the execution-time characteristics of flip decoders while minimizing the impact on error-correction performance and hardware resources. Among hardware resources, the primary focus is on the decoder memory, which occupies the largest part of the decoder area.

The first contribution of this thesis is the generalized restart mechanism (GRM) for the flip decoder based on SCF. By applying the GRM, a portion of the decoding tree can be skipped. This portion corresponds to parts of the tree to estimate the bit-flipping candidate and all the previous bits in each additional decoding trial. The GRM reduces the average execution time of the DSCF-3 decoder by 26% to 60% without any negative effect on the error-correction performance. The GRM results in approximately 4% of additional memory for this decoder.

The second contribution is the modified GRM for flip decoders with fast decoding techniques. Existing fast decoding techniques improve the execution-time characteristics of the flip decoders. Our proposed GRM is designed to be adaptable to these fast decoders. As a result of this combination, the average execution time of the flip decoders is further reduced while maintaining the original error-correction performance.

The third contribution is the limited-locations restart mechanism (LLRM) for the list-flip decoder (SCLF and its variations). Applying our proposed GRM to list-flip decoders results in enormous memory overhead. To overcome this issue, the LLRM, a modification of the GRM, is proposed. The probability-based method for selecting restart locations is proposed, which aims to maximize the execution-time reduction while minimizing the memory overhead. The LLRM reduces the average execution time by 10% to 40% when applied to DSCLF-3 decoder. For this decoder, the LLRM requires approximately 2% of additional memory. The LLRM does not modify original error-correction performance.

The fourth contribution is the early-termination mechanism for flip decoders. This contribution consists of two mechanisms. First, the early-stopping mechanism is introduced to differentiate undecodable codewords from

decodable ones by using our proposed early-stopping metric. If the metric suggests that a codeword is likely undecodable, the decoder attempts a reduced maximum number of trials, much smaller than the initial maximum number of trials. The early-stopping mechanism reduces the average execution time of the DSCF-1 decoder by 22% at the cost of minor error-correction loss of 0.05 dB. Second, the multi-threshold mechanism is proposed. This mechanism restrains the delay of a flip decoder depending on the state of the buffer to prevent overflow. This mechanism is implemented in the system where the channel produces data with a fixed rate. When applied to the DSCF-1 decoder, the multi-threshold mechanism allows to operate in a system with a fixed channel-production rate 1.13 times lower than the rate associated with a single decoding trial. This results in a minor in a minor error-correction loss of 0.06 dB.

Keywords: Polar Codes, Error-correction Codes, Decoding, Memory Management, Execution Time, Complexity, Energy Efficiency

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF GLOSSARIES

5G 3GPP's next-generation mobile-communication standard.

ASIC application-specific integrated circuit.

AWGN additive white Gaussian noise.

B-DMC binary discrete memoryless channel.

BER bit-error rate.

BPSK binary phase-shift keying.

CC clock cycle.

CRC cyclic-redundancy check.

DSCF dynamic successive-cancellation flip.

DSCLF dynamic successive-cancellation list flip.

eMBB enhanced mobile broadband.

FER frame-error rate.

GRM generalized restart mechanism.

IoT Internet of things.

LDPC low-density parity check.

LHS left-hand side.

LLR log-likelihood ratio.

LLRM limited-locations restart mechanism.

LRT latency-reducing technique.

LSB least significant bit.

mMTC massive machine-type communications.

PMF probability-mass function.

PS partial-sum.

PSCF partitioned SCF.

PSCLF partitioned SCLF.

R0 rate-0.

R1 rate-1.

REP repetition.

RHS right-hand side.

SC successive-cancellation.

SCF successive-cancellation flip.

SCL successive-cancellation list.

SCLF successive-cancellation list flip.

SNR signal-to-noise ratio.

SPC single-parity-check.

SRM simplified restart mechanism.

SSC Simplified SC.

INTRODUCTION

Digital communication systems are integrated into many areas of modern technology. Networks such as the Internet of things (IoT) require low-power transmitters and receivers to provide service for networks with strict requirements on production cost and battery life. The 5G networks have gained solid ground and are being actively implemented in modern digital systems. The massive machine-type communications (mMTC) service of 5G fits the requirements of Internet of things (IoT) networks, and it will be expanded and further improved in the 5G advanced 3GPP (2018).

The transmitted information can often get corrupted due to noises and disturbances present in the communication channel. A straightforward retransmission can solve the problem to a certain extent. However, it increases communication delay, which restrains communication capabilities. Error detection and correction algorithms on the receiver's side are crucial for reliable and sustainable communication systems, such as satellite and 5G networks. Nevertheless, blocks implementing error detection and correction, particularly decoding, are typically among the most resource-intensive blocks of the entire communication system. Therefore, developing efficient error-correction decoding algorithms is crucial, especially for systems with strict resource requirements, such as IoT networks.

Polar codes are linear block codes that asymptotically achieve the channel capacity under the low-complexity SC decoding as the code length approaches infinity (Arıkan, 2009). Polar codes have explicit construction and the low-complexity SC decoding algorithm. Unfortunately, as shown in the study by Balatsoukas-Stimming, Giard & Burg (2017), polar codes with the SC decoder provide mediocre error-correction performance at short-to-moderate code lengths compared to other modern channel codes like low-density parity check (LDPC). Nevertheless, low-complexity encoding and decoding algorithms make polar codes suitable for mMTC service of 5G, as stated in the work of Zhou, Miao, Liu & Liu (2021).

Problem Statement

Tal & Vardy (2011) have proposed the SCL decoding algorithm to improve the error-correction performance of the SC decoder for polar codes. This algorithm decodes a codeword by generating a list of L parallel candidates throughout decoding. In their following work, Tal & Vardy (2015) have concatenated the information block with the CRC code before passing it through a polar encoder. This allows the SCL decoder to select the most likely transmitted word from the final list of candidates if the initial selection strategy fails. The resulting CRC-aided SCL decoder significantly improves error-correction performance to the extent that polar codes are selected to protect the control channel of the enhanced mobile broadband (eMBB) service in 5G (3GPP, 2018). However, the SCL decoder implementations require more resources compared to the SC decoder because of their parallel decoding schedule.

Afisiadis, Balatsoukas-Stimming & Burg (2014) have proposed the SCF decoding to improve the error-correction performance of the SC decoding. Unlike the SCL, which decodes a codeword with L parallel candidates, SCF generates the candidates across multiple decoding trials, i.e., decoding attempts. A list of bit-flipping candidates is constructed at the end of an initial unsuccessful SC decoding trial. This list contains the bit locations estimated as the least reliable ones. At each additional trial, a decision bit from the list is flipped during the course of the SC decoding before normal decoding is resumed. A block of information bits is concatenated with the CRC code before passing it through a polar encoder. This allows the SCF decoder to validate an estimated information word and exit decoding when the CRC passes. In their study, Afisiadis *et al.* (2014) have presented that the SCF decoder achieves the same error-correction performance as the SCL with a small list size L.

The DSCF algorithm was proposed by Chandesris, Savin & Declercq (2018) with two major contributions: (a) a more accurate metric to construct the list of the bit-flipping candidates, and (b) algorithmic modifications where multiple bits can be simultaneously flipped per decoding trial and where the list of candidates is dynamically updated. As a result of these modifications, the DSCF decoding achieves the same performance as the SCL with a medium list size L.

The SCLF decoding is proposed by Yongrun, Zhiwen, Nan & Xiaohu (2018) with the idea of combining the SCL and SCF decoding strategies. The standard SCL decoding is performed at the initial trial. If none of the L paths pass the CRC at the end of this trial, a list of path-flipping candidates is constructed. The path-flipping strategy for the SCLF decoding was first introduced by Cheng, Liu, Zhang & Ren (2019). The additional decoding trial follows the standard SCL algorithm until the path-flip location, where all initially discarded paths are selected as opposed to the original selection criteria. The standard SCL decoding is then resumed to decode the remaining bits in a codeword. The SCLF decoder has the $T_{\max}$ trials to decode a codeword, and if failed, the codeword is deemed undecodable. Then, the DSCLF decoding was proposed by Shen, Balatsoukas-Stimming, You, Zhang & Burg (2022), where the DSCF algorithm strategy was adapted to SCLF. This adaptation has allowed the authors to perform multi-path flipping during an additional decoding trial. The SCLF decoders allow for a balance between SCL and SCF decoders to achieve the target error-correction performance.

Despite improved error-correction performance, the SCF and SCLF decoding algorithms and their variations introduce variability in execution time. This variability poses challenges in implementing receivers with fixed-time operations. Giard, Balatsoukas-Stimming, Müller, Bonetti, Thibeault, Gross, Flatresse & Burg (2017) and Ercan, Tonnellier, Doan & Gross (2020a) have proposed architectural designs. These designs demonstrate that the SCF and DSCF decoders are more efficient in hardware resources and energy requirements compared to the SCL decoder.

However, the average execution time of SCF and DSCF decoders is higher than the latency of the SCL decoder. Therefore, mechanisms that can reduce this average execution time are needed.

Research Objectives

The main objective of this doctoral study is to design energy-efficient flip decoders of polar codes, based on the SCF algorithm. In particular, the goal is to develop mechanisms to improve the execution-time characteristics of flip decoders with minimal to no impact on the error-correction performance and hardware resources. The analysis of hardware resources focuses on decoder memory.

To achieve the main objective, the following sub-objectives are identified:

1. Design universal restart mechanisms that skip computationally intensive operations in flip decoders.
2. Derive a methodology for analyzing resource requirements and execution-time characteristics for realistic hardware implementations.
3. Modify the mechanisms to integrate with existing fast decoding techniques to further improve the execution-time characteristics of flip decoders.
4. Design efficient early-termination mechanisms for flip decoders that are adapted to specific channel or latency conditions.

This study proposes the following mechanisms for flip decoders of polar codes:

1. **Generalized Restart Mechanism (GRM) for Flip Decoders.**
   The flip decoders can achieve the error-correction performance of the state-of-the-art SCL decoders. However, they can only achieve this with a high maximum number of trials, resulting in increased average and worst-case execution times. In our proposed GRM, the parts of the tree to estimate a bit-flipping candidate and all of the prior bits of each additional decoding trial are skipped. The decoding tree is traversed from the root along the restart path to directly estimate the restart bit. To perform such a restart, the bit estimates must be stored in memory at the end of the initial unsuccessful decoding trial. The GRM reduces the average execution time of the DSCF-3 decoder by 26% to 60% without any negative effect on the error-correction performance. This decoder achieves the error-correction performance of the state-of-the-art SCL decoder. The GRM results in approximately 4% of additional memory.

2. **Modified Generalized Restart Mechanism for Fast Decoding Techniques.**

   Existing fast decoding techniques improve the execution-time characteristics of the flip decoders. The GRM proposed in this study is designed to be adaptable to these fast decoding techniques. The modified GRM reduces the average execution time by 15% to 22% when applied to Fast-DSCF-3 decoder. This reduction is in addition to the reduction already attained by fast decoding. Compared to the original decoding, the modified GRM requires approximately 4% of additional memory.

3. **LLRM for List-Flip Decoders.**

   List-flip decoders based on SCLF allow for a balance between the SCL and SCF decoders to achieve the desired error-correction performance. Similar to flip decoders, they also exhibit high average and worst-case execution times. Applying the GRM to list-flip decoders highly improves the execution-time characteristics. However, this improvement results in enormous memory overhead. To overcome the issue of memory overhead, the LLRM, a modification of the GRM, is proposed. Alongside this, the probability-based method for selecting restart locations is developed. This method aims to maximize the execution-time reduction while minimizing the memory overhead. The LLRM reduces the average execution time by 10% to 40% when applied to DSCLF-3 decoder. This decoder provides the strongest error-correction performance compared to other list-flip decoders. The LLRM results in approximately 2% memory overhead, whereas the GRM incurred 170% to 1500% memory overhead. According to the results, the proposed LLRM is the more appropriate algorithm for hardware implementations of the list-flip decoders.

4. **Early-termination Mechanisms for Flip Decoders.**

   Two early-termination mechanisms, namely the early-stopping mechanism and the multi-threshold mechanism, are proposed to improve the execution time characteristics of flip decoders. The early-stopping mechanism attempts to differentiate undecodable codewords from decodable ones. It uses the proposed early-stopping metric and a predefined threshold. If the metric suggests a codeword is likely undecodable, the decoder attempts a reduced maximum number of trials. This number is much smaller than the initial maximum trials. The early-stopping mechanism reduces the average execution time of the DSCF-1 decoder by approximately 22% and execution-time variance by approximately 45%. This mechanism results in a minor error-correction loss of approximately 0.05 dB at the target FER of 10^{-2}. The proposed multi-threshold mechanism restrains the delay of a flip decoder depending on the state of the buffer to prevent overflow. This mechanism is implemented in the system where the channel produces data with a fixed rate, which closely approaches that of a single decoding trial. When applied to DSCF-1 decoder, our proposed multi-threshold mechanism allows to operate in a system with a fixed channel-production rate approximately 1.13 times lower than the rate associated with

a single decoding trial. This adjustment helps prevent buffer overflow. Compared to the ideal scenario, the proposed mechanism has a minor error-correction loss of approximately 0.06 dB at the target FER of 10^{-2}. The ideal scenario, where the decoder operates without constraints, leads to an unsustainable situation due to the rapid overflow of the buffer.

Related Publications

The following publications resulted throughout the course of the doctoral study:

- I. Sagitov and P. Giard, "An Early-Stopping Mechanism for DSCF Decoding of Polar Codes", *IEEE Workshop on Signal Processing Systems (SiPS),* 2020, Coimbra, Portugal.
- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Successive-Cancellation Flip Decoding of Polar Codes with a Simplified Restart Mechanism." *IEEE Wireless Communications and Networking Conference (WCNC),* 2023, Glasgow, United Kingdom.
- C. Pillet, I. Sagitov, G. Domer, and P. Giard, "Partitioned Successive-Cancellation List Flip Decoding of Polar Codes." *IEEE Workshop on Signal Processing Systems (SiPS),* 2024, Cambridge, USA.
- I. Sagitov, C. Pillet, and P. Giard, "Successive-Cancellation Flip Decoding of Polar Codes Under Fixed Channel-Production Rate." *arXiv preprint arXiv:2409.03051,* 2024.
- C. Pillet, I. Sagitov, D. Deslandes, and P. Giard, "Successive-Cancellation Flip and Perturbation Decoder of Polar Codes." *IEEE Wireless Communications and Networking Conference (WCNC),* 2025, Milan, Italy (*under review*).
- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Generalized Restart Mechanism for Successive-Cancellation Flip Decoding of Polar Codes." *J. Signal Process. Syst.,* 2024 (*under review*).
- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Restart Mechanisms for the Successive-Cancellation List-Flip Decoding of Polar Codes." *J. Signal Process. Syst.,* 2024 (*under review*).

Thesis Organization

The thesis is structured as follows to provide a comprehensive overview of the research problem, progress, and results.

- CHAPTER 1: Literature review describes the background of digital communication, channel coding, polar code construction, and polar encoding. This chapter also describes the SC, fast and simplified SC, and SCL decoding algorithms, along with a comparison of their error-correction performance. At the end of the chapter, the implementation results of the state-of-the-art decoders are summarized and compared.

- CHAPTER 2: This chapter discusses the background on flip decoding algorithms while describing the SCF and DSCF decoding algorithms, along with the SCLF and DSCLF list-flip decoding algorithms. This chapter provides detailed comparisons in terms of the error-correction performance for all flip decoders. Then, it describes the time model that aligns with the architectural design. Following this, a comparison of the average execution time for the flip decoders is provided. Then, the implementation results of the state-of-the-art flip decoders are summarized and compared.

- CHAPTER 3: This chapter describes two proposed mechanisms for flip decoders, namely, the SRM and the generalized restart mechanism (GRM). The SRM is presented with a detailed description of the algorithm, followed by methodologies for resource and execution-time analysis. The simulation results are presented, focusing on error-correction performance and average execution time. Following this, the effectiveness and drawbacks of the SRM are identified. This chapter also presents the GRM, an extension of the SRM. The algorithm of the GRM and methodologies for resource and execution-time analysis are presented. At the end of the chapter, the simulation results are provided to highlight the effectiveness of the proposed GRM to SCF and DSCF decoders.

- CHAPTER 4: This chapter describes our proposed modified GRM for flip decoders with fast decoding techniques. The algorithm and the assumptions for applying the proposed GRM in conjunction with fast decoding techniques are provided. Then, methodologies for resource and execution-time analysis are presented. At the end of the chapter, simulation results are presented, highlighting the effectiveness of the proposed GRM when combined with existing fast decoding techniques for flip decoders, such as Fast-SCF and Fast-DSCF decoders.

- CHAPTER 5: This chapter introduces the proposed limited-locations restart mechanism (LLRM) for list-flip decoders and explores the challenges associated with applying the previously proposed GRM to these decoders. It provides a detailed description of the LLRM algorithm and memory estimates. Following this, the chapter presents the proposed probability-based method to determine the restart locations in the LLRM. Then, the methodologies for resource and execution-time analysis are provided. At the end of the chapter, the simulation results are presented, demonstrating the effectiveness of the proposed LLRM to SCLF and DSCLF decoders.

- CHAPTER 6: This chapter discusses two early-termination mechanisms for flip decoders: the early-stopping mechanism and the multi-threshold mechanism. It begins with a comprehensive description of the proposed early-stopping mechanism, followed by simulation results that demonstrate its effectiveness for the DSCF decoder. This chapter then discusses flip decoding under a fixed channel-production rate, starting with a description of the system model. The core of the model is the control mechanism that regulates the decoder based on the state of the buffer. It introduces the straightforward codeword-dropping mechanism and then presents the proposed multi-threshold mechanism. The simulation results are provided, highlighting the effectiveness of the multi-threshold mechanism for the DSCF decoder under a fixed channel-production rate.

The thesis concludes with a summary of the work. It provides academic achievements and discusses the limitations of the proposed modifications. Additionally, it outlines potential future research directions.

CHAPTER 1

LITERATURE REVIEW

Throughout the thesis, we use bold notations for vectors and matrices, while normal notations are used for single values (for example, a single element of the vector).

1.1 Digital Communication System

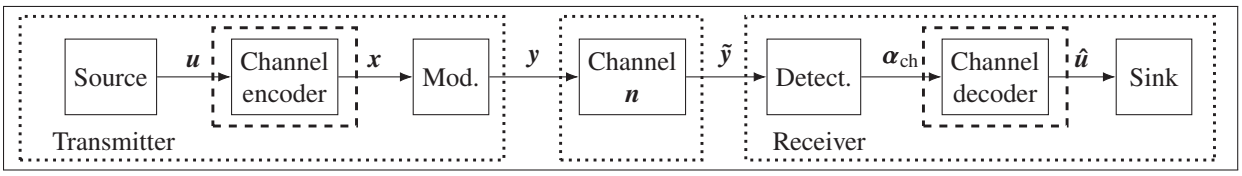


Figure 1.1 Simplified block diagram of the digital communication system

A digital communication system is often represented by three main blocks: the transmitter, the channel, and the receiver. The block diagram of this system is depicted in Figure 1.1, with these three blocks surrounded by the dotted line. In the transmitter block, the source delivers the information bit vector $\boldsymbol{u}$, which has the length of k. The channel encoder applies the encoding algorithm, creating a codeword $\boldsymbol{x} = \{x_0, \ldots, x_i, \ldots, x_{N-1}\}$, which has the length of N. The code rate R is defined as $R = {}^{k}/_{N}$, and $R < 1$ is required to enable error-correction coding. Next, modulation is applied, converting the resulting bit stream into a signal vector $\boldsymbol{y}$ that is transmitted over the channel. The channel affects the signal by adding noise vector $\boldsymbol{n}$ to the signal, and the resulting signal $\tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{n}$ is observed at the receiver. In the receiver, detection is made, where either soft or hard channel information is first observed. In the scope of this doctoral study, the soft channel information $\boldsymbol{\alpha}_{\text{ch}} = \{\alpha_{\text{ch}}(0), \ldots, \alpha_{\text{ch}}(i), \ldots, \alpha_{\text{ch}}(N-1)\}$ is considered. Next, $\boldsymbol{\alpha}_{\text{ch}}$ is delivered to the channel decoder, where the error-correction decoding algorithm is executed. Together, the channel encoder and decoder protect the transmitted information. Finally, the estimated information is received (forwarded to the sink).

Despite their important role in protecting transmitted information, the channel coding blocks, especially the decoder, are one of the most resource-consuming parts of the entire communication system. Therefore, the design of the decoder is the focus of this doctoral study. The channel encoder and decoder blocks are highlighted in the dashed line in Figure 1.1, and these blocks are where the polar codes are implemented. Additionally, a block of information bits is commonly concatenated with the cyclic-redundancy check (CRC) code, which is applied by the state-of-the-art polar codes to enhance the overall error-correction capability.

The choice of the channel model depends on the application environment. In this doctoral study, we consider wireless communication scenarios, thus the additive white Gaussian noise (AWGN) channel is used. The noise

follows a normal distribution $n \sim \mathcal{N}\left(\mu, \sigma_n^2\right)$, where the mean is $\mu = 0$ and the variance is σ_n^2. Furthermore, we use the binary phase-shift keying (BPSK) modulation scheme, where each bit in the transmitted codeword $x_i \in \mathbf{x}$ is mapped to a digital signal as:

$$y_i = \begin{cases} 1.0, & \text{when } x_i = 0, \\ -1.0, & \text{otherwise.} \end{cases} \tag{1.1}$$

Each bit of the signal is affected by the noise before it arrives at the receiver, as $\tilde{y}_i = y_i + n_i$. To decode a polar code, the demodulator performs a log-likelihood estimation of the input signal to obtain the soft estimated vector $\boldsymbol{\alpha}_{\text{ch}}$, also called the channel log-likelihood ratios (LLRs). In BPSK modulation, for each bit of the transmitted codeword, $\alpha_{\text{ch}}(i)$ is calculated as follows:

$$\alpha_{\text{ch}}(i) = \ln\left(\frac{p\left(\tilde{y}_i | y_i = 0\right)}{p\left(\tilde{y}_i | y_i = 1\right)}\right), \tag{1.2}$$

where "ln" denotes the natural logarithm and $p\left(\tilde{y}_i | y_i\right)$ represents the probabilities of transmitting each bit of the digital signal $y_i \in \{1, -1\}$. Those probabilities can be calculated by the probability density function as follows:

$$p\left(\tilde{y}_i | y_i\right) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \cdot \exp\left(-\frac{\left(\tilde{y}_i - y_i\right)}{2\sigma_n^2}\right). \tag{1.3}$$

Finally, applying (1.3) to (1.2), calculation of the channel LLRs for BPSK modulation can be simplified as:

$$\alpha_{\text{ch}}(i) = \frac{2\tilde{y}_i}{\sigma_n^2}, \tag{1.4}$$

The noise spectral density in the AWGN channel is estimated as $N_0 = 2\sigma_n^2$. The channel signal-to-noise ratio (SNR) is a common measure used to characterize the channel quality, which is computed as:

$$\text{SNR} = \frac{E_s}{N_0} = \frac{1}{2\sigma_n^2}, \tag{1.5}$$

where E_s denotes the signal energy, which is set to $E_s = 1$ to simplify the analysis.

Another common measure characterizing the channel is the ratio of the energy per information bit to the noise spectral density, denoted by E_b/N_0. It differs from the SNR by taking into account the code rate and the modulation applied to the transmitted message. It is calculated as follows:

$$E_b/N_0 = \frac{\text{SNR}}{R \cdot \log_2 M}, \tag{1.6}$$

where M is the size of the constellation, $R = {}^{k}/_{N}$ is the code rate, and the SNR is calculated by (1.5). For the BPSK, $M = 2$ is set. Note that both SNR and ${}^{E_b}/_{N_0}$ are typically expressed in dB.

Similarly to (1.6), the measures ${}^{E_b}/_{N_0}$ and SNR are related to each other in the logarithmic domain as follows:

$$ {}^{E_b}/_{N_0}|_{\text{dB}} = \text{SNR}|_{\text{dB}} - 10\log_{10}\left(R \cdot \log_2 M\right). \tag{1.7} $$

Note that since $M = 2$ applied in this work and $R \leq 1$, the relation ${}^{E_b}/_{N_0} \geq \text{SNR}$ in (1.7) holds.

Uncoded Versus Coded Communication

To demonstrate the advantages of using error-correction codes, we performed simulations to analyze the error-correction performance in terms of the bit errors of a message encoded with polar code transmitted over the channel. This performance is compared to the analytical expression of an uncoded message. The analysis is made only for demonstration purposes, while the details on polar code construction and encoding will be provided in the following sections of this chapter.

For the polar-encoded message, the block with the length $N = 128$ bits and with the rate $R = {}^{3}/_{4}$ is used. It contains $k = 96$ information bits, and the remaining $(N - k) = 32$ bits are used for error correction. These 32 bits, also known as parity bits, are formed as a combination of the 96 bits according to the encoding scheme. Simulations are made with the BPSK modulation over the AWGN channel, expressed through the ${}^{E_b}/_{N_0}$ with values $\{2.0, 2.125, \ldots, 7.25\}$ dB. The polar codes are decoded by the successive-cancellation (SC) decoder, the original decoding algorithm proposed in the seminal paper by Arıkan (2009). Simulations are performed with a minimum of $C = 2 \cdot 10^5$ messages and continue until at least 2000 message errors are observed for each ${}^{E_b}/_{N_0}$ point. The error-correction performance is analyzed using the bit-error rate (BER), i.e., the ratio of incorrectly received bits in messages to the total transmitted message bits. The BER is evaluated for each ${}^{E_b}/_{N_0}$ point.

For the uncoded message, the analytical expression for a bit-error probability, denoted by P_e, can be analytically estimated as:

$$ P_e = Q\left(\sqrt{2\frac{E_b}{N_0}}\right), \tag{1.8} $$

where Q-function is calculated for a given value of ${}^{E_b}/_{N_0}$ expressed in the linear domain. Note that (1.8) is valid for the BPSK modulation under the AWGN channel.

Figure 1.2 depicts the error-correction performance for uncoded and polar-encoded messages. The x axis represents ${}^{E_b}/_{N_0}$, where moving left indicates deterioration in the channel condition while moving right indicates improvement. The y axis, displayed on a logarithmic scale, corresponds to the BER. The top of the y axis indicates the high BER,

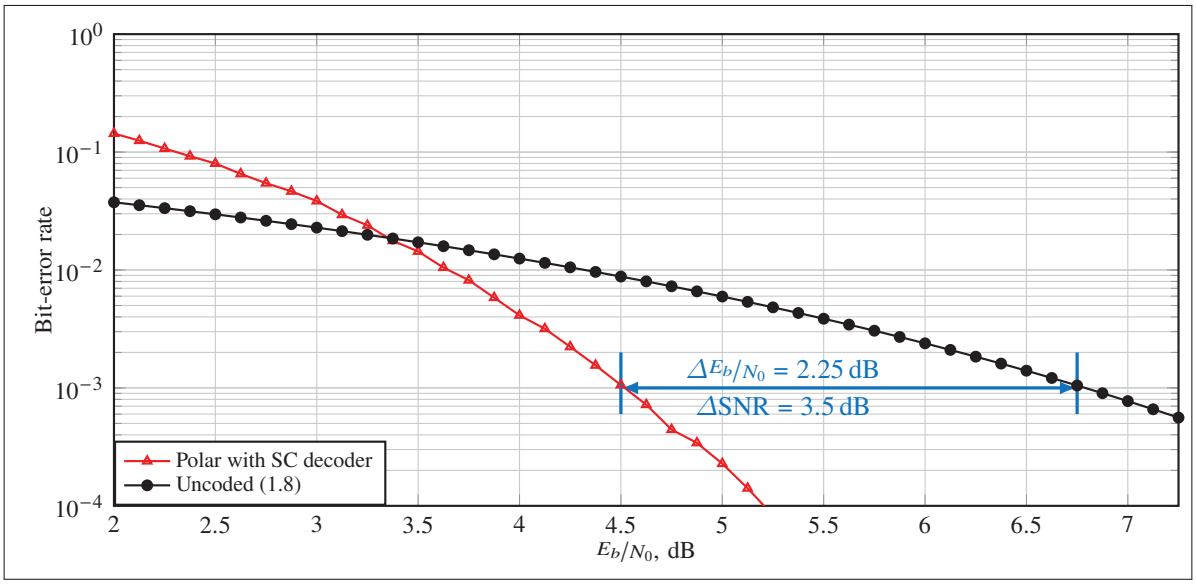


Figure 1.2 Error-correction performance of uncoded vs polar-encoded messages, both of length $N = 128$. Polar codes of rate $R = 3/4$ are decoded by the SC decoder

with the maximum value of $10^0 = 1$, indicating that 100% of the transmitted bits result in errors. As we descend the y axis, the BER decreases. The minimum displayed point is $10^{-4} = 0.0001$, indicating 0.01% of the bit errors.

The results of Figure 1.2 show that at low E_b/N_0, polar codes with SC decoder have worse performance than the uncoded messages. The region corresponding to BER of 10^{-1} is considered unreliable, while BER of 10^{-4} is the point of reference in many practical communication systems. In the BER region of 10^{-3} to 10^{-4}, the encoded messages highly surpass the performance of uncoded messages. At the BER of 10^{-3}, polar codes achieve a gain of $\Delta E_b/N_0 = 2.25$ dB and of ΔSNR = 3.5 dB. Therefore, to achieve the same error-correction performance as polar codes, the transmitter needs to send the signal of the uncoded messages with significantly more energy to compensate for the loss due to the channel noise.

Note that analyzing the error-correction performance using E_b/N_0 instead of SNR favors the uncoded messages over the polar-encoded messages. As shown in Figure 1.2, at the BER of 10^{-3}, the coding gain $\Delta E_b/N_0 = 2.25$ dB, while the coding gain in terms of SNR is ΔSNR = 3.5 dB. The difference occurs because the code rate R used for polar codes shifts the BER curve to the right along the E_b/N_0 axis relative to the SNR axis, according to (1.7). Thus, the gain achieved by polar codes in the BER region of 10^{-3} to 10^{-4} is significant.

1.2 Construction and Encoding of Polar Codes

Polar codes are the first linear block codes proven to asymptotically achieve the channel capacity over the binary-input discrete memoryless channel (B-DMC) (Arıkan, 2009). Polar codes have explicit construction, which is based on

the concept of channel polarization. This allows the assignment of k information bits among N total bits in the input vector. After constructing the code, encoding is performed, resulting in the creation of a codeword. These two operations are summarized below.

1.2.1 Code Construction

To understand the concept of channel polarisation, a simple example of a polarizing construction with $N = 2$ is shown in Figure 1.3 (a). Two bits, u_0 and u_1 are transformed into $x_0 = u_0 \oplus u_1$ and $x_1 = u_1$, where $\oplus$ is a bitwise XOR operation. Then, x_0 and x_1 each are sent over the combined channel W_2, whereas W is the original channel. On the receiver side, given that the channel noise affected the signal, the probability of correctly estimating the bit u_1 increases compared to the scenario without any transformation. Meanwhile, the probability of correctly estimating bit u_0 decreases. Thus, the effect of channel polarization is observed.

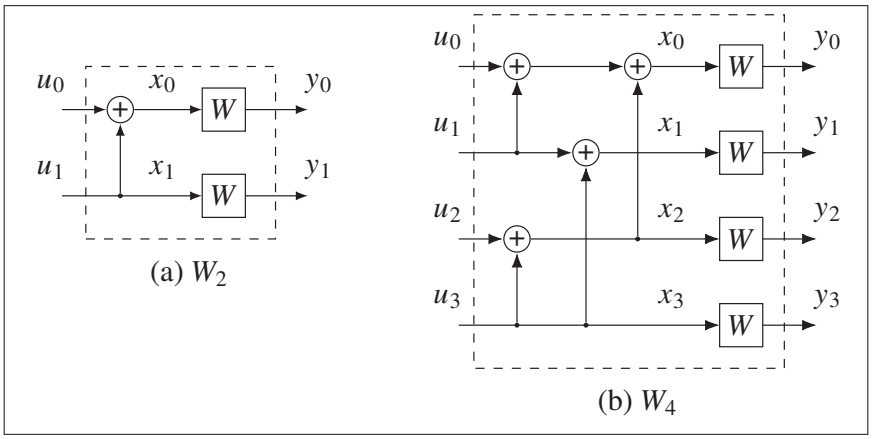


Figure 1.3 Construction of the channels W_N with $N = 2$ and $N = 4$

Such polarizing transformation can be combined recursively to create channels with longer codes, as shown in Figure 1.3 (b), for channel W_N with $N = 4$. As $N \to \infty$, the probability of successfully estimating each bit u_i approaches either 1 or 0.5, indicating perfectly reliable and completely unreliable bit-channels. The proportion of reliable bits approaches the symmetric capacity of W_N, as was shown by Arıkan (2009).

To construct an (N, k) polar code, the k most reliable positions are assigned to information bits, while the remaining $(N - k)$ bits, called frozen bits, are set to predefined values. These frozen bits, known by the decoder, are typically set to zero. The vector $\boldsymbol{u}$ contains the k information bits in their predefined locations and $(N - k)$ frozen bits. The set of information-bit indices is denoted by $\mathcal{A}$, and $\mathcal{A} = \{a_0, \ldots, a_j, \ldots, a_{k-1}\}$, with $a_0 < \cdots < a_j < \cdots < a_{k-1}$. Similarly, the set of frozen-bit indices is denoted by $\mathcal{A}^C$, and $\mathcal{A}^C = \{a^c_0, \ldots, a^c_j, \ldots, a^c_{N-k-1}\}$, with $a^c_0 < \cdots < a^c_j < \cdots < a^c_{N-k-1}$.

In the finite-length regime, not all bit-channel reliabilities correspond to 1 and 0.5, meaning they are not fully polarized. Therefore, an accurate ranking algorithm for reliabilities is required to select k most reliable channels, ensuring strong error-correction performance. The state-of-the-art works include the density evolution and the Gaussian approximation of density evolution methods, proposed by Tal & Vardy (2013) and Trifonov (2012). In these methods, the reliability ranking is determined for specific channel conditions.

Recent works on the partial reliability order, such as Mondelli, Hassani & Urbanke (2017) and Condo, Hashemi & Gross (2017), created an opportunity to derive a universal construction of polar codes independent of the channel conditions. Based on the results of these works, the 3GPP has derived a reliability sequence of 1024 bits, which is used as a basis to extract individual reliabilities. These reliabilities are then used to construct polar codes with $N \leq 1024$, as used in 3GPP's next-generation mobile-communication standard (5G) according to 3GPP (2018).

In the scope of this doctoral study, the proposed modifications to flip decoders demonstrated efficiency for any construction of the polar code, with negligible differences. Therefore, we selected the 5G code construction (3GPP, 2018) and presented all the results accordingly.

1.2.2 Encoding

An (N, k) polar code can be encoded by the generator matrix $\boldsymbol{G}^{\otimes n}$, where n is the Kronecker power of the binary kernel $\boldsymbol{G} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. The information-bit vector $\boldsymbol{u}$ is composed of information and frozen bits. Based on the reliability sequence of code construction, the k most reliable positions of $\boldsymbol{u}$ are assigned to information bits. The remaining $(N - k)$ bits are assigned to the frozen bits, and the code rate is $R = {}^{k}/_{N}$.

The encoding of the vector $\boldsymbol{u}$ to a codeword $\boldsymbol{x}$ is performed as follows:

$$\boldsymbol{x} = \boldsymbol{u}\boldsymbol{G}^{\otimes n}, \tag{1.9}$$

where vector operations are performed in the binary field.

The generator matrix $\boldsymbol{G}^{\otimes n}$ for a polar code of length $N = 2^3 = 8$ is:

$$\boldsymbol{G}^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.10}$$

Alternatively, polar encoding can also be performed with a factor graph. The encoder graph for $(8, 4)$ code is shown in Figure 1.4. The information propagates from the left to the right side of the graph. The vector $\boldsymbol{u}$ is entered to the left-hand side (LHS) of the graph with the set of information indices $\mathcal{A} = \{3, 5, 6, 7\}$ and the set of frozen indices $\mathcal{A}^C = \{0, 1, 2, 4\}$. Then, encoding is carried out through stages $s \in \{1, \ldots, n\}$, where $n = 3$ and $\oplus$ represents a bitwise XOR operation. Finally, the codeword $\boldsymbol{x}$ is output from the right-hand side (RHS) of the graph.

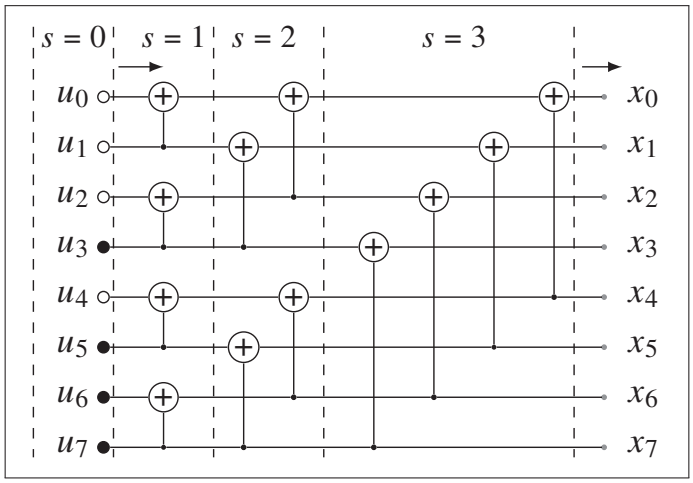


Figure 1.4 Encoder graph for an (8, 4) polar code

1.3 Successive-Cancellation (SC) Decoding

The successive-cancellation (SC) decoding can be represented through the information flow from the RHS to the LHS of the encoding graph as in Figure 1.4. Alternatively, the SC decoding can be effectively represented by a binary tree traversal, which was proposed by Alamdar-Yazdi & Kschischang (2011). This representation of an SC decoding is also used in this work. In this scheme, the tree is traversed from top to bottom and left to right along its branches. The SC decoding tree for an $(8, 4)$ polar code is depicted in Figure 1.5, where the stages are

denoted by $s \in \{n, \ldots, 0\}$ with the root node at the stage $s = n$. The received vector of channel LLRs, denoted by $\boldsymbol{\alpha}_{\text{ch}} = \{\alpha_{\text{ch}}(0), \ldots, \alpha_{\text{ch}}(N-1)\}$, is at the tree root.

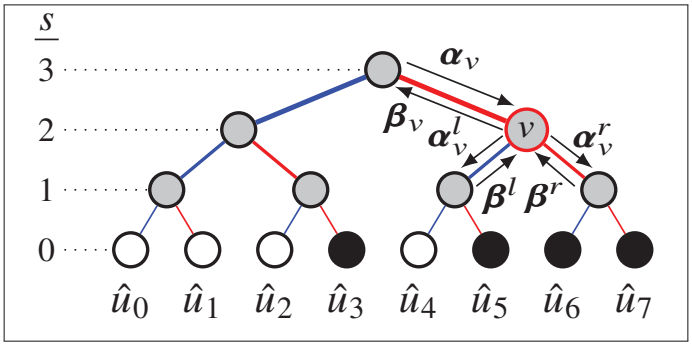


Figure 1.5 SC decoding tree for an $(8, 4)$ polar code with a focus on a node v of length $N_v = 4$

For any node, denoted by v and located at stage $s = s_v$, the input LLR vector $\boldsymbol{\alpha}_v = \{\alpha_v(0), \ldots, \alpha_v(2^{s_v} - 1)\}$ is received from the parent node, and two input partial-sum (PS) vectors $\boldsymbol{\beta}^l = \{\beta^l(0), \ldots, \beta^l(2^{s_v-1} - 1)\}$ and $\boldsymbol{\beta}^r = \{\beta^r(0), \ldots, \beta^r(2^{s_v-1} - 1)\}$ are received from its left and right child nodes, respectively. The left and right child nodes of the node v receive the vectors $\boldsymbol{\alpha}_v^l = \{\alpha_v^l(0), \ldots, \alpha_v^l(2^{s_v-1} - 1)\}$ and $\boldsymbol{\alpha}_v^r = \{\alpha_v^r(0), \ldots, \alpha_v^r(2^{s_v-1} - 1)\}$, respectively, where each LLR element is calculated as follows:

$$\alpha_v^l(j) = f\left(\alpha_v(j), \alpha_v\left(j + 2^{s_v-1}\right)\right), \tag{1.11}$$

$$\alpha_v^r(j) = g\left(\alpha_v(j), \alpha_v\left(j + 2^{s_v-1}\right), \beta^l(j)\right), \tag{1.12}$$

where $j \in \left\{0, \ldots, 2^{s_v-1} - 1\right\}$. Blue and red lines highlight the traversal to the left and right child nodes. The line thickness represents the vector sizes of the traversed data. Thus, a larger vector size corresponds to a thicker line. Going left, the f function is calculated.

The minimum function is introduced by Fossorier, Mihaljevic & Imai (1999), and it is further adapted to the SC decoder in the work of Leroux, Tal, Vardy & Gross (2011). Based on this, the f function is calculated as follows:

$$f(\alpha_1, \alpha_2) = \text{sgn}(\alpha_1)\,\text{sgn}(\alpha_2)\,\min(|\alpha_1|, |\alpha_2|), \tag{1.13}$$

where sgn (x) indicates the sign of value x, |x| indicates the absolute value of x, and min (x, y) returns the smaller value between x and y. Note that if x = 0, sign (x) returns 1, making it neutral to the overall outcome of the f function. Next, going right, the g function is calculated as:

$$g(\alpha_1, \alpha_2, \beta_1) = (1 - 2\beta_1)\,\alpha_1 + \alpha_2. \tag{1.14}$$

Bit estimates $\hat{\boldsymbol{u}} = \{\hat{u}_0, \ldots, \hat{u}_{N-1}\}$ are obtained by taking a hard decision on the decision LLRs, $\boldsymbol{\alpha}_{\text{dec}} = \{\alpha_{\text{dec}}(0), \ldots, \alpha_{\text{dec}}(N-1)\}$, that reach the leaf nodes. The hard decision is made based on BPSK as:

$$\hat{u}_i = \begin{cases} 0, \text{ if } \alpha_{\text{dec}}\,(i) \geq 0, \\ 0, \text{ if } i \in \mathcal{A}^C, \\ 1, \text{ otherwise.} \end{cases} \tag{1.15}$$

In Figure 1.5, the nodes represented in black and white correspond to information and frozen bits. The PS vector $\boldsymbol{\beta}$ is calculated from the bit estimates and is propagated up from the children to the parent nodes. At any node v, each bit of $\boldsymbol{\beta}_v$ is calculated as follows:

$$\beta_v\,(j) = \begin{cases} \beta^l\,(j) \oplus \beta^r\,(j)\,, \text{ if } j < 2^{s_v - 1}, \\ \beta^r\,(j)\,, \qquad \text{otherwise,} \end{cases} \tag{1.16}$$

where $j \in \{0, \ldots, 2^{s_v} - 1\}$.

1.4 Existing Fast Decoding Techniques

The construction of polar codes leads to special nodes, i.e., nodes at stages $s \geq 2$ with a recognizable pattern of frozen and information bits in SC decoding tree (Alamdar-Yazdi & Kschischang, 2011). A special node is denoted by v with the length $N_v = 2^{s_v - 1}$ with $N_v \geq 4$. A special node does not need to be traversed until the leaf node at stage $s = 0$. Instead, it can be decoded by a fast decoding technique that would significantly reduce computational complexity and decoding time. The Simplified SC (SSC) decoder, proposed by Alamdar-Yazdi & Kschischang (2011), does not traverse special nodes entirely composed of either frozen bits or information bits. These nodes are denoted by rate-0 (R0) and rate-1 (R1) nodes. The Fast-SSC decoder, proposed by Sarkis, Giard, Vardy, Thibeault & Gross (2014), further improves the decoding time of SSC by implementing additional types of special nodes. The most notable of these node types are the repetition (REP) and the single-parity-check (SPC) nodes.

The Fast-SSC with these four types of special nodes was adapted to successive-cancellation flip (SCF) decoding, and the Fast-SCF decoder was proposed by Giard & Burg (2018). The hardware implementation of the Fast-SCF decoding was proposed by Ercan, Tonnellier & Gross (2020c). Next, the Fast-dynamic successive-cancellation flip (DSCF) decoding algorithm and its hardware implementation were proposed by Ercan *et al.* (2020a).

Throughout this doctoral study, we also use these types of special nodes. The vector of PS bits $\boldsymbol{\beta}_v$ (1.16) is output from these nodes, and calculated according to the original schedule of SC decoding. Next, we describe the decoding schedule of four types of special nodes used in Fast-SSC.

1.4.1 Decoding of Special Nodes

Rate-0 (R0) and Rate-1 (R1) Nodes

A R0 node is entirely composed of frozen bits, and all N_v bit estimates are assigned to zero. A R1 node is entirely composed of information bits, and all N_v bit estimates are obtained by taking the hard decisions on the input LLR vector $\boldsymbol{\alpha}_v$ according to (1.15).

Repetition (REP) Node

A REP node of length N_v is composed of a single information bit at the right-most position and $N_v - 1$ frozen bits. The decision LLR α_{dec} for this information bit is calculated from the input vector $\boldsymbol{\alpha}_v$ according to Sarkis *et al.* (2014):

$$\alpha_{\text{dec}} = \sum_{i=0}^{N_v-1} \alpha_v(i) . \tag{1.17}$$

The bit estimate is obtained by taking a hard decision on α_{dec} of (1.17) according to (1.15).

Single-Parity-Check (SPC) Node

An SPC node of length N_v is composed of a single frozen bit at the left-most location and $N_v - 1$ information bits. Decoding is made by taking hard decisions on all elements of $\boldsymbol{\alpha}_v$ and satisfying parity constraint $p = 0$. The bit indices in the SPC node are denoted by $i \in \{0, \ldots, N_v - 1\}$, where the frozen bit is at $i = 0$. The SPC node decoding can be defined according to Sarkis *et al.* (2014) and Ercan *et al.* (2020c) as follows:

$$\hat{u}_i = \begin{cases} 0, \text{ if } i = 0, \\ 0, \text{ if } i > 0 \text{ and } \alpha_v(i) \geq 0, \\ 1, \text{ if } i > 0 \text{ and } \alpha_v(i) < 0, \end{cases} \quad \text{and} \quad p = \bigoplus_{i=0}^{N_v-1} \hat{u}_i . \tag{1.18}$$

If parity constraint $p = 1$ in (1.18), the least reliable information bit according to $|\alpha_v(d)|$ is flipped.

The Fast-SSC decoding is denoted by FSSC . The decoding tree of the $(8, 4)$ polar code, previously shown in Figure 1.5, can now be simplified. The resulting FSSC decoding tree composed of one REP node and one SPC node, both of lengths $N_v = 4$, is shown in Figure 1.6. The resulting tree shows how the SC tree with the total of 4 stages and 14 traversing edges is now simplified to the FSSC tree with only 2 stages and 2 traversing edges. Going left, the f function is calculated (highlighted in blue line) according to (1.11), and then the REP node is decoded. Next, going right, the g function is calculated (highlighted in red line) according to (1.12), and then the SPC node is decoded.

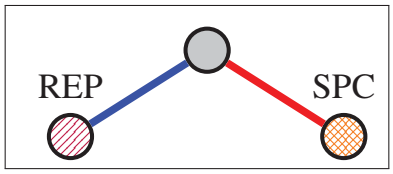


Figure 1.6 Fast-SSC decoding tree for an $(8, 4)$ polar code. Highlighted nodes are repetition (REP) and single-parity-check (SPC) nodes. Taken from the work of Sarkis *et al.* (2014)

1.4.2 Latency-Reducing Technique (LRT)

In their study, (Giard *et al.*, 2017) proposed an implementation of the SCF decoder, where each SC trial is modified by the latency-reducing technique (LRT). This technique is entirely based on skipping traversal of the left-most frozen bits in the decoding tree, meaning that it resumes the decoding at the first information-bit location a_0. The SC trial with the LRT is denoted by $\mathrm{SC_{LRT}}$. The $\mathrm{SC_{LRT}}$ decoding tree is depicted in Figure 1.7, where the nodes represented in black and white correspond to information and frozen bits. The nodes corresponding to the frozen bits at the LHS of the first information bit are visualized by the dashed line. These nodes are not traversed in $\mathrm{SC_{LRT}}$.

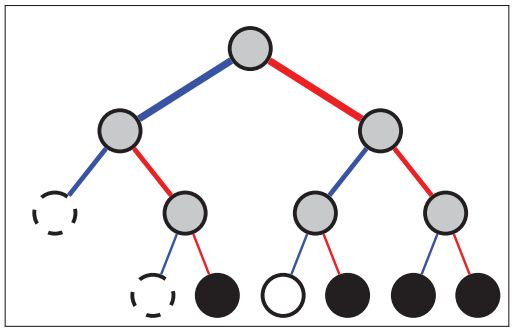


Figure 1.7 $\mathrm{SC_{LRT}}$ decoding tree for an $(8, 4)$ polar code. Dashed nodes are skipped from decoding. Based on the work of Giard *et al.* (2017)

Contrary to SC decoding, the execution times of FSSC and $\mathrm{SC_{LRT}}$ depend on a pattern of frozen and information bits. This applies to the changes in the code construction or the code rate R.

1.5 Successive-Cancellation List (SCL) Decoding

The successive-cancellation list (SCL) decoding algorithm, which decodes a codeword by generating a list of L candidates, was proposed by Tal & Vardy (2015). The SCL significantly improves the error-correction performance to the extent that polar codes were selected to protect the control channel of the enhanced mobile broadband (eMBB) service in 5G. During the SCL decoding, when the information bit $\hat{u}_i$, $i \in \mathcal{A}$, is estimated, both possible decisions

$\hat{u}_i \in \{0, 1\}$ are considered, i.e., the path split is made. The number of paths and thus decoding instances double after the estimation of each information bit. For the bit locations $i \in \{0, \ldots, \log_2 L - 1\}$, $i \in \mathcal{A}$, all possible path splits are kept. When $i \geq \log_2 L$, only L out of $2L$ possible paths are kept to continue decoding according to the path metrics. For each information-bit location i, the path metric, denoted by $PM_i\,(l)$, is calculated as follows (Stimming, Parizi & Burg, 2015):

$$PM_i\,(l) = \begin{cases} PM_{i-1}\,(l) + |\alpha_{\text{dec}}\,(i)|\,, & \text{if } \hat{u}_i \neq \frac{1-\text{sgn}(\alpha_{\text{dec}}(i))}{2}, \\ PM_{i-1}\,(l)\,, & \text{otherwise,} \end{cases} \tag{1.19}$$

where l denotes the path index, $l \in \{0, \ldots, 2L - 1\}$ and $\alpha_{\text{dec}}\,(i)$ is decision LLR of the bit $\hat{u}_i$. The expression $\hat{u}_i \neq \frac{1-\text{sgn}(\alpha_{\text{dec}}(i))}{2}$ in (1.19) implies checking whether the bit $\hat{u}_i \in \{0, 1\}$ corresponds to the sign of the decision LLR, denoted by $\text{sgn}\,(\alpha_{\text{dec}}\,(i))$, which ultimately corresponds to the BPSK mapping scheme of the digital signal y_i provided in (1.1).

The $2L$ path metrics are calculated according to (1.19) and then sorted in ascending order. After sorting, the L smallest path metrics are selected and stored to $\mathbf{PM} = \{PM\,(0)\,, \ldots, PM\,(l)\,, \ldots, PM\,(L-1)\}$. The paths associated with these path metrics are stored in the structure $\hat{\boldsymbol{\mathcal{U}}} = \{\hat{\boldsymbol{u}}_0, \ldots, \hat{\boldsymbol{u}}_l, \ldots, \hat{\boldsymbol{u}}_{L-1}\}$, while the remaining L paths are discarded. The SCL decoder continues decoding the L candidate paths $\hat{\boldsymbol{\mathcal{U}}}$ by SC decoding. When the next bit $\hat{u}_i$ is estimated, the $2L$ path metrics are calculated according to (1.19) and then sorted. Then, the same path-selecting strategy explained above is applied. Note that when estimating any frozen bit $\hat{u}_i \in \mathcal{A}^C$, all paths are extended with only one decision $\hat{u}_i = 0$, followed by the same path metric calculation according to (1.19) and sorting.

In the initial study of the SCL decoder, Tal & Vardy (2015) have shown that if a codeword is concatenated with a CRC, the error-correction performance of the SCL can be significantly improved. In our study, a block of information bits is concatenated with an r-bit CRC code before applying polar encoding. The CRC bits are concatenated to the set of information bits $\mathcal{A}$, increasing the number of information bits of the polar code to $k_{\text{tot}} = k + r$. The notations (N, k) and $(N, k + r)$ are used throughout this doctoral study to indicate a polar code of length N, code dimension k, and r additional CRC bits. At the end of SCL decoding, the CRC is performed for all L paths. The path with the smallest path metric out of L paths that passes the CRC is selected as the final bit estimates $\hat{\boldsymbol{u}}$ of the decoder.

The SC and SCL decoders for polar codes are implemented to analyze the error-correction performance. For this analysis, the 5G polar codes (3GPP, 2018) of length $N = 1024$ with the rate $R = 1/2$ are used. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. The CRC is applied by the SCL decoder, and it increases the code dimension to $k_{\text{tot}} = 512 + 11 = 523$ bits. The CRC is not applied by the SC decoder, thus the code dimension

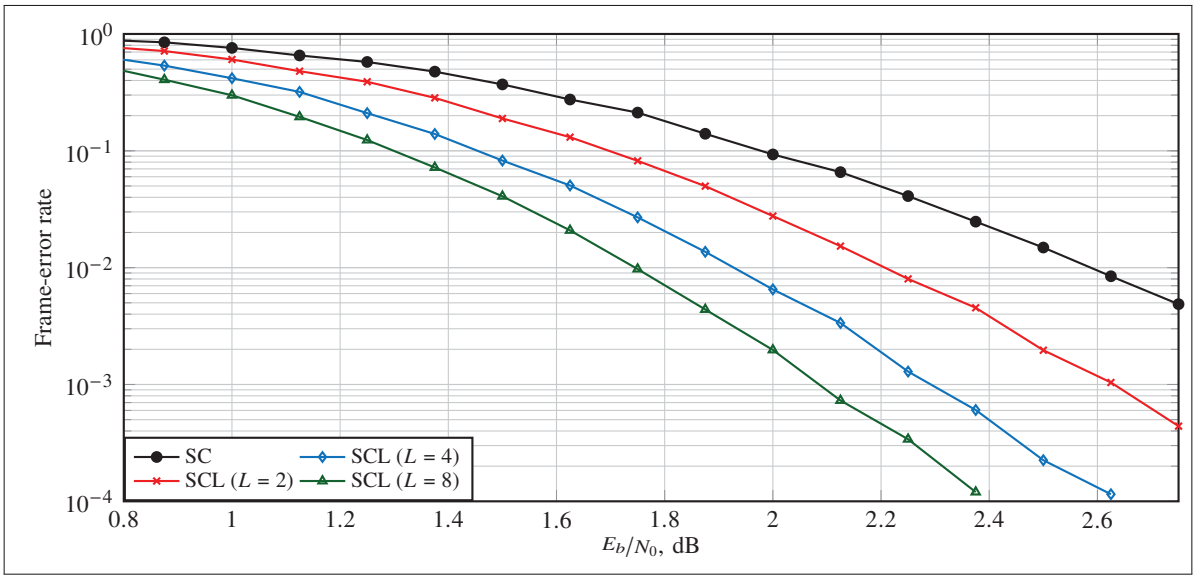


Figure 1.8 Error-correction performance of SC vs SCL decoders for a $(1024, 512)$ polar code. An 11-bit CRC is applied by SCL decoder

remains $k = 512$ bits. The number of processing elements is set to $P = 64$, following the recommended value for a code with $N = 1024$ as defined by Leroux, Raymond, Sarkis & Gross (2013). The list size of $L \in \{2, 4, 8\}$ is used by the SCL decoder. Simulations are made over the AWGN channel with BPSK modulation. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed for each E_b/N_0 point. The error-correction performance is analyzed using the frame-error rate (FER), i.e., the ratio of incorrectly received messages to the total transmitted messages. Throughout this doctoral study, the analysis is focused on the FER. The target FER is 10^{-2}, a typical region of interest for wireless communications.

Figure 1.8 depicts the error-correction performance for SC and SCL decoders. The x-axis represents E_b/N_0, while the y-axis represents the FER. The results show that SCL significantly outperforms SC decoder. As the list size L increases, the error-correction performance of the SCL decoder improves. The performance of SCL decoder with $L = 8$ represents the baseline for the 5G performance evaluation 3GPP (2018). Note that the Fast-SSC decoder achieves the same error-correction performance as the SC decoder but with significantly lower decoding latency.

1.6 Hardware Implementations of SC and SCL Decoders

In this section, implementation results are presented to demonstrate the tradeoff between implementation cost and error-correction performance for SC and SCL decoders.

The first application-specific integrated circuit (ASIC) implementation of the SC decoder was proposed by Mishra, Raymond, Amaru, Sarkis, Leroux, Meinerzhagen, Burg & Gross (2012). Then, the semi-parallel SC decoder,

which is used as a basis in the majority of subsequent works, was proposed by Leroux *et al.* (2013). The Fast-SSC decoder was proposed by Sarkis *et al.* (2014), followed by the hardware implementation proposed by Ercan *et al.* (2020c). The SCL decoder was proposed by Stimming *et al.* (2015), followed by the high-throughput SCL decoder (ChenYang, Ji, YouZhe, Chi-ying, Jie, Hui & Bin, 2018) and Fast-SSCL-SPC decoder (Hashemi, Condo & Gross, 2017).

Table 1.1 summarizes various existing implementations of SC, Fast-SSC and SCL decoders. The results are presented by normalizing the values to the 65 nm technology, using the scaling methodology presented by Stillmaker & Baas (2017). Given that the technology nodes of the compared works exceed 45 nm, we apply traditional geometry-based scaling methods. The technology coefficient s is found as the ratio of the target technology to the original technology. Then, the scaling parameters for area, frequency, and power are provided. To scale the results for SCL decoders with different list sizes L, the list-size ratio l is applied.

Table 1.1 Summary of implementations and scaling parameters to the target 65 nm technology

Decoder	Reference	Technology	Scaling			
			Coefficient	Area	Frequency	Power
SC	Mishra *et al.* (2012)	180 nm	$s = 65/180$	$s^2 l$	$1/s$	s^2
Fast-SSC	Ercan *et al.* (2020c)	65 nm	$s = 1$			
SCL	ChenYang *et al.* (2018)	90 nm	$s = 65/90$			
Fast-SSCL-SPC	Hashemi *et al.* (2017)	65 nm	$s = 1$			

Table 1.2 provides implementation results for works listed in Table 1.1 for a $(1024, 512)$ polar code. All results are scaled to 65 nm technology. Also, for all SCL decoders, results are scaled to $L = 4$. Note that the provided values for coded throughput are measured considering all N bits in a codeword.

Table 1.2 Hardware comparisons of state-of-the-art SC, Fast-SSC, and SCL decoders

	SC[(a)]	Fast-SSC	SCL[(a)]	Fast-SSCL-SPC
List size L	–	–	4	4
Area (mm^2)	0.22	0.38	2.16	1.82
Frequency (MHz)	415	455	558	840
Coded throughput (Mbps)	271	1557	1540	1608
Area Efficiency (Mbps/mm^2)	1217	4100	714	883
Power (mW)	67	79.90	–	517
Energy per bit (pJ/bit)	247	51.32	–	321

[(a)] Scaled to 65 nm technology using the scaling parameters provided in Table 1.1

The results for various SCL decoder implementations show that the chip area is increased by more than 100% compared to SC and Fast-SSC decoders. Moreover, the required area nearly doubles when doubling the list size, as noted by Stimming *et al.* (2015) and Hashemi *et al.* (2017). The SCL decoders also have more power consumption

and require more energy per bit. Despite significant improvement in error-correction performance, the SCL decoder is less energy- and area-efficient for hardware implementation compared to the SC decoder.

When comparing the results for the SC and Fast-SSC decoders, the Fast-SSC indicates significantly better coded throughput at the cost of a small area overhead. This resulted in better area efficiency and smaller energy per bit. However, the Fast-SSC decoder does not improve error-correction performance compared to the SC decoder.

Decoder Area

The area of the chip primarily determines its fabrication cost and energy requirements. Therefore, it is important to analyze the area and identify the largest blocks of the decoder. The area breakdown for the SC decoder is provided in the work of Leroux *et al.* (2013). Similarly, the area breakdown for the SCL decoder is provided in the work of Stimming *et al.* (2015). Table 1.3 summarizes the area breakdowns reported in these works. The first column provides the total area in absolute values, while the subsequent columns show the area distribution in percent between different parts of the decoder.

The results in Table 1.3 show that memory occupies the largest fraction of the total area of the chip. The memory accounts for approximately 90% of the area in both decoders. For the SC decoder, a more detailed breakdown of the memory usage is reported. This breakdown indicates that nearly 80% of the memory is allocated to blocks storing the LLR values.

Table 1.3 Area breakdown for the SC and SCL decoder implementations for $N = 1024$ polar code

Decoder	Total area mm^2	Memory %		Decoder core[(c)] %	Controller %	Sorter %	Other %
SCL[(a)(b)]	0.91	89.70		6.30	–	0.53	3.43
		LLR	PS				
SC	0.31	76.10	17.52	4.77	0.48	–	1.13

[(a)] For the list size $L = 4$
[(b)] Scaled to 65 nm technology using the scaling parameters provided in Table 1.1
[(c)] Using $P = 64$ processing elements

This doctoral study contributes to the design of energy-efficient flip decoders of polar codes. In particular, the proposed mechanisms improve the execution-time characteristics of the flip decoders. To analyze time and resource requirements, the methodology is derived based on existing hardware implementations. The memory overhead is carefully analyzed to estimate the impact of our proposed modifications. According to the area breakdown provided in Table 1.3, the memory overhead can be directly linked to area overhead in a potential hardware implementation. Therefore, minimizing memory overhead is crucial for achieving energy- and area-efficient decoder implementations.

1.7 Conclusion

The blocks of the channel coder and decoder ensure the robustness of the transmitted information in the digital communication system. Polar codes are linear block codes proven to asymptotically achieve the channel capacity in practically relevant channels under low-complexity SC decoding. However, for the codes with finite lengths, the error-correction performance of the SC decoder is lacking for many practical applications. The SCL decoder was proposed to improve the error-correction performance of the SC decoder by decoding a codeword by generating a list of L candidates. The simulation results of the SC and SCL decoders show that the SCL decoder significantly outperforms the SC when $L \geq 2$. However, the analysis of the implementation results indicates that the SCL decoder and its variations have a larger area and higher power consumption compared to the SC decoder.

In this doctoral study, the focus is on the area- and energy-efficient decoder implementations that can achieve the same error-correction performance as the SCL decoder. The SCF decoder is based on reusing a single instance of SC over multiple decoding attempts. The SCF decoder and its variations were shown to achieve the error-correction performance of the SCL decoder with the medium list sizes. The next chapter provides the background and basis of the flip decoding algorithms based on SCF.

CHAPTER 2

BACKGROUND AND BASIS OF FLIP DECODING ALGORITHMS

2.1 Successive-Cancellation Flip (SCF) Decoding

The SCF decoding algorithm was introduced by Afisiadis *et al.* (2014). The authors observed that detecting and correcting the first erroneously estimated bit before resuming SC decoding highly improves the error-correction performance of the decoder. To detect a decoding failure, information bits are concatenated with an r-bit CRC code before applying polar encoding, similarly to the SCL decoding.

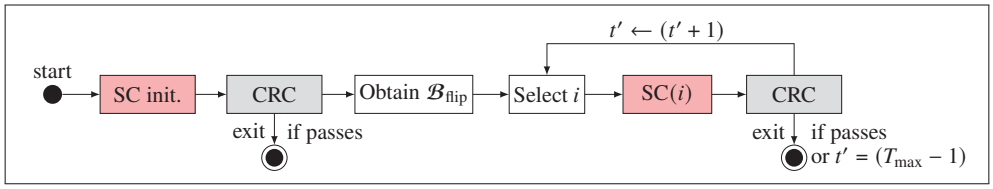


Figure 2.1 Block-diagram of SCF decoding algorithm

The block diagram of the SCF decoder is depicted in Figure 2.1. The initial state is represented by a filled circle with an arrow leading from it. The final state is represented by a filled circle nested inside another circle, with an arrow pointing to it. The decoding operations are represented by rectangle blocks. Arrows indicate the transitions between the operations. The SC decoding blocks for the initial and the additional trials are highlighted in red. The CRC blocks are highlighted in gray.

The SCF algorithm starts by decoding a codeword with the initial SC pass, followed by the CRC. If the CRC passes, decoding exits with success. Otherwise, a bit-flipping list $\mathcal{B}_{\text{flip}} = \left\{a_0, \ldots, a_{k_{\text{tot}}-1}\right\}$ that contains all the information bits is constructed. The information-bit locations, that are estimated as the least reliable according to the smallest metrics $\mathcal{M}_{\text{flip}} = \left\{\left|\alpha_{\text{dec}}\left(a_0\right)\right|, \ldots, \left|\alpha_{\text{dec}}\left(a_{k_{\text{tot}}-1}\right)\right|\right\}$, where $|\text{x}|$ indicates the absolute value of x, are identified. Then, the bit-flip location $i \in \mathcal{B}_{\text{flip}}$ is selected, and the additional trial, denoted by SC(i), is performed. During this additional trial, SC decoding restarts from the beginning, and when the bit-flipping candidate i is estimated, i.e., the bit $\hat{u}_i = \text{HD}\left(\alpha_{\text{dec}}\left(i\right)\right)$, the decision is inverted. The standard decoding is resumed for the remaining part of the SC decoding in this trial.

To constrain the latency, the maximum number of trials $T_{\max}$ is defined, where $T_{\max} \in \mathbb{N}^+$ and $1 \leq T_{\max} \leq (k_{\text{tot}} + 1)$, including the initial SC pass. Thus, the sizes of the bit-flipping list and the corresponding metrics are constrained to $\left|\mathcal{B}_{\text{flip}}\right| = \left|\mathcal{M}_{\text{flip}}\right| = T_{\max} - 1$. The *required additional* number of trials, denoted by τ', corresponds to the trials beyond the initial decoding trial that were applied to decode a codeword. The index t' denotes the current index of

the additional trial throughout decoding a codeword, and $1 \leq t' \leq \tau'$. Similarly, τ and t denote the required number of trials and the current trial, including the initial SC pass. If the CRC still fails after running $t' = T_{\max} - 1$ (or $t = T_{\max}$) trials, decoding is stopped and a frame error is declared. Note that $\tau' = 0$ indicates a successful decoding by the initial SC pass.

2.2 Dynamic Successive-Cancellation Flip (DSCF) Decoding

The DSCF decoding algorithm was proposed by Chandesris *et al.* (2018) with two major improvements to the original SCF. First, a novel metric for constructing $\mathcal{B}_{\text{flip}}$ is derived, which increases the probability of determining the bit-flip position that results in successful decoding with smaller $T_{\max}$. Second, an algorithmic improvement allowing multiple simultaneous bit flips per trial is proposed. The block diagram of the DSCF decoder is shown in Figure 2.2. In this diagram, the extra steps and modifications from the SCF decoding, provided in Figure 2.1, are surrounded by a dashed line.

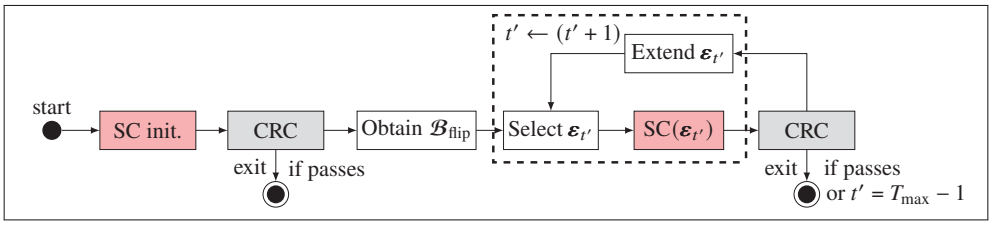


Figure 2.2 Block-diagram of DSCF decoding algorithm

A bit-flipping list $\mathcal{B}_{\text{flip}} = \left\{\boldsymbol{\varepsilon}_0, \ldots, \boldsymbol{\varepsilon}_{t'}, \ldots, \boldsymbol{\varepsilon}_{T_{\max}-1}\right\}$ now consists of sets of bit-flipping candidates that enable the multi-bit flipping approach. A set of bit-flipping candidates is denoted by $\boldsymbol{\varepsilon}_{t'} = \left\{i_1, \ldots, i_j, \ldots, i_\lambda\right\}$, where $i_j \in \mathcal{A}$ is a bit-flip location, $1 \leq j \leq \lambda$, and λ is the current set size $|\boldsymbol{\varepsilon}_{t'}|$. At each unsuccessful attempt t', a new information index j is progressively inserted to an extended set $\boldsymbol{\varepsilon}_{\text{ext}} = (\boldsymbol{\varepsilon}_{t'} \cup j)$, increasing its size to $\lambda = \lambda + 1$, and the metric $\mathcal{M}_{\text{flip}}(t') \in \boldsymbol{\mathcal{M}}_{\text{flip}}$ is calculated as:

$$\mathcal{M}_{\text{flip}}(t') = \sum_{j \in \boldsymbol{\varepsilon}_{\text{ext}}} |\alpha_{\text{dec}}(j)| + \sum_{\substack{j \leq i_\lambda \\ j \in \mathcal{A}}} \mathcal{J}\left(\alpha_{\text{dec}}(j)\right), \tag{2.1}$$

where $\mathcal{J}$ can be approximated according to Ercan, Tonnellier, Doan & Gross (2020b) as:

$$\mathcal{J}\left(\alpha_{\text{dec}}(j)\right) = \begin{cases} 1.5, & \text{if } |\alpha_{\text{dec}}(j)| \leq 5.0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

If the metric of a newly constructed set $\boldsymbol{\varepsilon}_{\text{ext}}$ exceeds the largest metric in $\mathcal{B}_{\text{flip}}$, it is discarded. If not, it is inserted to $\mathcal{B}_{\text{flip}}$ while keeping the list sorted in ascending order according to each $\mathcal{M}_{\text{flip}}(t')$. The name dynamic in DSCF stands for dynamic updates of the bit-flipping sets following each unsuccessful additional decoding trial.

A set is not extended further after reaching the maximum size ω. Throughout this doctoral study, (2.2) is used as it was shown to result in a negligible loss in error-correction performance while reducing computational overhead, as explained by Ercan *et al.* (2020b) and Ercan *et al.* (2020a). The decoding order ω defines the maximum size of the bit-flipping set $\boldsymbol{\varepsilon}_{t'}$ as $1 \leq \lambda \leq \omega$, i.e., $\omega = |\boldsymbol{\varepsilon}_{t'}|_{\max}$, according to Chandesris *et al.* (2018). If $\omega = 1$, the bit-flipping candidates are constructed at the initial unsuccessful SC trial as in SCF, but with the metric (2.1) with $\lambda = 1$, and no additional metric computations are performed.

The decoder is denoted by DSCF-ω to emphasize the dependence on the parameter ω. A total of $T_{\max}$ trials are run with a total of $T_{\max} - 1$ bit-flipping sets. Note that $T_{\max}$ may be higher than $(k_{\text{tot}} + 1)$ when the order is $\omega > 1$, since in those cases, sets containing multiple bit-flips can be constructed.

2.2.1 Error-Correction Performance Analysis

The SCF and DSCF decoders are implemented to analyze the error-correction performance. For this analysis, the 5G polar codes (3GPP, 2018) of the length $N = 1024$ with the rate $R = 1/2$ are used. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. The CRC is applied by the SCF, DSCF, and SCL decoders. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. Simulations are made over the AWGN channel with BPSK modulation. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed for each E_b/N_0 point. The hardware-friendly function (2.2) is applied to all DSCF decoders. For the DSCF-ω decoders, $\omega \in \{1, 2, 3\}$ is applied. Applying $\omega > 3$ does not result in significant error-correction improvement while increasing computational complexity, as was explained by Chandesris *et al.* (2018).

Figure 2.3 shows error-correction performance of SCF and DSCF-1 decoders. The maximum number of trials $T_{\max} \in \{13, 32\}$ are applied to SCF, and $T_{\max} = 8$ is applied to DSCF-1. The results show that DSCF-1 decoder with $T_{\max} = 8$ outperforms SCF with $T_{\max} = 13$. The results also indicate that both SCF and DSCF-1 decoders are able to achieve the error-correction performance of SCL with $L = 2$. However, DSCF-1 decoder can achieve it with much smaller $T_{\max} = 8$, while SCF can achieve it only with $T_{\max} = 32$. In this study, $T_{\max} = 13$ is set to the SCF decoder, which is close to $T_{\max} = 8$ in the DSCF-1 decoder.

The performance gain of DSCF-1 decoder is observed due to an improved metric function, which increases the probability of determining the bit-flip position that results in successful decoding with smaller $T_{\max}$. For both SCF and DSCF-1 decoders, increasing $T_{\max}$ further does not result in significant error-correction improvement at the

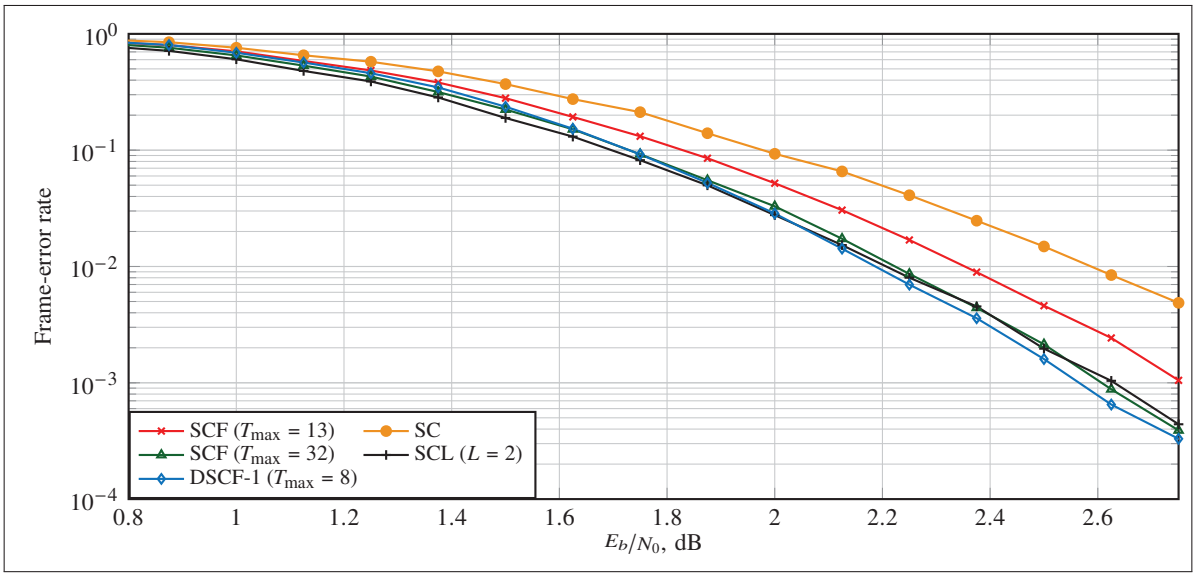


Figure 2.3 Error-correction performance of SCF and DSCF-1 decoders for a $(1024, 512 + 11)$ polar code

target FER of 10^{-2}, as the decoders reach their performance limits. This was explained in the work of Chandesris *et al.* (2018) with the illustration that the single-bit flip decoders can achieve the performance of the genie-aided decoder, corresponding to the performance bound. However, multi-bit flip decoding algorithms are required to break through this bound.

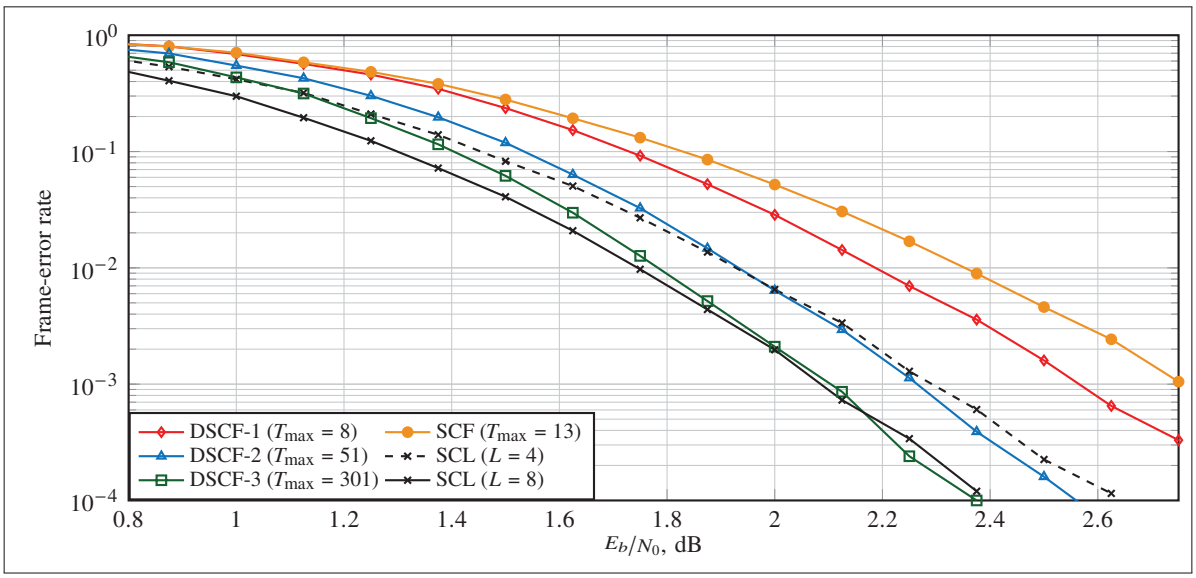


Figure 2.4 Error-correction performance of DSCF-ω decoder with the decoding order $\omega \in \{2, 3\}$ for a $(1024, 512 + 11)$ polar code

Figure 2.4 depicts error-correction performance of DSCF-ω decoders with the decoding orders $\omega \in \{1, 2, 3\}$. The maximum number of trials $T_{\max} \in \{8, 51, 301\}$ is chosen for each ω. Similarly to DSCF-1 decoder, the values of $T_{\max}$ are selected to achieve the error-correction performance that is close to the genie-aided decoders for $\omega \in \{2, 3\}$ at the target FER of 10^{-2} (Chandesris *et al.*, 2018). The results show that the multi-bit flip DSCF-ω decoders achieve significantly better error-correction performance than the single-bit flip DSCF-1 decoder. Finally, the results show that the DSCF-2 decoder matches the error-correction performance of the SCL decoder with $L = 4$, and that the DSCF-3 matches the performance of SCL $L = 8$ with only around 0.05 dB loss at FER of 10^{-2}. The latter property indicates that DSCF-3 decoder fits the requirements for the 5G performance evaluation for the eMBB service in 5G (3GPP, 2018).

To conclude the results, SCF decoder decodes a codeword with multiple SC decoding attempts, where at each attempt, one bit is flipped before resuming SC decoding. The DSCF decoding algorithm is a major improvement to SCF, where a novel metric to construct the list of bit-flipping candidates is derived. Also, multiple simultaneous bit flips per trial are performed. The DSCF-3 decoder can achieve the error-correction performance of SCL decoder with $L = 8$, which fulfills the requirements of the eMBB service in 5G. However, this comes at a cost of an increased $T_{\max}$, leading to higher average and worst-case execution times. Nevertheless, existing hardware implementations, such as Ercan *et al.* (2020c) and Ercan *et al.* (2020a), have shown that SCF and DSCF decoders are more efficient than SCL decoders in terms of area and energy per bit, at the same FER. Therefore, DSCF-3 decoder is considered the reference algorithm in this doctoral study.

2.3 Successive-Cancellation List-Flip (SCLF) Decoding

The successive-cancellation list flip (SCLF) decoding was first proposed by Yongrun *et al.* (2018) with the idea of combining the SCL and SCF decoding strategies. This strategy allows for a balance between parallel and sequential computations to achieve the target FER.

At the initial decoding trial, standard SCL decoding is performed. If none of the L paths pass the CRC at the end of this decoding trial, a *path-flipping list* $\mathcal{B}_{\text{flip}}$ is constructed. The additional SCL trial t' is then performed by following the standard SCL algorithm until the location $i = \mathcal{B}_{\text{flip}}(t')$ is met. When $\hat{u}_i$ is estimated, the path flipping is made, i.e., the paths that were initially discarded by a standard SCL are kept to continue decoding. The path-flipping strategy for the SCLF decoding was introduced by Cheng *et al.* (2019). This strategy is used throughout this doctoral study. The standard SCL decoding is then resumed to decode the remaining bits in a codeword. The SCLF decoder has the $T_{\max}$ trials to decode a codeword, and if failed, the codeword is deemed undecodable.

The path-flipping locations are obtained based on the metric $\mathcal{F}_{\varkappa}(t')$, which is calculated for each information bit-location i according to Pan, Wang & Ueng (2020), as follows:

$$\mathcal{F}_{\varkappa}(t') = \ln\left(\frac{\sum_{l=0}^{L-1}\exp\left(-PM_i(l)\right)}{\left(\sum_{l=0}^{L-1}\exp\left(-PM_i(l+L)\right)\right)^{\varkappa}}\right), \tag{2.3}$$

where "ln" denotes the natural logarithm, and the path metric $PM_i(l)$ is calculated for each path $l \in \{0,\ldots,2L-1\}$ according to (1.19). The constant value $\varkappa \approx 1.2$ is found by simulations as described by Pan *et al.* (2020).

The metric in (2.3) can be approximated by the differential function as shown by Ivanov, Morishnik & Krouk (2021):

$$\mathcal{F}'_{\varkappa}(t') = -PM_i(0) + \varkappa \cdot PM_i(L), \tag{2.4}$$

where $\varkappa \approx 1.0$ was derived by simulations (Ivanov *et al.*, 2021). According to our simulation results, the metric approximation in (2.4) results in a negligible error-correction loss of less than 0.01 dB at a FER of 10^{-2}. Thus, the SCLF decoder with the metric in (2.4) is used in this thesis. When calculated, the metric is stored in the list of path-flipping metrics in a position that keeps the list sorted in ascending order. This list is denoted by $\boldsymbol{\mathcal{M}}_{\text{flip}}$, similarly to SCF decoding, as both lists share the same properties. Note that candidate paths are evaluated for bit-indices where the path splits and selections are made, i.e., for $i \geq \log_2(L)$, $i \in \mathcal{A}$. Also note that $\mathcal{F}'_{\varkappa} \geq 0$ in (2.4) holds, given that the vector $\boldsymbol{PM}_i$ is sorted in ascending order, as described in Section 1.5 (for SCL decoding).

Similarly to SCF, the maximum number of trials is $T_{\max}$, and the maximum number of additional trials is $T_{\max}-1$. Thus, the sizes of the path-flipping list and the corresponding metrics are constrained to $\left|\boldsymbol{\mathcal{B}}_{\text{flip}}\right| = \left|\boldsymbol{\mathcal{M}}_{\text{Lflip}}\right| = T_{\max}-1$. If the CRC still fails after $t' = T_{\max}-1$ trials, the decoding is stopped, and a frame error is declared.

2.4 Dynamic Successive-Cancellation List-Flip (DSCLF) Decoding

The dynamic successive-cancellation list flip (DSCLF) decoding was proposed by Shen *et al.* (2022), where the DSCF algorithm strategy was adapted to SCLF. A path-flipping list $\boldsymbol{\mathcal{B}}_{\text{flip}} = \{\boldsymbol{\varepsilon}_0,\ldots,\boldsymbol{\varepsilon}_{t'},\ldots,\boldsymbol{\varepsilon}_{T_{\max}-1}\}$ now consists of sets of path-flipping candidates that enable the multi-path flipping approach. A set of path-flipping candidates is denoted by $\boldsymbol{\varepsilon}_{t'} = \{i_1,\ldots,i_j,\ldots,i_\lambda\}$, where $i_j \in \mathcal{A}$ is a path-flip location, $1 \leq j \leq \lambda$, and λ is the current set size $|\boldsymbol{\varepsilon}_{t'}|$. At each unsuccessful decoding attempt t', a new information index j is progressively inserted to an extended set $\boldsymbol{\varepsilon}_{\text{ext}} = (\boldsymbol{\varepsilon}_{t'} \cup j)$, increasing its size to $\lambda = \lambda + 1$, and the metric $\mathcal{M}_{\text{Lflip}}(t') \in \boldsymbol{\mathcal{M}}_{\text{Lflip}}$ is calculated according to Shen *et al.* (2022) as follows:

$$\mathcal{M}_{\text{Lflip}}(t') = \sum_{j\in\boldsymbol{\varepsilon}_{\text{ext}}} \mathcal{F}_{\varkappa}(t') + \sum_{\substack{j\leq i_\lambda \\ j\in\mathcal{A}}} \frac{1}{z}\cdot\ln\left(1+\exp\left(-z\cdot\mathcal{F}_{\varkappa}(t')\right)\right), \tag{2.5}$$

where z is a constant value, $0.0 < z \leq 1.0$, and is found by simulations. The metric $\mathcal{F}_{\varkappa}$ is calculated according to (2.3) or (2.4). This thesis utilizes the differential function $\mathcal{F}'_{\varkappa}$ (2.4), while the original function (2.3) is used by Shen *et al.* (2022). The second part of the metric function in (2.5), which includes logarithmic and exponential functions, can be simplified by the step-approximation function $\mathcal{J}(x) \approx \ln\left(1 + \exp\left(-z \cdot \mathcal{F}'_{\varkappa}(t)\right)\right)$, as shown for the DSCF decoding in the work of Ercan *et al.* (2020b):

$$\mathcal{J}(x) = \begin{cases} 1.5\,, & \text{if } \ 0 \leq x \leq 5.0, \\ 0\,, & \text{otherwise.} \end{cases} \tag{2.6}$$

According to our simulation results, the approximation $\mathcal{J}(x)$ in (2.6) results in a negligible error-correction loss. In this thesis, the DSCLF decoder is used with the metric calculated by (2.5) and (2.6).

If the metric $\mathcal{M}_{\text{Lflip}}(t')$ of a newly constructed set $\boldsymbol{\varepsilon}_{t'}$ exceeds the largest metric $\mathcal{M}_{\text{Lflip}}(T_{\max} - 1)$, it is discarded. If not, this set is inserted to $\mathcal{B}_{\text{flip}}$, and its $\mathcal{M}_{\text{Lflip}}(t')$ is inserted to $\boldsymbol{\mathcal{M}}_{\text{Lflip}}$, while keeping the list $\boldsymbol{\mathcal{M}}_{\text{Lflip}}$ sorted in ascending order. The set is not extended further after reaching the maximum size ω, i.e., the decoding order. The decoder is denoted by DSCLF-ω, similarly to DSCF-ω as discussed in Section 2.2.

2.4.1 Error-Correction Performance Analysis

The SCLF and DSCLF decoders are implemented to analyze the error-correction performance. The results are obtained using the same simulation parameters as those described in Subsection 2.2.1. For all list-flip decoders, the path-flipping metrics are calculated using the differential function in (2.4) with $\varkappa = 1.0$. For the DSCLF decoders, the metric for path-flipping candidates is calculated according to (2.5) and with the step-approximation function in (2.6). The list sizes $L = 2$ and $L = 4$ are applied for all list-flip decoders.

The maximum number of trials $T_{\max} \in \{21, 51, 301\}$ is set to all DSCLF-ω decoders with $\omega \in \{1, 2, 3\}$. For the SCLF decoder, it is set to $T_{\max} = 31$. For the DSCLF-ω decoder, the $T_{\max}$ is selected to closely approach the error-correction performance limit of its respective decoder. For the SCLF decoder, $T_{\max}$ is selected to achieve the performance of the SCL decoder with $L = 4$.

Figure 2.5 shows error-correction performance of SCLF and DSCLF-1 decoders with $L = 2$. The results indicate that SCLF achieves the error-correction performance of SCL decoder with $L = 4$. The results also indicate that the DSCLF-1 decoder outperforms the SCLF decoder, as well as the SCL decoder with $L = 4$. However, the gap between the DSCLF-1 decoder and the SCL with $L = 8$ is noticeably large.

Figure 2.6 depicts error-correction performance of DSCLF-ω decoders with $L = 2$. The results show that DSCLF-2 and DSCLF-3 decoders improve the error-correction performance compared to DSCLF-1 and SCLF.

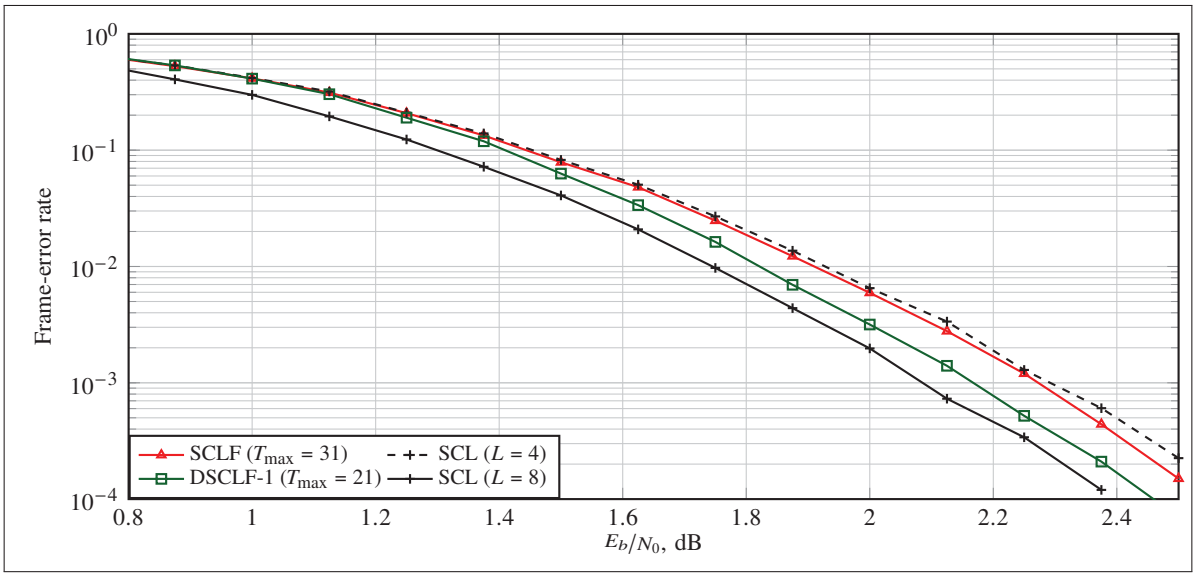


Figure 2.5 Error-correction performance of SCLF and DSCLF-1 decoders with $L = 2$ for a $(1024, 512 + 11)$ polar code

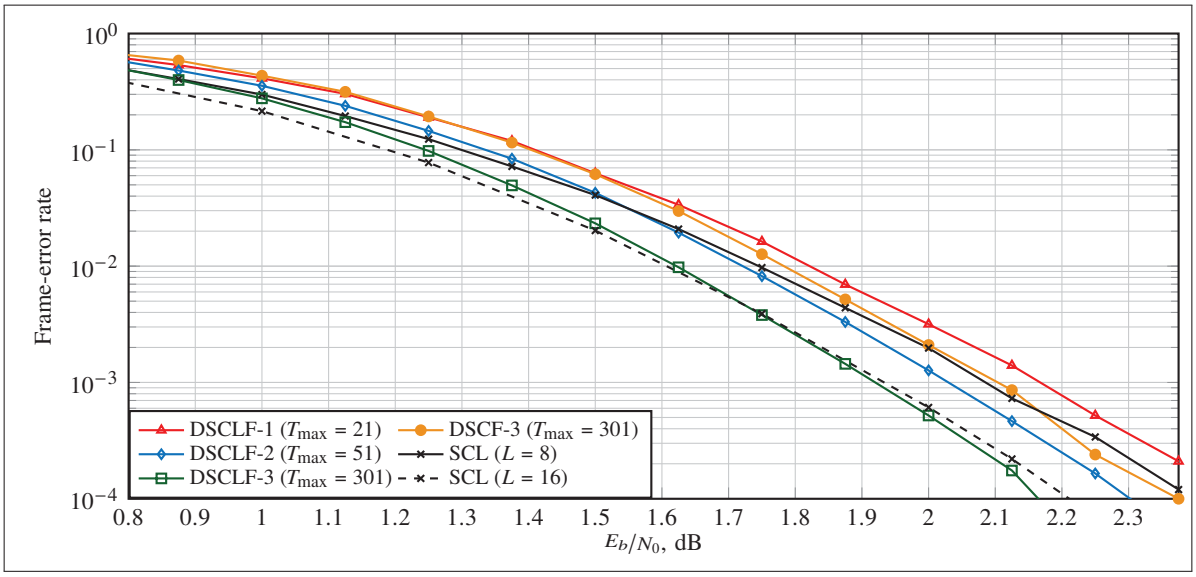


Figure 2.6 Error-correction performance of DSCLF-ω decoder with the decoding order $\omega \in \{2, 3\}$ and $L = 2$ for a $(1024, 512 + 11)$ polar code

Then, DSCLF-2 closely approaches the SCL with $L = 8$ at the target FER of 10^{-2} and outperforms it at lower FER. Furthermore, DSCLF-3 decoder virtually matches the performance of the SCL with $L = 16$. Finally, both DSCLF-2 and DSCLF-3 decoders outperform DSCF-3 (with $L = 1$).

Figure 2.7 depicts error-correction performance of DSCLF-ω decoders with $L = 2$ and $L = 4$. The results show that error-correction performance improves with an increase in the list size L. That is, DSCLF-2 with $L = 4$ and $T_{\max} = 51$ closely achieves the performance of DSCLF-3 with $L = 2$ and $T_{\max} = 301$ and the SCL with $L = 16$. The DSCLF-3 with $L = 4$ and $T_{\max} = 301$ closely achieves the performance of the SCL with $L = 32$. This is expected and in line with the definition of SCLF decoder. That is, SCLF allows for a balance between the SCL and SCF decoders to achieve the target FER.

As discussed in Section 1.6 of Chapter 1, increasing L nearly doubles the decoder area in SCL decoder implementations. On the other hand, increasing ω and $T_{\max}$ leads to a small increase in the area of the SCF decoder, which will be further discussed in Section 2.6 of this chapter. Therefore, it is beneficial to keep a low list size L while increasing ω and $T_{\max}$ in SCLF decoders to derive area- and energy-efficient implementations with strong error-correction performance. Detailed memory analysis of the SCLF decoders with $L = 2$ and $L = 4$ will be provided in Subsection 5.2.2 of Chapter 5 and in Appendix I.

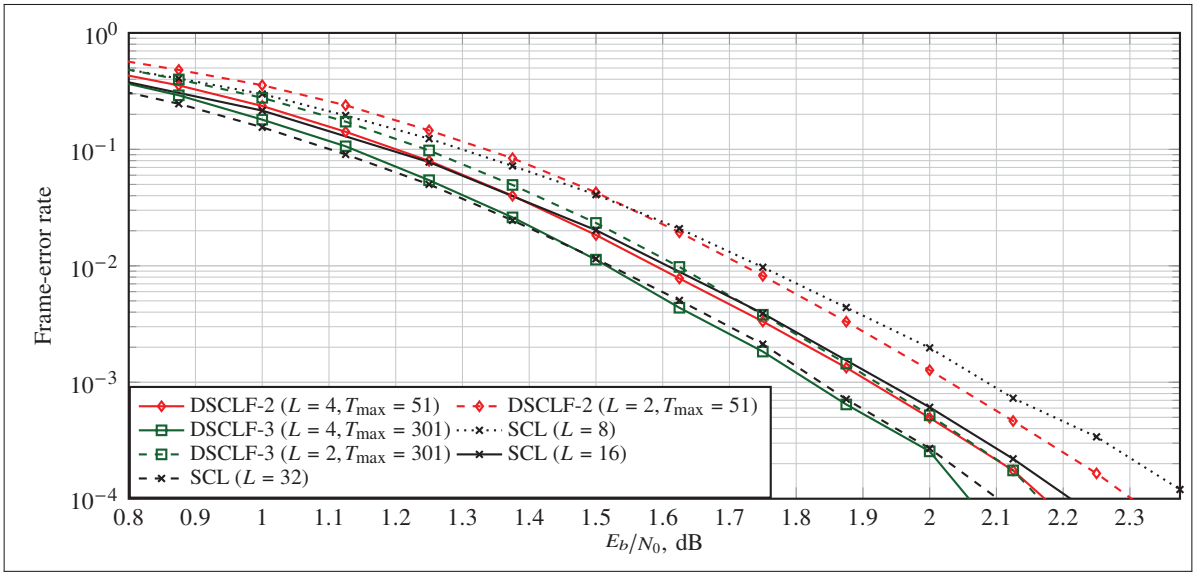


Figure 2.7 Error-correction performance of DSCLF-ω decoder with $\omega \in \{2, 3\}$ and $L = \{2, 4\}$ for a $(1024, 512 + 11)$ polar code

2.5 Execution-Time Model

The execution-time model in this doctoral study is based on estimations from a realistic hardware implementation. Specifically, this model is based on the semi-parallel SC decoder implementation proposed by Leroux *et al.* (2013). During the tree traversal of the SC decoder (shown in Figure 1.5), LLR values are calculated with (1.11) and (1.12) in one clock cycle (CC), using a limited number of processing elements, denoted by P. The PS bits are calculated by (1.16). These calculations are done with P parallel XOR blocks that generate $2P$ bits in one CC according to

Sarkis *et al.* (2014). In this doctoral study, the number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a polar code with $N = 1024$.

The execution time of a standard SC trial, denoted by $\mathcal{L}_{\text{SC}}$, is calculated in CCs as follows:

$$\mathcal{L}_{\text{SC}} = \mathcal{L}_{\alpha}(N) + \mathcal{L}_{\beta}(N), \tag{2.7}$$

where $\mathcal{L}_{\alpha}(N)$ and $\mathcal{L}_{\beta}(N)$ denote execution time of LLR and PS calculations for the polar code of length N, and each of them is calculated by the following equations:

$$\mathcal{L}_{\alpha}(N) = 2N + \frac{N}{P} \cdot \log_2\left(\frac{N}{4P}\right), \tag{2.8}$$

$$\mathcal{L}_{\beta}(N) = \sum_{s=1}^{n-1} (2^{n-s} - 1) \cdot \left\lceil \frac{2^s}{2P} \right\rceil, \tag{2.9}$$

where $\mathcal{L}_{\alpha}(N)$ (2.8) was given by Leroux *et al.* (2013), while $\mathcal{L}_{\beta}(N)$ (2.9) is derived by us using the results of Sarkis *et al.* (2014). The term $(2^{n-s} - 1)$ in (2.9) indicates the number of nodes at tree stage s, excluding the right-most node. The term $2^s/2P$ corresponds to the number of CCs required to compute PS for each node. For $P \geq N/4$, expression in (2.9) is simplified to $\mathcal{L}_{\beta}(N) = N - n - 1$ (Sagitov, Pillet, Balatsoukas-Stimming & Giard, 2023).

The latency of SCL decoding consists of the latency of SC decoding plus the additional delay associated with sorting paths during the estimation of each information bit. According to Stimming *et al.* (2015), the latency of SCL decoding can be approximated as follows:

$$\mathcal{L}_{\text{SCL}} = \mathcal{L}_{\text{SC}} + \mathcal{L}_{\text{sort}}, \tag{2.10}$$

where $\mathcal{L}_{\text{sort}} \approx k_{\text{tot}}$ accounts for the CCs required to sort the paths at the information-bit locations as defined in the work of Stimming *et al.* (2015).

Before analyzing the execution time of the flip decoders, we provide an important detail. For DSCF-ω and DSCLF-ω decoders with $\omega > 1$, the delay associated with updating the bit-flipping sets can occur. However, this delay is not included in the time model of this thesis for simplicity. Additionally, it has been shown to have a minimal impact on overall execution time compared to SC tree traversal, as explained by Ercan *et al.* (2020a). Thus, the equations and derivations provided in this section are all applied to characterize both single- and multi-bit flip decoders (DSCF-ω and DSCLF-ω).

The SCF, SCLF decoders, and their dynamic variations are referred to as "flip" decoders. The flip decoder runs the core SC or SCL algorithm with a maximum T_{max} trials to decode a codeword. The total number of trials, including the initial trial, is denoted by $\tau(c)$, where c represents the codeword index. The execution time to decode a codeword

is the product of SC or SCL latency by $\tau(c)$. This execution time is further denoted by $l_{\text{flip}}(c)$, expressed in CCs, and is computed as:

$$l_{\text{flip}}(c) = \tau(c) \cdot \mathcal{L}_{\text{dec}}, \tag{2.11}$$

where $\mathcal{L}_{\text{dec}}$ denotes the baseline algorithm latency with dec $\in \{\text{SC}, \text{SCL}\}$. Also, the total number of estimated codewords is denoted by C, and $0 \leq c \leq C-1$.

The average execution time of the flip decoder, denoted by $\overline{\mathcal{L}}_{\text{flip}}$, is calculated using a total of C simulated codewords as follows:

$$\overline{\mathcal{L}}_{\text{flip}} = \frac{1}{C} \sum_{c=0}^{C-1} l_{\text{flip}}(c). \tag{2.12}$$

The average execution time of the flip decoders is analyzed through simulations. A simulation setup similar to that described in Section 2.4 and Section 2.2 is applied.

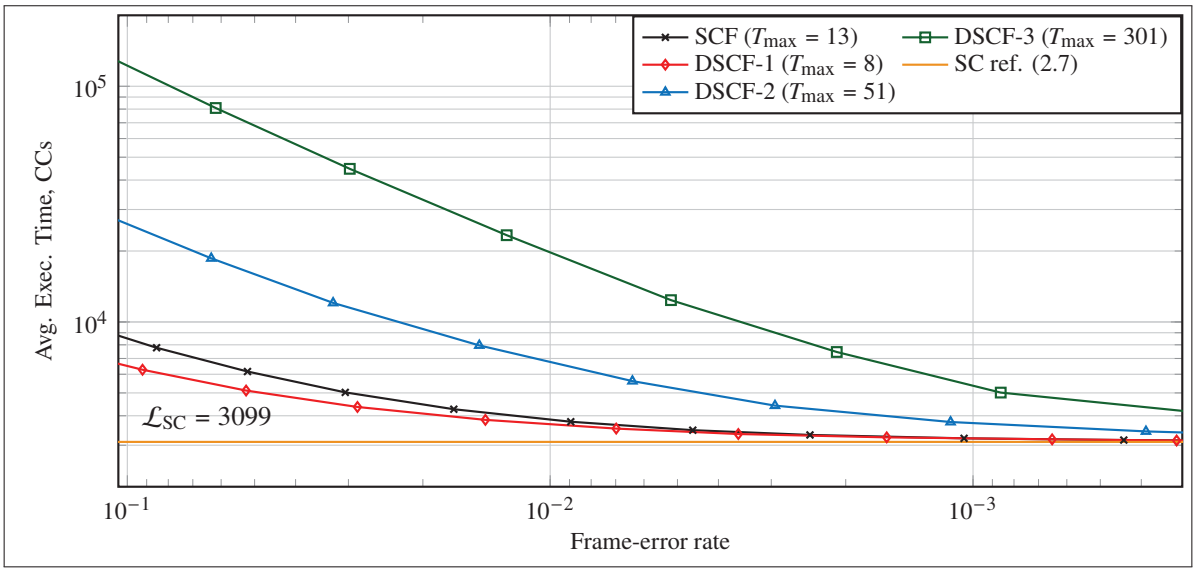


Figure 2.8 Average execution time of flip SCF and DSCF-ω decoders with the decoding order $\omega \in \{1, 2, 3\}$ for a $(1024, 512 + 11)$ polar code

Figure 2.8 shows the average execution time of SCF and DSCF-ω decoders for a $(1024, 512 + 11)$ polar code. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. The x axis represents the FER, where moving right indicates a reduction in the FER or an improvement in error-correction performance, while moving left indicates an increase in the FER or decrease in error-correction performance. The y axis, displayed on a logarithmic scale, corresponds to the average execution time $\overline{\mathcal{L}}_{\text{flip}}$, expressed in CCs and calculated by (2.12). The latency of SC decoder, calculated by (2.7), is depicted as a constant line, and it serves as a reference for SCF and DSCF-ω decoders.

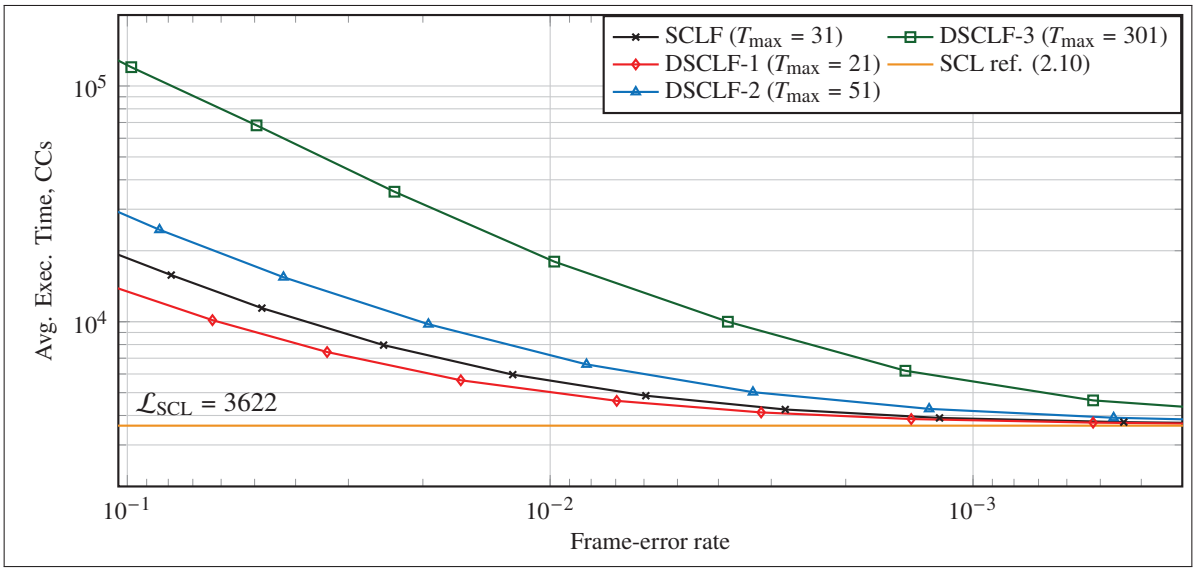


Figure 2.9 Average execution time of list-flip SCLF and DSCLF-ω decoders with the decoding order $\omega \in \{1, 2, 3\}$ and $L = 2$ for a $(1024, 512 + 11)$ polar code

Figure 2.9 shows the average execution time of SCLF and DSCLF-ω decoders. The latency of SCL decoder, calculated by (2.10), is depicted as a constant line and it serves as a reference for SCLF and DSCLF-ω decoders.

The results from both Figure 2.8 and Figure 2.9 indicate that the average execution time of all flip decoders is high when the FER is high. It then exponentially reduces as the FER decreases, closely approaching the latency of the SC or SCL decoder. Throughout the entire region of the FER, the highest average execution time is observed for the DSCF-3 and DSCLF-3 decoders. The single flip SCF and DSCF-1, as well as the SCLF and DSCLF-1 decoders, have the smallest average execution times, quickly approaching the latency of SC or SCL at low FER.

2.6 Hardware Implementations of SCF Decoders

This section summarizes the existing implementation results to demonstrate the tradeoff between implementation costs of SCF and SCL decoders. The implementations of SC and SCL decoders are discussed in detail in Section 1.6 of Chapter 1. Here, only the results for the Fast-SSC decoder (Sarkis *et al.*, 2014) and the Fast-SSCL-SPC decoder (Hashemi *et al.*, 2017) are provided. The implementations of SCF decoders are summarized below.

The Fast-SCF decoder was proposed by Giard & Burg (2018). Then, the hardware implementation of the Fast-SCF decoder was proposed by Ercan *et al.* (2020c). The Fast-DSCF decoding algorithm and its hardware implementation were proposed by Ercan *et al.* (2020a).

Another variation of SCF, partitioned SCF (PSCF) decoding was proposed by Ercan, Condo & Hashemi (2018). In PSCF decoding, a codeword is divided into partitions, each with its own CRC. The SCF decoding is applied to each partition separately. This technique allows to potentially exit decoding trials earlier without decoding the whole codeword. As a result, the average execution time is reduced. Also, partitioning improves the error-correction performance compared to the standard SCF. Hardware implementation of the PSCF decoder variation was proposed by Lee, Roh, Hwangbo & Sunwoo (2021).

The results are presented through normalization of the values to the 65 nm technology, using the scaling methodology described in Section 1.6 of Chapter 1. This methodology is based on the guidelines provided by Stillmaker & Baas (2017). As the technology nodes of the compared works exceed 45 nm, we apply traditional geometry-based scaling methods. The scaling parameters provided in Table 1.1 are applied.

Table 2.1 provides various existing implementations of SC, SCF, and SCL decoders for a $(1024, 512)$ polar code. For all SCL decoders, the results are normalized for the list size $L = 4$. The provided results for coded throughput are measured considering all N bits in a codeword. For the SCF decoder and its variations, the average values of coded throughput are provided at the FER of 10^{-4}.

The results of Table 2.1 show that the SCF decoder and its variations require less area and show better area efficiency compared to the SCL decoder. Additionally, the SCF decoders have less power consumption and require less energy per bit. However, they achieve lower throughput compared to the SCL decoders. This is due to the high average and worst-case execution times of the SCF decoder caused by the high $T_{\max}$.

Table 2.1 Hardware comparisons of state-of-the-art SC, SCF, and SCL decoders for a $(1024, 512)$ code

	Fast-SSC	Fast-SCF	PSCF[(a)]	SCL[(a)]	Fast-SSCL-SPC
Maximum number of trials $T_{\max}$	–	20	8	–	–
List size L	–	–	–	4	4
Area (mm^2)	0.38	0.56	0.43	2.16	1.82
Frequency (MHz)	455	455	520	558	840
Coded throughput (Mbps)	1557	1511[(b)]	252[(b)]	1540	1608
Area Efficiency (Mbps/mm^2)	4100	2710	586	714	883
Power (mW)	79.90	83.44	–	–	517
Energy per bit (pJ/bit)	51.32	55.22	–	–	321

[(a)] Scaled to 65 nm technology using the scaling parameters provided in Table 1.1
[(b)] Average value at FER of 10^{-4}

Table 2.2 provides existing implementations of the DSCF and SCL decoders. The results are for the Fast-DSCF-ω decoder (Ercan *et al.*, 2020a) with orders $\omega = \{1, 2, 3\}$ and for the Fast-SSCL-SPC decoder (Hashemi *et al.*, 2017) with list sizes $L = \{2, 4, 8\}$. The average values of coded throughput are provided at the FER of 10^{-3}. Furthermore, Figure 2.10 depicts the area efficiency of the DSCF-ω and SCL decoders at various FER points. The values of

Table 2.2 Hardware comparisons of state-of-the-art DSCF and SCL decoders for a $(1024, 512)$ code

	Fast-DSCF-ω			Fast-SSCL-SPC		
Decoding order ω	1	2	3	–	–	–
Maximum number of trials $T_{\max}$	10	100	400	–	–	–
List size L	–	–	–	2	4	8
Area (mm^2)	0.55	0.69	1.00	1.05	1.82	3.98
Frequency (MHz)	455	425	410	885	840	722
Coded throughput (Mbps)	557[(a)]	402[(a)]	277[(a)]	1861	1608	1198
Area Efficiency (Mbps/mm^2)	1012	583	277	1776	883	301
Power (mW)	118	136.35	201.61	119.11	517	1128
Energy per bit (pJ/bit)	215	339.04	727.62	64	321	942.17

[(a)] Average value at FER of 10^{-3}

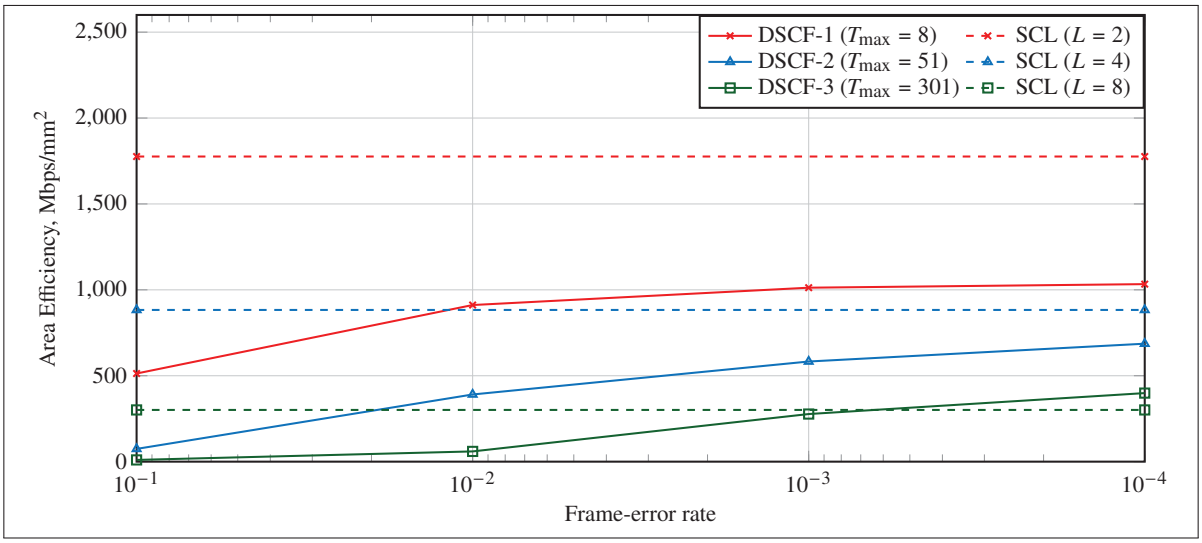


Figure 2.10 Area efficiency of Fast-DSCF-ω and SCL decoders at various FER points for a $(1024, 512)$ polar code

average execution time used for throughput calculations are from our estimated results, which will be discussed in Chapter 4.

The results in Table 2.2 show that the DSCF-ω decoder with any ω requires less area compared to the SCL decoder. The throughput, area efficiency, and energy per bit vary due to the variable execution time of the DSCF decoder. According to the results in Figure 2.10, the area efficiency for the DSCF-ω decoder is low at high FER, and it improves as the FER decreases. Also, as ω increases, the area efficiency of DSCF-ω decoder decreases. When comparing the DSCF-3 decoder with the SCL decoder with $L = 8$, both of which have close error-correction performance, their area efficiencies are also similar at the FER of 10^{-3}, as shown in Table 2.2 and Figure 2.10.

However, the DSCF-3 decoder has 22% lower energy per bit and 82% lower power consumption compared to the SCL decoder with $L = 8$.

The results of Table 2.2 also show that the area nearly doubles for the SCL decoder by increasing L, while for the DSCF-ω decoder, the area increases only by 25% to 45% with an increase in ω. Similarly to the SCF, the DSCF decoder achieves lower throughput compared to the SCL decoder in the FER region of 10^{-2} to 10^{-3}.

There are no hardware implementations in literature for the SCLF decoder. However, based on the implementation results of the SCL, SCF, and DSCF decoders, we can project that SCLF and DSCLF decoders will also be more efficient in terms of area and energy per bit compared to the SCL decoder. This is expected to be true when the decoders have the same error-correction performance.

2.7 Conclusion

The SCF decoder decodes a codeword with multiple SC decoding attempts, where at each attempt, one bit is flipped before resuming SC. The DSCF decoding algorithm is a major improvement to SCF, where a novel metric to construct the list of bit-flipping candidates is derived, and multiple simultaneous bit flips per trial are performed. The DSCF decoder can achieve the error-correction performance of the SCL decoder with medium list size L, which fulfills the requirements of the eMBB service in 5G. The SCLF decoding was proposed with the idea of combining the SCL and SCF decoding strategies. Then, the DSCLF decoding was proposed to adapt the DSCF algorithm strategy to the SCLF.

The considered flip and list-flip decoding algorithms based on the SCF and SCLF indicate strong error-correction performance. However, this comes at a cost of an increased $T_{\max}$, leading to higher average and worst-case execution times. In a hardware implementation, this indicates low throughput. Nevertheless, when analyzing implementation results, the SCF decoders are more efficient than the SCL decoders. In particular, the DSCF-ω decoder has less area and requires less energy per bit, compared to the SCL decoder at matched error-correction performance.

The simulation results show that the DSCF-3 and DSCLF-3 decoders have the highest average execution times at the target FER of 10^{-2}. These decoders also have the strongest error-correction performances compared to other flip decoders. Therefore, mechanisms that can reduce this average execution time without significantly affecting the original error-correction performance of these decoders are needed. Additionally, the mechanisms should not impose significant memory overhead to maintain efficiency in a potential hardware implementation. The DSCF-3 and DSCLF-3 decoder are considered the reference algorithms in this doctoral study.

CHAPTER 3

EXECUTION-TIME REDUCTION MECHANISMS FOR FLIP DECODERS

This chapter describes our two proposed execution-time reduction mechanisms for flip decoders based on SCF: the simplified restart mechanism (SRM) and the generalized restart mechanism (GRM). The simplified restart mechanism (SRM) is presented with a detailed description of the algorithm. In the SRM, conditional restart for the additional trials is performed from the right-hand side (RHS) of the decoding tree if that is where the bit-flipping candidate is located. To perform such a restart, the bit estimates and the partial-sum (PS) bits from the left-hand side (LHS) tree are required to be stored in memory following the initial unsuccessful decoding trial. As a result, the SRM improves the execution-time characteristics of the original decoders.

This chapter then presents the generalized restart mechanism (GRM), an extension of the SRM. The parts of the tree in the GRM are skipped to estimate a bit-flipping candidate and all of the prior bits of each additional decoding trial. The decoding tree is traversed from the root along the restart path to directly estimate the restart bit. To perform a restart, the partial-sum (PS) bits are restored from the bit estimates stored in memory following the initial unsuccessful decoding trial. These PS restorations are shown to be made by low-complexity encoding operations.

When applied to the DSCF-3 decoder, the proposed SRM is shown to reduce the average execution time by 4% to 30%. Similarly, the proposed GRM for the DSCF-3 decoder reduces the average execution time by 26% to 60%. Both mechanisms result in approximately 4% of additional memory for the DSCF-3 decoder.

The following publications resulted from the work presented in this chapter:

- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Successive-Cancellation Flip Decoding of Polar Codes with a Simplified Restart Mechanism", *IEEE Wireless Communications and Networking Conference (WCNC),* 2023, Glasgow, United Kingdom, https://ieeexplore.ieee.org/document/10119097.
- In part: I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Generalized Restart Mechanism for Successive-Cancellation Flip Decoding of Polar Codes", *J. Signal Process. Syst.,* 2024 (*under review*).

In the time analysis provided in Section 2.5 of Chapter 2, the notation $\mathcal{L}_{\text{dec}}$ was used to denote the latency of the baseline decoding algorithm SC or SCL. However, in this chapter, only SC is considered as the baseline algorithm. Also, the delay associated with updating the bit-flipping sets for the DSCF-ω decoders is not considered in the time model of this thesis, as explained in Section 2.5. Therefore, execution times of an additional trial of the SCF and DSCF decoders are considered equivalent, and both are referred to as *flip decoders*.

3.1 Simplified Restart Mechanisms (SRM) For Flip Decoder

3.1.1 Introduction

This section describes our proposed SRM to flip decoders based on SCF. The SRM reduces the average execution time, the average additional execution time, and the execution-time variance of SCF and DSCF decoders. The central idea of the mechanism is to conditionally restart additional decoding trials from the second half of the codeword by using computations stored following the initial SC pass. The error-correction performance is identical to the original non-SRM decoders. When applied to DSCF-3 decoder, the average execution time, the average additional execution time, and the execution-time variance are reduced by 4% to 55%. This decoder achieves the error-correction performance of the state-of-the-art SCL decoder. For the DSCF-3 decoder, the SRM requires approximately 4% of additional memory.

3.1.2 Description of the SRM

Throughout the course of SCF decoding, each additional trial starts by redoing the SC decoding from the beginning to estimate the very first information bit and then proceeds all the way to the location that corresponds to the information bit that needs to be flipped, $i_1 = \varepsilon_{t'}(0)$. However, we observe that decoding of both bits $\hat{u}_0$ and $\hat{u}_{N/2}$ begins from the stage $s = n$ of the decoding tree, where channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$ are used. This list remains constant throughout the SCF decoding of the current codeword. Thus, these observations allow us to propose the SRM for SCF decoding.

In the SCF with the SRM, for each additional trial where the flipping index is on the RHS of the decoding tree, we propose to skip the (unchanged) LHS of the decoding tree, i.e., keep the initial $\{\hat{u}_0, \ldots, \hat{u}_{N/2-1}\}$ and directly decode $\{\hat{u}_{N/2}, \ldots, \hat{u}_{N-1}\}$. This modification requires the LHS computations from the initial SC pass, i.e., the partial-sum (PS) and bit estimates. These vectors are further denoted by $\boldsymbol{\beta}_{\text{rest}}$ and $\hat{\boldsymbol{u}}_{\text{rest}}$, where $\boldsymbol{\beta}_{\text{rest}} = \{\beta_0, \ldots, \beta_{N/2-1}\}$ and $\hat{\boldsymbol{u}}_{\text{rest}} = \{\hat{u}_0, \ldots, \hat{u}_{N/2-1}\}$.

The restart location of the additional trial is denoted by $\psi_{t'}$ with t' indicating the additional trial index. The modified SC trial is denoted by $\text{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, indicating the restart location $\psi_{t'}$ and the bit-flipping set $\boldsymbol{\varepsilon}_{t'}$. According to the definition, the SRM has two possible restart locations $\psi_{t'} \in \{0, N/2\}$. The binary representation of $\psi_{t'}$ is denoted by $\boldsymbol{\mathcal{H}}_{\psi_{t'}} = \{\mathcal{H}_{\psi_{t'}}(0), \ldots, \mathcal{H}_{\psi_{t'}}(n-1)\}$, where the least significant bit (LSB) is on the right.

The modified SC trial in the SCF with the SRM, $\text{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, is illustrated in Figure 3.1 for an $(8, 4)$ polar code. In this example, the bit-flipping index is $i_1 = 5$, and the restart location is $\psi_{t'} = 4$. The part of the vector of bit estimates, $\hat{\boldsymbol{u}}_{\text{rest}}(0, \psi_{t'} - 1) = \{\hat{u}_0, \ldots, \hat{u}_3\}$ is assigned to the pre-stored vector $\hat{\boldsymbol{u}}_{\text{rest}}$. Similarly, the part of the PS

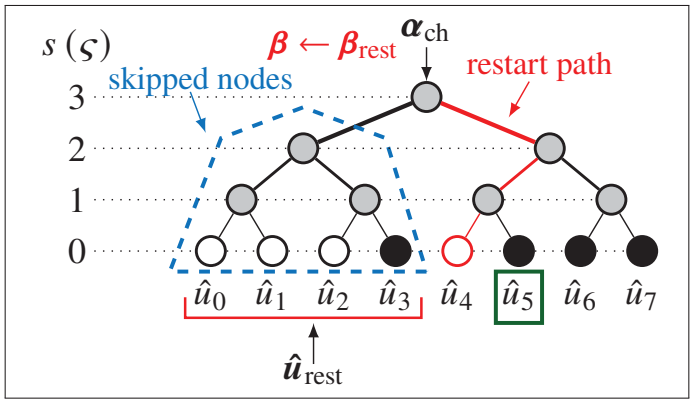


Figure 3.1 The modified SC trial SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ with the bit-flipping index $i_1 = 5$ in SCF with the SRM for an $(8, 4)$ polar code

vector, $\boldsymbol{\beta}\,(0, \psi_{t'} - 1) = \{\beta_0, \ldots, \beta_3\}$ is assigned to $\boldsymbol{\beta}_{\text{rest}}$. Both $\hat{\boldsymbol{u}}_{\text{rest}}$ and $\boldsymbol{\beta}_{\text{rest}}$ are stored to the memory after the initial unsuccessful SC trial. As a result of applying the SRM, the part of the tree to estimate the bits $\{\hat{u}_0, \ldots, \hat{u}_3\}$, surrounded by the dashed blue line in Figure 3.1, is skipped. The restart path, which connects the root of the tree at stage $s = n = 3$ with the leaf $\hat{u}_4$ at stage $s = 0$, is highlighted in red in Figure 3.1. The type of function (1.13)–(1.14) to perform at the decoding stage $\varsigma \in \{n - 1, \ldots, 0\}$ is determined by the binary representation $\mathcal{H}_{\psi_{t'}}$ of $\psi_{t'}$. If $\mathcal{H}_{\psi_{t'}}\,(n - 1 - \varsigma) = 0$, the f function (1.13) is performed at stage ς. If $\mathcal{H}_{\psi_{t'}}\,(n - 1 - \varsigma) = 1$, the g function (1.14) is performed at stage ς.

The construction of polar codes normally results in patterns, where most of the frozen bits are located on the LHS, while the information bits are located on the RHS. In the example tree of Figure 3.1, the $3/4$ of the information bits are found on the RHS. Naturally, the bit-flipping locations in SCF will often occur on the RHS. Therefore, the proposed SRM has high execution-time reduction capability for additional decoding trials of the SCF decoder. Moreover, no further computations are needed to restart an additional trial from $\psi_{t'}$.

Algorithm 1 describes how the SCF-based decoder with the SRM works. The algorithm follows the original course of decoding at the initial trial where $t' = 0$. If the CRC fails after the initial trial, the $\mathcal{B}_{\text{flip}}$ is constructed according to the selected decoding algorithm (SCF or DSCF). Then, the bit estimates and PS are stored to $\hat{\boldsymbol{u}}_{\text{rest}}$ and $\boldsymbol{\beta}_{\text{rest}}$. If during additional trials the bit-flipping index $i_1 = \varepsilon_{t'}\,(0)$ at the RHS of the tree is identified, the SRM flag is raised, and the lists $\hat{\boldsymbol{u}}$ and $\boldsymbol{\beta}$ are assigned to the pre-stored $\hat{\boldsymbol{u}}_{\text{rest}}$ and $\boldsymbol{\beta}_{\text{rest}}$. The SC decoding is then resumed for the remaining part of the decoding tree to estimate $\hat{\boldsymbol{u}}\,(\psi_{t'} + 1, \ldots, N - 1)$.

Algorithm 1 Flip (SCF or DSCF) decoding with the SRM

procedure SCF_ WITH_SRM($\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}, T_{\max}, \omega$)
  $\psi_{t'} \leftarrow 0$
  for $t = 1, t \leq T_{\max}, t = t + 1$ **do**
    $t' \leftarrow t - 1$
    if $t' > 0$ **then** ▷ For additional decoding trials
      $\boldsymbol{\varepsilon}_{t'} \leftarrow \mathcal{B}_{\text{flip}}(t' - 1)$
      $i_1 \leftarrow \boldsymbol{\varepsilon}_{t'}(0)$
      if $i_1 > N/2 - 1$ **then** ▷ Acivate SRM
        $\psi_{t'} \leftarrow N/2$
        $\hat{\boldsymbol{u}}(0, \ldots, \psi_{t'} - 1) \leftarrow \hat{\boldsymbol{u}}_{\text{rest}}(0, \ldots, \psi_{t'} - 1)$ ▷ Restore the half of $\hat{\boldsymbol{u}}$
        $\boldsymbol{\beta}(0, \ldots, \psi_{t'} - 1) \leftarrow \boldsymbol{\beta}_{\text{rest}}(0, \ldots, \psi_{t'} - 1)$ ▷ Restore the half of $\boldsymbol{\beta}$
        $S_f \leftarrow$ True ▷ Set SRM flag
      else
        $S_f \leftarrow$ False ▷ Reset SRM flag to default
        $\psi_{t'} \leftarrow 0$
      end if
    else
      $S_f \leftarrow$ False
    end if
    $(\hat{\boldsymbol{u}}, \boldsymbol{\alpha}_{\text{dec}}) \leftarrow \text{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}, \boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}, S_f)$
    if $CRC(\hat{\boldsymbol{u}}) =$ failure **then**
      if $t = 1$ **then** ▷ For init. SC trial
        $\mathcal{B}_{\text{flip}} \leftarrow$ CONST_FLIP_SET$(\boldsymbol{\alpha}_{\text{dec}}, T_{\max})$
        $\boldsymbol{\beta}_{\text{rest}} \leftarrow \boldsymbol{\beta}(0, \ldots, \psi_{t'} - 1)$ ▷ Store $\boldsymbol{\beta}$ at $t' = 0$
        $\hat{\boldsymbol{u}}_{\text{rest}} \leftarrow \hat{\boldsymbol{u}}(0, \ldots, \psi_{t'} - 1)$ ▷ Store $\hat{\boldsymbol{u}}$ at $t' = 0$
      else
        if $|\boldsymbol{\varepsilon}_{t'}| < \omega$ **then**
          UPDATE_FLIP_SET$(\mathcal{B}_{\text{flip}}, \boldsymbol{\alpha}_{\text{dec}}, \boldsymbol{\varepsilon}_{t'}, \omega)$ ▷ Update $\boldsymbol{\varepsilon}_{t'} = \mathcal{B}_{\text{flip}}(t')$ in DSCF (2.1)
        end if
      end if
    else
      break
    end if
  end for
  return $\hat{\boldsymbol{u}}$
end procedure

3.1.3 Time and Resource Analysis

3.1.3.1 Memory Structure

The sketch of the memory of an SCF-based decoder with the SRM is provided in Figure 3.2. It is largely based on the memory architectures provided by Hashemi *et al.* (2017) and Leroux *et al.* (2013). For the SC decoding, the channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$ and intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$ are quantized using Q_{ch} and Q_{int} bits, respectively. The bit estimates $\hat{\boldsymbol{u}}$ are stored using N bits of memory and the intermediate partial-sums $\boldsymbol{\beta}_{\text{int}}$ require with $N - 1$ bits of storage. For the SCF decoding, the metrics of the bit-flipping candidates are quantized using Q_{flip} bits. Each bit-flipping set $\boldsymbol{\varepsilon}_{t'} \in \mathcal{B}_{\text{flip}}$ requires $\omega \times n$ bits, where $n = \log_2(N)$ is the length of the binary representation of bit-flipping candidates, $\left|\mathcal{B}_{\text{flip}}\right| = T_{\max} - 1$, and $\omega = |\boldsymbol{\varepsilon}_{t'}|_{\max}$ is the maximum size of a bit-flipping set.

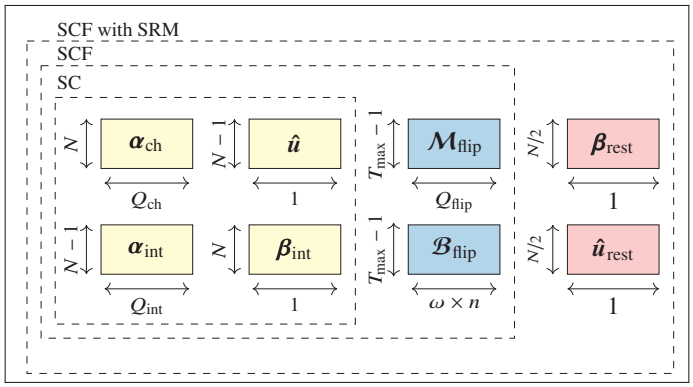


Figure 3.2 Memory sketch of SC, SCF and SCF with SRM

Embedding the SRM to SCF-based decoder requires additional memory in order to perform the restart path. It corresponds to the storage of the LHS calculations of the initial SC trial, bit estimates $\hat{\boldsymbol{u}}_{\text{rest}}$, and partial-sums $\boldsymbol{\beta}_{\text{rest}}$, each of length $N/2$. Thus, the memory overhead of the proposed mechanism is N bits regardless of the value of ω.

The total memory Λ_{SRM} of the SCF with the SRM is then:

$$\Lambda_{\text{SRM}} = \Lambda_{\text{SC}} + \Lambda_{\text{flip}} + \Lambda_{\text{rest}}, \tag{3.1}$$

where Λ_{SC} is memory of SC decoder, Λ_{flip} is the memory of the bit-flipping candidates in SCF Λ_{flip}, and Λ_{rest} is memory of the modified restart in the SRM. The size of each of those memories is estimated according to:

$$\Lambda_{\text{SC}} = Q_{\text{ch}} \cdot N + Q_{\text{int}} \cdot (N-1) + 2N - 1, \tag{3.2}$$

$$\Lambda_{\text{flip}} = Q_{\text{flip}} \cdot (T_{\text{max}} - 1) + \omega \cdot n \cdot (T_{\text{max}} - 1)\,, \tag{3.3}$$

$$\Lambda_{\text{rest}} = 2 \cdot N/2. \tag{3.4}$$

3.1.3.2 Execution-Time Estimations

The execution-time model is described in Section 2.5 of Chapter 2, which is based on the semi-parallel SC decoder implementation proposed by Leroux *et al.* (2013). This model is used throughout the doctoral study and in this chapter. Based on this model, the execution time of the standard SC trial $\mathcal{L}_{\text{SC}}$, expressed in CCs, is calculated as:

$$\mathcal{L}_{\text{SC}} = \mathcal{L}_{\alpha}(N) + \mathcal{L}_{\beta}(N), \tag{3.5}$$

where $\mathcal{L}_\alpha(N)$ and $\mathcal{L}_\beta(N)$ are the LLR and PS calculations for the polar code of length N, and each of them is calculated by the following equations:

$$\mathcal{L}_\alpha(N) = 2N + \frac{N}{P} \cdot \log_2\left(\frac{N}{4P}\right), \tag{3.6}$$

$$\mathcal{L}_\beta(N) = \sum_{s=1}^{n-1} (2^{n-s} - 1) \cdot \left\lceil \frac{2^s}{2P} \right\rceil. \tag{3.7}$$

The modified SC trial SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ in the proposed SRM provides the execution-time reduction when SC is resumed from the index $\psi_{t'} = N/2$ while flipping the first bit $i_1 = \varepsilon_{t'}(0)$ at the trial t'. We further denote the execution time of the SC trial by $\mathcal{L}_{\mathrm{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})}$, which is obtained as follows:

$$\mathcal{L}_{\mathrm{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})} = \mathcal{L}_{\mathrm{SC}} - \Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}), \tag{3.8}$$

where $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ denotes the execution-time reduction of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, that conditionally restarts SC from $\psi_{t'} \in \{0, N/2\}$. Observe that $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}) = 0$ when $\psi_{t'} = 0$, and $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}) > 0$ when $\psi_{t'} = N/2$.

The execution time of estimating one codeword c with total $\tau(c)$ trials under the standard flip decoder is:

$$l_{\mathrm{flip}}(c) = \tau(c) \cdot \mathcal{L}_{\mathrm{SC}}. \tag{3.9}$$

By applying the SRM, the execution time to estimate one codeword c is modified as follows:

$$l_{\mathrm{flipSRM}}(c) = l_{\mathrm{flip}}(c) - \sum_{t'=1}^{\tau'(c)} \Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}). \tag{3.10}$$

The average execution time of the standard flip decoder, denoted by $\overline{\mathcal{L}}_{\mathrm{flip}}$, is calculated using a total of C estimated codewords as follows:

$$\overline{\mathcal{L}}_{\mathrm{flip}} = \frac{1}{C} \sum_{c=0}^{C-1} l_{\mathrm{flip}}(c). \tag{3.11}$$

By applying l_{flipSRM} in (3.10), the average execution time of the flip decoder with the SRM, denoted by $\overline{\mathcal{L}}_{\mathrm{flipSRM}}$, is calculated as follows:

$$\overline{\mathcal{L}}_{\mathrm{flipSRM}} = \frac{1}{C} \sum_{c=0}^{C-1} l_{\mathrm{flipSRM}}(c). \tag{3.12}$$

The reduction of the average execution time of a flip decoder with the SRM compared to the original flip decoder, denoted by $\Delta\overline{\mathcal{L}}_{\text{flip}}$, is obtained as:

$$\Delta\overline{\mathcal{L}}_{\text{flip}} = \frac{\overline{\mathcal{L}}_{\text{flip}} - \overline{\mathcal{L}}_{\text{flipSRM}}}{\overline{\mathcal{L}}_{\text{flip}}}. \tag{3.13}$$

Furthermore, we analyze the average additional execution time and the execution-time variance. These characteristics are particularly significant, as the impact on additional decoding trials can be analyzed in detail. According to the definition, the SRM modifies the additional trials while keeping the initial SC trial untouched. The average additional execution time, denoted by $\mathcal{L}'_{\text{flipSRM}}$, is obtained as:

$$\overline{\mathcal{L}'}_{\text{flipSRM}} = \frac{1}{C'} \sum_{c'=0}^{C'-1} \left(l_{\text{flipSRM}}\left(c'\right) - \mathcal{L}_{\text{SC}} \right), \tag{3.14}$$

where $C' \leq C$ denotes the total simulated codewords that required additional trials, and c' is an index of those codewords. Finally, the execution-time variance, denoted by $\mathcal{V}_l$, is calculated as follows:

$$\mathcal{V}_l = \frac{1}{C-1} \sum_{c=1}^{C} \left(l_{\text{flipSRM}}\left(c\right) - \overline{\mathcal{L}}_{\text{flipSRM}} \right)^2, \tag{3.15}$$

where $\overline{\mathcal{L}}_{\text{flipSRM}}$ is the average execution time obtained by (3.12).

3.1.4 Simulation Results

The effects of our proposed SRM to flip (SCF and DSCF-ω) decoders are analyzed through simulations. A simulation setup similar to that described in Subsection 2.2.1 of Chapter 2 is applied. The 5G polar codes from 3GPP (2018) with length $N = 1024$ and with the rates $R = \{1/8, 1/4, 1/2\}$ are used. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed.

The values of the maximum number of trials $T_{\max}$ are selected as described in Subsection 2.2.1 of Chapter 2. For the single bit-flip SCF and DSCF-1 decoders, $T_{\max} \in \{13, 8\}$ is set, and for the multi-bit flip DSCF-ω decoders with $\omega \in \{2, 3\}$, $T_{\max} \in \{51, 301\}$ is set. For the DSCF-ω decoders, selection of $T_{\max}$ is motivated by achieving the error-correction performance that is close to the genie-aided decoders for $\omega \in \{1, 2, 3\}$ at the target FER of 10^{-2} (Chandesris *et al.*, 2018). For the SCF decoder, $T_{\max} = 13$ is set, which is close to $T_{\max} = 8$ in the DSCF-1 decoder. The hardware-friendly function (2.2) is applied as part of the metric calculations to all DSCF decoders. We compare decoders with and without the SRM in terms of error-correction performance, memory requirements, and execution-time characteristics. These characteristics include the average execution time, average

additional execution time, and execution-time variance, estimated by (3.13)–(3.15). We highlight the results of the execution-time characteristics at the target FER of 10^{-2}.

3.1.4.1 Error-Correction Performance

Figure 3.3 depicts the error-correction performance for a (1024, 128 + 11) 5G polar code with the SCF, DSCF-1, DSCF-2 and DSCF-3 decoders. The error-correction performance results of the SCL decoder with list sizes of $L = 4$ and $L = 8$ are provided for reference. The results show that the DSCF-2 and DSCF-3 decoders achieve the performance of SCL with $L = 4$ and $L = 8$. These results were also discussed in Subsection 2.2.1 of Chapter 2. As discussed in Section 1.5 of Chapter 1, performance of the SCL decoder with $L = 8$ represents the baseline for the 5G performance evaluation 3GPP (2018). Since 5G polar codes are the targeted code in this thesis and given the performance, the DSCF-3 is the decoding algorithm of reference in the remaining part of this chapter. Our proposed SRM provides a higher execution-time reduction for this decoder.

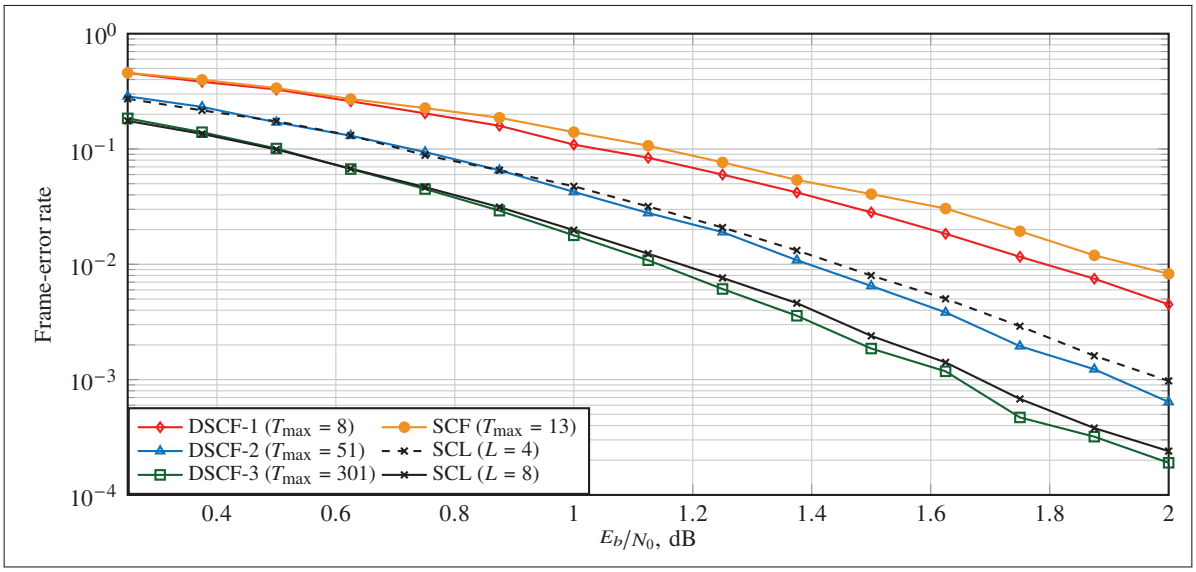


Figure 3.3 Error-correction performance of DSCF-3 decoder for a (1024, 128 + 11) polar code

Figure 3.4 shows the error-correction performance of the DSCF-3 decoder with and without the SRM. The results indicate that the SRM has no detrimental effect on the error-correction performance. This is expected and in line with the description and the algorithm of the SRM provided in Subsection 3.1.2.

3.1.4.2 Memory Estimates

Memory requirements are estimated with (3.1), as described in Subsection 3.1.3.1. The same quantization scheme as that of Ercan *et al.* (2020a) is applied. The blocks with LLR values are quantized by $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$, and

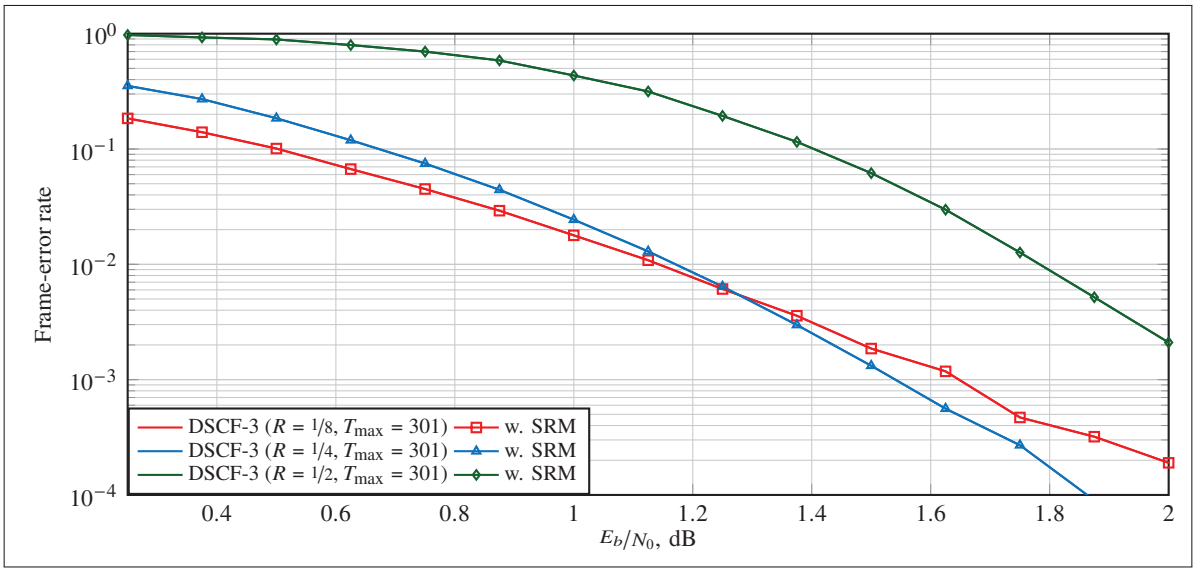


Figure 3.4 Error-correction performance of DSCF-3 decoder with and without the SRM for polar code with $N = 1024$ and $R = \{1/8, 1/4, 1/2\}$

$Q_{\text{flip}} = 7$ bits, respectively. The sizes of these blocks also depend on the values of N and $T_{\max}$. The memory estimates in bits and the memory overhead are provided in Table 3.1. The estimations show that the proposed SRM leads to a memory overhead of 3.87% to 6.58%. Applying the SRM to DSCF-3 decoder results in the smallest memory compared to other decoders (listed in Table 3.1). This happens because the DSCF-3 decoder requires the largest memory due to the highest $T_{\max}$, which defines the size of both the bit-flipping list $\mathcal{B}_{\text{flip}}$ and corresponding list of metrics $\mathcal{M}_{\text{flip}}$.

Table 3.1 Memory estimates and overhead for decoders with and without the SRM

Decoders	$T_{\max}$	no SRM (bits)	w. SRM (bits)	Mem. overh. (%)
SCF	13	15556	16580	6.58
DSCF-1	8	15471	16495	6.62
DSCF-2	51	16702	17726	6.13
DSCF-3	301	26452	27476	3.87

3.1.4.3 Execution-Time Characteristics

Figure 3.5 shows the average execution time of the SCF and DSCF-ω decoders with and without the SRM for a $(1024, 128 + 11)$ polar code. The decoders with and without the SRM are displayed in dashed and solid lines. Additionally, the latency of the SC decoder is provided for reference. The results show that the SRM provides a higher reduction for the DSCF-2 and DSCF-3 decoders throughout the FER range. We explain this by a higher

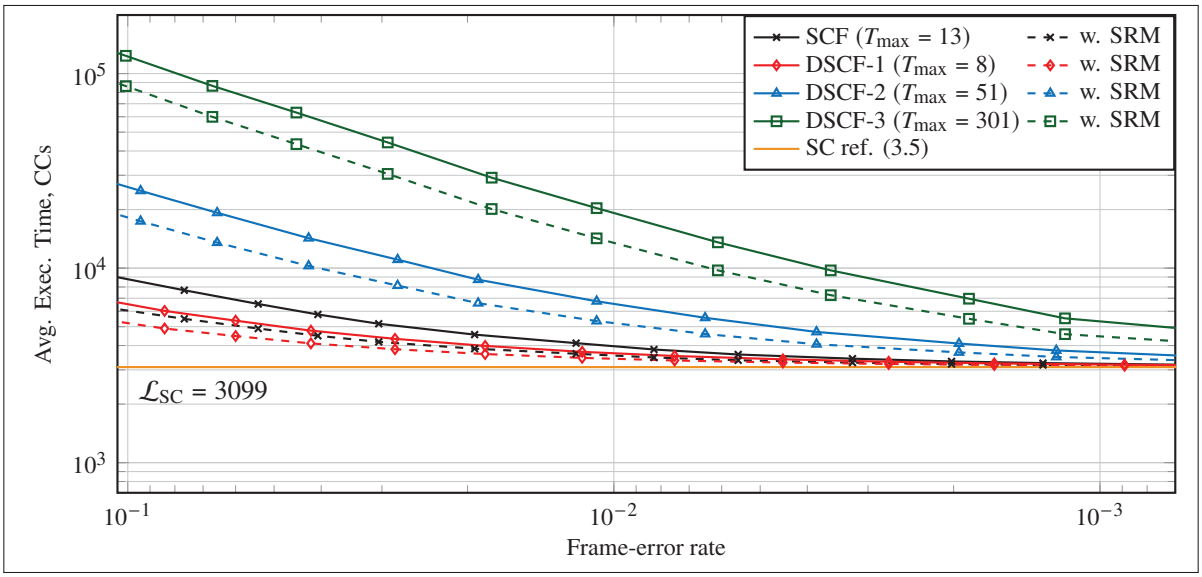


Figure 3.5 Average execution time of SCF and DSCF-ω decoders with and without the SRM for a $(1024, 128 + 11)$ polar code

number of additional trials performed by the multi-bit flip decoders on average. We also observe that applying the DSCF-1 decoder with the SRM provides the smallest reduction among the other decoders. This can be explained by a low number of additional decoding attempts and low decoding latency (smallest T_{max}) compared to other decoders. The results also show that at lower FER, the average execution time of all decoders with and without the SRM closely approaches the latency of the SC decoder.

Figure 3.6 shows the average additional execution time of the SCF and DSCF-ω decoders with and without the SRM for a $(1024, 128 + 11)$ polar code. The results show that the SRM provides a high reduction for all decoders. Contrary to the average execution time provided in Figure 3.5, where reduction is decreased at low FER, the reduction of the average additional execution time remains high throughout the whole FER range. This is because the average additional execution time, calculated by (3.14), includes only codewords that required additional trials. Codewords that were successfully decoded on the first SC decoding attempt are excluded from this analysis. Also, for these codewords, initial SC latency was subtracted from the total execution time. The proposed SRM reduces the execution time of the additional trials without altering the initial trial. Therefore, the impact of this modification is more pronounced when analyzing those codewords. However, for practical applications, the average execution time is more critical than the average additional execution time and the execution-time variance.

The reduction of the execution-time characteristics compared to the original decoding is summarized in Table 3.2 for polar code with $N = 1024$ and rates $R = \{1/8, 1/4, 1/2\}$. The E_b/N_0 points for each decoder are indicated that correspond to target FER of 10^{-2}. The results provided in the table indicate a higher reduction from applying the SRM to a polar code with a lower rate $R = 1/8$. For this rate, compared to the original SCF decoder, the SCF

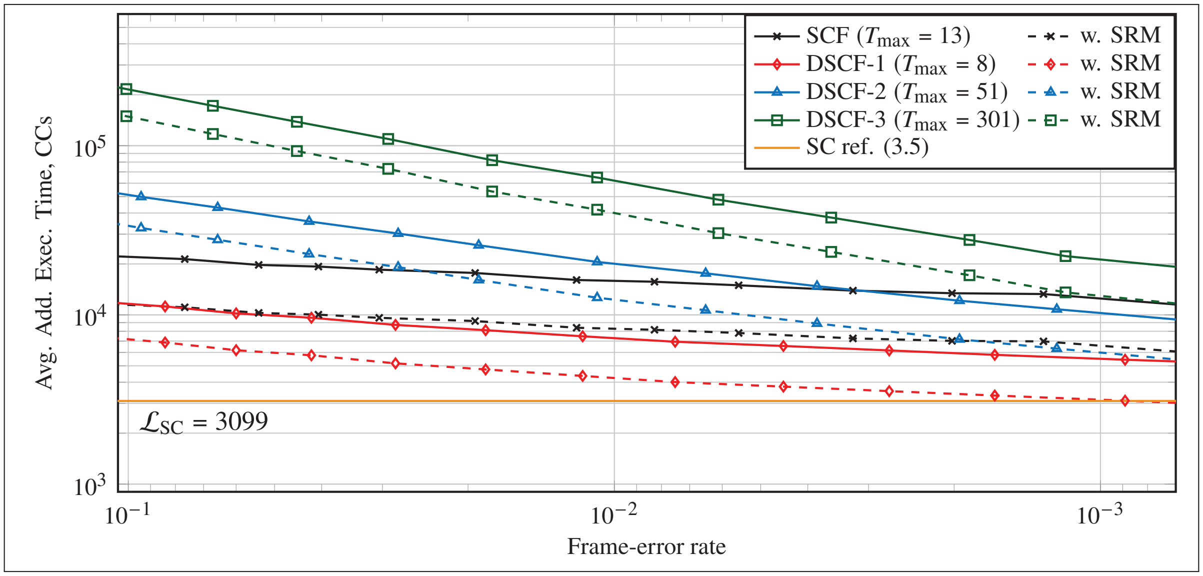


Figure 3.6 Average additional execution time of SCF and DSCF-ω decoders with and without the SRM for a $(1024, 128 + 11)$ polar code

Table 3.2 Reductions of the execution-time characteristics by applying the SRM to flip decoders compared to original decoders at the FER of 10^{-2}

		$R = 1/8$				$R = 1/4$				$R = 1/2$			
Decoder	T_{max}	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{flipSRM}$ %	$\Delta\overline{\mathcal{L}'}_{flipSRM}$ %	$\Delta\mathcal{V}_l$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{flipSRM}$ %	$\Delta\overline{\mathcal{L}'}_{flipSRM}$ %	$\Delta\mathcal{V}_l$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{flipSRM}$ %	$\Delta\overline{\mathcal{L}'}_{flipSRM}$ %	$\Delta\mathcal{V}_l$ %
SCF	13	2.00	9.04	48.00	73.06	1.75	10.52	44.32	68.66	2.375	6.03	33.70	55.32
DSCF-1	8	1.75	6.90	41.67	64.56	1.625	5.52	32.00	47.92	2.25	1.92	15.73	22.53
DSCF-2	51	1.375	20.87	38.56	59.63	1.375	13.41	26.58	39.66	2.00	4.34	9.70	12.16
DSCF-3	**301**	**1.125**	**30.05**	**35.46**	**54.75**	**1.125**	**17.90**	**20.70**	**32.04**	**1.75**	**4.04**	**4.66**	**5.70**

decoder with the SRM achieves the reductions of $\{9.04\%, 48.00\%, 73.06\%\}$ of the average execution time, average additional execution time and execution-time variance, respectively. Highlighting the results for the DSCF-3 decoder (which has the strongest error-correction performance at a rate $R = 1/8$), applying the SRM achieves the reductions of $\{30.05\%, 35.46\%, 54.75\%\}$ compared to the original decoder. This DSCF-3 decoder has a 3.87% memory overhead.

3.1.5 Conclusion on Simplified Restart Mechanism

In this section, we described our proposed SRM, a mechanism that improves the execution-time characteristics of the flip (SCF and DSCF) decoders for polar codes without affecting original error-correction performance. The SRM conditionally restarts additional SC decoding trial from the second half of the codeword if that is where the first bit-flipping candidate is located. As a result, the flip decoder with the SRM reduces the execution-time

characteristics compared to the original decoder, particularly for low-rate polar codes. For the DSCF-3 decoder, applying the SRM permits to achieve the reductions of $\{30.05\%, 35.46\%, 54.75\%\}$ of the average execution time, average additional execution time, and execution-time variance, respectively. For the DSCF-3 decoder, the proposed mechanism results in approximately 4% of additional memory. This decoder achieves the error-correction performance of the state-of-the-art SCL decoder.

Although the SRM highly improves the execution-time characteristics for flip decoders, it is only effective for the low-rate polar codes. We explain this effectiveness by a high number of information bits located at the second half of the codeword when the code rate is low. Therefore, the bit-flipping candidates are pushed into the second half of the codeword, and the restart location $\psi_{t'} = N/2$ is frequently selected at the additional decoding trials. However, for the codes with higher rates, the flip decoder with the SRM rarely restarts from $\psi_{t'} = N/2$, which reduces the effectiveness of the SRM. A mechanism that can restart from various locations in a codeword would improve the effectiveness of the codes with higher rates. Hence, the GRM is proposed with this motivation and is described in the following section.

3.2 Generalized Restart Mechanism (GRM) For Flip Decoders

3.2.1 Introduction

In this section, we describe our proposed GRM for flip decoders for polar codes with the central idea to skip decoding computations that are identical between the initial trial and any additional trial. For any additional trial, the restart is performed from the location where the course of SC is altered for the first time, i.e., the information bit that follows the bit-flipping candidate. This allows us to avoid computations to decode previous bits. The restart is achieved by calculating the PS bits from the bit estimates that have been stored in memory following the initial unsuccessful SC pass.

When applied to DSCF-3 decoder, our proposed GRM is shown to reduce the average execution time by 26% to 60% without any negative effect on error-correction performance. For the DSCF-3, the proposed mechanism requires 4% additional memory.

In the previous section, we described the proposed SRM. The SRM conditionally restarts additional trials from the second half of the codeword if that is where the bit-flipping candidate is located. In this section, we describe the proposed GRM, an extension of the SRM. The GRM can restart from any location in the codeword.

3.2.2 Distribution of the Bit-Flipping Candidates

A statistical analysis is carried out to analyze the probability-mass function (PMF) of the bit-flipping candidates. This is beneficial to understand the limitations of the previously proposed SRM and to motivate the use of the proposed GRM. In other existing works, the analyses are mainly focused on identifying unsuccessfully decoded codewords throughout the simulation. Hence, such analysis is made for the SCF decoder by Condo, Ercan & Gross (2018) and Ercan, Condo & Gross (2019), and for the DSCF decoder by Sagitov & Giard (2020). By identifying these codewords, decoding is terminated at earlier decoding trials. However, this approach negatively impacts the error-correction performance. In this work, we focus on both successfully and unsuccessfully decoded codewords to improve execution-time characteristics without affecting original error-correction performance.

For this analysis, 5G polar codes 3GPP (2018) with length $N = 1024$ and rates $R = \{1/8, 1/4, 1/2\}$ are used. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. Simulations are made over the AWGN channel with BPSK modulation. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed. We simulated the DSCF-3 decoding algorithm with $T_{\max} = 301$. The PMF distribution includes only the first bit in a bit-flipping set, i.e, bit $i_1 = \varepsilon_{t'}(0)$, for the set of information bits $\mathcal{A} = \{a_0, \ldots, a_j, \ldots, a_{k_{\text{tot}}-1}\}$, such that $0 \leq j \leq k_{\text{tot}} - 1$. The location i_1 is stored for each trial index t' for the codewords with $\tau' > 0$, i.e., the codewords that required additional trials. The probability that a_j is the first bit to be flipped during an additional trial is further denoted by $\mathbb{P}\left(i_1 = a_j\right)$.

In the previously proposed SRM, additional trials are restarted on the LHS, i.e., at 0 or the RHS, i.e., at $N/2$ of the decoding tree depending on the location of i_1. The probability of a restart from the LHS in the SRM is

$$\mathbb{P}_{\text{LHS}} = \mathbb{P}\left(i_1 < \frac{N}{2}\right) = \sum_{j=0}^{j_{\text{RHS}}-1} \mathbb{P}\left(i_1 = a_j\right), \tag{3.16}$$

where j_{RHS} is location of the first information bit on the RHS of the decoding tree. Likewise, the probability to restart from the RHS in the SRM is the probability

$$\mathbb{P}_{\text{RHS}} = \mathbb{P}\left(i_1 \geq \frac{N}{2}\right) = \sum_{j=j_{\text{RHS}}}^{k_{\text{tot}}-1} \mathbb{P}\left(i_1 = a_j\right) = 1 - \mathbb{P}_{\text{LHS}}. \tag{3.17}$$

The SRM improves the execution-time characteristics when restarting from the RHS tree that has a high probability. Figure 3.7 depicts the resulting PMF for different code rates in each subfigure. The red dashed line indicates the first information index at the RHS tree, i.e., j_{RHS}. Additionally, the probabilities $\mathbb{P}_{\text{LHS}}$ and $\mathbb{P}_{\text{RHS}}$ are overlayed on each subfigure. Looking at the results, the correlation between j_{RHS} and $\mathbb{P}_{\text{LHS}}$ (or $\mathbb{P}_{\text{RHS}}$) is evident. Notably, for the rate $R = 1/8$, the tree-division index is very low such that $j_{\text{RHS}} = 10 \ll k_{\text{tot}}$. That pushes the bit-flipping candidates

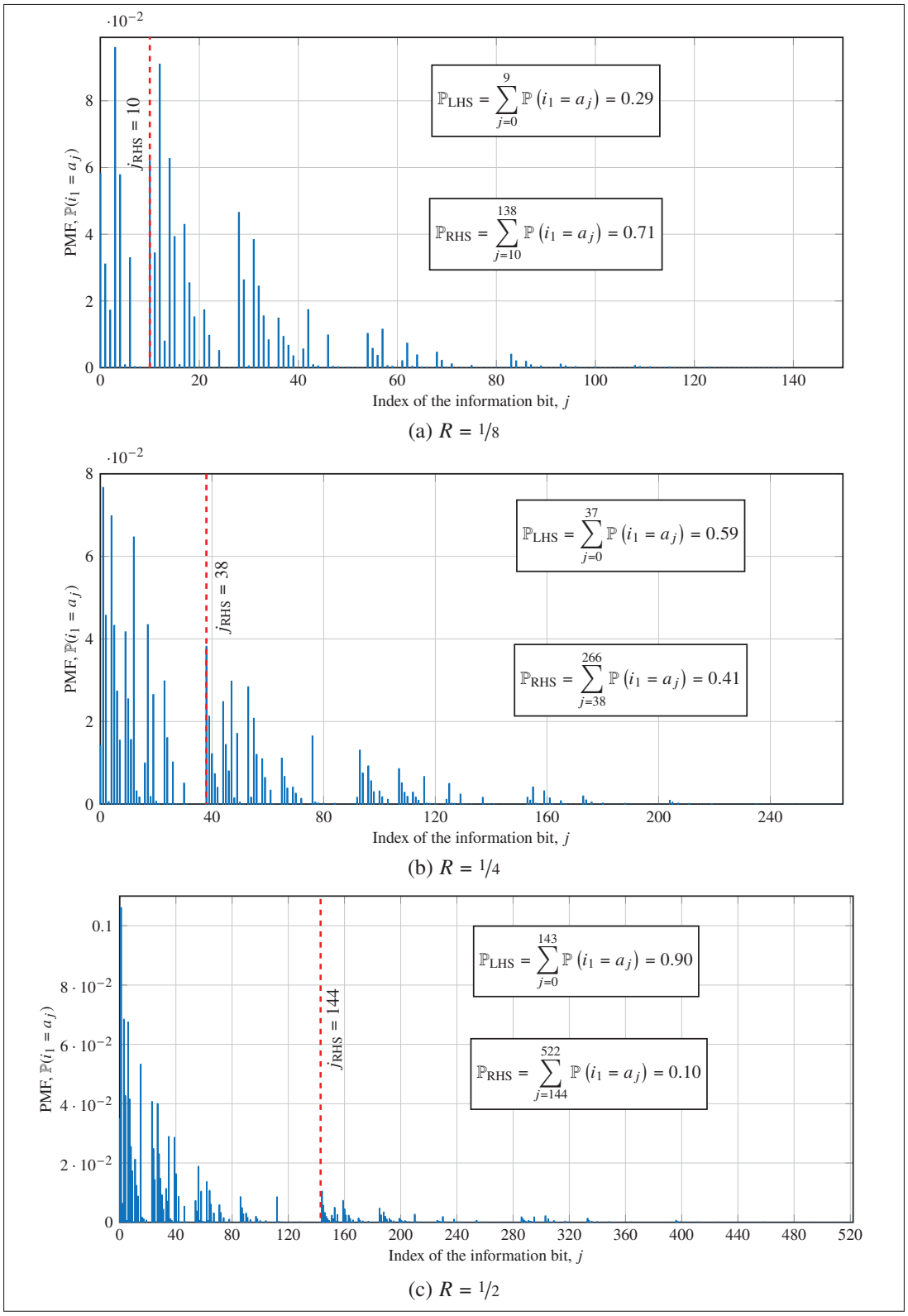


Figure 3.7 PMF of the information-bit location $a_j \in \mathcal{A}$ being the first bit-flipping candidate $i_1 = \varepsilon_{t'}$ (0) under DSCF-3 decoding for the code rates $R = \{1/8, 1/4, 1/2\}$. Probabilities $\mathbb{P}_{\text{LHS}}$ and $\mathbb{P}_{\text{RHS}}$ are inside each plot

more towards the RHS, resulting in a smaller $\mathbb{P}_{\text{LHS}}$. As the code rate increases, the bit-flipping candidates tend to be located on the LHS, resulting in a higher $\mathbb{P}_{\text{LHS}}$.

This distribution of $\mathbb{P}_{\text{LHS}}$ and $\mathbb{P}_{\text{RHS}}$ explains why the SRM was shown to be more effective for a low-rate code such as $R = 1/8$ compared to a high-rate code such as $R = 1/2$. A mechanism that can restart anywhere in a codeword may further reduce the execution-time characteristics compared to the SRM. Hence, the GRM is proposed with this motivation and is described in the following section.

3.2.3 Description of the GRM

At additional decoding trials, the flip decoder based on SCF starts by traversing the whole decoding tree to estimate each information bit until the one that needs to be flipped, at index $i_1 = \varepsilon_{t'}(0)$. After flipping this bit, the standard course of decoding is resumed for the remaining part of the decoding tree when $|\boldsymbol{\varepsilon}_{t'}| = 1$. However, if $|\boldsymbol{\varepsilon}_{t'}| > 1$, at least one additional bit-flip will be performed in the remaining part of the decoding tree. The GRM aims to reduce the number of computations for each additional decoding trial t'. This results in a reduced decoding time of t', and ultimately, a reduced average execution time of the flip decoder.

First, a fundamental definition of the proposed GRM is provided in Definition 3.1. Second, a definition of the restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ is provided in Definition 3.2. This is followed by a definition of the partial-sum restoration provided in Definition 3.3.

Definition 3.1 (The GRM of SCF-based decoder)**.** For an additional trial t', the flip decoding algorithm with the proposed GRM skips the non-negligible part of the decoding tree by restarting at location $\psi_{t'}$. This modified SC trial is denoted by SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, indicating the restart location $\psi_{t'}$ and the bit-flipping set $\boldsymbol{\varepsilon}_{t'}$. The savings in terms of computations and decoding time result from storing the N bit estimates $\hat{\boldsymbol{u}}$ of the initial SC trial. These bit estimates are stored to $\hat{\boldsymbol{u}}_{\text{rest}}$.

When performing an additional trial of the SC decoding with the set of bit-flipping candidates $\boldsymbol{\varepsilon}_{t'}$, the additional trial does not differ from the initial SC pass until the first flipping location $i_1 = \varepsilon_{t'}(0)$. Hence, all bit estimates $\{\hat{u}_0, \ldots, \hat{u}_{i_1-1}\}$ are identical to the estimations of the initial SC trial. Their storage in $\hat{\boldsymbol{u}}_{\text{rest}}$ permits to assign

$$\hat{u}_j = \hat{u}_{\text{rest}}(j), \tag{3.18}$$

where $0 \leq j \leq i_1 - 1$. For position i_1, the value of the bit $\hat{u}_{i_1}$ is flipped with respect to $\hat{u}_{\text{rest}}(i_1)$. Hence, no computations are required to determine the value of the bit $\hat{u}_{i_1}$. For positions greater than i_1, the bit estimates are known if they are frozen and unknown if the bit belongs to $\mathcal{A}$. Hence, no computations are required until the first information bit on the RHS of i_1 is met. The tree traversal to compute $\alpha_{\text{dec}}(\psi_{t'})$ to estimate $\hat{u}_{\psi_{t'}}$ corresponds to the

restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$. After the restart path, the SC trial with $\boldsymbol{\varepsilon}_{t'}$ is resumed for the remaining part of the decoding tree to estimate bits $\left\{\hat{u}_{\psi_{t'}+1}, \ldots, \hat{u}_{N-1}\right\}$.

Definition 3.2 (Restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$)**.** In an additional trial t', the restart path is the tree traversal from the root $s = n$ to the leaf $s = 0$, leading to the decision LLR $\alpha_{\text{dec}}(\psi_{t'})$. It allows to compute $\hat{u}_{\psi_{t'}} = \text{HD}\left(\alpha_{\text{dec}}\left(\psi_{t'}\right)\right)$ (1.15). The type of function (1.13)–(1.14), performed at each decoding stage $\varsigma \in \{n-1, \ldots, 0\}$ is determined by the vector $\boldsymbol{\mathcal{H}}_{\psi_{t'}}$, which is the binary representation of $\psi_{t'}$.

The binary representation of $\psi_{t'}$ is denoted by $\boldsymbol{\mathcal{H}}_{\psi_{t'}} = \left\{\mathcal{H}_{\psi_{t'}}(0), \ldots, \mathcal{H}_{\psi_{t'}}(n-1)\right\}$, where the least significant bit (LSB) is on the right, as defined in Subsection 3.1.2. At the decoding stage $\varsigma \in \{n-1, \ldots, 0\}$ of the restart path, the bit $\mathcal{H}_{\psi_{t'}}(n-1-\varsigma)$ is used as flag to determine whether an f function or a g function is performed to return the LLR vector $\boldsymbol{\alpha}_\varsigma = \left\{\alpha_\varsigma(0), \ldots, \alpha_\varsigma(2^\varsigma - 1)\right\}$ that will be used at stage $\varsigma - 1$. If $\mathcal{H}_{\psi_{t'}}(n-1-\varsigma) = 0$, f function is performed at decoding stage ς and

$$\alpha_\varsigma(j) = f\left(\alpha_{\varsigma+1}(j), \alpha_{\varsigma+1}(j+2^\varsigma)\right), \tag{3.19}$$

where $j \in \{0, \ldots, 2^\varsigma - 1\}$ (1.11). If $\mathcal{H}_{\psi_{t'}}(n-1-\varsigma) = 1$, g function is performed at decoding stage ς. If a g function is required at stage ς at the restart path, a segment of 2^ς partial-sums, denoted by $\boldsymbol{\beta}_\varsigma$, is required and is obtained by the partial-sum restoration. The partial-sum restoration is explained in Definition 3.3.

The resulting vector $\boldsymbol{\alpha}_\varsigma$ is retrieved as:

$$\alpha_\varsigma(j) = g\left(\alpha_{\varsigma+1}(j), \alpha_{\varsigma+1}(j+2^\varsigma), \beta_\varsigma(j)\right), \tag{3.20}$$

where $j \in \{0, \ldots, 2^\varsigma - 1\}$ (1.12). Note that the computations of the vector $\boldsymbol{\alpha}_\varsigma$ do not need extra memory and are being stored in the memory for SC intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$.

Definition 3.3 (Partial-sum restoration)**.** At any decoding stage $\varsigma \in \{n-1, \ldots, 0\}$, where g function is to be performed, the partial-sum restoration corresponds to the computations of 2^ς partial-sums $\boldsymbol{\beta}_\varsigma$ used in the g function on the basis of a segment of 2^ς bit estimates $\boldsymbol{\eta}_\varsigma$. The start index of the segment $\boldsymbol{\eta}_\varsigma$ is obtained by subtracting an offset from $\psi_{t'}$. This offset is further denoted by ϕ_ς and can be obtained as follows:

$$\phi_\varsigma = \sum_{s=0}^{\varsigma} \left(\mathcal{H}_{\psi_{t'}}(n-1-s) \cdot 2^s\right). \tag{3.21}$$

A polar encoder of size 2^ς (1.16) on $\boldsymbol{\eta}_\varsigma$ allows to retrieve $\boldsymbol{\beta}_\varsigma$.

The partial-sum restoration in the architectural model is discussed in more detail in Subsection 3.2.4.3 of this chapter.

Illustration of Restart Path in GRM

The modified SC trial in a flip decoder based on SCF with the GRM, SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, is illustrated in Figure 3.8 with a $(16, 8)$ code defined by $\mathcal{A} = \{6, 7, 9, 11, 12, 13, 14, 15\}$. In this example, the bit-flipping location is $i_1 = a_2 = 9$, and the restart location is set to the next information bit with index $\psi_{t'} = a_3 = 11$. Bit estimates at the LHS of $\hat{u}_9$, i.e, bits $\{\hat{u}_0, \ldots, \hat{u}_8\}$, are restored from the memory $\hat{\boldsymbol{u}}_{\text{rest}}$ according to (3.18). Then, the bit $\hat{u}_9 = \hat{u}_{i_1}$ is restored from $\hat{u}_{\text{rest}}(9)$ and flipped, i.e., $\hat{u}_9 = \hat{u}_{\text{rest}}(9) \oplus 1$. The bit $\hat{u}_{10}$ is assigned to 0 since $10 \in \mathcal{A}^C$. As a result of applying the GRM, the parts of the tree to estimate the bits $\{\hat{u}_0, \ldots, \hat{u}_{\psi_{t'}-1}\}$, surrounded by the dashed blue line in Figure 3.8, are skipped. The restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, which connects the root of the tree at stage $s = n = 4$ with the leaf $\hat{u}_{11}$ at stage $s = 0$, is highlighted in red in Figure 3.8. During the traversal, at the decoding stages $\varsigma \in \{n-1, \ldots, 0\}$ and given $\mathcal{H}_{11} = \{1, 0, 1, 1\}$, g functions are performed at stages $\varsigma \in \{3, 1, 0\}$, and f functions at stage $\varsigma = 2$. The standard course of SC decoding is resumed for the remaining part of the decoding tree upon traversing the restart path and estimating $\hat{u}_{11}$.

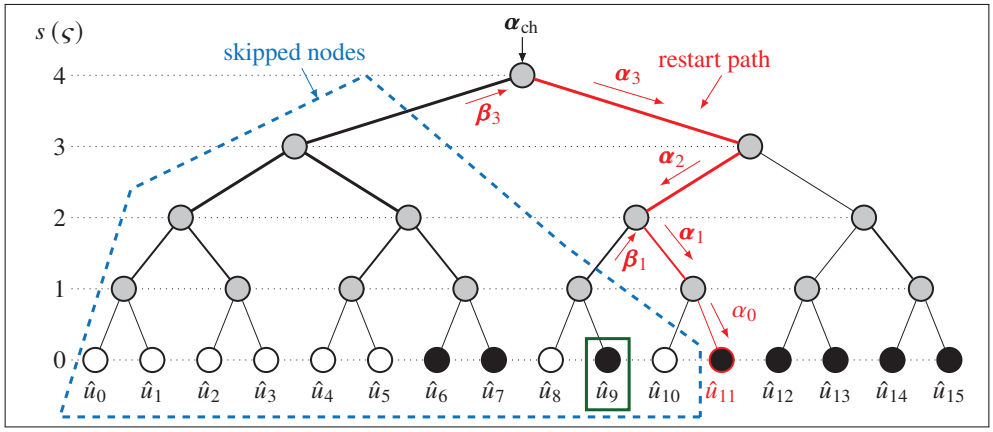


Figure 3.8 The modified SC trial SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ with the bit-flipping index $i_1 = 9$ and the restart location $\psi_{t'} = 11$, in SCF with the GRM for a $(16, 8)$ polar code

3.2.4 Time and Resource Analysis of the GRM

This subsection begins by analyzing the execution-time reduction capability of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ in the GRM. The execution-time reduction is derived as a function of the restart location and then compared to the original decoding, with the results visualized in a graph. Next, the memory resources and the partial-sum restoration resources are discussed.

3.2.4.1 Execution-Time Reduction Capability

The modified SC trial SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ in GRM provides the execution-time reduction by resuming SC from the index $\psi_{t'}$ while flipping the bits in $\boldsymbol{\varepsilon}_{t'}$. The execution time of the modified SC trial SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, denoted by $\mathcal{L}_{\mathrm{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})}$, was derived by (3.8) in Section 3.1.3 of this chapter. This execution time $\mathcal{L}_{\mathrm{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})}$ is calculated as:

$$\mathcal{L}_{\mathrm{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})} = \mathcal{L}_{\mathrm{SC}} - \Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}), \tag{3.22}$$

where $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ denotes the execution-time reduction of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$. Then, the reduction $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ provided by the GRM corresponds to the skipped CCs that would otherwise be required to calculate the LLR and PS values for the modified restart. This reduction can be computed as follows:

$$\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}) = \Delta\mathcal{L}_{\alpha}(\psi_{t'}) + \Delta\mathcal{L}_{\beta}(\psi_{t'}) - \Theta(\psi_{t'}), \tag{3.23}$$

where $\Delta\mathcal{L}_{\alpha}(\psi_{t'})$ and $\Delta\mathcal{L}_{\beta}(\psi_{t'})$ denote the skipped CCs by avoiding LLR and PS computations required to decode bit estimates $\{\hat{u}_0, \ldots, \hat{u}_{\psi_{t'}-1}\}$. The component $\Theta(\psi_{t'})$ corresponds to the additional CCs required to calculate all PS at the restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$. It is visualized with the vectors $\boldsymbol{\beta}_{\varsigma} \in \{\boldsymbol{\beta}_3, \boldsymbol{\beta}_1\}$ shown in red in Figure 3.8.

The skipped nodes inside the dashed-blue area in Figure 3.8 permit to visualize the reductions $\Delta\mathcal{L}_{\alpha}(11)$ and $\Delta\mathcal{L}_{\beta}(11)$. These functions are calculated as follows:

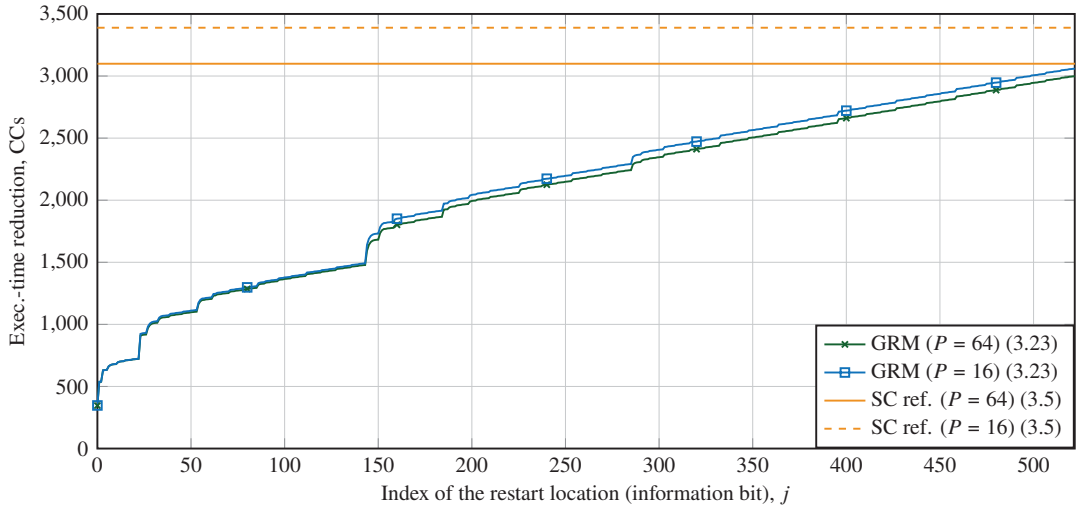
$$\Delta\mathcal{L}_{\alpha}(\psi_{t'}) = \sum_{s=0}^{n-1} \left(\left\lfloor \frac{\psi_{t'}}{2^s} \right\rfloor \cdot \left\lceil \frac{2^s}{P} \right\rceil \right), \tag{3.24}$$

$$\Delta\mathcal{L}_{\beta}(\psi_{t'}) = \sum_{s=1}^{n-1} \left(\left\lfloor \frac{\psi_{t'}}{2^s} \right\rfloor \cdot \left\lceil \frac{2^s}{2P} \right\rceil \right). \tag{3.25}$$

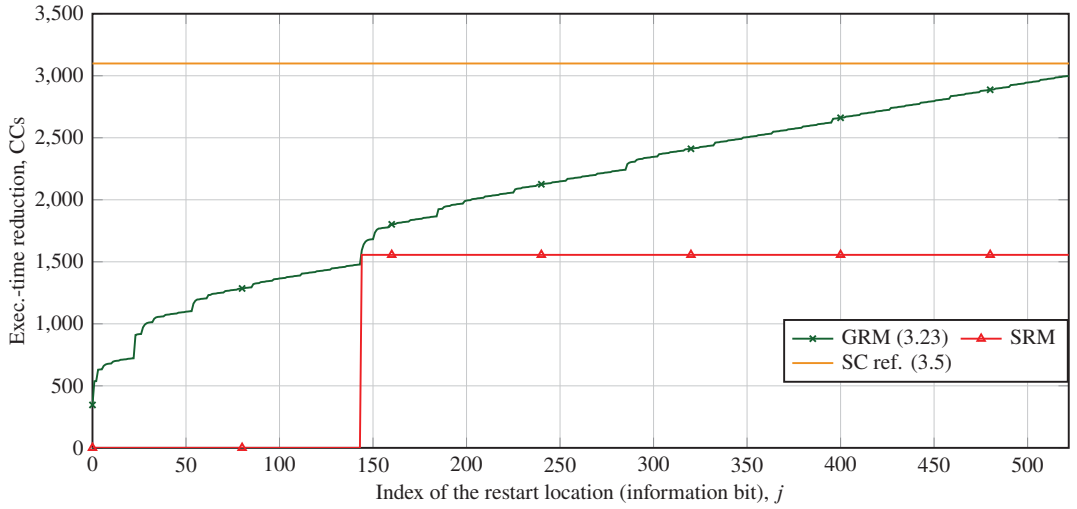
In both equations, $\left\lfloor \frac{\psi_{t'}}{2^s} \right\rfloor$ indicates the number of skipped nodes at tree stage s. The second term corresponds to the number of skipped CCs to compute LLR and PS for each of those nodes. As a remainder, P processing elements are used for LLRs (Leroux *et al.*, 2013) while $2P$ are used for PS (Sarkis *et al.*, 2014). Note that for $s = 0$, the PS correspond to bit estimates, hence no computations are performed, thus the summation begins from $s = 1$ (3.25).

The component $\Theta(\psi_{t'})$ in (3.23) is the additional CCs for the partial-sum restoration during the restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, thus being subtracted from $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$. In this doctoral study, the time model proposed by Sarkis *et al.* (2014) is applied. In that scheme, $2P$ bits of PS are calculated according to (1.16) in one CC. Based on this, the additional time component $\Theta(\psi_{t'})$, expressed in CCs, is calculated as follows:

$$\Theta(\psi_{t'}) = \sum_{s=1}^{n-1} \left(\mathcal{H}_{\psi_{t'}}(n-1-s) \cdot \left\lceil \frac{2^s}{2P} \right\rceil \cdot s \right). \tag{3.26}$$



(a) GRM with $P \in \{16, 64\}$



(b) GRM Vs. SRM with $P = 64$

Figure 3.9 Execution-time reduction of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ in the GRM estimated by (3.23) for a $(1024, 512 + 11)$ code with $\psi_{t'} = a_j \in \mathcal{A}$

Figure 3.9 illustrates the execution-time reduction $\Delta\mathcal{L}_{\text{SC}}(\psi_{t'})$ in GRM estimated by (3.23), expressed in CCs, for each information bit being the restart location, $\psi_{t'} = a_j \in \mathcal{A}$, of a $(1024, 512 + 11)$ polar code designed according to the 5G reliability order (3GPP, 2018).

Figure 3.9 (a) compares the execution-time reduction induced by the GRM for two different values $P \in \{16, 64\}$. Looking at Figure 3.9, the reduction $\Delta\mathcal{L}_{\text{SC}}(\psi_{t'})$ linearly increases as $\psi_{t'}$ moves further to the RHS of the codeword, and it is upper bounded by the SC reference line. For lower P, $\Delta\mathcal{L}_{\text{SC}}(\psi_{t'})$ is slightly higher for the bits further away on the RHS.

Notably, the SRM utilizes only one possible restart location $\psi_{t'} = N/2$, resulting in a step function. Note that in Figure 3.9 (b), the reductions achieved by the SRM and GRM are not equal for the information bit $j_{\text{RHS}} = 144$. This difference occurs for two reasons. First, unlike the GRM, the SRM cannot restart from the information bit $j_{\text{RHS}} = 144$, which corresponds to the location $j = a_{144} = 543$. Instead, it can only restart from the frozen location $j = 512$. Second, the GRM requires additional CCs for the PS restoration $\Theta(543)$ (3.26). Overall, the reduction of the GRM at $j_{\text{RHS}} = 144$ is higher than the reduction of SRM.

3.2.4.2 Memory Structure

The sketch of the memory of a flip decoder based on SCF with the GRM is provided in Figure 3.10. It is based on the memory architecture proposed by Hashemi *et al.* (2017). For SC decoding, the channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$ and intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$ are quantized using Q_{ch} and Q_{int} bits, respectively. The bit estimates $\hat{\boldsymbol{u}}$ are stored using N bits of memory and the intermediate partial-sums $\boldsymbol{\beta}_{\text{int}}$ require with $N - 1$ bits of storage. For SCF decoding, the metrics of the bit-flipping candidates are quantized using Q_{flip} bits. Each bit-flipping set $\boldsymbol{\varepsilon}_{t'} \in \mathcal{B}_{\text{flip}}$ requires $\omega \times n$ bits, where $n = \log_2(N)$ is the length of the binary representation of a bit-flipping location, $\left|\mathcal{B}_{\text{flip}}\right| = T_{\max} - 1$, and $\omega = |\boldsymbol{\varepsilon}_{t'}|_{\max}$ is the maximum size of a bit-flipping set.

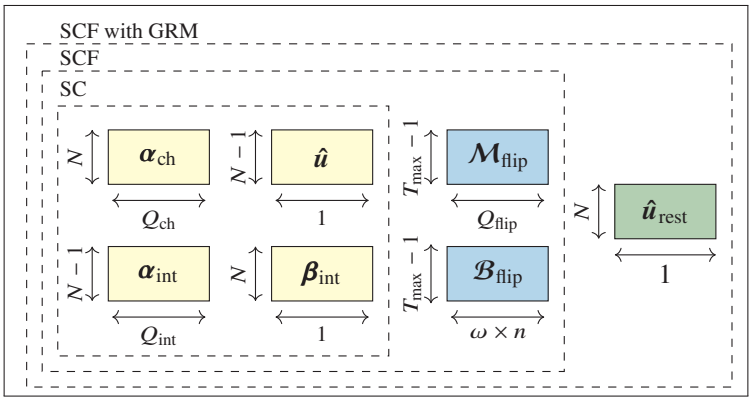


Figure 3.10 Memory sketch of SC, SCF and SCF decoders with the GRM

Embedding the GRM to SCF-based decoder requires additional memory in order to perform the restart path. It corresponds to the storage of the bit estimates $\hat{\boldsymbol{u}}_{\text{rest}}$ of initial SC trial. During the restart path, the intermediate memory $\boldsymbol{\alpha}_{\text{int}}$ is restored with $\boldsymbol{\alpha}_{\varsigma}$, $\varsigma = \{n-1, \ldots, 0\}$, while the $\boldsymbol{\beta}_{\text{int}}$ is restored with $\boldsymbol{\beta}_{\varsigma}$. Thus, the memory overhead of the proposed mechanism is N bits regardless of the value ω. The total memory Λ_{GRM} required for SCF with the GRM is then:

$$\Lambda_{\text{GRM}} = \Lambda_{\text{SC}} + \Lambda_{\text{flip}} + \Lambda_{\text{rest}}. \tag{3.27}$$

The Λ_{GRM} is composed of memories for SC decoding Λ_{SC}, bit-flipping candidates Λ_{flip}, and modified restart in the GRM Λ_{rest}. The size of each of those memories is estimated according to:

$$\Lambda_{\text{SC}} = Q_{\text{ch}} \cdot N + Q_{\text{int}} \cdot (N-1) + 2N - 1, \tag{3.28}$$

$$\Lambda_{\text{flip}} = Q_{\text{flip}} \cdot (T_{\max} - 1) + \omega \cdot n \cdot (T_{\max} - 1)\,, \tag{3.29}$$

$$\Lambda_{\text{rest}} = N. \tag{3.30}$$

3.2.4.3 Partial-Sum Restoration

Using the architecture proposed by Sarkis *et al.* (2014), the encoding operations are performed with (1.16) by calculating $2P$ PS bits in a single encoding stage $s \in \{1, \ldots, \varsigma\}$ in one CC. In this, ς represents a decoding stage where g functions are performed. The total execution time required to restore all PS vectors of interest, $\boldsymbol{\beta}_{\varsigma}$, is denoted $\Theta\left(\psi_{t'}\right)$, and depends on the binary representation $\mathcal{H}_{\psi_{t'}}$ (3.26). At the restart path, a controller has to select the relevant segment of $\hat{\boldsymbol{u}}$, referred to as $\boldsymbol{\eta}_{\varsigma}$ in Definition 3.3, which is on the basis for restoring $\boldsymbol{\beta}_{\varsigma}$. These segments are specific to each of the k_{tot} possible restart locations $\psi_{t'}$, as they require the offset ϕ_{ς} obtained by (3.21). The complexity of this controller is expected to be negligible in terms of resource requirements. Also, the PS restoration does not require additional memory as it is being stored in the intermediate partial-sums $\boldsymbol{\beta}_{\text{int}}$.

3.2.5 Simulation Results

The effects of the GRM on the SCF and DSCF-ω decoders are observed through the simulations. A simulation setup similar to that described in Subsection 2.2.1 of Chapter 2 is applied. The 5G polar codes (3GPP, 2018) of length $N = 1024$ are used with three different rates $R \in \{1/8, 1/4, 1/2\}$. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed.

The values of the maximum number of trials $T_{\max}$ are selected as described in Subsection 2.2.1 of Chapter 2. For the single bit-flip SCF and DSCF-1 decoders, $T_{\max} \in \{13, 8\}$ is set, and for the multi-bit flip DSCF-ω decoders with $\omega \in \{2, 3\}$, $T_{\max} \in \{51, 301\}$ is set. The hardware-friendly metric calculation function (2.2) is applied for all DSCF-ω decoders. The comparisons are made in terms of error-correction performance, memory requirements, and average execution time.

3.2.5.1 Error-Correction Performance

Figure 3.11 depicts the error-correction performance for a $(1024, 512 + 11)$ 5G polar code with SCF, DSCF-1, DSCF-2 and DSCF-3 decoders. The error-correction performance of the SCL decoders with list sizes of $L = 4$ and $L = 8$ are provided for reference. As discussed in Chapter 1, the performance of the SCL decoder with $L = 8$ represents the baseline for the 5G performance evaluation (3GPP, 2018). The results in Figure 3.11 show that the DSCF-3 decoder closely approaches the SCL with $L = 8$ at the target FER of 10^{-2}. Since the 5G polar codes are the targeted codes in this thesis and according to the performance, DSCF-3 is the reference algorithm in the remaining part of this chapter. Our proposed GRM provides a higher execution-time reduction for this decoder.

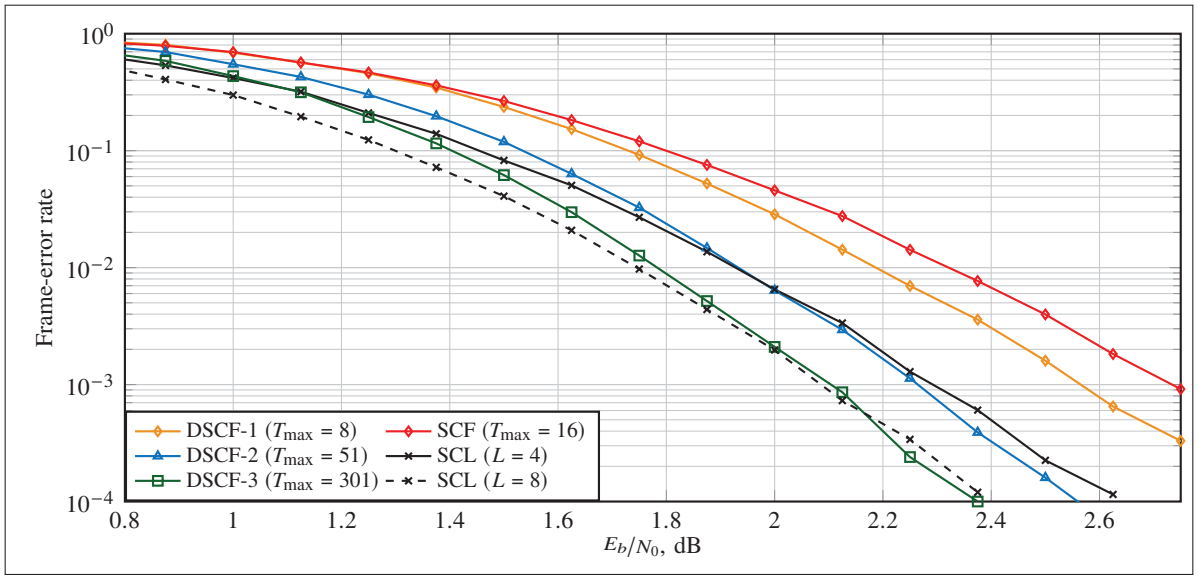


Figure 3.11 Error-correction performance for a $(1024, 512 + 11)$ code of SCF and DSCF-ω decoders with decoding order $\omega \in \{2, 3\}$, and of SCL decoder with $L = 4$ and $L = 8$

Figure 3.12 shows the error-correction performance of the DSCF-3 decoder with and without the GRM. The results indicate that the GRM has no detrimental effect on the error-correction performance. This is expected and in line with the mechanism definition provided in Subsection 3.2.3.

3.2.5.2 Memory Estimations

Memory requirements are estimated with (3.27), where the same quantization scheme described in Subsection 3.1.4.2 is used. This quantization scheme is based on that of Ercan *et al.* (2020a). The blocks based on the LLR values are quantized by $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$, and $Q_{\text{flip}} = 7$ bits, respectively. The sizes of these blocks also depend on the values of N and T_{max}. The memory estimates in bits and the memory overhead are provided in Table 3.3. The estimations show that the proposed GRM leads to a memory overhead of 3.87% to 6.58%. The smallest memory overhead is

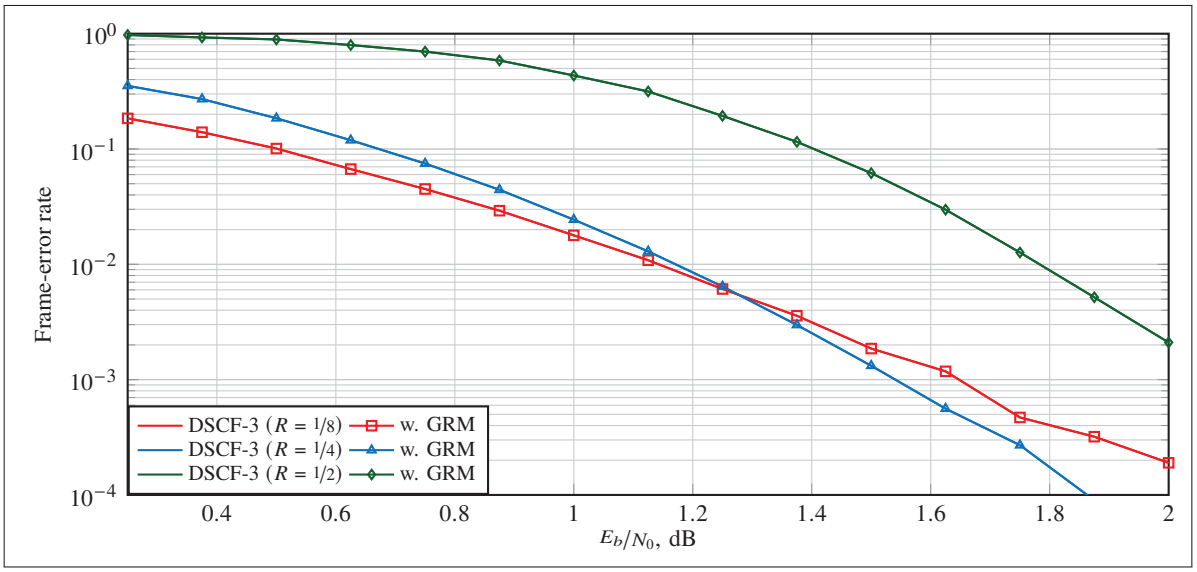


Figure 3.12 Error-correction performance for a $N = 1024$ polar code with $R = \{1/8, 1/4, 1/2\}$ of DSCF-3 decoder with and without the GRM

obtained for the DSCF-3 algorithm. This happens because the DSCF-3 decoder requires the largest memory due to a higher value of $T_{\max}$, which reduces the impact of the GRM.

Table 3.3 Memory estimates and overhead for flip decoders with and without the GRM for a $(1024, 512 + 11)$ code

Decoders	$T_{\max}$	no GRM (bits)	w. GRM (bits)	Mem. overh. (%)
SCF	13	15556	16580	6.58
DSCF-1	8	15471	16495	6.62
DSCF-2	51	16702	17726	6.13
DSCF-3	**301**	**26452**	**27476**	**3.87**

3.2.5.3 Average Execution Time

The equations (3.9)–(3.13) are used to estimate the average execution time and its reduction by applying the GRM to the original flip decoders. The execution-time reduction of the SC trial, $\Delta\mathcal{L}_{\text{SC}}(\psi_{t'})$, calculated by (3.23), is applied in these derivations.

Figure 3.13 shows the average execution time of a DSCF-3 decoder for polar code of the length $N = 1024$ and rates $R = \{1/8, 1/4, 1/2\}$. The presented decoding algorithms are original DSCF-3, DSCF-3 with the GRM, and DSCF-3 with the SRM. Additionally, the execution time of SC decoding is shown for reference, which corresponds to the bound of these decoders. The results show that the decoder with the GRM achieves significant reduction for all code rates and throughout the practical range of the FER.

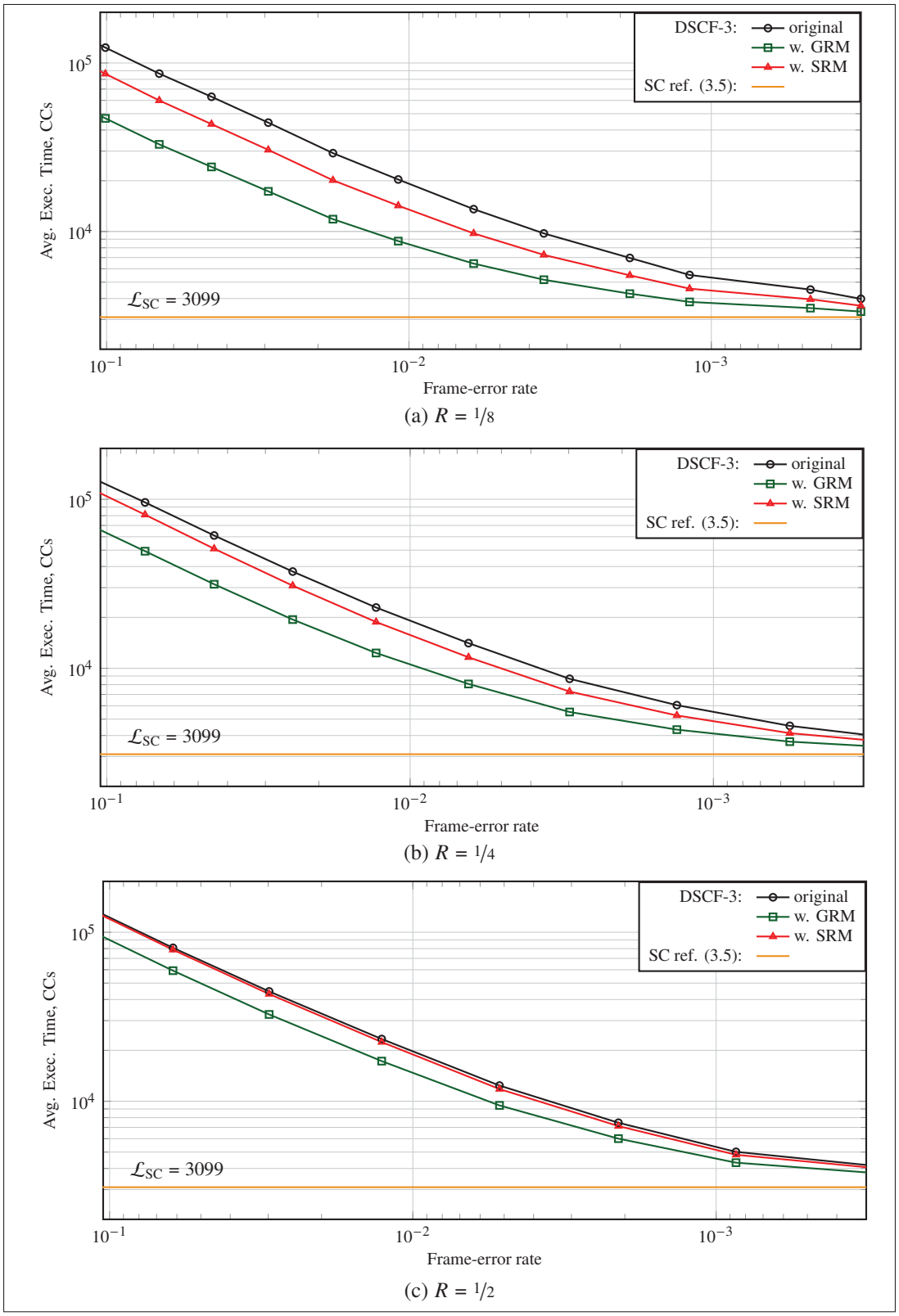


Figure 3.13 Average execution time of DSCF-3 decoder with the GRM, SRM and original decoder for $R \in \{1/8, 1/4, 1/2\}$

The numerical results of the average execution-time reduction compared to the original decoding $\Delta\overline{\mathcal{L}}_{\text{flip}}$, estimated by (3.13) for the GRM, at the target FER of 10^{-2} in percent, are provided in Table 3.4 and Table 3.5. Table 3.4 provides results of applying the GRM to flip decoders, and Table 3.5 summarizes reductions for DSCF-3 decoder with the GRM and SRM. Compared to original decoding, the DSCF-3 with the GRM achieves the reductions in $\Delta\overline{\mathcal{L}}_{\text{flip}}$ of $\{56.90\%, 46.18\%, 26.00\%\}$, while for SRM, it achieves $\{30.05\%, 17.90\%, 4.04\%\}$. Comparing two mechanisms, the GRM provides higher reductions for all code rates. The GRM requires the N bits of additional memory.

Table 3.4 Reduction $\Delta\overline{\mathcal{L}}_{\text{flip}}$ (3.13) by applying the GRM to flip decoders at the FER of 10^{-2}

Decoders		$R = 1/8$		$R = 1/4$		$R = 1/2$	
	$T_{\max}$	E_b/N_0 dB	$\Delta\mathcal{L}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\mathcal{L}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\mathcal{L}_{\text{flip}}$ %
SCF	13	2.00	15.81	1.75	18.06	2.375	10.50
DSCF-1	8	1.75	12.27	1.625	10.81	2.25	5.00
DSCF-2	51	1.375	38.00	1.375	29.46	2.00	15.71
DSCF-3	**301**	**1.125**	**56.90**	**1.125**	**46.18**	**1.75**	**26.00**

Table 3.5 Reduction $\Delta\overline{\mathcal{L}}_{\text{flip}}$ (3.13) by applying the GRM and SRM to DSCF-3 decoder at the FER 10^{-2}

		$R = 1/8$		$R = 1/4$		$R = 1/2$	
mechanism	$T_{\max}$	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %
GRM	301	1.125	**56.90**	1.125	**46.18**	1.75	**26.00**
SRM			30.05		17.90		4.04

3.2.6 Conclusion on Generalized Restart Mechanism

This section described our proposed GRM, an extension of our previously proposed SRM. The GRM improves the execution-time characteristics of the flip decoders by allowing restart from any location of the codeword. In the GRM, the parts of the tree are skipped to estimate a bit-flipping candidate and all of the prior bits of each additional decoding trial. The decoding tree is traversed from the root over the restart path to directly estimate the restart bit. To calculate any g function at the restart path, the PS bits are restored from the bit estimates stored in memory following the initial unsuccessful decoding trial. The partial-sum restorations are shown to be made by low-complexity encoding operations. When applied to DSCF-3 decoder, the proposed GRM reduced the average execution time by 26% to 60%. For the DSCF-3 decoder, this reduction resulted in approximately 4% of additional memory.

3.3 Conclusion

This chapter described two proposed execution-time reduction mechanisms for flip decoders based on SCF: the simplified restart mechanism (SRM) and the generalized restart mechanism (GRM). The SRM is presented with a detailed description of the algorithm. In the SRM, conditional restart for the additional trials is performed from the second half of the codeword if that is where the bit-flipping candidate is located. The proposed SRM improves the execution-time characteristics of the original decoders for the low-rate codes.

The proposed GRM allows skipping parts of the decoding tree to estimate the bit-flipping candidate and all the previous bits in each additional decoding trial. The decoding tree is traversed from the root along the restart path to directly estimate the restart bit. To perform such a restart, the partial-sum (PS) bits are restored from the bit estimates stored in memory following the initial unsuccessful decoding trial.

When applied to DSCF-3, the proposed SRM reduced the average execution time by 4% to 30%. Similarly, the proposed GRM reduced the average execution time by 26% to 60%. For the DSCF-3 decoder, both mechanisms resulted in approximately 4% of additional memory. This decoder achieves the error-correction performance of the state-of-the-art SCL decoder.

Overall, both proposed SRM and GRM have high execution-time reduction capability for flip decoders (SCF and DSCF) without any negative effect on error-correction performance. Both mechanisms require only small additional memory for SCF and DSCF decoders, which is a fractional part of the memory required by the full decoder. Therefore, the proposed mechanisms can result in efficient hardware implementations of flip decoders.

CHAPTER 4

MODIFIED GRM FOR FAST DECODING TECHNIQUES

This chapter describes our proposed modified GRM for flip decoders with fast decoding techniques. The fast decoder techniques, Fast-SSC and LRT, are existing algorithms that reduce the execution time of the flip decoders. In the previous chapter, we described the GRM for flip decoders: SCF and DSCF decoders. We showed that the GRM reduces the average execution time of flip decoders by skipping computations that are identical between the initial and additional trials.

We demonstrate that our proposed GRM can be combined with fast decoding techniques. This combination further reduces the average execution time of flip decoders, in addition to the improvements already achieved by these existing techniques. When applied to the Fast-DSCF-3 decoder, our proposed modified GRM reduces the average execution time by 15% to 22%, in addition to the improvements already achieved by fast decoding. Similar to the standard DSCF decoder, the GRM results in approximately 4% of additional memory.

The following publication resulted from the work presented in this chapter:

- In part: I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Generalized Restart Mechanism for Successive-Cancellation Flip Decoding of Polar Codes", *J. Signal Process. Syst.,* 2024 (*under review*).

4.1 Fast Decoding Techniques

The fast decoding techniques, such as Fast-SSC and LRT, are existing algorithms aimed at reducing the execution time of flip decoders. A comprehensive discussion of the fast decoding techniques is provided in Chapter 1. In this section, we summarize the key aspects of these techniques to establish context and simplify understanding of the proposed GRM. The term *flip decoders* refers to the SCF and DSCF decoders, with SC, FSSC , and SC_{LRT} serving as the baseline algorithms for comparison.

4.1.1 Fast-SSC Decoding

The fast decoding techniques are based on recognizing special nodes at stages $s \geq 2$ of the SC decoding tree. Instead of traversing these nodes as in standard SC, they can be fast decoded. As a result, execution time can be reduced. The SSC decoder, proposed by Alamdar-Yazdi & Kschischang (2011), does not traverse special nodes entirely composed of either frozen bits or information bits: R0 and R1 nodes. The Fast-SSC decoder, proposed by Sarkis *et al.* (2014), further improves the decoding time of SSC by implementing additional types of special nodes. The most notable of these node types are the REP and the SPC nodes.

The Fast-SSC with these four types of special nodes is adapted to SCF decoding, and the Fast-SCF decoder is proposed by Giard & Burg (2018). The hardware implementation of the Fast-SCF decoding is proposed by Ercan *et al.* (2020c). Also, in their study, Ercan *et al.* (2020a) proposed the Fast-DSCF decoding algorithm and its hardware implementation.

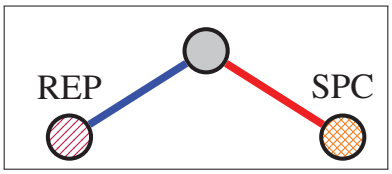


Figure 4.1 Fast-SSC decoding tree for an $(8, 4)$ polar code. Highlighted nodes are REP and SPC nodes

A decoding trial in the Fast-SSC is denoted by FSSC . Figure 1.5 in Chapter 1 illustrated a decoding tree of a polar code with the length of $N = 8$ and $k = 4$ information bits. This tree can be simplified by recognizing one REP node and one SPC node at stage $s = 2$, both with lengths $N_v = 4$. Figure 4.1 illustrates FSSC decoding tree for this scenario. The resulting tree shows how the SC tree with the total of 4 stages and 14 traversing edges is now simplified to the FSSC tree with only 2 stages and 2 traversing edges. Starting from the root node, going left, the f function is calculated (highlighted in blue line) according to (1.11), and then the repetition (REP) node is decoded according to (1.17). Then, going right, the g function is calculated (highlighted in red line) according to (1.12), and then the single-parity-check (SPC) node is decoded according to (1.18). The bit estimates $\hat{\boldsymbol{u}}$ are available right after decoding of these two nodes.

4.1.2 SC with LRT Decoding

In their study, (Giard *et al.*, 2017) proposed an implementation of the SCF decoder, where each SC trial is modified by the LRT. This technique is entirely based on skipping traversal of the left-most frozen bits in the decoding tree, meaning that it resumes the decoding at the first information bit-location a_0. The SC trial with the LRT is denoted by SC_{LRT}, which is defined as $\text{SC}_{\text{LRT}} = \text{SC}\left(a_0, \boldsymbol{\varepsilon}_{t'}\right)$. The SC_{LRT} decoding tree is depicted in Figure 4.2, where the nodes represented in black and white correspond to information and frozen bits. The nodes corresponding to the frozen bits at the LHS of the first information bit are visualized by the dashed line. These nodes are not traversed.

Contrary to SC decoding, the FSSC and SC_{LRT} have variable execution times when a pattern of frozen and information bits changes. This applies to the changes in the code construction or the code rate R. However, decoding latency remains deterministic for the same construction of the polar code.

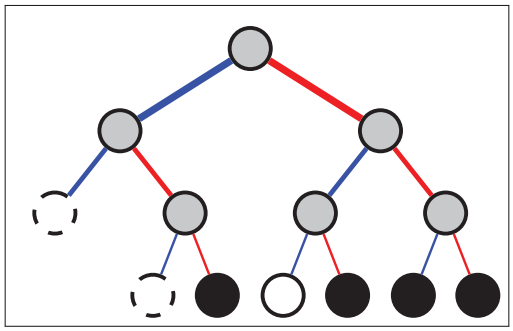


Figure 4.2 $\mathrm{SC}_{\mathrm{LRT}}$ decoding tree for an $(8, 4)$ polar code.
Dashed nodes are skipped from decoding

4.2 Applying GRM with Fast Decoding Techniques

The fast decoding techniques, Fast-SSC and LRT, are the existing techniques that reduce the execution time of the SC decoding. When applied to the flip decoders, SCF or DSCF, these techniques permit to reduce the average execution time. Our proposed GRM, presented in Chapter 3, is flexible to apply with these fast decoding techniques. As a result of this combination, the average execution time of flip decoders can be further reduced.

In Chapter 3, the standard SC was considered as the baseline algorithm (dec = SC). Also, the modified SC for an additional trial of the flip decoder with the GRM was denoted by $\mathrm{SC}\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$, where $\boldsymbol{\varepsilon}_{t'}$ is the bit-flipping set and t' is the index of an additional trial. In this chapter, the FSSC and $\mathrm{SC}_{\mathrm{LRT}}$ serve as the baseline algorithms "dec" for the flip decoders (dec $\in \{\mathrm{FSSC}, \mathrm{SC}_{\mathrm{LRT}}\}$). By applying GRM to the additional trial of SC with LRT, it is denoted by $\mathrm{SC}\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$, similarly to the standard SC.

In a flip decoder with Fast-SSC as the baseline algorithm, an additional trial is denoted by $\mathrm{FSSC}\left(\boldsymbol{\varepsilon}_{t'}\right)$. The restart path of $\mathrm{FSSC}\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$ is made with the following assumptions:

1. If the bit-flipping index $i_1 = \varepsilon_{t'}(0)$ belongs to a special node, the FSSC $\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$ begins with this special node.
2. If $i_1 = \varepsilon_{t'}(0)$ is located outside of a special node, the FSSC $\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$ begins by decoding the first information bit on the RHS of i_1, same as for SC $\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$.
   - If $\hat{u}_{\psi_{t'}}$ falls in a special node, the FSSC $\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$ begins with this special node.

The same assumptions are made for the Fast-DSCF decoder. The Fast-SCF and Fast-DSCF decoders with the GRM will further reduce the average execution time of the standard SCF and DSCF decoders.

Table 4.1 summarizes execution-time reduction mechanisms, where the GRM is applied with various baseline algorithms. The notations provided in Table 4.1 are used throughout this chapter for simplicity.

Table 4.1 Summary of the key differences between execution-time reduction mechanisms for flip decoders based on SCF

Flip decoder	Baseline algorithm dec	Additional trial decoder	Restart Mechanism	Restart index (add. trial)	Memory overhead	Label
SCF, DSCF	SC	SC ($\boldsymbol{\varepsilon}_{t'}$)	–	0	0%	SCF, DSCF-ω
SCF with LRT	SC_{LRT}	SC ($a_0, \boldsymbol{\varepsilon}_{t'}$)	–	a_0	0%	LRT
Fast-SCF, Fast-DSCF	Fast-SSC	FSSC($\boldsymbol{\varepsilon}_{t'}$)	–	a_0	0%	FAST
SCF with SRM	SC	SC(0 or $N/2, \boldsymbol{\varepsilon}_{t'}$)	SRM	0 or $N/2$	4% to 7%	SRM
This study	SC	SC ($\psi_{t'}, \boldsymbol{\varepsilon}_{t'}$)	GRM	$\psi_{t'}$	4% to 7%	GRM
	SC_{LRT}	SC ($\psi_{t'}, \boldsymbol{\varepsilon}_{t'}$)	GRM	$\psi_{t'}$	4% to 7%	LRT+GRM
	Fast-SSC	FSSC ($\psi_{t'}, \boldsymbol{\varepsilon}_{t'}$)	GRM	$\psi_{t'}$	4% to 7%	FAST+GRM

4.3 Time and Resource Analysis

In this chapter, the execution-time model described in Section 2.5 of Chapter 2 is applied. This model is based on the semi-parallel SC decoder implementation proposed by Leroux *et al.* (2013). The equations (4.1)–(4.3) provided below correspond to equations (3.5)–(3.7) provided in Chapter 3.

The execution time of the standard SC decoding algorithm, denoted by $\mathcal{L}_{\text{SC}}$, is summarized as follows:

$$\mathcal{L}_{\text{SC}} = \mathcal{L}_{\alpha}(N) + \mathcal{L}_{\beta}(N), \tag{4.1}$$

where $\mathcal{L}_{\alpha}(N)$ and $\mathcal{L}_{\beta}(N)$ are the LLR and PS calculations for the polar code of length N, and each of them is calculated by the following equations:

$$\mathcal{L}_{\alpha}(N) = 2N + \frac{N}{P} \cdot \log_2\left(\frac{N}{4P}\right), \tag{4.2}$$

$$\mathcal{L}_{\beta}(N) = \sum_{s=1}^{n-1} (2^{n-s} - 1) \cdot \left\lceil \frac{2^s}{2P} \right\rceil, \tag{4.3}$$

where $\mathcal{L}_{\alpha}(N)$ (4.2) was given by Leroux *et al.* (2013), while $\mathcal{L}_{\beta}(N)$ (4.3) is derived by using the results of Sarkis *et al.* (2014). The term $(2^{n-s} - 1)$ in (4.3) indicates the number of nodes at tree stage s, excluding the right-most node. The term $2^s/2P$ corresponds to the number of CCs required to compute PS for each node. For the values of $P \geq N/4$, (4.3) is simplified to $\mathcal{L}_{\beta}(N) = N - n - 1$, as shown in our work of Sagitov *et al.* (2023). Note that "-1" in (4.3) indicates that the final computations (going up on the right edges of the last leaf in the tree) are not performed.

The execution time of PS calculations, which include the final computations, is further denoted by $\mathcal{L}_{\beta_{\text{full}}}(N)$. It differs from (4.3) and is calculated as follows:

$$\mathcal{L}_{\beta_{\text{full}}}(N) = \sum_{s=1}^{n-1} \left(2^{n-s} \cdot \left\lceil \frac{2^s}{2P} \right\rceil \right). \tag{4.4}$$

The decoding of a node of size N_v is related to a full SC decoding of a code of size N, where the additional PS calculations are made after estimating the last bit. Hence, the execution time $\mathcal{L}_{\text{node}}(N_v)$ to decode a node of size $N_v = 2^s$ is obtained using (4.1), (4.2) and (4.4) as follows:

$$\mathcal{L}_{\text{node}}(N_v) = \mathcal{L}_{\alpha}(N_v) + \mathcal{L}_{\beta_{\text{full}}}(N_v). \tag{4.5}$$

The execution time to decode the node v (4.5) is introduced to estimate the decoding time of special nodes in FSSC decoding.

The following assumptions are made to estimate the execution time of the FSSC $(\boldsymbol{\varepsilon}_{t'})$ and FSSC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ trials:

1. The lengths of the special nodes N_v are based on hardware implementations proposed by Giard & Burg (2018) and Ercan *et al.* (2020a).
2. Decoding of a special node permits to retrieve the N_v bit estimates and N_v PS bits in $\left\lceil \frac{N_v}{P} \right\rceil$ CCs.
3. At the restart path of FSSC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, the PS restorations are performed in parallel with a maximum of $2P$ operations in one CC within each encoding stage.
4. Outside of the special nodes, we keep the same assumptions as for the SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$.

Applying the FSSC incurs additional complexity to flip decoders. This complexity can increase with the higher decoding order ω, resulting from the flipping and fast decoding of special nodes. Upper bounds on the special node lengths N_v are applied to reduce this complexity.

Table 4.2 Upper bounds on the length N_v of special nodes in FSSC for the Fast-SCF and Fast-DSCF-ω decoders with $\omega = \{1, 2, 3\}$

Order ω	REP	R1	SPC
1	$N_v \leq 32$	$N_v \leq 64$	$N_v \leq 64$
2	$N_v \leq 32$	$N_v \leq 64$	$N_v \leq 8$
3	$N_v \leq 32$	$N_v \leq 64$	$N_v = 4$

Table 4.2 provides the upper bounds on the lengths N_v of special nodes in FSSC for the Fast-SCF and Fast-DSCF-ω decoders. The values of N_v for the REP and R1 nodes are selected according to the implementation results of

Giard & Burg (2018), and for the SPC nodes according to the implementation of Ercan *et al.* (2020a). The R0 nodes are unlimited in size and can be decoded in 1 CC according to Sarkis *et al.* (2014).

Based on the information set of the polar code and the node constraints provided in Table 4.2, a sequence of special nodes can be found. This sequence is denoted by $\{n_1, \ldots, n_j, \ldots, n_i\}$, where i is the total number of special nodes, and n_j is the j^{th} node in the sequence (classified according to the order of apparition). Following that, the execution time of the FSSC trial, denoted by $\mathcal{L}_{\text{FSSC}}$, is estimated as follows:

$$\mathcal{L}_{\text{FSSC}} = \mathcal{L}_{\text{SC}} - \Delta_{n_q}, \tag{4.6}$$

where $\mathcal{L}_{\text{SC}}$ denotes the execution time of the SC decoder (4.1), and Δ_{n_q} is the execution-time reduction by fast decoding the q special nodes of a polar code.

4.3.1 Execution-Time Reduction Capability of the GRM

The execution-time reduction provided by our proposed GRM to the standard SC trial is described in detail in Chapter 3. This subsection summarizes key equations of the GRM to establish the context and simplify the explanations for the FSSC and SC_{LRT} decoding. The equations (4.7)–(4.11) provided below correspond to equations (3.22)–(3.26) defined in Chapter 3.

The execution time of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ is denoted by $\mathcal{L}_{\text{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})}$, which is obtained as follows:

$$\mathcal{L}_{\text{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})} = \mathcal{L}_{\text{SC}} - \Delta\mathcal{L}_{\text{SC}}\left(\psi_{t'}\right), \tag{4.7}$$

where $\Delta\mathcal{L}_{\text{SC}}\left(\psi_{t'}\right)$ denotes the execution-time reduction of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$. The execution-time reduction provided by the GRM corresponds to the skipped CCs that would otherwise be required to calculate LLR and PS for the modified restart. This reduction is denoted by $\Delta\mathcal{L}_{\text{SC}}\left(\psi_{t'}\right)$ and is summarized as:

$$\Delta\mathcal{L}_{\text{SC}}\left(\psi_{t'}\right) = \Delta\mathcal{L}_{\alpha}\left(\psi_{t'}\right) + \Delta\mathcal{L}_{\beta}\left(\psi_{t'}\right) - \Theta\left(\psi_{t'}\right), \tag{4.8}$$

where $\Delta\mathcal{L}_{\alpha}\left(\psi_{t'}\right)$ and $\Delta\mathcal{L}_{\beta}\left(\psi_{t'}\right)$ denote the skipped CCs by avoiding LLR and PS computations required to decode bit estimates $\{\hat{u}_0, \ldots, \hat{u}_{\psi_{t'}-1}\}$. The component $\Theta\left(\psi_{t'}\right)$ corresponds to the additional CCs required to calculate all

PS at the restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$. These functions, expressed in CCs, are calculated as follows:

$$\Delta\mathcal{L}_{\alpha}(\psi_{t'}) = \sum_{s=0}^{n-1}\left(\left\lfloor\frac{\psi_{t'}}{2^s}\right\rfloor \cdot \left\lceil\frac{2^s}{P}\right\rceil\right), \tag{4.9}$$

$$\Delta\mathcal{L}_{\beta}(\psi_{t'}) = \sum_{s=1}^{n-1}\left(\left\lfloor\frac{\psi_{t'}}{2^s}\right\rfloor \cdot \left\lceil\frac{2^s}{2P}\right\rceil\right). \tag{4.10}$$

In both equations, $\left\lfloor\frac{\psi_{t'}}{2^s}\right\rfloor$ indicates the number of skipped nodes at tree stage s. The second term corresponds to the number of skipped CCs to compute LLR and PS for each node of length $N_v = 2^s$ according to (4.5). As a reminder, P processing elements are used for LLRs, as proposed by Leroux *et al.* (2013), while $2P$ processing elements are used for PS as proposed by Sarkis *et al.* (2014). If $s = 0$, the PS bits correspond to bit estimates, and no computations are required. In this case, the summation starts from $s = 1$ as indicated in (4.10).

The component $\Theta(\psi_{t'})$ in (4.8) corresponds to the additional CCs for the PS restoration during the restart path of SC $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$. This component is subtracted from overall reduction $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ (4.7). In this study, the time model proposed by Sarkis *et al.* (2014) is applied. In that scheme, $2P$ bits of PS are calculated according to (1.16) in one CC. Based on this, the additional time component $\Theta(\psi_{t'})$, expressed in CCs, is calculated as follows:

$$\Theta(\psi_{t'}) = \sum_{s=1}^{n-1}\left(\mathcal{H}_{\psi_{t'}}(n-1-s) \cdot \left\lceil\frac{2^s}{2P}\right\rceil \cdot s\right), \tag{4.11}$$

where $\boldsymbol{\mathcal{H}}_{\psi_{t'}} = \left\{\mathcal{H}_{\psi_{t'}}(0), \ldots, \mathcal{H}_{\psi_{t'}}(n-1)\right\}$ is the binary representation of $\psi_{t'}$, with the LSB is on the right, as defined in Subsection 3.1.2 of Chapter 3.

Execution-Time Reduction of the LRT

The execution time of $\mathrm{SC}_{\mathrm{LRT}}$ decoding trial can be calculated using the same equation (4.7) as for the GRM, where the restart location is $\psi_{t'} = a_0$. This execution time, denoted by $\mathcal{L}_{\mathrm{SC}_{\mathrm{LRT}}}$, is estimated as follows:

$$\mathcal{L}_{\mathrm{SC}_{\mathrm{LRT}}} = \mathcal{L}_{\mathrm{SC}} - \Delta\mathcal{L}_{\alpha}(a_0) - \Delta\mathcal{L}_{\beta}(a_0), \tag{4.12}$$

where $\Delta\mathcal{L}_{\alpha}(a_0)$ and $\Delta\mathcal{L}_{\beta}(a_0)$ denote the skipped CCs by avoiding LLR and PS computations brought by resuming SC trial from the first information bit a_0. Both are calculated using (4.9) and (4.10).

Table 4.3 displays the execution time of SC, $\mathrm{SC}_{\mathrm{LRT}}$ and Fast-SSC decoders, estimated by (4.5), (4.12) and (4.6). For these estimations, polar codes with a length $N = 1024$ and rates $R \in \{1/8, 1/4, 1/2\}$ are applied, designed according to the 5G standard. For the Fast-SSC decoder, the limits to N_v are applied according to the line with $\omega = 1$ in Table 4.2, and the number of processing elements $P = 64$.

Table 4.3 Execution time (in CCs) of SC, $\mathrm{SC_{LRT}}$ and Fast-SSC decoders for the 5G polar codes. The limits on length N_v in Fast-SSC are applied according to $\omega = 1$ in Table 4.2

Polar code	$\mathcal{L}_{\mathrm{SC}}$	$\mathcal{L}_{\mathrm{SC_{LRT}}}$	$\mathcal{L}_{\mathrm{FSSC}}$
$(1024, 128 + 11)$	3099	1671	483
$(1024, 256 + 11)$	3099	2349	666
$(1024, 512 + 11)$	3099	2732	816

4.3.2 Execution-Time Reduction of Combined GRM with $\mathrm{SC_{LRT}}$ and FSSC

The execution-time reduction achieved by applying the GRM to the additional trial of SC with LRT, referred to as $\mathrm{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, is denoted by $\Delta\mathcal{L}_{\mathrm{SC_{LRT}}}(\psi_{t'})$. This reduction, expressed in CCs, is calculated as follows:

$$\Delta\mathcal{L}_{\mathrm{SC_{LRT}}}(\psi_{t'}) = \Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}) - \Delta\mathcal{L}_{\alpha}(a_0) - \Delta\mathcal{L}_{\beta}(a_0), \tag{4.13}$$

where $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ is the reduction brought by the GRM applied to the standard SC trial calculated by (4.8). The component $\Theta(\psi_{t'})$ (4.11), added by the GRM, is included in $\Delta\mathcal{L}_{\mathrm{SC_{LRT}}}(\psi_{t'})$ through $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ (4.8). By combining two mechanisms, the total execution-time reduction will be increased. However, the reduction brought by the GRM with $\mathrm{SC_{LRT}}$ as a baseline (4.13) is lower than that with SC as a baseline (4.8), i.e., $\Delta\mathcal{L}_{\mathrm{SC_{LRT}}}(\psi_{t'}) < \Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$.

The execution-time reduction of $\mathrm{FSSC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ by restarting at $\psi_{t'}$ is denoted by $\Delta\mathcal{L}_{\mathrm{FSSC}}(\psi_{t'})$. This reduction is estimated as:

$$\Delta\mathcal{L}_{\mathrm{FSSC}}(\psi_{t'}) = \Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'}) - \Delta_{n_j}, \tag{4.14}$$

where Δ_{n_j} denotes the reduction from the standard SC by decoding the special nodes $\{n_1, \ldots, n_j\}$, with j being the index of the last special node on the LHS of $\psi_{t'}$. In (4.14), additional CCs to calculate all PS induced by the GRM, $\Theta(\psi_{t'})$ (4.11), are included in $\Delta\mathcal{L}_{\mathrm{SC}}(\psi_{t'})$ (4.8).

Figure 4.3 illustrates the execution-time reduction $\Delta\mathcal{L}_{\mathrm{dec}}(\psi_{t'})$ with the baseline algorithms $\mathrm{dec} \in \{\mathrm{SC}, \mathrm{SC_{LRT}}, \mathrm{FSSC}\}$, estimated by (4.8), (4.13) and (4.14). Estimations are made for each information bit being the restart location, $\psi_{t'} = a_j \in \mathcal{A}$, of a $(1024, 256 + 11)$ polar code. In all scenarios, the number of processing elements is set to $P = 64$, and the code is designed according to 5G specifications. The special nodes in the Fast-SSC are applied with $\omega = 1$ according to Table 4.2. The lines representing the execution time of SC, $\mathrm{SC_{LRT}}$, and FSSC are displayed for reference and correspond to a bound on the reduction.

Figure 4.3 (a) compares the execution-time reduction of the Fast-SSC with the GRM and the standard SC with the GRM. The results show that the execution-time reduction capability brought by GRM follows a similar pattern

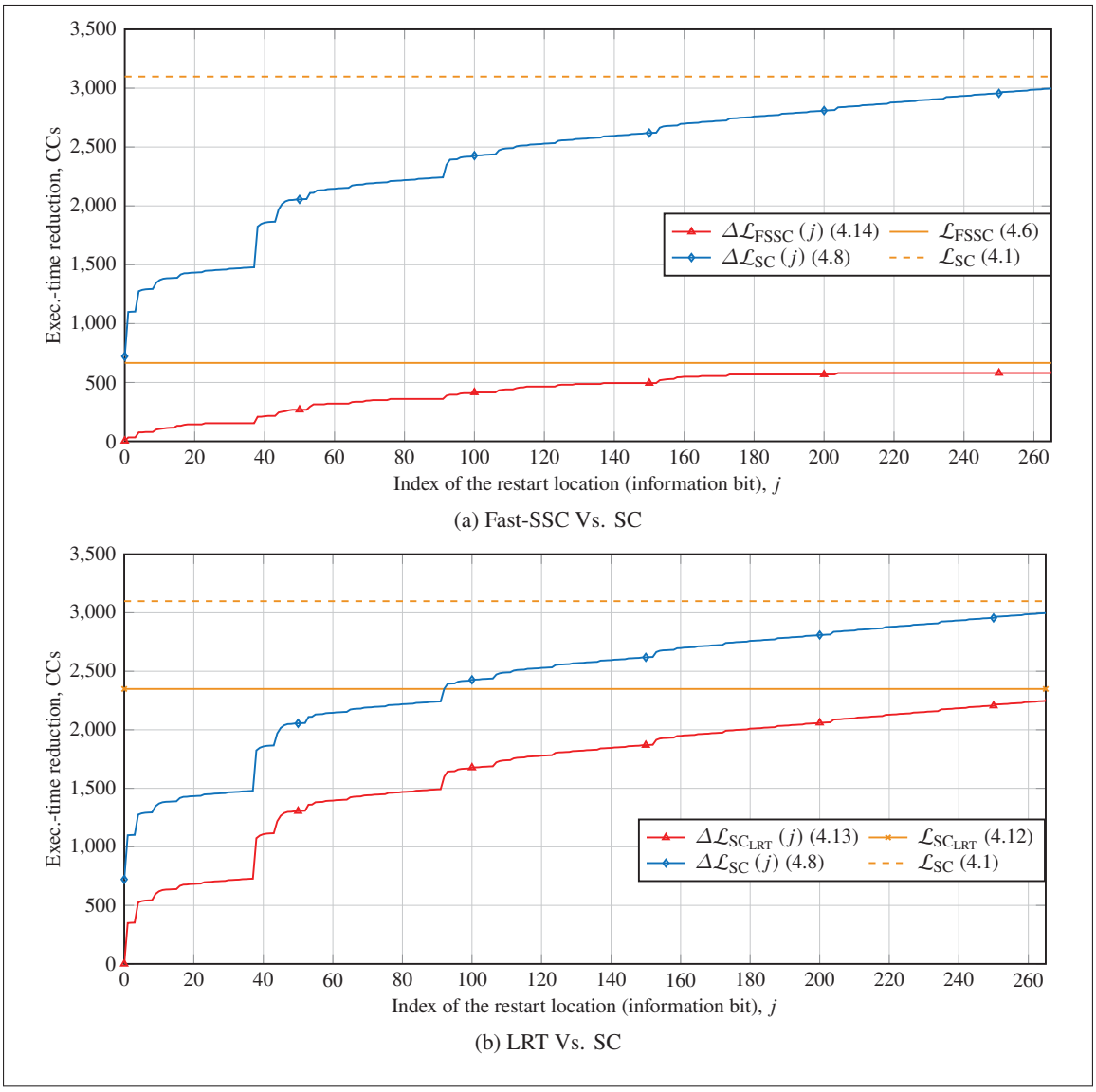


Figure 4.3 Execution-time reduction capability of the GRM with FSSC and SC_{LRT} with $P = 64$ for a $(1024, 256 + 11)$ polar code with $\psi_{t'} = a_j \in \mathcal{A}$

when applied to both the Fast-SSC and SC decoder. This reduction increases as $\psi_{t'}$ moves further to the RHS of the codeword, and it is upper bounded by the reference Fast-SSC or SC decoding algorithm. The results also show that the execution time of the FSSC is lower compared to the standard SC. However, the GRM can further reduce the execution time when applied to FSSC.

Figure 4.3 (b) compares the LRT with the GRM and the standard SC with the GRM. The results show that the execution-time reduction capability brought by GRM follows a similar pattern when applied to both SC_{LRT} and SC decoders. This reduction increases as $\psi_{t'}$ moves further to the RHS of the codeword, and it is upper bounded by the

reference SC_{LRT} or SC. Also, the results of Figure 4.3 (b) indicate that the execution time of the SC_{LRT} is lower compared to the standard SC. However, the GRM can further reduce the execution time when applied to the SC_{LRT}.

4.4 Simulation Results

The effects of the proposed GRM to flip decoders implementing fast decoding techniques are observed through the simulations. A simulation setup similar to that described in Subsection 2.2.1 of Chapter 2 is applied. The 5G polar codes (3GPP, 2018) of length $N = 1024$ are used with three different rates $R \in \{1/8, 1/4, 1/2\}$. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. The BPSK modulation is used over an AWGN channel. The results are focused on the DSCF-3 decoder (original, DSCF-3 with LRT, and Fast-DSCF-3). This decoder provides the error-correction performance, which fulfills the requirements of the eMBB service in 5G, as explained in Chapter 2.

The values of the maximum number of trials $T_{\max}$ are selected as described in Subsection 2.2.1 of Chapter 2. For all variations of the single bit-flip SCF and DSCF-1 decoders, the maximum number of trials is set to $T_{\max} \in \{13, 8\}$. For all variations of the multi-bit flip DSCF-ω decoders with $\omega \in \{2, 3\}$, $T_{\max} \in \{51, 301\}$ is set. The lengths of special nodes and the baseline latency of Fast-SSC in the Fast-SCF and the Fast-DSCF-ω decoders are set as described in Subsection 4.3.1 of this chapter. The comparisons are made in terms of the error-correction performance and average execution time. In this study, the memory of the flip decoders with fast decoding techniques is considered the same as that of standard flip decoders. This memory structure was discussed in Subsection 3.1.3.1 of Chapter 3.

4.4.1 Average Execution Time

The average execution time of applying the proposed GRM to flip decoders is estimated using C simulated codewords. The baseline decoding algorithms dec $\in \{\text{SC}, \text{SC}_{\text{LRT}}, \text{FSSC}\}$ are applied to flip decoders. In this chapter, we focus on SC_{LRT} and FSSC baseline algorithms. The key equations (4.15), (4.16), and (4.17) provided below correspond to equations (3.10), (3.12), and (3.13) in Chapter 3.

By applying the GRM, the average execution time to decode one codeword is derived as follows:

$$l_{\text{flipGRM}}(c) = l_{\text{flip}}(c) - \sum_{t'=1}^{\tau'(c)} \Delta\mathcal{L}_{\text{dec}}(\psi_{t'}), \tag{4.15}$$

where $\Delta\mathcal{L}_{\text{dec}}(\psi_{t'})$ indicates the execution-time reduction of a baseline decoding algorithm dec $\in \{\text{SC}_{\text{LRT}}, \text{FSSC}\}$. These reductions are calculated by (4.13) and (4.14). Then, the average execution time of the flip decoder with the GRM, denoted by $\overline{\mathcal{L}}_{\text{flipGRM}}$, is obtained using C simulated codewords as follows:

$$\overline{\mathcal{L}}_{\text{flipGRM}} = \frac{1}{C}\sum_{c=0}^{C-1} l_{\text{flipGRM}}(c)\,. \tag{4.16}$$

The reduction of the average execution time of a flip decoder with the GRM compared to the original flip decoder, denoted by $\Delta\overline{\mathcal{L}}_{\text{flip}}$ (2.12), is obtained as follows:

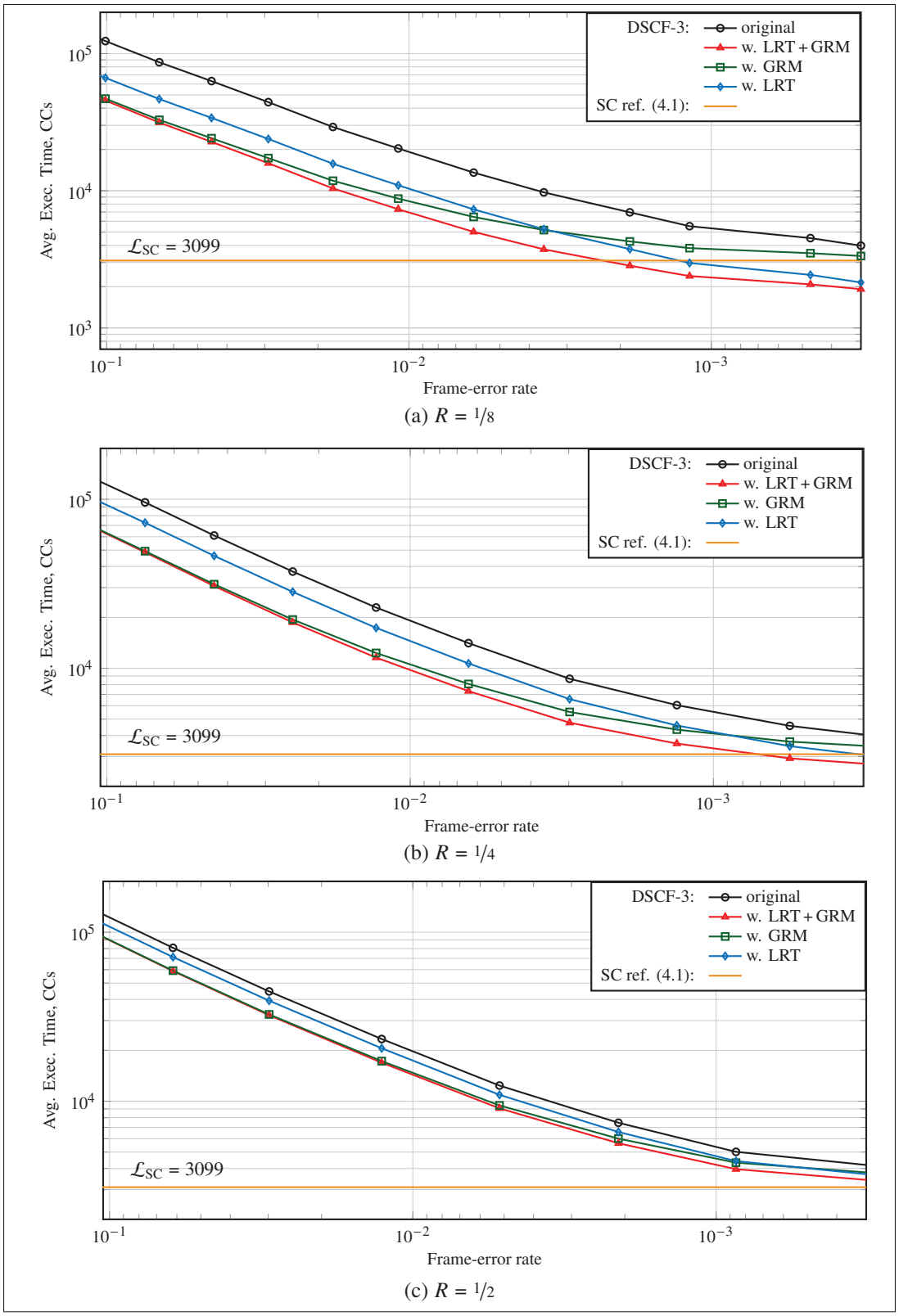
$$\Delta\overline{\mathcal{L}}_{\text{flip}} = \frac{\overline{\mathcal{L}}_{\text{flip}} - \overline{\mathcal{L}}_{\text{flipGRM}}}{\overline{\mathcal{L}}_{\text{flip}}}. \tag{4.17}$$

Figure 4.4 shows the average execution time of a DSCF-3 decoder with and without the GRM. In these results, dec $\in \{\text{SC}, \text{SC}_{\text{LRT}}\}$ are applied as baseline algorithms for the DSCF-3 decoder. The legend labels correspond to the mechanisms listed in Table 4.1. Comparing the results of the sole GRM and sole LRT, the GRM shows a higher reduction. However, as the FER decreases, LRT surpasses the GRM and even SC. This is especially evident for a low-rate code of $R = 1/8$, where LRT surpasses the reference SC at around FER point of 10^{-3}. This happens due to the differences in baseline algorithms. The SC_{LRT} is the baseline algorithm in DSCF-3 with the LRT, whereas the original SC is the baseline in DSCF-3 with the GRM. This difference is more noticeable at low FER, where the initial trials tend to prevail on average.

The results in Figure 4.4 also indicate that the combined mechanism, LRT+GRM, achieves a lower average execution time compared to other mechanisms. Notably, for a low-rate code of $R = 1/8$, the LRT+GRM has a higher reduction at the target FER of 10^{-2}, compared to other mechanisms. However, for rates $R \in \{1/4, 1/2\}$, the difference between this combined mechanism and the sole GRM is negligible at this FER.

Figure 4.5 shows the average execution time of a DSCF-3 decoder with and without the GRM. In these results, dec $\in \{\text{SC}, \text{FSSC}\}$ are applied as baseline algorithms for a DSCF-3 decoder. The results show that the Fast-DSCF-3 decoder significantly reduces the average execution time compared to the original DSCF-3 decoder. The results also show that the Fast-DSCF-3 with the GRM achieves a noticeable reduction compared to the standard Fast-DSCF-3 decoder. This reduction remains significant throughout the practical FER range of 10^{-2} to 10^{-3}.

The numerical results of the average execution-time reduction by applying the GRM compared to the original decoders $\Delta\overline{\mathcal{L}}_{\text{flip}}$, estimated by (4.17), are provided in Table 4.4 and Table 4.5. The results are provided in percentages at the target FER of 10^{-2}. The reductions for flip decoders with the GRM using different baseline algorithms are provided in Table 4.4. Compared to the original decoding, DSCF-3 with the GRM and with SC



(a) $R = 1/8$

(b) $R = 1/4$

(c) $R = 1/2$

Figure 4.4 Average execution time of DSCF-3 decoder with and without the proposed GRM for dec $\in \{\text{SC}, \text{SC}_{\text{LRT}}\}$ and for $R \in \{1/8, 1/4, 1/2\}$

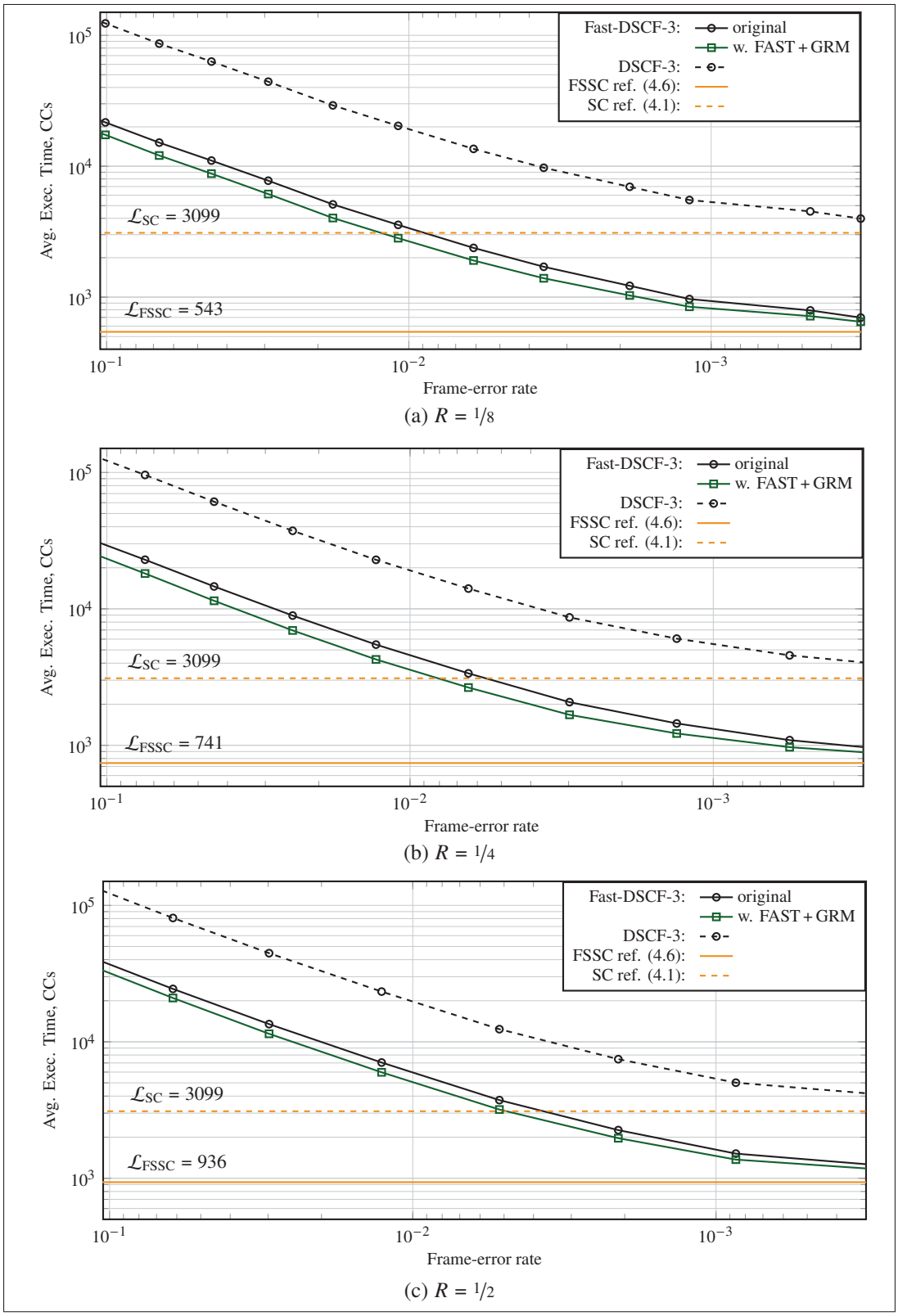


Figure 4.5 Average execution time of DSCF-3 decoder with and without the proposed GRM for dec $\in$ {SC,FSSC} and for $R \in \{1/8, 1/4, 1/2\}$

as a baseline algorithm achieves the reductions $\Delta\overline{\mathcal{L}}_{\text{flip}}$ of $\{56.90\%, 46.18\%, 26.00\%\}$, for rates $R \in \{1/8, 1/4, 1/2\}$, respectively. On the other hand, the DSCF-3 with GRM and SC_{LRT} as a baseline achieves reductions $\Delta\overline{\mathcal{L}}_{\text{flip}}$ of $\{33.09\%, 33.32\%, 17.83\%\}$. The overall reductions of DSCF-3 with LRT + GRM compared to the original DSCF-3 decoder are approximately $\{64\%, 50\%, 28\%\}$.

Table 4.4 Reduction $\Delta\overline{\mathcal{L}}_{\text{flip}}$ (4.17) by applying the GRM to flip decoders with different baseline algorithms at the FER 10^{-2}

			$R = 1/8$		$R = 1/4$		$R = 1/2$	
dec	flip	$T_{\max}$	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %
SC	SCF	13	2.00	15.81	1.75	18.06	2.375	10.50
	DSCF-1	8	1.75	12.27	1.625	10.81	2.25	5.00
	DSCF-2	51	1.375	38.00	1.375	29.46	2.00	15.71
	DSCF-3	**301**	**1.125**	**56.90**	**1.125**	**46.18**	**1.75**	**26.00**
SC_{LRT}	SCF	13	2.00	13.22	1.75	16.24	2.375	9.50
	DSCF-1	8	1.75	8.53	1.625	8.76	2.25	4.03
	DSCF-2	51	1.375	24.09	1.375	22.61	2.00	11.81
	DSCF-3	**301**	**1.125**	**33.09**	**1.125**	**33.32**	**1.75**	**17.83**
FSSC	SCF	13	2.00	10.27	1.75	14.67	2.375	10.60
	DSCF-1	8	1.75	5.90	1.625	7.10	2.25	4.24
	DSCF-2	51	1.375	16.03	1.375	16.24	2.00	11.10
	DSCF-3	**301**	**1.125**	**20.87**	**1.125**	**22.14**	**1.75**	**15.21**

Table 4.5 Reduction $\Delta\overline{\mathcal{L}}_{\text{flip}}$ (4.17) by applying the GRM, SRM and LRT to DSCF-3 decoder at the FER 10^{-2}

		$R = 1/8$		$R = 1/4$		$R = 1/2$	
Mechanism	$T_{\max}$	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %	E_b/N_0 dB	$\Delta\overline{\mathcal{L}}_{\text{flip}}$ %
GRM			**56.90**		**46.18**		**26.00**
SRM	301	1.125	30.05	1.125	17.90	1.75	4.04
LRT			46.08		24.20		11.84

The results in Table 4.4 also indicate that DSCF-3 with the GRM and FSSC as a baseline algorithm achieves the reductions $\Delta\overline{\mathcal{L}}_{\text{flip}}$ of $\{20.87\%, 22.14\%, 15.21\%\}$ for rates $R \in \{1/8, 1/4, 1/2\}$, respectively. The overall reductions of DSCF-3 with FSSC + GRM are approximately $\{86\%, 81\%, 74\%\}$. The reductions for DSCF-3 decoder with the

GRM, SRM, and LRT compared to the original DSCF-3 are provided in Table 4.5. These results indicate that the GRM achieves higher reductions compared to SRM and LRT.

According to the results of Table 4.4 and Table 4.5, the higher reductions are achieved for the flip decoders with SC as the baseline algorithm, while the lower reductions are observed for those with FSSC as the baseline. This is expected, as GRM is an additional mechanism applied to the fast decoder, which already provides a reduction compared to the original decoding. Nevertheless, the reduction provided by the GRM is significant. The proposed GRM requires N bits of additional memory for all flip decoders with any baseline algorithm.

4.5 Conclusion

This chapter described our proposed modified GRM for flip decoders with fast decoding techniques. Our previously proposed GRM can reduce the average execution time of flip decoders by skipping computations that are identical between the initial and additional trials.

We demonstrated that our proposed GRM can be combined with state-of-the-art fast decoding techniques. When applied to the Fast-DSCF-3 decoder, our proposed modified GRM reduced the average execution time by 15% to 22%, in addition to the improvements already achieved by fast decoding. Similar to the standard DSCF-3 decoder, the GRM resulted in approximately 4% of additional memory.

CHAPTER 5

RESTART MECHANISMS FOR LIST-FLIP DECODERS

This chapter describes how the restart mechanisms we proposed in the previous chapters can be adapted to the list-flip decoders based on SCLF decoding algorithm. In the GRM, the part of the tree to estimate a path-flipping candidate and all the previous bits is skipped for each additional decoding trial. This chapter describes that applying the GRM to the list-flip decoder results in enormous memory overhead.

To overcome the issue of memory overhead, this chapter further proposes the limited-locations restart mechanism (LLRM), a modification of the GRM. In the LLRM, the restart locations are constrained to a small set of values. The path information of the initial SCL decoding is stored only for these locations. Therefore, memory requirements can be significantly reduced. These locations must be carefully selected to maximize the reduction of average execution time.

This chapter also describes our proposed probability-based method to determine the restart locations in the LLRM. This method selects the locations that equally divide a codeword in a way that each segment has an equal path-flipping probability distribution.

Compared to original decoding, DSCLF-3 with the LLRM with probability-based restart locations reduced the average execution time by 10% to 40%. Similarly, the GRM reduced the average execution time by 13% to 43%. For this decoder, the LLRM resulted in approximately 2% memory overhead, whereas the GRM resulted in 170% to 1500% memory overhead. According to these results, the LLRM is the more suitable algorithm for hardware implementations.

The following publication was produced from the work presented in this chapter:

- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming and P. Giard, "Restart Mechanisms for the Successive-Cancellation List-Flip Decoding of Polar Codes", *J. Signal Process. Syst.,* 2024 (*under review*).

5.1 Summary of List-Flip Decoders

The detailed introduction to the list-flip decoders is provided in Section 2.3 of Chapter 2. The SCLF decoding was proposed by Yongrun *et al.* (2018) with the idea of combining the SCL and SCF decoding strategies. This strategy allows for a balance between parallel and sequential computations to achieve the target FER.

At the initial decoding trial, standard SCL decoding is performed. If none of the L paths at the end of decoding pass the CRC, a list of path-flipping candidates $\mathcal{B}_{\text{flip}}$ is constructed. The additional SCL trial t' is then performed

by following the standard SCL algorithm until the location $i = \mathcal{B}_{\text{flip}}(t')$ is met. When $\hat{u}_i$ is estimated, the path flipping is made, i.e., the paths that were initially discarded by a standard SCL are kept to continue decoding. The path-flipping strategy for the SCLF decoding was introduced by Cheng *et al.* (2019) and is used in this thesis. The standard SCL is then resumed to decode the remaining bits in a codeword. The SCLF decoder has the $T_{\max}$ trials to decode a codeword, and if failed, the codeword is deemed undecodable.

The DSCLF decoding was proposed by Shen *et al.* (2022), where DSCF algorithm strategy was adapted to SCLF. A path-flipping list $\mathcal{B}_{\text{flip}} = \{\boldsymbol{\varepsilon}_0, \ldots, \boldsymbol{\varepsilon}_{t'}, \ldots, \boldsymbol{\varepsilon}_{T_{\max}-1}\}$ now consists of sets of path-flipping candidates that enable the multi-path flipping approach. At the additional decoding trial t', a new information index j is progressively inserted to an extended set $\boldsymbol{\varepsilon}_{\text{ext}} = (\boldsymbol{\varepsilon}_{t'} \cup j)$, and the metric is calculated according to Shen *et al.* (2022) as (2.5).

Ultimately, list-flip decoders based on SCLF allow for an effective balance between parallel and sequential computations to achieve the target FER. The DSCLF decoder highly improves the error-correction performance, which is achieved with an increased $T_{\max}$, leading to higher average and worst-case execution times. Therefore, execution-time reduction mechanisms are beneficial for SCLF and DSCLF decoders.

5.2 Challenge of Applying GRM to List-Flip Decoder

In Chapter 3 and Chapter 4 of this thesis, our proposed GRM was applied to flip decoders (SCF and DSCF). These decoders were used with the baseline algorithms SC, Fast-SSC, and SC with LRT. The GRM modifies additional trials of the flip decoder. This modified SC trial is denoted by $\text{SC}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, where t' is the trial index, and $\boldsymbol{\varepsilon}_{t'}$ is the set of bit-flipping candidates.

For the list-flip decoders, SCL is the baseline decoding algorithm. The modified SCL trial in GRM is denoted by $\text{SCL}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, where t' is the trial index, and $\boldsymbol{\varepsilon}_{t'}$ is the set of *path-flipping candidates.* In this case, restarting from any bit-location of the codeword requires storing additional information, corresponding to the state memory for each information bit, which can be significantly large. Further details on the restart requirements are provided in the following subsection.

5.2.1 Restart Path Requirements in SCLF with GRM

The SCL decoder was introduced in Chapter 1 of this thesis. In the SCL, the L paths $\hat{\boldsymbol{U}} = \{\hat{\boldsymbol{u}}_0, \ldots, \hat{\boldsymbol{u}}_l, \ldots, \hat{\boldsymbol{u}}_{L-1}\}$ with the smallest path metrics $\mathbf{PM} = \{PM(0), \ldots, PM(l), \ldots, PM(L-1)\}$ are kept and updated throughout decoding a codeword.

The following notations are used throughout this chapter. The paths that are kept throughout the SCL decoding are referred to as the *best paths*, and their path metrics are referred to as the *best path metrics*. Similarly, the paths that

are discarded during the SCL decoding are referred to as the *worst paths*, and their path metrics are referred to as the *worst path metrics*. The information-bit locations $j \in \mathcal{A}$, where $\log_2 L \leq j \leq k_{\text{tot}} - 1$, are referred to as the *path split and select locations.* For these j, where the path sorting is performed, the set $\mathcal{A}_{\text{sort}} \subseteq \mathcal{A}$ is introduced. The term *state memory* refers to the paths and path metrics of the SCL decoder together, based on the notation used in the work of Stimming, Parizi & Burg (2014). In our study, the state memory does not include intermediate LLR and PS values, unlike in the work of Stimming *et al.* (2014). The binary representation of the restart location $\psi_{t'}$ is denoted by $\boldsymbol{\mathcal{H}}_{\psi_{t'}} = \left\{\mathcal{H}_{\psi_{t'}}(0), \ldots, \mathcal{H}_{\psi_{t'}}(n-1)\right\}$, where the least significant bit (LSB) is on the right, as defined in Subsection 3.1.2 of Chapter 3.

Chapter 3 described the GRM for flip decoders based on SCF. The GRM reduces the execution time of an additional SC trial by skipping computations that are the same between the initial and additional trials. The N bit estimates after the initial unsuccessful SC trial are stored to $\hat{\boldsymbol{u}}_{\text{rest}}$. In an additional trial t', the GRM restarts decoding at the restart location $\psi_t \in \mathcal{A}$, which is the next information bit location after the bit-flipping location $i_1 \in \mathcal{A}$. The intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$ are retrieved from the channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$ and the intermediate PS $\boldsymbol{\beta}_{\text{int}}$. To do so, the bit estimates are restored from $\hat{\boldsymbol{u}}_{\text{rest}}$, and they are also used to retrieve the intermediate PS bits $\boldsymbol{\beta}_{\text{int}}$.

The GRM for list-flip decoders based on SCLF follows the same restart schedule as that for the flip decoder. However, the following modifications are required. Throughout the process of SCL decoding, at any path split and select location, L best paths are kept to continue decoding while the L worst paths are discarded. In an additional trial of SCLF decoding, when SCL reaches the path-flipping location, this decision is reversed. Consequently, at this location, the worst paths are kept to continue decoding while the best paths are discarded. The final candidate paths in $\hat{\boldsymbol{\mathcal{U}}}$ do not necessarily contain the bits that were selected at each path split and select location. In many cases, paths are overwritten by duplicates from other candidates, resulting in the loss of the original path. In order to resume decoding at any path-flipping location $\psi_{t'}$ in GRM, the state memories of the L worst paths and the L worst path metrics at that location must be available.

The additional trial with the modified SCL decoding algorithm is denoted by SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, indicating the restart location $\psi_{t'}$ and path-flipping set $\boldsymbol{\varepsilon}_{t'}$. The data structure that stores the worst paths for each path split and select location $j \in \{\log_2 L, \ldots, k_{\text{tot}} - 1\}$ is denoted by $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}}$, and $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}} = \left\{\hat{\boldsymbol{\mathcal{U}}}_{\log_2 L}, \ldots, \hat{\boldsymbol{\mathcal{U}}}_{j}, \ldots, \hat{\boldsymbol{\mathcal{U}}}_{k_{\text{tot}}-1}\right\}$. Similarly, the data structure that stores the worst path metrics is denoted by $\mathbf{PM}_{\text{worst}}$, and $\mathbf{PM}_{\text{worst}} = \left\{\mathbf{PM}_{\log_2 L}, \ldots, \mathbf{PM}_{j}, \ldots, \mathbf{PM}_{k_{\text{tot}}-1}\right\}$. Both structures, $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}}$ and $\mathbf{PM}_{\text{worst}}$, must be stored in memory for every path split and select locations during the initial trial.

Unlike the SCF, the GRM for SCLF decoder does not restart from the information bit that follows the path-flipping location. This is because the path metrics for the possible frozen bits located between the path-flipping location and the following information bit must also be calculated. It cannot be done without actual decoding of these bits

and obtaining their decision LLRs. To avoid these calculations, the GRM restarts directly from the path-flipping location.

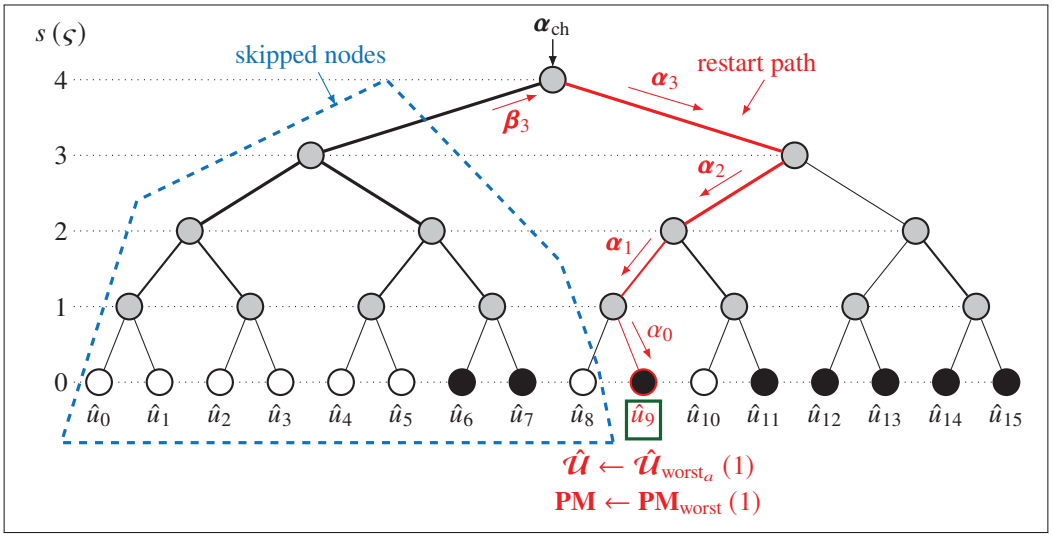


Figure 5.1 The modified SCL trial SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ with the path-flipping index $i_1 = 9$ and the restart location $\psi_{t'} = 9$, in SCLF with the GRM for a $(16, 8)$ polar code

Figure 5.1 illustrates the modified trials SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ for a $(16, 8)$ code with $\mathcal{A}_{\text{sort}} = \{7, 9, 11, 12, 13, 14, 15\}$. In this example, the path-flipping location for a trial t' is $i_1 = a_1 = 9$, where $a_1 = \mathcal{A}_{\text{sort}}\,(1)$ and $i_1 = \varepsilon_{t'}\,(0)$. The restart location is also set to $\psi_{t'} = 9$. The bit estimates $\hat{\boldsymbol{U}}$ required to restart SCL decoding from $\psi_{t'} = 9$ are restored from the memory $\hat{\boldsymbol{U}}_{\text{worst}_a}\left(j_{\psi_{t'}}\right)$, where $0 \leq j_{\psi_{t'}} \leq |\mathcal{A}_{\text{sort}}| - 1$ and $j_9 = 1$. Note that the memory $\hat{\boldsymbol{U}}_{\text{worst}_a}$ differs from $\hat{\boldsymbol{U}}_{\text{worst}}$ in that the former stores only the paths composed of information bits, while the latter stores both the information and frozen bits. The vector $\mathbf{PM}_{\text{worst}}$ stores the worst path metrics of the path split and select locations. To restore $\hat{\boldsymbol{U}}$, the information bits are restored from $\hat{\boldsymbol{U}}_{\text{worst}_a}\left(j_{\psi_{t'}}\right)$ according to the set $\mathcal{A}$. The path metrics $\mathbf{PM}$ are also restored from $\mathbf{PM}_{\text{worst}}\left(j_{\psi_{t'}}\right)$. Then, the partial-sum bits $\boldsymbol{\beta}_{\text{int}}$ for each of the L paths are restored based on $\hat{\boldsymbol{U}}$, and $\boldsymbol{\beta}_{\text{int}} = \boldsymbol{\beta}_3$ in this example. Note that the information bits at $j \in \left\{0, \ldots, \log_2 L - 1\right\}$, $j \in \mathcal{A}$, are stored in $\hat{\boldsymbol{U}}$ and never appear in path-flipping sets throughout the decoding of a codeword.

As a result of applying the GRM, the part of the tree to estimate the bits $\left\{\hat{u}_0, \ldots, \hat{u}_{\psi_{t'}-1}\right\}$, surrounded by the dashed blue line in Figure 5.1, is skipped. The restart path of SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, which connects the root of the tree at stage $s = n = 4$ with the leaf $\hat{u}_9$ at stage $s = 0$, is highlighted in red in Figure 5.1. During the traversal through the decoding stages $\varsigma \in \{n-1, \ldots, 0\}$, the g function is performed at stages $\varsigma \in \{3, 0\}$, and the f function is performed at stages $\varsigma \in \{2, 1\}$. These functions are chosen based on $\mathcal{H}_9 = \{1, 0, 0, 1\}$. Thus, if $\mathcal{H}_{\psi_{t'}}\,(n - 1 - \varsigma) = 0$, the f function (1.13) is performed at stage ς. If $\mathcal{H}_{\psi_{t'}}\,(n - 1 - \varsigma) = 1$, the g function (1.14) is performed at stage ς. The standard course of the SCL decoding is resumed for the remaining part of the decoding tree upon traversing the restart path and estimating $\hat{u}_9$, and by using the state memories restored from $\hat{\boldsymbol{U}}_{\text{worst}_a}\,(1)$ and $\mathbf{PM}_{\text{worst}}\,(1)$. Note

that storing $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}$ instead of $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}}$ reduces the size of the additional memory, as paths are only stored at the information-bit locations.

5.2.2 Memory Estimates

According to the requirements for the restart path in SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, the state memories, which contain the worst paths $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}$ and the worst path metrics $\mathbf{PM}_{\text{worst}}$, need to be stored for every path split and select locations $j \in \left\{\log_2 L, \ldots, k_{\text{tot}} - 1\right\}$. The data structure $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}$ stores the paths $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a} = \{\hat{\boldsymbol{u}}_{\log_2 L}(l), \ldots, \hat{\boldsymbol{u}}_j(l), \ldots, \hat{\boldsymbol{u}}_{k_{\text{tot}}-1}(l)\}$, where the list index is $l \in \{0, \ldots, L-1\}$. For each l, the j^{th} path contains the bit estimates $\hat{\boldsymbol{u}}_j = \left\{\hat{u}_{\log_2 L}, \ldots, \hat{u}_j\right\}$. Note that the paths $\hat{\boldsymbol{u}}_j$ for each j vary in size.

The additional memory to restart in the list-flip decoder with the GRM is denoted by Λ_{rest}. The size of Λ_{rest} can be calculated as follows:

$$\Lambda_{\text{rest}} = \Lambda\left(\text{PM}_{\text{worst}}\right) + \Lambda\left(\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}\right) = L \cdot Q_{\text{PM}} \cdot |\mathcal{A}_{\text{sort}}| + L \cdot S_j, \tag{5.1}$$

where $\Lambda(\cdot)$ represents the size of each list in bits. The quantization coefficient for the path metrics is denoted by Q_{PM}. The total size of each path $\hat{\boldsymbol{u}}_j = \{\hat{u}_{\log_2 L}, \ldots, \hat{u}_j\}$ with $j \in \{\log_2 L, \ldots, k_{\text{tot}} - 1\}$, is denoted by S_j, which is estimated as:

$$S_j = \sum_{j=\log_2(L)}^{k_{\text{tot}}-1} j = \frac{|\mathcal{A}_{\text{sort}}|}{2} \times \left(\log_2(L) + k_{\text{tot}} - 1\right). \tag{5.2}$$

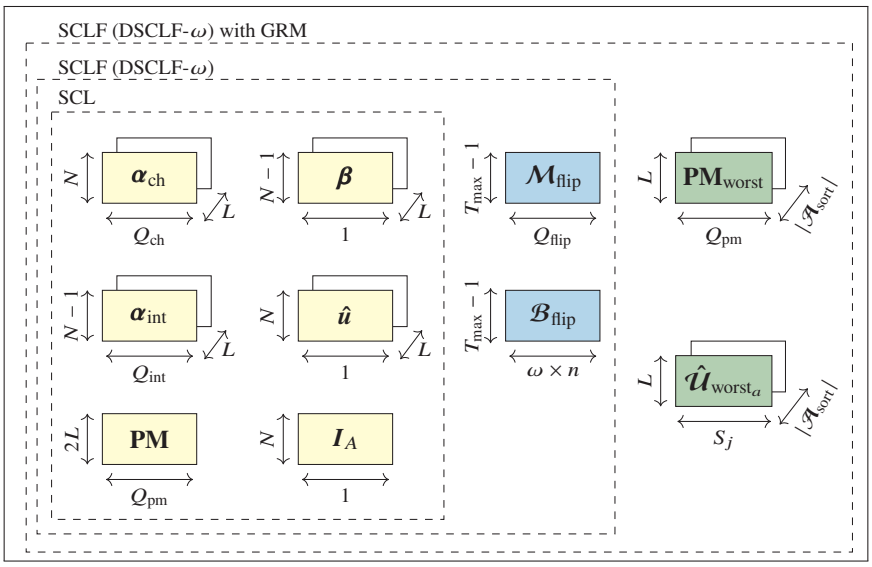


Figure 5.2 Memory sketch of SCL, SCLF and SCLF with GRM

The sketch of the memory of a list-flip decoder based on SCLF with the GRM is provided in Figure 5.2. This sketch is similar to one we derived in Subsection 3.2.4.2 of Chapter 3. For SCL decoding, the channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$ and intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$ are quantized using Q_{ch} and Q_{int} bits. The bit estimates $\hat{\boldsymbol{u}}$ are stored using N bits of memory, and the intermediate partial-sums $\boldsymbol{\beta}_{\text{int}}$ require $N-1$ bits of storage. The path metrics require storing of $2L$ elements, each quantized using Q_{PM} bits. The vector $\boldsymbol{I}_A$ of length N stores binary flags indicating the positions of information and frozen bits. The entries of $\boldsymbol{I}_A$ are 1-s at the information-bit positions in $\mathcal{A}$ and 0-s at the frozen-bit positions in $\mathcal{A}^C$. For the SCLF or DSCLF-ω decoding, the metrics of the path-flipping candidates are quantized using Q_{flip} bits. Each path-flipping set $\boldsymbol{\varepsilon}_{t'} \in \mathcal{B}_{\text{flip}}$ requires $\omega \times n$ bits, where $n = \log_2(N)$, and $\omega = |\boldsymbol{\varepsilon}_{t'}|_{\max}$. The total memory Λ_{LGRM} required by SCLF with the GRM is then calculated as follows:

$$\Lambda_{\text{LGRM}} = \Lambda_{\text{SCLF}} + \Lambda_{\text{rest}}, \tag{5.3}$$

where the Λ_{rest} is estimated by (5.1), and Λ_{SCLF} is composed of Λ_{SCL} and Λ_{flip}. These memories are estimated as:

$$\Lambda_{\text{SCLF}} = \Lambda_{\text{SCL}} + \Lambda_{\text{flip}}, \tag{5.4}$$

$$\Lambda_{\text{SCL}} = Q_{\text{ch}} \cdot N + Q_{\text{int}} \cdot (N-1) + 3N - 1 + 2L \cdot Q_{\text{PM}}, \tag{5.5}$$

$$\Lambda_{\text{flip}} = Q_{\text{flip}} \cdot (T_{\max} - 1) + \omega \cdot n \cdot (T_{\max} - 1)\,. \tag{5.6}$$

In the following, two examples are provided to estimate the memory by applying the GRM to list-flip decoders for a $(1024, 512 + 11)$ code:

- Scenario 1: DSCLF-3 decoder with $L = 2$ and $T_{\max} = 301$.
- Scenario 2: DSCLF-2 decoder with $L = 4$ and $T_{\max} = 51$.

The values of L, $T_{\max}$, and ω in both scenarios are selected to achieve the error-correction performance of the state-of-the-art SCL decoder with $L = 16$. The error-correction performance of these two decoders was provided in Figure 2.7 in Chapter 2.

Memory requirements are estimated with (5.3) using the quantization schemes of Ercan *et al.* (2020a) and Hashemi *et al.* (2017). Therefore, the blocks based on LLR values are quantized by $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$, $Q_{\text{PM}} = 8$, and $Q_{\text{flip}} = 7$ bits. The memory estimates and overhead for both scenarios are summarized in Table 5.1.

According to the estimations in Table 5.1, applying the GRM to list-flip decoders results in enormous memory overhead. The overheads of 658.90% and 884.57% correspond to approximately 7× and 9× increases compared to the standard decoder memory in each scenario. Therefore, applying the GRM to the list-flip decoders is inefficient for hardware implementation.

Table 5.1 Memory estimates and overhead by applying GRM for DSCLF-2 and DSCLF-3 decoders in two considered scenarios

Scenario	Decoder	L	$T_{\max}$	Λ_{SCL} (5.5) bits	Λ_{flip} (5.6) bits	Λ_{SCLF} (5.4) bits	Λ_{rest} (5.1) bits	Λ_{LGRM} (5.3) bits	Overhead %
1	DSCLF-3	2	301	31760	11100	42860	282402	325262	658.90
2	DSCLF-2	4	51	62496	1350	63846	564764	628610	884.57

Table 5.1 also provides memory estimations of the original (non-GRM) decoders, where Λ_{SCLF} is the total memory. These results show that the DSCLF-2 decoder with $L = 4$ requires 50% more memory than DSCLF-3 with $L = 2$ under the corresponding $T_{\max}$ and ω. The SCL memory Λ_{SCL} is the largest component, which accounts for 74% and 96% of Λ_{SCLF} in both scenarios. Moreover, Λ_{SCL} doubles when increasing the list size from $L = 2$ to $L = 4$. These results also agree with the implementation results provided in Chapter 2. The results of Table 2.1 indicated that the chip area of the SCL decoder nearly doubles when L is doubled. Therefore, it is beneficial to keep a low list size L while increasing ω and $T_{\max}$ to achieve an area- and energy-efficient decoder implementation with strong error-correction performance. The remaining part of this chapter focuses on list-flip decoders with $L = 2$, while the results for $L = 4$ are presented in Appendix I. The impact of increasing L, ω, and $T_{\max}$ on memory is also provided in Appendix I.

5.3 List-Flip Decoder with LLRM

The key issue of the memory overhead of GRM for the list-flip decoder is the vector $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}$, which stores the worst L paths for each path split and select location $j \in \mathcal{A}_{\text{sort}}$ during the initial trial. To reduce the size of $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}$, we propose limiting the possible restart locations to a smaller set. The resulting mechanism is referred to as the limited-locations restart mechanism (LLRM).

A fundamental definition of the proposed LLRM for the list-flip decoder is provided in Definition 5.1.

Definition 5.1 (The LLRM for SCLF decoder)**.** For an additional trial t', the list-flip decoder embedding the proposed LLRM skips non-negligible parts of the decoding tree traversal by resuming at the location $\psi_{t'} \in \boldsymbol{\Upsilon}$, where $\boldsymbol{\Upsilon} = \{\Upsilon(0), \ldots, \Upsilon(Y-1)\} \subseteq \mathcal{A}_{\text{sort}}$ represents the set of limited restart locations. The number of possible restart locations is denoted by Y, $Y = |\boldsymbol{\Upsilon}|$, and $Y \ll |\mathcal{A}_{\text{sort}}|$. The additional trial with the modified SCL decoding algorithm is denoted by SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, indicating the restart location $\psi_{t'}$ and path-flipping set $\boldsymbol{\varepsilon}_{t'}$. The restart is performed from $\psi_{t'} \in \boldsymbol{\Upsilon}$, which is the closest to the path-flipping location i_1 on its LHS. The execution-time reduction results from storing the complete state memories for the best and the worst paths, $\hat{\boldsymbol{\mathcal{U}}}_a = \{\hat{\boldsymbol{\mathcal{U}}}_{\text{best}_a}, \hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}\}$ and $\mathbf{PM}_a = \{\mathbf{PM}_{\text{best}}, \mathbf{PM}_{\text{worst}}\}$, for each location $j \in \boldsymbol{\Upsilon}$ during the initial SCL trial. This is different from the GRM, where the path information of only the worst paths $\hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}$ and $\mathbf{PM}_{\text{worst}}$ is stored for all locations $j \in \mathcal{A}_{\text{sort}}$. This

difference occurs because, in the LLRM, the restart location $\psi_{t'}$ may either match with or precede the path-flipping location. In the latter case, decoding must be resumed with the initially selected paths at $\psi_{t'}$. In the GRM, the location $\psi_{t'}$ always corresponds to the path-flipping location, so decoding is resumed with the initially discarded paths at $\psi_{t'}$.

Note that, unlike the GRM, the LLRM requires storing the state memories for both the best and the worst paths. To clarify this property, two scenarios are considered. In the first scenario, $\psi_{t'} < i_1$, i.e., the restart location is on the LHS of the path-flipping location. The SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ must be resumed at location $\psi_{t'}$ exactly as in the initial SCL trial. Therefore, the state memories are restored from the best paths $\hat{\boldsymbol{U}}_{\text{best}_a}$ and the best path metrics $\mathbf{PM}_{\text{best}}$ at $\psi_{t'}$. In the second scenario, $\psi_{t'} = i_1$, i.e., the restart location matches with the path-flipping location. In this case, SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ must be resumed with the initially discarded paths at $\psi_{t'}$. Therefore, the state memories are restored from the worst paths $\hat{\boldsymbol{U}}_{\text{worst}_a}$ and the worst path metrics $\mathbf{PM}_{\text{worst}}$.

Chapter 3 described the SRM for the SCF decoder. The SRM and LLRM share common properties. Both mechanisms use a limited set of restart locations. The SRM applies only two restart locations $\boldsymbol{\Upsilon} = \{0, N/2\}$. These locations were selected based on recognizing the parts of the decoding tree that can be skipped from decoding. In the SRM, for each additional trial where the flipping index is on the RHS of the tree, the LHS tree can be skipped, and decoding can resume from the location $\psi_{t'} = N/2$. The LLRM also applies a limited set of restart locations. However, unlike the SRM, these locations are selected to approach the execution-time reduction of the GRM. At the same time, the number of locations must be small to reduce the memory overhead resulting from storing the state memories at the possible restart locations.

Before discussing the methodologies for selecting the restart locations, the algorithm of the LLRM is explained in detail using an illustrative example.

5.3.1 Illustration of Restart Path in LLRM

The modified SCL trial in the list-flip decoder based on SCLF with the LLRM, SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, is illustrated in Figure 5.3. This example is for a $(16, 8)$ code with the set of path split and select locations $\mathcal{A}_{\text{sort}} = \{7, 9, 11, 12, 13, 14, 15\}$. In this example, the path-flipping location for a trial t' is $i_1 = a_1 = 9$, where $a_1 = \mathcal{A}_{\text{sort}}(1)$ and $i_1 = \varepsilon_{t'}(0)$. The set of restart locations is assigned to $\boldsymbol{\Upsilon} = \{7, 12\}$ or $\boldsymbol{\Upsilon}_{\mathcal{A}} = \{1, 4\}$. The restart location is set to $\psi_{t'} = \Upsilon(0) = 7$, which is the closest one to i_1 on its LHS. To restore $\hat{\boldsymbol{U}}$, the information bits are restored from the paths in $\hat{\boldsymbol{U}}_{\text{best}_a}\left(j_{\psi_{t'}}\right)$, where $j_{\psi_{t'}} = 0$. The path metrics $\mathbf{PM}$ are also restored from $\mathbf{PM}_{\text{best}}\left(j_{\psi_{t'}}\right)$. Then, the partial-sum bits $\boldsymbol{\beta}_{\text{int}}$ for each of the L paths are restored based on $\hat{\boldsymbol{U}}$, and in this example, $\boldsymbol{\beta}_{\text{int}} = \{\boldsymbol{\beta}_2, \boldsymbol{\beta}_1, \boldsymbol{\beta}_0\}$. As a result of applying the LLRM, the part of the tree to estimate the bits $\{\hat{u}_0, \ldots, \hat{u}_{\psi_{t'}-1}\}$, surrounded by the dashed blue line in Figure 5.3, is skipped. The restart path of SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$, which connects the root of the tree at stage

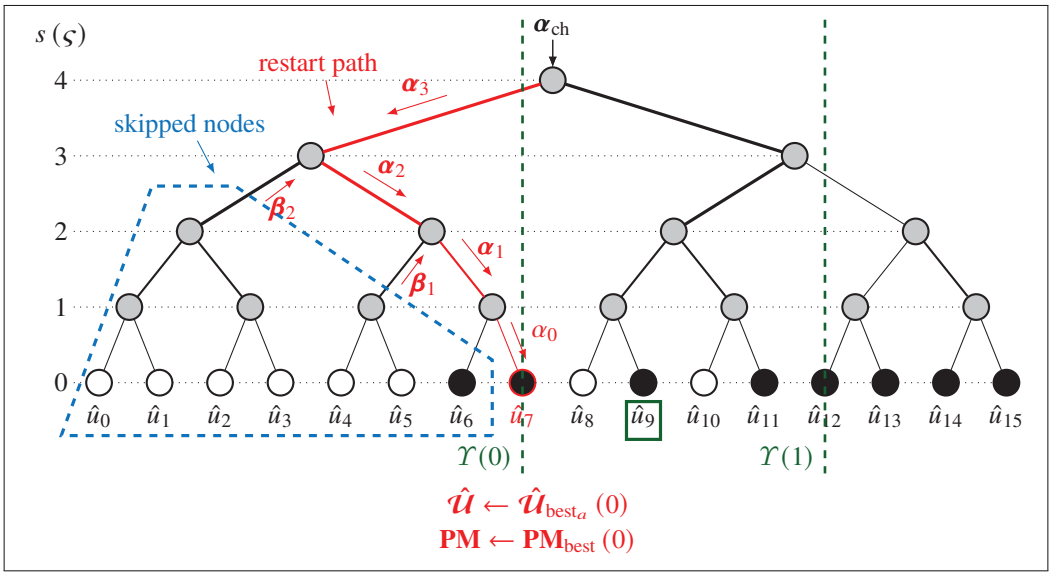


Figure 5.3 The modified SCL trial SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ with the path-flipping index $i_1 = 9$ and the restart location $\psi_{t'} = \Upsilon(0) = 7$, in SCLF with the LLRM for a $(16, 8)$ polar code

$s = n = 4$ with the leaf $\hat{u}_7$ at stage $s = 0$, is highlighted in red in Figure 5.1. During the traversal, at the decoding stages $\varsigma \in \{n-1, \ldots, 0\}$ and given $\mathcal{H}_{\psi_{t'}} = \{0, 1, 1, 1\}$, the g functions are performed at stages $\varsigma \in \{2, 1, 0\}$, and the f function is performed at stage $\varsigma = 3$. The standard course of SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ trial is resumed for the remaining part of the decoding tree. When the bit $\hat{u}_9$ is estimated, the path flip is made according to standard SCLF decoding. Note that if $i_1 = \psi_{t'} = 7$, the state memories are restored from $\hat{\mathcal{U}}_{\text{worst}_a}$ and $\mathbf{PM}_{\text{worst}}$, thus directly making a path flip.

5.3.2 Memory Estimates

In order to resume list-flip decoding at location in $\boldsymbol{\Upsilon}$, the LLRM requires the storage of the following information:

1. The set of limited restart locations $\boldsymbol{\Upsilon} = (\Upsilon(0), \ldots, \Upsilon(Y-1))$, where $Y \ll |\mathcal{A}_{\text{sort}}|$.
2. The data structure $\hat{\mathcal{U}}_a$ that stores the best and the worst paths for locations $j \in \boldsymbol{\Upsilon}$ during initial SCL.
3. The data structure $\mathbf{PM}_a$ that stores the best and the worst path metrics for locations $j \in \boldsymbol{\Upsilon}$ during initial SCL.

The additional memory required by the LLRM Λ_{rest} can be calculated as follows:

$$\begin{aligned}\Lambda_{\text{rest}} &= \Lambda(\Upsilon) + \Lambda(\text{PM}_a) + \Lambda\left(\hat{\mathcal{U}}_a\right) = \\ &= \Lambda(\Upsilon) + \Lambda(\text{PM}_{\text{best}}) + \Lambda(\text{PM}_{\text{worst}}) + \Lambda\left(\hat{\mathcal{U}}_{\text{best}_a}\right) + \Lambda\left(\hat{\mathcal{U}}_{\text{worst}_a}\right) = \\ &= n \cdot Y + 2L \cdot Q_{\text{PM}} \cdot Y + 2L \cdot S'_j,\end{aligned} \tag{5.7}$$

where $\Lambda(\cdot)$ represents the size of each list in bits. The memory in bits for each path $\hat{\boldsymbol{u}}_j$ is denoted by S_j, and it is estimated as follows:

$$S'_j = \sum_{j=0}^{Y-1} \Upsilon_{\mathcal{A}}(j), \tag{5.8}$$

where $\Upsilon_{\mathcal{A}}(j) \in \boldsymbol{\Upsilon}_{\mathcal{A}}$ differs from $\Upsilon(j) \in \boldsymbol{\Upsilon}$ in that the former set iterates over the information bits, whereas the latter iterates over all indices in a codeword.

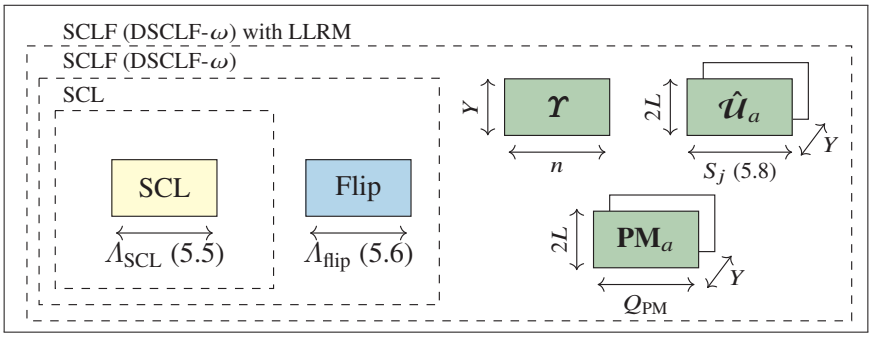


Figure 5.4 Memory sketch of SCL, SCLF and SCLF with LLRM

Figure 5.4 shows the memory sketch of the list-flip decoder based on SCLF with the LLRM. The memory sketch of the list-flip decoder based on SCLF with the LLRM is provided in Figure 5.4. This sketch compresses all the memories for SCL and SCLF decoding, containing the same blocks as depicted in Figure 5.2. The quantization coefficients are applied also the same as stated in Subsection 5.2.2 of this chapter. The total memory Λ_{LLRM} required by SCLF with the LLRM is computed as:

$$\begin{aligned} \Lambda_{\text{LLRM}} &= \Lambda_{\text{SCLF}} + \Lambda_{\text{rest}} = \\ &= \Lambda_{\text{SCL}} + \Lambda_{\text{flip}} + \Lambda_{\text{rest}}, \end{aligned} \tag{5.9}$$

where Λ_{SCLF}, Λ_{SCL}, and Λ_{flip} are estimated by (5.4), (5.5), and (5.6).

5.4 Selection of Restart Locations in LLRM

The set of restart locations $\boldsymbol{\Upsilon}$ in LLRM needs to be carefully selected to achieve execution-time reduction close to that of the GRM. At the same time, the number of restart locations $Y = |\boldsymbol{\Upsilon}|$ must be small to reduce the memory overhead resulting from storing the state memories at possible restart locations.

We propose a probability-based design for selecting the restart locations. The proposed design requires prior simulations to retrieve the probability-mass function (PMF) of the first location in the path-flipping sets $i_1 = \varepsilon_{t'}(0)$ in list-flip decoder based on SCLF. This PMF is obtained according to methodology, which was described in

Subsection 3.2.2 of Chapter 3. Additionally, we present two design methods for $\boldsymbol{\Upsilon}$ based on the construction of polar codes.

5.4.1 Design Based on Code Construction

Two simple methodologies based on code construction are considered for the design of restart locations $\boldsymbol{\Upsilon}$ in LLRM. In the first design, the codeword is equally divided into Y segments, where $Y = |\boldsymbol{\Upsilon}|$. This set is further denoted by $\boldsymbol{\Upsilon}_{\text{divN}}$, and $\boldsymbol{\Upsilon}_{\text{divN}} = \left\{0, \left\lfloor \frac{N}{Y} \right\rfloor, \ldots, \left\lfloor \frac{N(Y-1)}{Y} \right\rfloor \right\}$. Note that for $Y = 2$, the set $\boldsymbol{\Upsilon}_{\text{divN}} = \{0, N/2\}$ corresponds to the SRM, which was described in Section 3.1 of Chapter 3.

The second design takes into account the set of information bits $\mathcal{A}_{\text{sort}}$ and divides it equally into Y equal parts. The set is denoted by $\boldsymbol{\Upsilon}_{\text{divK}}$, and $\boldsymbol{\Upsilon}_{\text{divK}} = \left\{a_{\log_2(L)}, a_{\text{x}}, \ldots, a_{\text{x}(Y-1)}\right\}$, where $\text{x} = \left\lfloor \frac{k_{\text{tot}}}{Y} \right\rfloor$. Note that the first location is set to $\Upsilon_{\text{divK}}(0) = \mathcal{A}_{\text{sort}}(0)$, i.e., to the first path split and select location $j_0 = \log_2 L$. Hence, for any additional trial t', tree traversal to estimate frozen bits on the LHS of $\Upsilon_{\text{divK}}(0)$ will be always avoided.

These two construction-based designs are not very effective in terms of execution-time reduction, as they do not consider the distribution of the path-flipping location and do not allocate them near these locations. However, these designs do not require prior simulations, which makes them relevant for practical applications. Also, it can be noted that any location in $\boldsymbol{\Upsilon}_{\text{divN}}$ may correspond to information or frozen bit. If it is a frozen bit, the state memory contains only the best L paths, as the paths are not split for frozen bits. Nevertheless, memory is allocated for all $2L$ paths for consistency, regardless of whether any location in $\boldsymbol{\Upsilon}_{\text{divN}}$ corresponds to an information or a frozen bit.

5.4.2 Probability-based Design

We propose to use a design based on PMF of the path-flipping locations. The PMF is obtained by simulation as follows. For this analysis, 5G polar codes (3GPP, 2018) with length $N = 1024$ are used with the rate $R = 1/2$. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed at the target FER of 10^{-2}. We simulate DSCLF-3 decoder with $L = 2$, $\omega = 3$, and $T_{\max} = 301$. The PMF distribution includes only the first bit in a path-flipping set, i.e., the bit $i_1 = \varepsilon_{t'}(0)$, for the set of information bits $\mathcal{A} = \left\{a_0, \ldots, a_j, \ldots, a_{k_{\text{tot}}-1}\right\}$, such that $0 \leq j \leq k_{\text{tot}} - 1$. The location i_1 is stored for each trial index t' for the codewords with $\tau' > 0$, i.e., the codewords that required additional trials. The probability that a_j is the first bit where the path flip occurs in an additional trial is denoted by $\mathbb{P}\left(i_1 = a_j\right)$.

The algorithm to obtain the PMF by simulation is described in Algorithm 2. For the codewords where additional trials were applied, each occurrence of the path-flipping locations $i_1 = \varepsilon_{t'}(0)$ is tracked by the counter. The counter

Algorithm 2 Obtaining the PMF of the path-flipping locations $i_1 = a_j$ for an SCLF decoder by simulations

procedure DISTR_BIT_FLIP_SCLF(C, $\mathcal{A}$, $\mathcal{A}_{\text{sort}}$, $T_{\max}$, L, ω, E_b/N_0)
  for $j = 0 : (|\mathcal{A}_{\text{sort}}| - 1)$ **do**
    $\mathbb{P}(j) \leftarrow 0$ ▷ Initialize PMF for all path split and select locations
    $P_{\text{cnt}}(j) \leftarrow 0$ ▷ Initialize counter $\boldsymbol{P}_{\text{cnt}}$ for all path split and select locations
  end for
  $\mathcal{T}_{\text{sim}} \leftarrow 0$ ▷ Total number of additional trials throughout simulation
  for $c = 0 : C - 1$ **do**
    $\boldsymbol{x} \leftarrow$ POLAR_ENCODING $(\boldsymbol{u})$
    $\boldsymbol{\alpha}_{\text{ch}} \leftarrow$ AWGN $(E_b/N_0, \boldsymbol{x})$ ▷ Obtain channel LLRs for decoding a codeword $\boldsymbol{x}$
    $(\mathcal{B}_{\text{flip}}, \tau') \leftarrow$ SCLF $(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}, T_{\max}, \omega)$ ▷ Run SCLF or DSCLF-ω decoder
    if $\tau' > 0$ **then** ▷ Select codewords with additional decoding trials
      $\mathcal{T}_{\text{sim}} \leftarrow \mathcal{T}_{\text{sim}} + \tau'$ ▷ Accumulate additional number of trials
      for $t' = 0 : \tau' - 1$ **do**
        $\boldsymbol{\varepsilon}_{t'} \leftarrow \mathcal{B}_{\text{flip}}(t')$
        $i_1 \leftarrow \varepsilon_{t'}(0)$ ▷ Extract the first path-flipping location $i_1 \in \mathcal{A}$
        $P_{\text{cnt}}(i_1 = a_j) \leftarrow P_{\text{cnt}}(i_1 = a_j) + 1$ ▷ Increment the occurrence of bit-location $i_1 = a_j$
      end for
    end if
  end for
  for $j = 0 : (|\mathcal{A}_{\text{sort}}| - 1)$ **do**
    $\mathbb{P}(j) \leftarrow P_{\text{cnt}}(j) / \mathcal{T}_{\text{sim}}$ ▷ Estimate PMF $\mathbb{P}(i_1 = a_j)$
  end for
  return the PMF $\mathbb{P}(i_1 = a_j), \forall a \in \mathcal{A}_{\text{sort}}$
end procedure

Algorithm 3 Design of restart locations $\boldsymbol{\Upsilon}_{\text{prob}}$ based on PMF of path-flipping locations for an SCLF decoder

procedure DESIGN_RESTART_WITH_PMF($\mathbb{P}$, $\mathcal{A}_{\text{sort}}$, Y)
  $S_p \leftarrow 0$ ▷ Probability summation counter
  $P_{\text{segm}} \leftarrow \frac{1}{Y}$ ▷ Segment Probability
  for $j = 0 : (|\mathcal{A}_{\text{sort}}| - 1)$ **do**
    $\mathbb{P}'(j) \leftarrow \frac{\mathbb{P}(j)}{\max(\mathbb{P})}$ ▷ Obtain the normalized PMF
  end for
  $\rho \leftarrow 1$ ▷ Index of restart location
  $\Upsilon_{\text{prob}}(0) \leftarrow \mathcal{A}_{\text{sort}}(0)$ ▷ The first restart location is the first path split and select location
  for $j = 0 : (|\mathcal{A}_{\text{sort}}| - 1)$ **do**
    $S_p \leftarrow (S_p + \mathbb{P}(i_1 = a_j))$ ▷ Accumulate probability summation counter
    if $S_p > \rho \cdot P_{\text{segm}}$ **then**
      $\Upsilon_{\text{prob}}(\rho) \leftarrow a_j$ ▷ Assign restart location when exceeding P_{segm}
      $\rho \leftarrow \rho + 1$ ▷ Next restart location
    end if
    if $\rho > Y$ **then**
      exit loop ▷ Already found Y restart locations in $\boldsymbol{\Upsilon}_{\text{prob}}$
    end if
  end for
  return restart locations $\boldsymbol{\Upsilon}_{\text{prob}}$
end procedure

is then normalized by the total number of applied trials to obtain the PMF. After transmitting $C = 2 \cdot 10^5$ codewords at the FER of 10^{-2}, a reliable PMF $\mathbb{P}(i_1 = a_j)$ is obtained for all $a_j \in \mathcal{A}_{\text{sort}}$.

The PMF obtained by Algorithm 2 is now used to design the set of restart locations. This set is denoted by $\boldsymbol{\Upsilon}_{\text{prob}}$. The set of restart locations equally divides a set of information bit-locations $\mathcal{A}_{\text{sort}}$, all sharing the same PMF

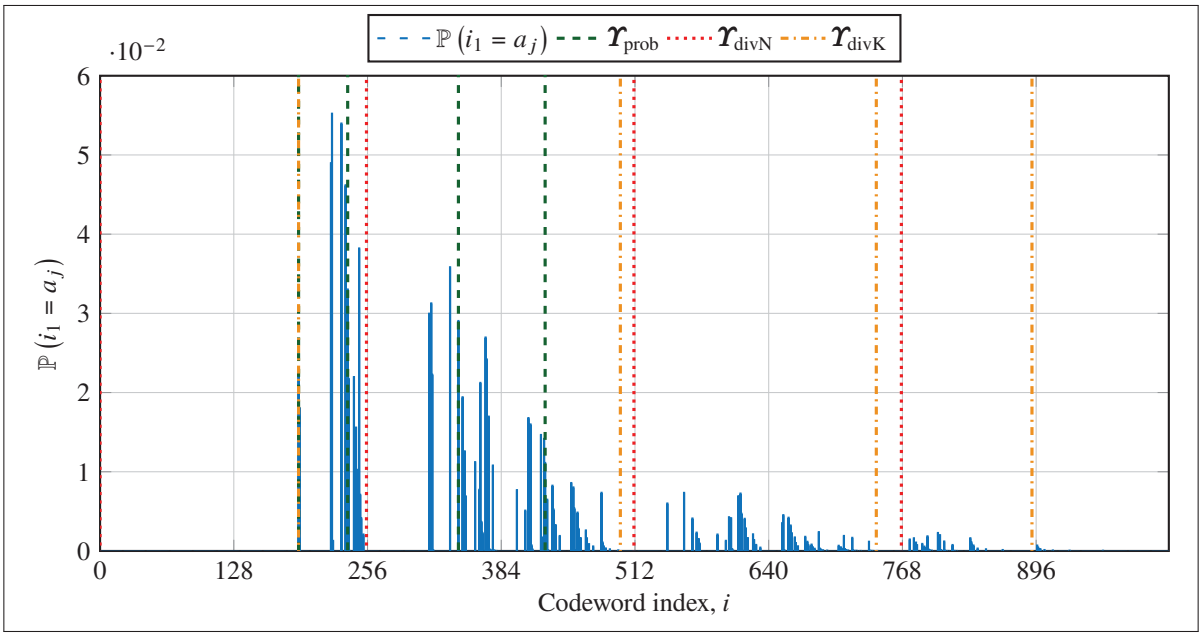


Figure 5.5 PMF of the bits i being the first path-flipping candidate $i_1 = \varepsilon_{i'}(0)$ of the DSCLF-3 decoder for a $(1024, 512 + 11)$ code. Restart locations $\boldsymbol{\Upsilon}$ are represented by the vertical lines

$\mathbb{P}(i_1 = a_j)$, $a_j \in \mathcal{A}_{\text{sort}}$. This means that the information bits have the same probabilities of being path-flipping locations within each segment of $\mathcal{A}_{\text{sort}}$. The algorithm to obtain the set of restart locations $\boldsymbol{\Upsilon}_{\text{prob}}$ according to the PMF is described in Algorithm 3. The number of restart locations Y is selected in advance. The first restart location is set to the first path split and select location, $\Upsilon_{\text{prob}}(0) = a_0 = \mathcal{A}_{\text{sort}}(0)$. The remaining locations $\boldsymbol{\Upsilon}_{\text{prob}}(\rho)$, $1 \leq \rho \leq Y - 1$, are selected verifying that PMF of each segment $\mathbb{P}\left(\Upsilon_{\text{prob}}(\rho - 1) \leq i_1 \leq \Upsilon_{\text{prob}}(\rho)\right) > \frac{\rho}{Y}$.

Figure 5.5 depicts the resulting PMF of all bits in a codeword $0 \leq i \leq N - 1$. The results are obtained for the DSCLF-3 decoder and for a $(1024, 512 + 11)$ polar code. Additionally, $Y = 4$ restart locations, designed by three methods, $\boldsymbol{\Upsilon}_{\text{divN}}$, $\boldsymbol{\Upsilon}_{\text{divK}}$ and $\boldsymbol{\Upsilon}_{\text{prob}}$ are represented by the vertical lines. The locations in $\boldsymbol{\Upsilon}_{\text{prob}}$ are obtained using Algorithm 2 and Algorithm 3. Looking at the resulting PMF, the path flipping occurs more often on the LHS of the codeword. Therefore, probability-based design selects all $Y = 4$ restart locations on the LHS of the codeword. The design methods based on code construction select the locations distributed at both LHS and RHS of the codeword. For the example shown in Figure 5.5, the restart locations designed based on code construction are $\boldsymbol{\Upsilon}_{\text{divN}} = \{0, 256, 512, 768\}$ and $\boldsymbol{\Upsilon}_{\text{divK}} = \{190, 498, 743, 892\}$. The probability-based restart locations are $\boldsymbol{\Upsilon}_{\text{prob}} = \{190, 237, 343, 426\}$.

5.5 Time and Resource Analysis

The execution-time model, whch was described in Section 2.5 of Chapter 2, is applied. This model is based on the semi-parallel SC decoder implementation proposed by Leroux *et al.* (2013). The equations (5.10)–(5.12) provided below correspond to equations (3.5)–(3.7) provided in Subsection 3.1.3.2 of Chapter 3. The derivations in this section are presented for the LLRM, but they are also applicable to the GRM.

The execution time of the SC decoder is calculated as:

$$\mathcal{L}_{\mathrm{SC}} = \mathcal{L}_{\alpha}\left(N\right) + \mathcal{L}_{\beta}\left(N\right), \tag{5.10}$$

where $\mathcal{L}_{\alpha}\left(N\right)$ and $\mathcal{L}_{\beta}\left(N\right)$ are the LLR and PS calculations for the polar code of length N, and each is computed by the following equations:

$$\mathcal{L}_{\alpha}(N) = 2N + \frac{N}{P} \cdot \log_2\left(\frac{N}{4P}\right), \tag{5.11}$$

$$\mathcal{L}_{\beta}(N) = \sum_{s=1}^{n-1} (2^{n-s} - 1) \cdot \left\lceil \frac{2^s}{2P} \right\rceil. \tag{5.12}$$

The execution time of the SCL decoder is derived in the decoder implementation in the work of Stimming *et al.* (2015). This execution time, denoted by $\mathcal{L}_{\mathrm{SCL}}$, is calculated as follows:

$$\mathcal{L}_{\mathrm{SCL}} = \mathcal{L}_{\mathrm{SC}} + \mathcal{L}_{\mathrm{sort}}, \tag{5.13}$$

where $\mathcal{L}_{\mathrm{sort}} = |\mathcal{A}_{\mathrm{sort}}|$ accounts for the CCs required to sort paths for all path split and select locations. Recall the approximation $\mathcal{L}_{\mathrm{sort}} \approx k_{\mathrm{tot}}$ defined in (2.10) in Chapter 2. This approximation is also valid because the relation $\log_2 L \ll k_{\mathrm{tot}}$ holds in practical scenarios.

The execution time of the modified SCL trial SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$ of the LLRM is denoted by $\mathcal{L}_{\mathrm{SCL}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})}$. By using the derivation (3.8) for the SC decoding in Chapter 3, this execution time can be derived as:

$$\mathcal{L}_{\mathrm{SCL}(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})} = \mathcal{L}_{\mathrm{SCL}} - \Delta\mathcal{L}_{\mathrm{SCL}}\left(\psi_{t'}\right), \tag{5.14}$$

where $\Delta\mathcal{L}_{\mathrm{SCL}}\left(\psi_{t'}\right)$ denotes the execution-time reduction of SCL $(\psi_{t'}, \boldsymbol{\varepsilon}_{t'})$.

The execution-time reduction provided by the LLRM corresponds to the skipped CCs that would otherwise be required to calculate LLR and PS for the modified restart. This reduction is denoted by $\Delta\mathcal{L}_{\mathrm{SCL}}\left(\psi_{t'}\right)$ and can be

computed as follows:

$$\Delta\mathcal{L}_{\text{SCL}}\left(\psi_{t'}\right) = \Delta\mathcal{L}_{\alpha}\left(\psi_{t'}\right) + \Delta\mathcal{L}_{\beta}\left(\psi_{t'}\right) + \Delta\mathcal{L}_{\text{sort}} - \Theta\left(\psi_{t'}\right), \tag{5.15}$$

where $\Delta\mathcal{L}_{\alpha}\left(\psi_{t'}\right)$, $\Delta\mathcal{L}_{\beta}\left(\psi_{t'}\right)$, and $\Delta\mathcal{L}_{\text{sort}}$ denote the skipped CCs by avoiding LLR computations, PS computations, and sorting required to decode the bit estimates $\{\hat{u}_0, \ldots, \hat{u}_{\psi_{t'}-1}\}$. The component $\Theta\left(\psi_{t'}\right)$ corresponds to the additional CCs required to calculate all intermediate PS bits $\boldsymbol{\beta}_{\text{int}}$ at the restart path of SCL $\left(\psi_{t'}, \boldsymbol{\varepsilon}_{t'}\right)$. The derivations (5.17)–(5.19) provided below correspond to (3.24)–(3.26) defined in Subsection 3.2.4.1 of Chapter 3.

The reductions $\Delta\mathcal{L}_{\text{sort}}$, $\Delta\mathcal{L}_{\alpha}\left(\psi_{t'}\right)$, and $\Delta\mathcal{L}_{\beta}\left(\psi_{t'}\right)$ are calculated as follows:

$$\Delta\mathcal{L}_{\text{sort}} = \left|\mathcal{A}_{\text{sort}}\left(0, \ldots, \psi_{t'} - 1\right)\right|, \tag{5.16}$$

$$\Delta\mathcal{L}_{\alpha}\left(\psi_{t'}\right) = \sum_{s=0}^{n-1}\left(\left\lfloor\frac{\psi_{t'}}{2^s}\right\rfloor \cdot \left\lceil\frac{2^s}{P}\right\rceil\right), \tag{5.17}$$

$$\Delta\mathcal{L}_{\beta}\left(\psi_{t'}\right) = \sum_{s=1}^{n-1}\left(\left\lfloor\frac{\psi_{t'}}{2^s}\right\rfloor \cdot \left\lceil\frac{2^s}{2P}\right\rceil\right). \tag{5.18}$$

The component $\Theta\left(\psi_{t'}\right)$ is calculated as:

$$\Theta\left(\psi_{t'}\right) = \sum_{s=1}^{n-1}\left(\mathcal{H}_{\psi_{t'}}\left(n-1-s\right) \cdot \left\lceil\frac{2^s}{2P}\right\rceil \cdot s\right). \tag{5.19}$$

The average execution time of applying the LLRM to the list-flip decoder is estimated using C simulated codewords. The key derivations (5.20)–(5.24) provided below are equivalent to (3.9)–(3.13) defined in Subsection 3.1.3.2 of Chapter 3. The only difference is the baseline algorithm. In this chapter, it is the SCL decoder, while in Chapter 3, it is the SC decoder.

The execution time of one codeword c, $0 \leq c \leq C-1$, with total $\tau\left(c\right)$ trials under the standard list-flip decoder is:

$$l_{\text{lflip}}\left(c\right) = \tau\left(c\right) \cdot \mathcal{L}_{\text{SCL}}, \tag{5.20}$$

where $\mathcal{L}_{\text{SCL}}$ is derived by (5.13).

The execution time to decode one codeword c with the list-flip decoder with LLRM is denoted by $l_{\text{lfipllrm}}\left(c\right)$ and is obtained as follows:

$$l_{\text{lfipllrm}}\left(c\right) = l_{\text{lflip}}\left(c\right) - \sum_{t'=1}^{\tau'(c)} \Delta\mathcal{L}_{\text{SCL}}\left(\psi_{t'}\right), \tag{5.21}$$

where $\tau'(c)$ is the number of additional trials applied to decode a codeword c, and $\Delta\mathcal{L}_{\text{SCL}}(\psi_{t'})$ indicates the execution-time reduction of an SCL decoding trial estimated by (5.15).

The average execution times of the list-flip decoder and the list-flip decoder with the LLRM, each denoted by $\overline{\mathcal{L}}_{\text{lflip}}$ and $\overline{\mathcal{L}}_{\text{lfliplrm}}$, are calculated as follows:

$$\overline{\mathcal{L}}_{\text{flip}} = \frac{1}{C}\sum_{c=0}^{C-1} l_{\text{flip}}(c), \tag{5.22}$$

$$\overline{\mathcal{L}}_{\text{lfliplrm}} = \frac{1}{C}\sum_{c=0}^{C-1} l_{\text{lfliplrm}}(c), \tag{5.23}$$

where C is the total number of simulated codewords. Then the reduction of the average execution time of a list-flip decoder with the LLRM compared to the original list-flip decoder, denoted by $\Delta\overline{\mathcal{L}}_{\text{lflip}}$, is estimated as:

$$\Delta\overline{\mathcal{L}}_{\text{lflip}} = \frac{\overline{\mathcal{L}}_{\text{lflip}} - \overline{\mathcal{L}}_{\text{lfliplrm}}}{\overline{\mathcal{L}}_{\text{lflip}}}. \tag{5.24}$$

5.6 Simulation Results

The effects of the proposed LLRM to list-flip decoders are observed through the simulations. A simulation setup similar to that described in Subsection 2.4.1 of Chapter 2 is applied. The 5G polar codes (3GPP, 2018) of length $N = 1024$ are used with three different rates $R \in \{1/8, 1/4, 1/2\}$. The 5G polynomial $z^{11} + z^{10} + z^9 + z^5 + 1$ generates $r = 11$ CRC codes. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed. For all simulations, the number of restart locations for the LLRM is set to $Y = 4$.

The values of the maximum number of trials $T_{\max}$ are selected as described in Subsection 2.4.1. For the DSCLF-ω decoders with $\omega \in \{1, 2, 3\}$, $T_{\max} \in \{21, 51, 301\}$ is set. For the SCLF decoder, $T_{\max} = 31$ is set. For the DSCLF-ω decoder, $T_{\max}$ is selected to closely approach the error-correction performance limit of its respective decoder. For the SCLF decoder, $T_{\max}$ is selected to achieve the performance of the SCL decoder with $L = 4$.

In this section, all list-flip decoders are analyzed with the list size $L = 2$. Focusing results on $L = 2$ is motivated by minimizing the memory overhead. In Subsection 5.2.2 of this chapter, it was shown that increasing the list size from $L = 2$ to $L = 4$ results in approximately 50% memory overhead for the DSCLF-ω decoders with the matched FER. However, the GRM and LLRM can also be applied to the list-flip decoders with $L > 2$. For the interested reader, results for $L = 4$ are provided in Appendix I.

For the LLRM, the restart locations $\boldsymbol{\Upsilon}_{\text{divN}}$, $\boldsymbol{\Upsilon}_{\text{divK}}$ and $\boldsymbol{\Upsilon}_{\text{prob}}$ designed by three methods are applied, all with the size $Y = 4$. The locations $\boldsymbol{\Upsilon}_{\text{prob}}$ designed with the probability-based method are obtained with prior simulations according to Algorithm 2 and Algorithm 3. These locations are obtained by simulation at the FER of 10^{-2} and then used for all other channel conditions.

5.6.1 Error-correction Performance

Figure 5.6 depicts the FER for SCLF and DSCLF-ω decoders with various ω for a $(1024, 512 + 11)$ polar code. The standard SCL decoders with $L = 8$ and $L = 16$ are provided for reference. The results in Figure 5.6 show that the DSCLF-ω with higher ω provides stronger error-correction performance compared to the SCLF. The results also show that the DSCLF-3 decoder with $T_{\text{max}} = 301$ has the strongest performance among the other decoders and that it closely approaches the SCL decoder with $L = 16$. However, this performance improvement results in worse execution time characteristics compared to other decoders (provided in this figure). The proposed LLRM will be more beneficial for DSCLF-3 decoder, which will serve as the algorithm of reference in the remaining part of this chapter.

Similar to the GRM and SRM that applied to flip decoders, applying LLRM and GRM to list-flip decoders does not affect the error-correction performance. The error-correction performance of the flip decoders with the GRM was provided in Figure 3.12 in Chapter 3.

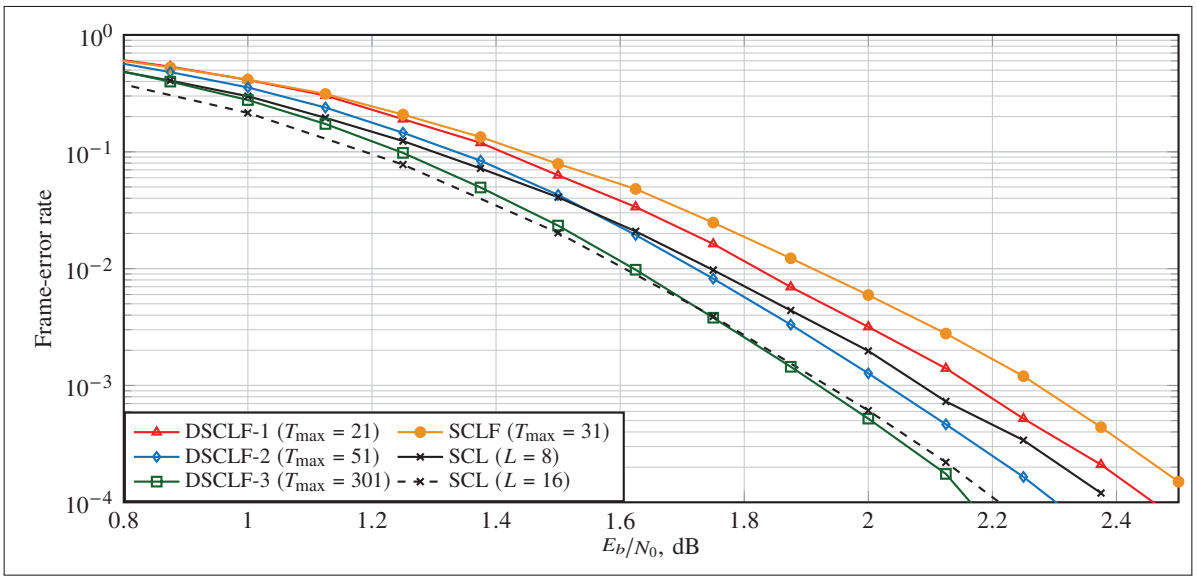


Figure 5.6 Error-correction performance of DSCLF-ω decoder with the decoding order $\omega \in \{1, 2, 3\}$ and $L = 2$ for a $(1024, 512 + 11)$ polar code

5.6.2 Memory Estimates

Memory requirements are estimated by (5.3)–(5.9) for the list-flip decoders with LLRM and GRM. The blocks of $\boldsymbol{\alpha}_{\text{ch}}$, $\boldsymbol{\alpha}_{\text{int}}$ and $\boldsymbol{\mathcal{M}}_{\text{flip}}$ are quantized by $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$, $Q_{\text{flip}} = 7$ bits, where the same quantization scheme described in Subsection 3.1.4.2 is used. This quantization scheme is based on that of Ercan *et al.* (2020a). All memory blocks storing the path metrics **PM** are quantized by $Q_{\text{pm}} = 8$ bits according to Hashemi *et al.* (2017).

Table 5.2 Memory estimates and overhead for list-flip decoders with $L = 2$ with LLRM, with GRM and the original decoders

Decoder	$T_{\max}$	Λ_{SCLF} (5.4)	R	Overh. w. LLRM			Overh. w. GRM
				$\boldsymbol{\Upsilon}_{\text{prob}}$	$\boldsymbol{\Upsilon}_{\text{divK}}$	$\boldsymbol{\Upsilon}_{\text{divN}}$	
		bits			%		%
SCLF	31	32270	1/4	1.82	5.53	2.14	235.00
			1/2	7.66	10.29	6.14	875.12
			3/4	4.62	15.05	11.96	1921.5
DSCLF-1	21	32100	1/4	2.69	5.56	2.16	236.17
			1/2	3.29	10.34	6.17	880.00
			3/4	3.71	15.13	12.02	1931.7
DSCLF-2	51	33110	1/4	2.20	5.39	2.09	228.96
			1/2	2.16	10.03	5.98	853.00
			3/4	2.69	14.67	11.66	1872.7
DSCLF-3	**301**	**42860**	1/4	**1.51**	4.16	1.61	**176.88**
			1/2	**1.38**	7.75	4.62	**658.90**
			3/4	**1.81**	11.33	9.01	**1446.70**

Table 5.2 provides the memory overhead, expressed percent, induced by embedding the GRM and LLRM to the list-flip decoders with $L = 2$ for a code with length $N = 1024$ and rates $R \in \{1/4, 1/2, 3/4\}$. For the LLRM, the restart locations $\boldsymbol{\Upsilon}_{\text{prob}}$, $\boldsymbol{\Upsilon}_{\text{divK}}$ and $\boldsymbol{\Upsilon}_{\text{divN}}$ are applied. In addition to overhead values, memory estimates of original list-flip decoders in bits are also provided. The smallest memory overhead is obtained for the DSCLF-3 decoder algorithm. The results indicate that applying the LLRM to the DSCLF-3 decoder leads to the smallest memory overhead compared to other decoders. This happens because the DSCLF-3 decoder requires the largest memory due to a higher value of $T_{\max}$, which reduces the impact of LLRM. The results also indicate that our proposed probability-based design $\boldsymbol{\Upsilon}_{\text{prob}}$ achieves the smallest overhead compared to the two other construction-based designs. When applying the LLRM to DSCLF-3 decoder, the memory overhead is of $\{1.51\%, 1.38\%, 1.81\%\}$ at the code rates $R \in \{1/4, 1/2, 3/4\}$. In comparison, when applying the GRM, the overhead is of $\{176.88\%, 658.90\%, 1446.70\%\}$. Due to the enormous memory overhead, implementation of the list-flip decoder with the GRM is unfeasible. On the other hand, the small overhead of the LLRM makes it feasible for hardware implementation.

5.6.3 Average Execution Time

The average execution time and its reduction by applying LLRM and GRM to list-flip decoders are estimated by equations (5.20)–(5.24). Figure 5.7 depicts the average execution time of the DSCLF-3 decoder for a polar code of the length $N = 1024$ and rates $R = \{1/4, 1/2, 3/4\}$. The presented decoding algorithms are the original DSCLF-3, DSCLF-3 with the GRM, and DSCLF-3 with the LLRM. For the LLRM, the results are shown for sets $\boldsymbol{\Upsilon}_{\text{prob}}$, $\boldsymbol{\Upsilon}_{\text{divN}}$ and $\boldsymbol{\Upsilon}_{\text{divK}}$. The execution time of the SCL decoder is shown for reference and corresponds to the lower bound of these decoding algorithms. The results indicate that the decoders with the GRM and LLRM achieve a significant reduction of the average execution time for all code rates and throughout the practical range of the FER. The results also show that this reduction is higher when applied to the codes with lower rates, and it gets smaller as the rate increases. The results also show that the LLRM has a higher reduction when applied with the proposed $\boldsymbol{\Upsilon}_{\text{prob}}$, closely approaching the reduction of the GRM.

Table 5.3 provides numerical results of the average execution-time reduction estimated by (5.24) for the list-flip decoders with $L = 2$. The reductions are provided by applying the GRM and LLRM to the original decoding at the target FER of 10^{-2}, in percent. The results show that a higher reduction is achieved for the DSCLF-3 decoder, which also provides the strongest error-correction performance compared to other decoders. Compared to original decoder, DSCLF-3 with the LLRM with $\boldsymbol{\Upsilon}_{\text{prob}}$ achieves the reductions $\Delta\overline{\mathcal{L}}_{\text{lflip}}$ of $\{39.16\%, 19.71\%, 10.60\%\}$ for code rates $R \in \{1/4, 1/2, 3/4\}$. When LLRM is applied with $\boldsymbol{\Upsilon}_{\text{divN}}$ and $\boldsymbol{\Upsilon}_{\text{divK}}$, the reductions are $\{30.56\%, 13.13\%, 6.09\%\}$ and $\{32.30\%, 15.15\%, 6.40\%\}$. When comparing the results of $\boldsymbol{\Upsilon}_{\text{divN}}$ and $\boldsymbol{\Upsilon}_{\text{divk}}$, the latter provides slightly better reduction for all scenarios. The results in Table 5.3 also show that DSCLF-1 decoder has a smaller reduction compared to other decoders. When applied with LLRM and $\boldsymbol{\Upsilon}_{\text{prob}}$, DSCLF-1 achieves reductions of $\{14.03\%, 6.60\%, 5.00\%\}$ for the code rates $R \in \{1/4, 1/2, 3/4\}$. Then, the results show that list-flip decoders with the LLRM and with $\boldsymbol{\Upsilon}_{\text{prob}}$ provide higher reductions compared to other mechanisms, with only around 1% to 3% loss from that provided by the GRM. The results in Table 5.3 show that applying the LLRM to list-flip decoders leads to memory overhead ranging from 2% to 15%, while for the GRM, this overhead ranges from 200% to 1500%.

5.7 Conclusion

This chapter described our proposed restart mechanisms for the list-flip decoders based on SCLF. The challenges of applying our previously proposed GRM to the list-flip decoder were explained. In the GRM, the part of the tree to estimate a path-flipping candidate and all the previous bits is skipped for each additional decoding trial. To perform such a restart, the state memories of each information bit are stored in memory after the initial SCL pass. However, this results in a significant increase in memory usage that exceeds the total memory of the standard list-flip decoder.

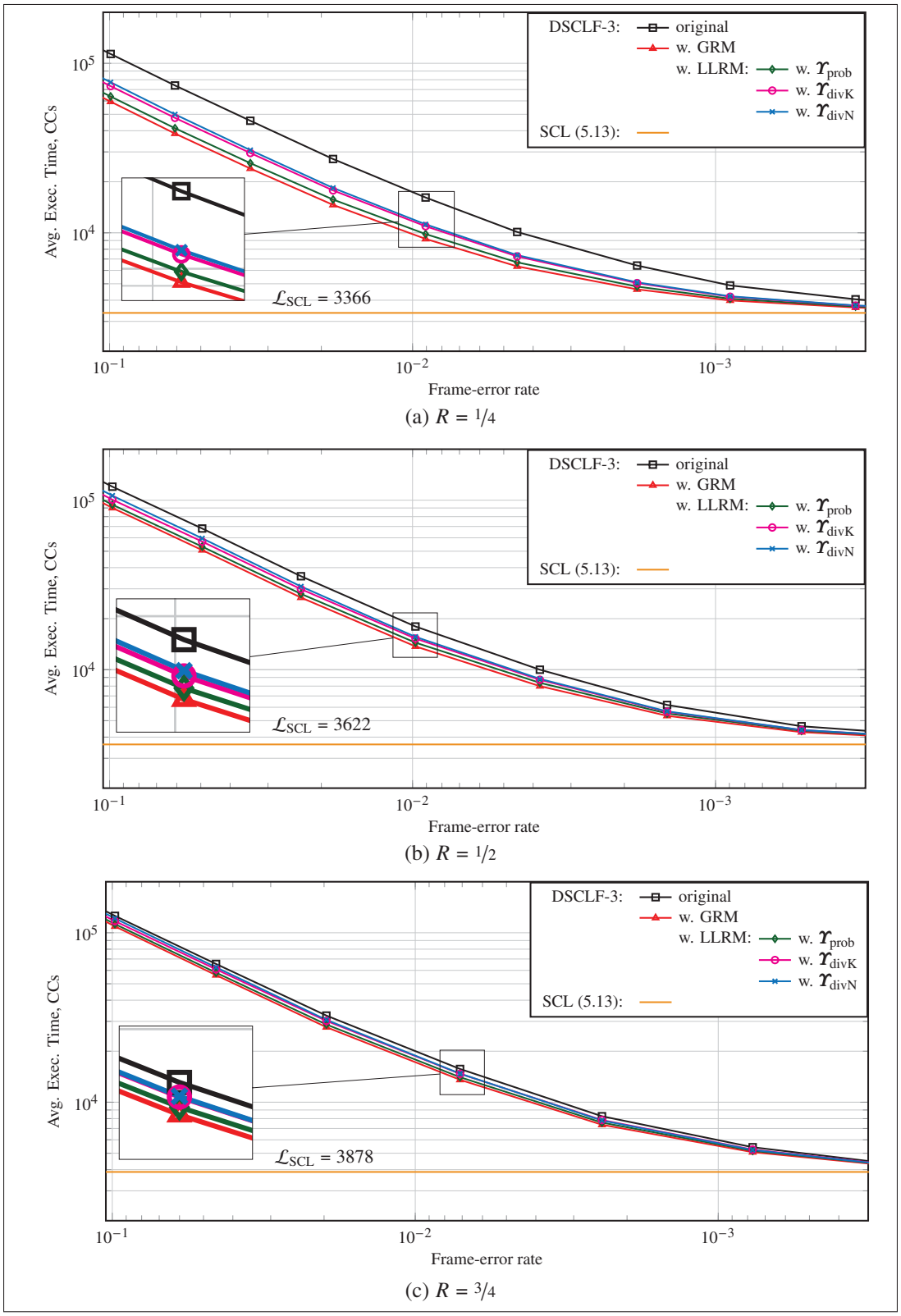


Figure 5.7 Average execution time of DSCLF-3 decoder with $L = 2$ with LLRM, GRM, and original decoder for $R \in \{1/4, 1/2, 3/4\}$

Table 5.3 Reduction $\Delta\overline{\mathcal{L}}_{\text{lflip}}$ (5.24) in % by applying the LLRM and GRM to list-flip decoders with $L = 2$ at FER 10^{-2}

R	Decoders	$T_{\max}$	E_b/N_0 (dB)	$\Delta\overline{\mathcal{L}}_{\text{lflip}}$, GRM	$\Delta\overline{\mathcal{L}}_{\text{lflip}}$, LLRM		
					Υ_{prob}	Υ_{divN}	Υ_{divK}
1/4	SCLF	31	1.250	20.13	17.66	13.68	15.88
	DSCLF-1	21	1.250	15.65	14.03	11.77	12.23
	DSCLF-2	51	1.125	27.20	24.42	19.98	20.90
	DSCLF-3	**301**	**1.000**	**43.02**	**39.16**	**30.56**	**32.30**
1/2	SCLF	31	1.875	22.06	18.60	18.00	17.00
	DSCLF-1	21	1.875	7.91	6.60	5.27	5.24
	DSCLF-2	51	1.750	14.88	11.63	9.15	9.73
	DSCLF-3	**301**	**1.625**	**23.70**	**19.71**	**13.13**	**15.15**
3/4	SCLF	31	3.000	8.50	6.32	5.45	4.66
	DSCLF-1	21	3.000	6.50	5.00	3.81	3.17
	DSCLF-2	51	2.875	11.16	8.55	5.65	5.30
	DSCLF-3	**301**	**2.875**	**13.67**	**10.60**	**6.09**	**6.40**

To overcome the issue of memory overhead, the LLRM, a modification of the GRM, was proposed. In the LLRM, an additional trial is resumed from a limited set of restart locations. The state memories of only these locations are required to be stored, which highly reduces the memory requirements. Similarly to GRM, the LLRM does not affect the original error-correction performance.

This chapter also described the three proposed design methods to determine the set of restart locations for the LLRM. The first two methods are based on code construction, where the locations equally divide the codeword or the information set of the code. These two methods are not very effective in terms of execution-time reduction capability, as they do not use any prior information about the path-flipping locations. However, they are simple and thus relevant for hardware implementation. The third design method is our proposed probability-based method, where the locations are derived by conducting simulations. This method selects the locations that equally divide a codeword in a way that each segment has an equal path-flipping probability distribution.

When applied to DSCLF-3 decoder, a decoder with the strongest error-correction performance among list-flip decoders, the proposed LLRM resulted in the most significant execution-time improvements. Compared to the original decoding, the DSCLF-3 with the LLRM and probability-based restart locations resulted in a reduction of the average execution time by 10% to 40%. When applied with the GRM, this reduction ranged from 13% to 43%. Compared to original DSCLF-3 decoding, the proposed LLRM resulted in approximately 2% memory overhead,

whereas the GRM resulted in 170% to 1500% memory overhead. According to these results, the LLRM is the more suitable algorithm for hardware implementations.

CHAPTER 6

EARLY-TERMINATION MECHANISMS FOR FLIP DECODERS

This chapter describes the two proposed execution-time reduction mechanisms for flip decoders (SCF). The central idea of these mechanisms is early termination depending on specific channel conditions or input data rates.

The early-stopping mechanism is presented with a detailed description of the algorithm and the key elements – the early-stopping metric and the metric threshold. If the metric suggests a codeword is likely undecodable, the decoder attempts a reduced maximum number of trials, much smaller than the initial maximum number of trials. However, if decoding is unsuccessful after these reduced trials, the decoder will stop. When applied to DSCF-1 decoder, the proposed early-stopping mechanism achieved a reduction of the average execution time by approximately 22% and execution-time variance by approximately 45%. This results in a minor error-correction loss of approximately 0.05 dB at the target FER of 10^{-2}.

This chapter further presents our proposed DSCF decoding under a fixed channel-production rate. The system model with a fixed channel-production rate is described, where the control mechanism is at the core of the system. This model also contains a buffer and decoder. This chapter also describes the proposed multi-threshold control mechanism. To prevent buffer overflow, this mechanism restrains the delay of a flip decoder depending on the state of the buffer. The results have shown that applying the multi-threshold mechanism with the DSCF-1 decoder enables a fixed channel production rate approximately 1.13 times lower than a single decoding trial while preventing buffer overflow. Compared to the ideal scenario, the proposed mechanism has an error-correction loss of approximately 0.06 dB at the target FER of 10^{-2}. The ideal scenario, where the decoder operates without constraints, corresponds to the unsustainable case, as the buffer quickly overflows.

The following publications resulted from the work presented in this chapter:

- I. Sagitov and P. Giard, "An early-stopping mechanism for DSCF decoding of polar codes." *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2020, Coimbra, Portugal, https://ieeexplore.ieee.org/document/9195213.
- I. Sagitov, C. Pillet, and P. Giard, "Successive-Cancellation Flip Decoding of Polar Codes Under Fixed Channel-Production Rate." *arXiv preprint arXiv:2409.03051,* 2024, https://doi.org/10.48550/arXiv.2409.03051.

In this chapter, we consider the execution time of the flip decoder as an integer multiple of the execution time of one SC decoding pass. This is different from the rest of this thesis, where execution time is derived in CCs based on a semi-parallel SC decoder implementation (Leroux *et al.*, 2013), which uses a limited number of processing elements executing in parallel. This approach is used for simplicity, but it does not limit generalization.

6.1 Early-Stopping Mechanism For Flip Decoder

6.1.1 Introduction

This section describes our proposed early-stopping mechanism, which differentiates undecodable codewords from decodable ones using the proposed early-stopping metric with a metric threshold. If the metric suggests a codeword is likely undecodable, the decoder attempts a reduced maximum number of trials, much smaller than the initial maximum number of trials. Simulation results are presented to demonstrate the effectiveness of the proposed mechanism when applied to the DSCF-1 decoder.

6.1.2 DSCF Decoder with Early-Stopping Mechanism

6.1.2.1 Summary of DSCF Decoding

The detailed introduction to the flip decoding, SCF and DSCF, is provided in Section 2.2 of Chapter 2. The DSCF decoding algorithm was proposed by Chandesris *et al.* (2018) with two major improvements to the original SCF. The first improvement is a more accurate metric for constructing the list of bit-flipping candidates, $\mathcal{B}_{\text{flip}}$. The second improvement is an algorithmic improvement allowing multiple simultaneous bit flips per trial.

In this chapter, only the single bit-flip DSCF decoder is used, i.e., DSCF-1. The list of bit-flipping candidates is denoted by $\mathcal{B}_{\text{flip}} = \{i_0, \ldots, i_{t'}, \ldots, i_{T_{\max}-1}\}$, where $i_{t'} \in \mathcal{A}$ is a bit-flipping location. The metrics $\mathcal{M}_{\text{flip}}(t') \in \boldsymbol{\mathcal{M}}_{\text{flip}}$, $t' \in \{0, \ldots, T_{\max} - 2\}$ are calculated for each bit-flipping candidate $i_{t'}$. As $\omega = 1$ is fixed in this chapter, (2.1) simplifies as follows:

$$\mathcal{M}_{\text{flip}}(t) = |\alpha_{\text{dec}}(j)| + \sum_{\substack{j \le i \\ j \in \mathcal{A}}} \mathcal{J}(\alpha_{\text{dec}}(j)), \tag{6.1}$$

where $\alpha_{\text{dec}}(j)$ is a decision LLR for the information bit-locations $j \in \mathcal{A}$. The function $\mathcal{J}$ can be approximated according to Ercan *et al.* (2020b) as:

$$\mathcal{J}(\alpha_{\text{dec}}(j)) = \begin{cases} 1.5, & \text{if } |\alpha_{\text{dec}}(j)| \le 5.0, \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

6.1.2.2 Early-Stopping Metric

The core idea of our proposed method is an early-stopping mechanism that attempts to distinguish the undecodable codewords from the decodable ones. When a codeword that is likely undecodable is identified, our decoder uses a

lower maximum number of trials. Then, we denote the maximum number of additional trials by T, and $T = T_{\max} - 1$. The early-stopping metric aims to identify undecodable codewords is the variance of the metric of the bit-flipping candidates $\boldsymbol{\mathcal{M}}_{\text{flip}}$. It is denoted by ϕ and is calculated as follows:

$$\phi = V_M = \frac{1}{T-1} \sum_{t=0}^{T-1} \left(\mathcal{M}(t) - m_{\text{av}} \right)^2 , \tag{6.3}$$

where $\mathcal{M}(t) \in \boldsymbol{\mathcal{M}}_{\text{flip}}$, and m_{av} is the mean of all elements of $\boldsymbol{\mathcal{M}}_{\text{flip}}$. The mean m_{av} is calculated as:

$$m_{\text{av}} = \frac{1}{T} \sum_{t=0}^{T-1} m_t . \tag{6.4}$$

The early-stopping metric ϕ is not to be confused with metrics associated with the bit-flipping candidates $\boldsymbol{\mathcal{M}}_{\text{flip}}$ of the DSCF decoder. The metric ϕ of our proposed algorithm aims to distinguish the codewords that are likely undecodable from the decodable ones.

6.1.2.3 Early-Stopping Algorithm

The early-stopping metric ϕ can be integrated to the DSCF decoder and is calculated for the current codeword after obtaining the list $\boldsymbol{\mathcal{M}}_{\text{flip}}$. Then, the resulting metric is compared against the pre-calculated threshold metric ϕ_{thr}. If $\phi > \phi_{\text{thr}}$, the maximum number of trials becomes the pre-defined reduced maximum number of trials $T = T_{\text{red}}$. Otherwise, the initial maximal number of trials T is kept. The simulation set-up is created and its corresponding algorithm is summarized in Algorithm 4.

Algorithm 4 DSCF decoding algorithm with early stopping

procedure DSCF_With_Early_Stop(T, ϕ_{thr}, T_{red})
  $(\hat{\boldsymbol{u}}, \boldsymbol{\alpha}_{\text{dec}}) \leftarrow \text{SC}(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A})$ ▷ Run initial SC trial
  if $CRC(\hat{u}^N) =$ failure **then**
    Init $(\mathcal{B}_{\text{flip}}, \boldsymbol{\mathcal{M}}_{\text{flip}})$
    $\phi \leftarrow V_{\mathcal{M}}(\boldsymbol{\mathcal{M}}_{\text{flip}})$ ▷ Calculate early-stopping metric ψ (6.3)
    if $\phi > \phi_{\text{thr}}$ **then** ▷ If metric exceeds threshold
      $T \leftarrow T_{\text{red}}$ ▷ Apply reduced maximum number of trials
    end if
    $(\hat{\boldsymbol{u}}, \boldsymbol{\alpha}_{\text{dec}}, \tau') \leftarrow \text{SCF}(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}, \mathcal{B}_{\text{flip}}, \boldsymbol{\mathcal{M}}_{\text{flip}}, T)$ ▷ Run SCF with selected T
  end if
  return $\{\hat{\boldsymbol{u}}, \tau'\}$ ▷ Return the resulting bit estimates and applied trials
end procedure

Simulations are for various E_b/N_0 points. The output number of trials τ', *beyond the initial trial,* can be used to build statistics of the execution time averaged over the number of simulated codewords. The average execution time

T_{av}, in terms of average of number of trials τ', can be calculated as:

$$T_{\text{av}} = \frac{1}{C} \sum_{c=0}^{C-1} \tau'(c), \tag{6.5}$$

where C corresponds to the total number of simulated codewords per a E_b/N_0 point, c is the codeword index, and $\tau'(c)$ is the number of resulting trials for each codeword.

The variance of the execution time V_T can also be obtained from the output τ and is calculated as follows:

$$V_T = \frac{1}{C-1} \sum_{c=0}^{C-1} (\tau'(c) - T_{\text{av}})^2. \tag{6.6}$$

6.1.3 Obtaining the Threshold from the Metric Distribution

The DSCF algorithm with an early stopping requires the threshold metric ϕ_{thr} and the reduced maximum number of trials T_{red}, values determined by way of simulation. The distribution of the average ϕ is obtained as a function of the number of additional trials τ' required by the original DSCF decoder. Then, the corresponding thresholds can be defined based on the resulting distribution.

Simulation Setup

The simulation setup is created to obtain the average metric distribution depending on the resulting number of trials t, where $0 \leq t \leq T$. We simulate the random codewords encoded by a $(1024, 512 + 16)$ polar code with the CRC polynomial $z^{16} + z^{15} + z^2 + 1$. That polar-code length and rate, and that CRC length, were chosen to ease comparison with other works as they are commonly found in the literature. The polar code is constructed for an approximate design E_b/N_0 of 2.365 dB with the construction method proposed by Tal & Vardy (2013). The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating a minimum of $C = 10^7$ codewords. The DSCF-1 decoder is simulated with the maximum number of trials beyond the initial SC decoding pass of $T = 10$.

6.1.3.1 Average Early-Stopping Metric Distribution

The Algorithm 5 describes how to obtain the average early-stopping metric distribution, where c stands for the simulation index and C corresponds to the total number of simulated codewords. The list of metrics $\boldsymbol{\Phi}$ has the length $T + 2$, and it includes the resulting early-stopping metrics obtained from the codewords successfully decoded by initial SC trials $\tau' = 0$, those corresponding to undecodable codewords, and those for each $1 \leq \tau' \leq T$ corresponding to successfully-decoded codewords with additional trials. It can be seen that the metric ϕ is calculated on line 8

and is accumulated into the list $\boldsymbol{\Phi}$ at the appropriate position based on τ', either on line 13 or 16 if decoding is successful or not, respectively. The list of counters $\boldsymbol{F}$, which has the same length as $\boldsymbol{\Phi}$, increments its element according to the output τ' on line 17 in the case of successful decoding or on line 14 otherwise. At the end of the simulation, the elements of the list of accumulated metrics $\boldsymbol{\Phi}$ are normalized by their corresponding counters $\boldsymbol{F}$ to obtain the average value. This simulation is run for each E_b/N_0 point.

Algorithm 5 Obtaining the metric distribution by simulation

1: **procedure** GET_DISTRIBUTION(C, T)
2: $\quad\boldsymbol{\Phi}^{T+2} \leftarrow \{0,\ 0,\ \ldots,0\}$ ▷ Initialize the list of metrics $\boldsymbol{\Phi}$ of the length $T+2$
3: $\quad\boldsymbol{F}^{T+2} \leftarrow \{0,\ 0,\ \ldots,0\}$ ▷ Initialize the list of metric counter of the length $T+2$
4: $\quad$**for** $c = 0;\ c < C;\ c++$ **do**
5: $\quad\quad(\hat{\boldsymbol{u}},\ \boldsymbol{\alpha}_{\text{dec}}) \leftarrow \text{SC}\,(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A})$ ▷ Run initial SC trial
6: $\quad\quad\tau' \leftarrow 0$
7: $\quad\quad\text{Init}\left(\mathcal{B}_{\text{flip}}, \boldsymbol{\mathcal{M}}_{\text{flip}}\right)$
8: $\quad\quad\phi \leftarrow V_M\left(\boldsymbol{\mathcal{M}}_{\text{flip}}\right)$ ▷ Calculate early-stopping metric ϕ (6.3)
9: $\quad\quad$**if** $CRC\,(\hat{\boldsymbol{u}}) =$ failure **then**
10: $\quad\quad\quad(\hat{\boldsymbol{u}}, \boldsymbol{\alpha}_{\text{dec}}, \tau') \leftarrow \text{SCF}\left(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}, \mathcal{B}_{\text{flip}}, T\right)$ ▷ Run SCF with T trials
11: $\quad\quad$**end if**
12: $\quad\quad$**if** $(\boldsymbol{u} \neq \hat{\boldsymbol{u}})$ **then**
13: $\quad\quad\quad\Phi\,(T+1) \leftarrow \Phi\,(T+1) + \phi$ ▷ Accumulate metric ϕ for undecodable codewords
14: $\quad\quad\quad F\,(T+1) \leftarrow F\,(T+1) + 1$ ▷ Increment the metric counter
15: $\quad\quad$**else**
16: $\quad\quad\quad\Phi\,(\tau') \leftarrow \Phi\,(\tau') + \phi$ ▷ Accumulate metric ϕ for decoded codewords
17: $\quad\quad\quad F\,(\tau') \leftarrow F\,(\tau') + 1$ ▷ Increment the metric counter
18: $\quad\quad$**end if**
19: $\quad$**end for**
20: $\quad$**for** $j = 0;\ j \leq (T+1);\ j++$ **do**
21: $\quad\quad\Phi\,(j) \leftarrow \frac{\Phi(j)}{F(j)}$ ▷ Normalize the metric list $\boldsymbol{\Phi}$
22: $\quad$**end for**
23: $\quad$**return** $\boldsymbol{\Phi}$ ▷ Return the normalized list of early-stopping metrics $\boldsymbol{\Phi}$
24: **end procedure**

The bar diagram of the resulting average early-stopping metric distribution is presented in Figure 6.1. The distribution is obtained by simulating at $E_b/N_0 = 2.25$ dB, that correspond to FER of 10^{-2} of the DSCF-1 decoder. From this figure, it can be seen that the average early-stopping metrics are distributed exponentially within the range $1 \leq \tau' \leq 10$. The average early-stopping metric corresponding to undecodable codewords is clearly higher than those of decodable codewords with the highest values of τ' and is close to that of $\tau' = 2$.

6.1.3.2 Defining the Threshold Metric and the Reduced Number of Maximum Trials

According to the distribution of the average early-stopping metrics shown in Figure 6.1, the threshold metric ϕ_{thr} gets the value of the average metric obtained for the undecodable codewords (denoted as "f" in the figure). A threshold metric ϕ_{thr} is defined for each E_b/N_0 point of interest. The relative position of the normalized metric for the undecodable codewords has been observed to be independent of the channel E_b/N_0. As a consequence, a single value for the reduced maximum number of trials T_{red} is obtained and then used for all E_b/N_0. In the following section, simulation results will be presented and discussed for $T_{\text{red}} \in \{2,\ 3,\ 4\}$.

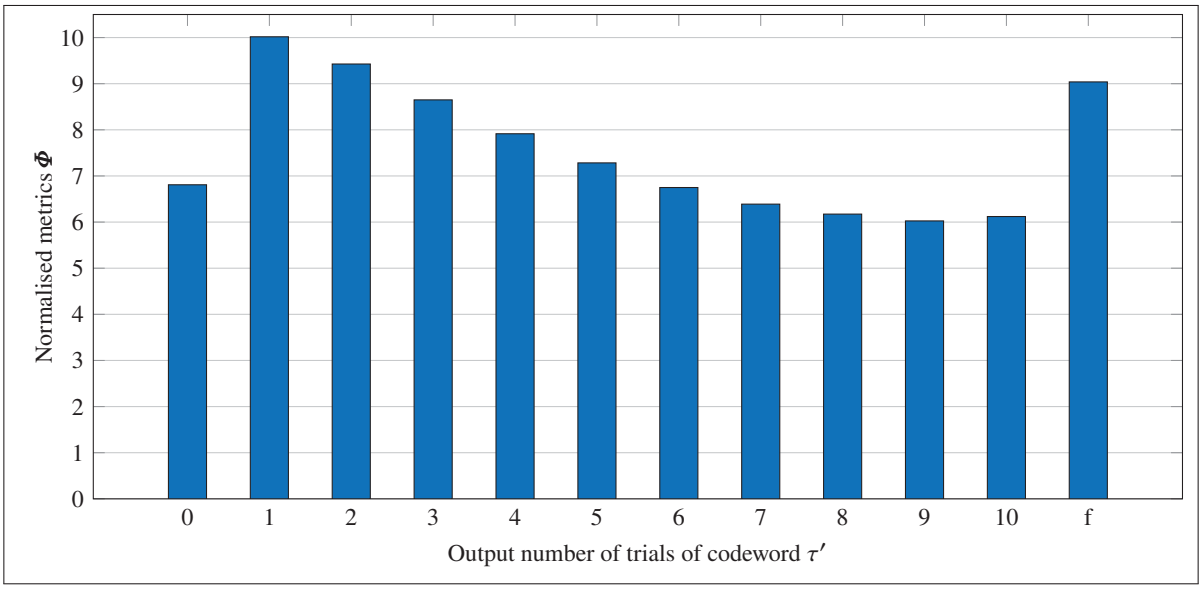


Figure 6.1 Distribution of early-stopping metrics depending on the number of trials required by the original DSCF-1 decoder at the FER of 10^{-2} for a $(1024,\ 512 + 16)$ polar code

6.1.4 Simulation Results

In this section, we compare the performance of DSCF decoding with our proposed early-stopping mechanism against the original decoder. The comparison is made in terms of the average execution time, execution-time variance, and error-correction performance.

Simulation Setup

The results are obtained using the same simulation parameters as those described in Subsection 6.1.3 except that the simulations are conducted by generating a minimum of $C = 10^5$ codewords or continuing until 1000 frame errors are observed. The values of E_b/N_0 in the range of $\{1.0,\ 1.25,\ \ldots\ 2.25\}$ dB are applied. Then, the minimum number of frame errors is 500 and 300 for the E_b/N_0 points of 2.5 dB and 2.75 dB accordingly. Several reduced numbers of maximum trials $T_{\text{red}} \in \{2,\ 3,\ 4\}$ are simulated for the DSCF decoder with the proposed early-stopping mechanism. The DSCF-1 decoder is simulated with the maximum number of trials beyond the initial SC decoding pass of $T = 10$. The metrics for the bit-flipping candidates $\mathcal{M}_{\text{flip}}$ are calculated according to (6.1).

6.1.4.1 Average Execution Time and Variance

Figure 6.2 shows the average execution time for the proposed DSCF decoder that incorporates our early-stopping mechanism, with $T_{\text{red}} \in \{2,\ 3,\ 4\}$. The average execution time for the original DSCF decoding algorithm is included

for comparison. Similarly, Figure 6.3 shows the execution-time variance for our proposed decoding algorithm and includes that of the original DSCF algorithm for comparison. In both figures, the original algorithm is depicted as a black curve with circle markers and the remainder of the curves are for our proposed modified algorithm.

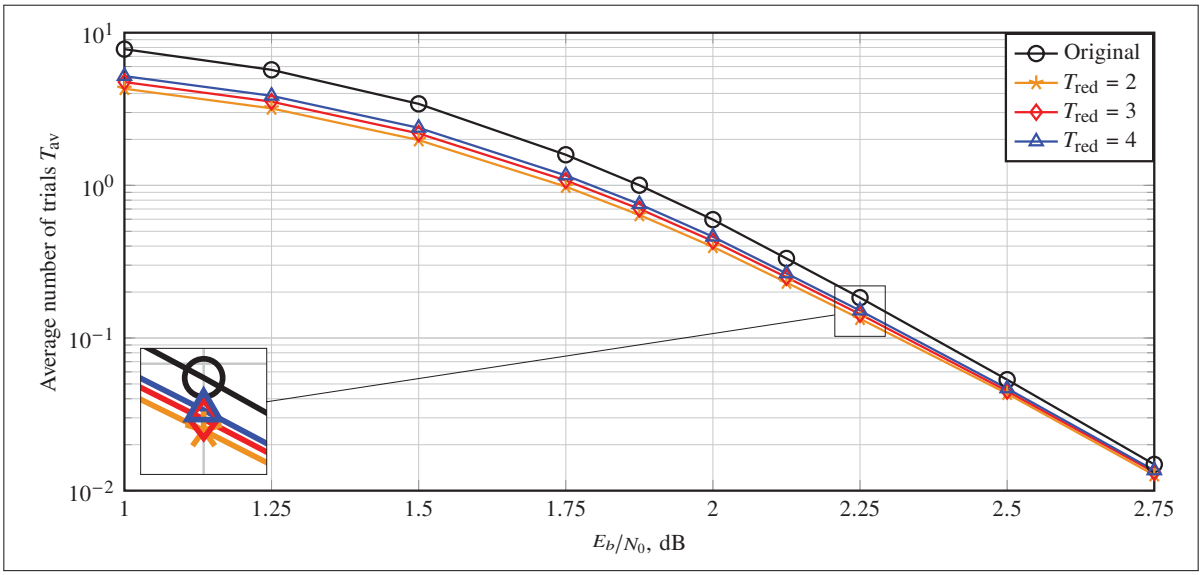


Figure 6.2 Average number of trials beyond the first SC decoding pass for various E_b/N_0 points of the DSCF-1 decoder for a $(1024, 512 + 16)$ code, with and without the proposed early-stopping mechanism

Figure 6.2 also shows that the proposed early-stopping mechanism results in a reduction in average execution time compared to the original algorithm. This improvement is observed even with the reduced maximum number of trials used. This reduction is more significant at low E_b/N_0 and gradually vanishes as the channel condition improves. At the $E_b/N_0 = 2.25$ dB, which corresponds to the FER of 10^{-2} as shown in Figure 6.4, the reduction is of 22% for $T_{red} = 3$. Another observation is that the differences between the three curves corresponding to the modified DSCF are very small and the curve representing the decoder with $T_{red} = 2$ achieves the smallest average execution time.

Figure 6.3 reveals that similar to the average execution time, our proposed early-stopping mechanism also leads to a reduction in execution-time variance. Contrary to the reduction in the average execution time, the reduction in the execution-time variance is very much similar across all E_b/N_0 points. For the E_b/N_0 point of interest, i.e., $E_b/N_0 = 2.25$ dB, the reduction is of 45% for $T_{red} = 3$. Again, we see that the differences between curves corresponding to $T_{red} \in \{2, 3, 4\}$ are very small.

6.1.4.2 Error-Correction Performance

Figure 6.4 shows the error-correction performance in terms of FER of the modified DSCF decoder with the proposed early-stopping mechanism as well as that of the original DSCF decoder. Similarly to Figure 6.2 and Figure 6.3, the

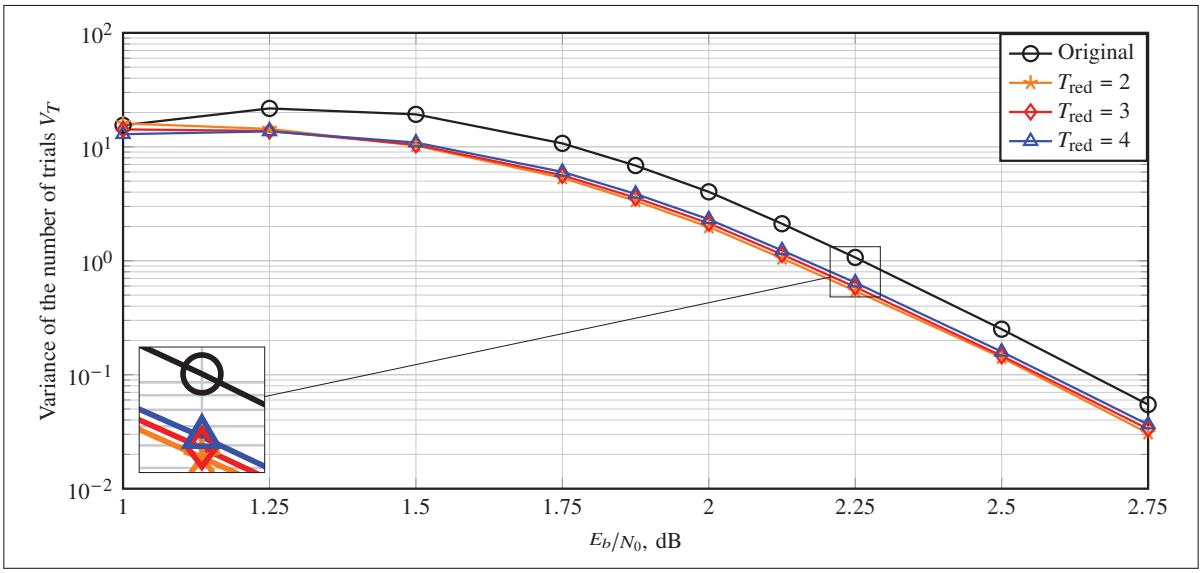


Figure 6.3 Variance of the number of trials for various E_b/N_0 points for a DSCF-1 decoder for a $(1024,\ 512 + 16)$ code, with and without the proposed early-stopping mechanism

original DSCF algorithm is depicted as a black curve with circle markers and the remainder of the curves are for our modified algorithm. It can be seen that a reduced maximum number of trials T_{red} of 2 with the codewords identified as likely undecodable leads to a coding loss that is little under 0.1 dB at the FER of 10^{-2} compared to the original DSCF algorithm. Increasing T_{red} to 3 significantly reduces that loss. The gap between the proposed algorithm and the original one is under 0.05 dB at the same FER. Increasing T_{red} to 4 further reduces the error-correction loss but the improvement between $T_{\text{red}} = 3$ and $T_{\text{red}} = 4$ is not as important as the one created by going from 2 to 3. For reference, we note that the FER of the DSCF decoder with $T = 10$, with and without the proposed early-stopping algorithm, falls between the FERs of a CRC-aided SCL decoder with list sizes $L = 2$ of $L = 4$.

Taking all three metrics into consideration—the average execution time, the execution-time variance, and the error-correction performance—, with a $(1024,\ 512 + 16)$ polar code, using $T_{\text{red}} = 3$ appears to offer the best tradeoff for a modified DSCF decoder that implements our proposed early-stopping mechanism.

6.1.5 Conclusion on Early-Stopping Mechanism for Flip Decoder

This section presented the proposed early-stopping mechanism for the flip decoder. The mechanism attempts to distinguish undecodable codewords from decodable ones. The key ingredient of this mechanism is the combination of an early-stopping metric with the metric threshold found through simulation. Based on that metric, a codeword may be classified as likely undecodable, in which case the decoder attempts a reduced maximum number of trials, much smaller than the initial maximal number of trials. After those trials, if decoding is not successful, the decoder

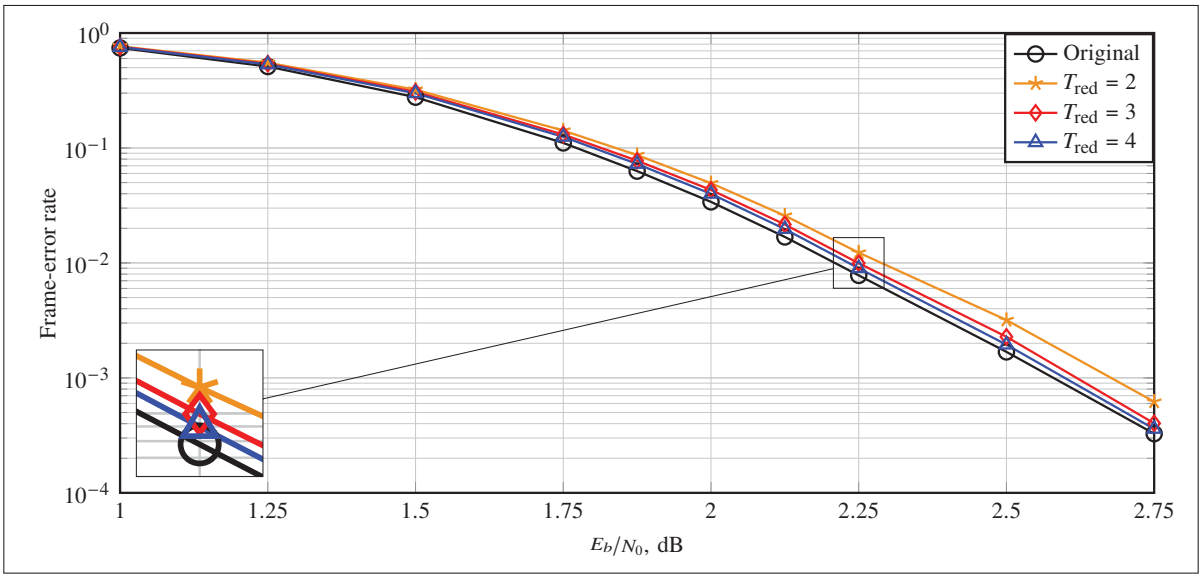


Figure 6.4 FER of the DSCF-1 decoder for a $(1024, 512 + 16)$ with and without the proposed early-stopping mechanism

stops. When applied to DSCF-1 decoder, the proposed early-stopping mechanism achieves the reduction of the average execution time by approximately 22% and execution-time variance by around 45%. These reductions resulted in a minor error-correction loss of approximately 0.05 dB at the FER of 10^{-2}. Overall, the proposed early-stopping mechanism is simple and effective for implementation to flip decoders.

6.2 Flip Decoder Under Fixed Channel-Production Rate

6.2.1 Introduction

This section presents the proposed multi-threshold mechanism for the flip decoder, which regulates the decoding delay to prevent buffer overflow. The proposed mechanism is at the core of the system model, which contains the channel, buffer, controller, and decoder. This system model allows the flip decoder to operate with a fixed channel-production rate, meaning that incoming data blocks are delivered from the channel at a fixed time interval. The multi-threshold mechanism aims to avoid buffer overflow while minimizing the error-correction performance loss. Simulation results are presented to demonstrate the effectiveness of the proposed multi-threshold mechanism when applied to the DSCF-1 decoder.

6.2.2 System Model with Fixed Channel Production Rate

In this work, we use a system model where the communication chain is simplified such that parts of the transmitter, the channel, and the detector, are lumped into one block denoted as the channel. Figure 6.5 illustrates this simplified model, where the channel acts as a data generator to the remainder of the model that is the central part of this work, i.e., the buffer, the controller, and the decoder.

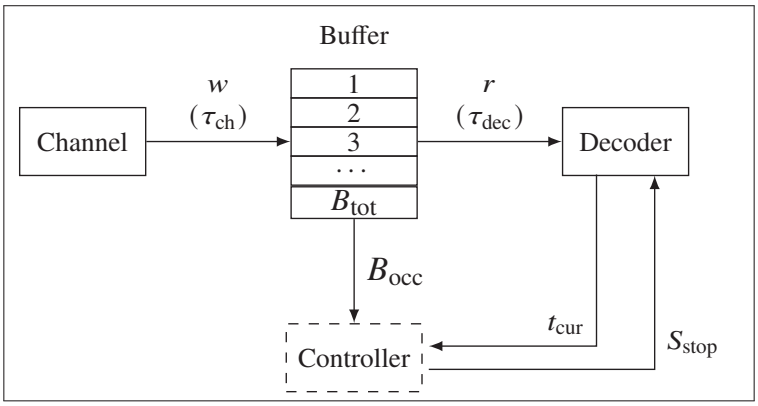


Figure 6.5 System model containing simplified blocks of the channel, buffer, controller, and flip decoder. Arrows indicate the data flow between the blocks

By denoting the latency of an SC trial by τ_{sc}, the execution time of one word under the flip decoding is calculated as:

$$\tau_{dec} = t_{req} \cdot \tau_{sc}, \tag{6.7}$$

where t_{req} is the required number of trials to decode a given word.

6.2.2.1 Channel Block

The channel block in our model acts as the generator that delivers incoming data blocks (words) to the decoder. The words are generated at a fixed time interval τ_{ch} and stored in the buffer. The direction of the data write operation is illustrated by the arrow that is denoted by w in Figure 6.5. We define the channel-production interval τ_{ch} as follows:

$$\tau_{ch} = \upsilon_{pr} \cdot \tau_{sc}, \tag{6.8}$$

where $\upsilon_{pr} \in \mathbb{R}^+$ is an additional coefficient that we call the production coefficient, and τ_{sc} corresponds to the latency of one SC decoding trial (6.7). The channel-production interval τ_{ch} cannot be lower than the latency of a single trial τ_{sc}, as the buffer will quickly overflow, even if the decoder with only a single SC trial is used. This is necessary because a single SC trial is the minimum delay required for the decoder to process any word. Thus, the relation

$\upsilon_{\text{pr}} \geq 1$ is maintained. An increase in the production coefficient corresponds to an increase in the data-production interval by the channel. The term channel-production rate used in this section corresponds to the inverse of the channel-production interval τ_{ch}.

6.2.2.2 Buffer Block

The buffer is used as memory to store words coming from the channel. The buffer is divided into slots, where each slot can accommodate one word. In this work, we consider a circular buffer. We denote the size of the buffer by the total number of slots B_{tot}, and the number of occupied slots is denoted by B_{occ}. One received word takes one slot in the buffer. The number of occupied slots B_{occ} is provided to the controller block.

6.2.2.3 Decoder Block

The decoder block reads the received words from the buffer. The reading event is illustrated by the arrow denoted by r in Figure 6.5. We implement a single bit-flip DSCF-1 decoder. The DSCF-1 decoder differs from the SCF only with the metric function to obtain the bit-flipping candidates, which is calculated by (6.1).

The flip decoder operates with a maximum number of trials T_{max}. However, the behavior of the block can change if the controller asserts its S_{stop} signal. When S_{stop} is $True$, the decoder immediately ceases the current decoding attempt, declares decoding failure, and starts processing the next word. Reading it from the buffer releases a memory slot, moving away from overflow. While S_{stop} is $False$, the decoder maintains its usual behavior, i.e., attempts up to T_{max} trials. The decoder provides the current number of fully applied trials t_{cur} to the controller.

6.2.3 Control Mechanisms

As illustrated in Figure 6.5, the controller is a key ingredient to our model. It regulates the decoder based on the number of available memory slots in the buffer. The controller aims to avoid buffer overflowing while minimizing the error-correction performance loss.

The buffer has two critical states in processing: buffer underflow and buffer overflow. Buffer underflow can easily be avoided, e.g., by suspending the decoder until the buffer is further filled with data. Buffer underflow does not affect the error-correction performance. However, buffer overflow is more challenging to deal with as it essentially requires controlling the worst-case execution time of the decoder which affects the error-correction performance. Therefore, our work focuses on control mechanisms that can prevent buffer overflow.

In our model, the controller regulates the operation of the flip decoder using thresholds. As the number of occupied slots in the buffer gets closer to overflow, pre-defined thresholds are violated. The decoding delay is then restricted by reducing the maximum number of decoding trials.

In this work, the controller can implement two different mechanisms, namely codeword dropping or multi-threshold mechanisms. Algorithm 6 illustrates the GEN_CTRL_SIGS algorithm that generates the control signals. This algorithm considers both codeword-dropping and multi-threshold mechanisms. The inputs of the GEN_CTRL_SIGS algorithm are the sets of buffer-size and trial-decoding thresholds, denoted by $\boldsymbol{B}$ and $\boldsymbol{T}$, respectively. The sets consist of multiple thresholds, where $\boldsymbol{B} = \{B_1, B_2, \ldots, B_Y\}$ and $\boldsymbol{T} = \{T_1, T_2, \ldots, T_Y\}$ with Y being the number of thresholds in each set. The set of thresholds $\boldsymbol{B}$ is sorted in descending order while the set $\boldsymbol{T}$ is sorted in ascending order. Once sorted, each threshold from $\boldsymbol{B}$ corresponds to the threshold of $\boldsymbol{T}$ located at the same position, i.e., they form a threshold pair according to their index.

As illustrated by Algorithm 6, the states of the buffer and the decoder are obtained through the number of occupied buffer slots B_{occ} and the current number of decoding trials t_{cur}. The buffer state B_{occ} is compared to the elements $B_i \in \boldsymbol{B}$. When the first violation is detected, the decoder state t_{cur} is compared to the threshold $T_i \in \boldsymbol{T}$ of the corresponding index i. If a violation is detected, the controller stops the decoder.

Algorithm 6 Generating the controller signals according to the states of the buffer and decoder

```
1:  procedure GEN_ CTRL_SIGS(B, T)
2:      B_occ ← buf.getOccSlots()          ▷ Obtain current number of occupied buffer slots
3:      t_cur ← decoder.getCurTrials()     ▷ Obtain current number of decoding trials
4:      S_stop ← False                     ▷ Reset the decoder stop flag
5:      for i in 1 ... Y do
6:          if B_occ > B_i and t_cur ≥ T_i then   ▷ Check threshold violations for buffer and decoder
7:              S_stop ← True              ▷ Set the decoder stop flag if any threshold is violated
8:              break
9:          end if
10:     end for
11:     return S_stop                      ▷ Return the decoder stop flag
12: end procedure
```

6.2.3.1 Codeword-Dropping Mechanism

The codeword-dropping mechanism follows Algorithm 6 with a single threshold pair $\{B_1, T_1\}$. Note that the threshold-violation check loop is executed only once since $Y = 1$. The codeword-dropping mechanism is activated only when the buffer is very close to overflow, i.e., B_1 is almost equal to B_{tot}. In this case, the trial-decoding threshold is set to $T_1 = 0$. This way, when $B_{\text{occ}} > B_1$, the decoder is immediately stopped regardless of how many trials have been attempted, i.e., the current codeword is dropped.

6.2.3.2 Multi-Threshold Mechanism

The multi-threshold mechanism follows Algorithm 6 with sets of multiple thresholds. For simplicity, we propose to use sets composed of $Y = 3$ thresholds. The buffer-size thresholds satisfy $B_3 < B_2 < B_1 < B_{\text{tot}}$ while the trial-decoding thresholds are $T_1 < T_2 < T_3 \leq T_{\max}$. The threshold pair $\{B_1, T_1\}$ is the same as codeword dropping. To obtain the best performance and tradeoff, the number of buffer-size thresholds and their values are expected to vary depending on code length and rate, channel condition, and $T_{\max}$. The general goal remains the same: evenly set the buffer-size thresholds throughout the buffer to achieve gradual control. The methodology to derive the trial-decoding thresholds is provided in the following subsection.

6.2.4 Threshold-Selection Methodology

As discussed in Subsection 6.2.3.2, the trial-decoding threshold $T_1 = 0$, and the buffer-size thresholds $\{B_1, B_2, \ldots, B_Y\}$ are evenly distributed across the buffer. By setting $Y = 3$, only the thresholds T_2 and T_3 need to be derived. The proposed threshold-selection methodology requires obtaining the balanced number of trials of flip decoding from offline simulations at the target FER of 10^{-2} and selecting the targeted production coefficient υ_{pr}.

The key metric for determining the balanced number of trials T_{bal} is the average number of decoding trials T_{av} derived from offline simulations. Experiments have shown that our system model can operate with a fixed channel-production rate without buffer overflow if the average number of trials T_{av} of the decoder, restricted by $T_{\max}$ alone, does not exceed the production coefficient υ_{pr}. To establish a good tradeoff between error-correction performance and buffer-overflow prevention, we start by defining the balanced number of trials as $T_{\text{bal}} = \max(T_{\max}) | T_{\text{av}} < \upsilon_{\text{pr}}$.

Further details on the simulation setup are provided in Subsection 6.2.5. In this subsection, only the key settings are provided to facilitate derivations of T_{bal}. Simulations are conducted by generating $C = 10^6$ codewords, decoded by the DSCF-1 decoder. For each simulation, the maximum number of trials $T_{\max} \in \{1, \ldots, 11\}$ is applied. Simulations are run for the ideal case, where $T_{\max}$ is the only decoding latency restriction. A channel E_b/N_0 of 2.25 dB is used in all simulations, as it corresponds to the FER of 10^{-2} for the DSCF-1 decoder with $T_{\max} = 11$ in the ideal (unrestricted) case.

Figure 6.6 shows the resulting average number of trials T_{av} for each $T_{\max}$ value. For the illustration, consider the two production coefficients $\upsilon_{\text{pr}} = 1.091$ and $\upsilon_{\text{pr}} = 1.125$ represented by the horizontal lines in Figure 6.6, highlighted in solid green and dashed red, respectively. In this example, the balanced number of trials is $T_{\text{bal}} = 2$ for $\upsilon_{\text{pr}} = 1.091$ whereas it is $T_{\text{bal}} = 4$ for $\upsilon_{\text{pr}} = 1.125$.

For our proposed multi-threshold mechanism, we suggest setting thresholds T_2 and T_3 as T_{bal} and $T_{\text{bal}} + 1$, respectively. As discussed in Subsection 6.2.3.2, the thresholds B_2 and B_3 are set to the middle and the head slots of the buffer.

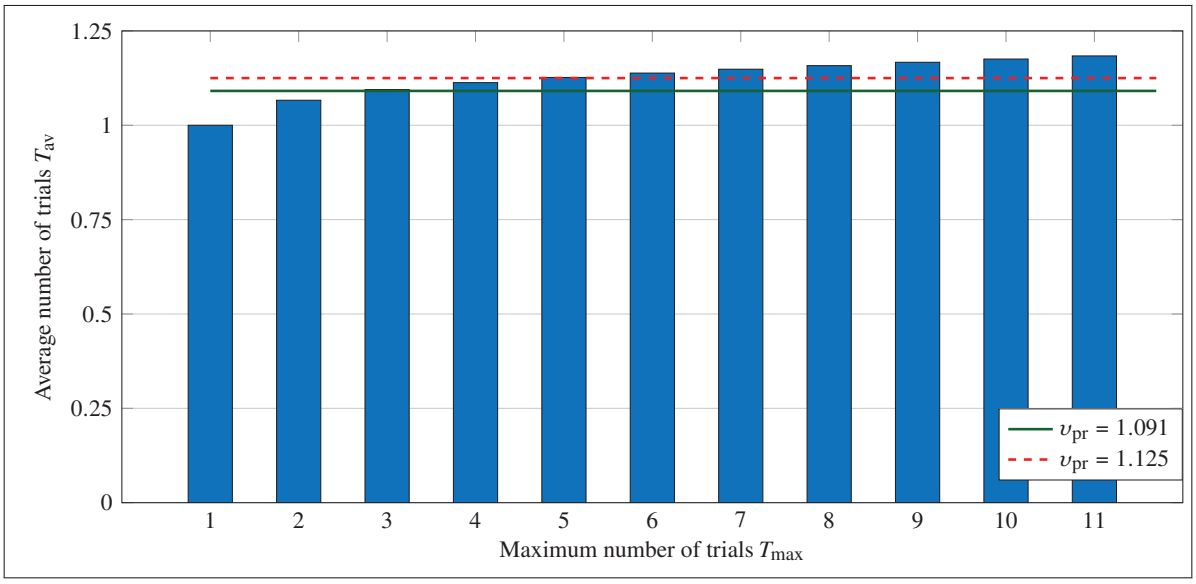


Figure 6.6 Average execution time of a DSCF-1 decoder with various maximum number of trials $T_{\max}$. Two examples of production coefficients υ_{pr} are shown as horizontal lines

This way, the multi-threshold mechanism cuts off the high decoding trials exceeding T_3 once the buffer is filled up to B_3, and further restricts decoding to T_2 trials when the buffer is half full. As a further protection against buffer overflow, codeword dropping is activated when the buffer is full. The proposed methodology is applicable to other configurations, i.e., different N and k of the polar code, channel E_b/N_0, and $T_{\max}$.

We highlight that applying too low υ_{pr}, i.e., high data rates, will put too much pressure on the multi-threshold mechanism resulting in the equivalent of the codeword-dropping mechanism. Therefore, when possible, we recommend selecting a data rate that results in a $T_{\text{bal}} \geq 2$. On the other hand, if too low data rates are applied to the extent that $T_{\text{bal}} = T_{\max}$, the multi-threshold mechanism is not necessary to avoid buffer overflow; $T_2 = T_3 = \ldots = T_Y = T_{\max}$.

6.2.5 Simulation Results

This subsection describes the simulation setup, followed by a detailed explanation of the simulation algorithm. Then, the results are provided comparing the buffer utilization and error-correction performance of the decoder with the multi-threshold mechanism and the ideal decoder.

6.2.5.1 Simulation Setup

The simulation of the system model consists of a series of iterations with each iteration being a single unit of time. In order to represent the channel data-production interval with the production coefficient υ_{pr}, channel and decoder blocks need to perform their operations at particular loop iterations. For simplicity, the time is normalized by a latency equivalent to a single SC pass. For example, with a production coefficient $\upsilon_{\text{pr}} = 1.125$ and a decoding latency τ_{sc} of 8 units, the channel generates data every $\tau_{\text{ch}} = \upsilon_{\text{pr}} \cdot \tau_{\text{sc}} = 9$ time units (6.8).

Before simulating our system model, we run simulations of the DSCF-1 decoder within the ideal system, i.e., with the initial maximum number of trials as the only decoding latency constraint. To illustrate the functionality of our proposed algorithm, the random blocks of data were encoded with a $(1024, 512 + 16)$ polar code. The polynomial $z^{16} + z^{15} + z^2 + 1$ (Stimming *et al.*, 2015) generates $r = 16$ CRC code. The polar code is constructed for an approximate design E_b/N_0 of 2.365 dB with the construction method proposed by Tal & Vardy (2013). The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating $C = 10^6$ codewords for the channel E_b/N_0 ranging from 1.75 dB to 2.5 dB. The maximum number of trials of DSCF decoder is set to $T_{\text{max}} = 11$, and the bit-flipping candidates are defined according to the metric calculated by (6.1). For each codeword, the required number of trials is stored in the list $\boldsymbol{F}_{\text{req}}$. The frame-error flag, indicating whether the word was successfully decoded or not, is stored in the list of frame-error flags $\boldsymbol{E}$. At the end of simulations, the lists $\boldsymbol{F}_{\text{req}}$ and $\boldsymbol{E}$ are stored and then used for further analysis of the system model.

Simulation of our system model (Figure 6.5) is performed using the results obtained from the ideal system simulation. To illustrate our algorithm, the total size of the buffer is fixed to $B_{\text{tot}} = 100$ memory slots. Both codeword-dropping and multi-threshold mechanisms are applied. For the codeword-dropping mechanism, the thresholds $B_1 = 99$ and $T_1 = 0$ are set. For the codeword-dropping mechanism, the thresholds are derived based on the methodology provided in Subsection 6.2.4. Therefore, the set of the buffer-size thresholds is $\boldsymbol{\mathcal{B}} = \{99, 50, 10\}$. The set of corresponding trial-decoding thresholds is $\boldsymbol{T} = \{0, T_{\text{bal}}, T_{\text{bal}} + 1\}$, which varies depending on the specific channel E_b/N_0 and υ_{pr}. For the applied configurations, the sets of three thresholds result in the optimal balance between complexity and error-correction performance.

The system model is illustrated with production coefficients that are close to the bound of 1.0. The values $\upsilon_{\text{pr}} \in \{1.091, 1.11, 1.125, 1.15, 1.2\}$ are considered. Selection of υ_{pr} is motivated by operating close to the bound and demonstrating that the mechanism maintains the FER close to 10^{-2}. At the same time, the mechanism prevents the system model from running into a buffer overflow even with very aggressive channel-production rates. For all simulations, the resulting metrics are analyzed when the buffer is filled with a substantial amount of words, i.e., when the system is at the steady-state, such that the comparison is fair for different values of υ_{pr} and E_b/N_0.

6.2.5.2 Simulation Algorithm

The simulation algorithm of our system model is summarized in Algorithm 7. The algorithm contains a loop, where functions corresponding to each block of the system model are called at each iteration. Each iteration of the loop corresponds to one time unit, that is used as reference to all processes in the system. The function generating the channel data is denoted by GEN_DATA, the function generating the controller signals is GEN_CTRL_SIGS, and the decoder function is DECODE.

Algorithm 7 Simulation algorithm of the system model with the fixed channel production data rate

Inputs:
$C, B_{\text{tot}}, \tau_{\text{ch}}, \tau_{\text{sc}}, \boldsymbol{F}_{\text{req}}, \boldsymbol{E}, \boldsymbol{B}, \boldsymbol{T}$
procedure SIM_SYST_MODEL
  $\boldsymbol{F}_{\text{res}} \leftarrow \{0, 0, \ldots 0\}, \boldsymbol{\chi}_{\text{occ}} \leftarrow \{0, 0, \ldots 0\}$ ▷ Initialize the lists of states for the decoder and buffer
  $buf \leftarrow$ CREATE_BUF (B_{tot}) ▷ Initialize buffer with B_{tot} slots
  $t_{\text{req}} \leftarrow F_{\text{req}}(1)$ ▷ Obtain required number of trials for the first word
  $c \leftarrow 1, i \leftarrow 1$
  while $c \neq C$ **do**
    GEN_DATA (buf, τ_{ch}) ▷ Store the received word to the buffer with delay τ_{ch}
    $S_{\text{stop}} \leftarrow$ GEN_CTRL_SIGS $(\boldsymbol{B}, \boldsymbol{T})$ ▷ Generate decoder stop flag by executing Algorithm 6
    $t_{\text{cur}} \leftarrow$ DECODE $(buf, S_{\text{stop}}, t_{\text{req}}, \tau_{\text{sc}})$
    if $(S_{\text{stop}} == True)$ **then** ▷ If stop signal is generated
      $\boldsymbol{F}_{\text{res}}(c) \leftarrow t_{\text{cur}}$ ▷ Store resulting number of trials for analysis
      $c \leftarrow c + 1$
      $t_{\text{req}} \leftarrow F_{\text{req}}(c)$ ▷ Obtain required number of trials for the next next word
    end if
    $\boldsymbol{\chi}_{\text{occ}}(i) \leftarrow buf.B_{\text{occ}}$ ▷ Store the currently occupied buffer slot for analysis
    $i \leftarrow i + 1$
  end while
  $\boldsymbol{E}' \leftarrow$ CALC_FER_IMPACT $(\boldsymbol{F}_{\text{res}}, \boldsymbol{F}_{\text{req}}, \boldsymbol{E})$ ▷ Generate flags indicating decoding success or failure for each word
  return $(\boldsymbol{\chi}_{\text{occ}}, \boldsymbol{E}')$ ▷ Return the lists to analyze buffer utilization and performance
end procedure

The functions of channel and decoder are passthrough functions with behavior that depends on the state of their internal counters. The function GEN_DATA will add a word to the buffer after every τ_{ch} iteration loops. The function DECODE reads the word from the buffer at every $\tau_{\text{dec}} = t_{\text{req}} \cdot \tau_{\text{sc}}$ iteration loops (6.7), where the required number of trials t_{req} for each decoding word c is obtained from the list $\boldsymbol{F}_{\text{req}}$.

The word counter c is incremented when the flag S_{stop} is raised, i.e., when either one of the thresholds is violated or when the decoder completes decoding according to t_{req}. At the same condition, the final current number of trials is saved to the list of the resulting number of trials $\boldsymbol{F}_{\text{res}}$. The simulation ends when all C decoding words are processed. The number of occupied buffer slots is stored in the list $\boldsymbol{\chi}_{\text{occ}}$ at every loop iteration.

At the end of the simulation, the function CALC_FER_IMPACT calculates the binary list of resulting frame-error flags $\boldsymbol{E}'$ indicating which words were successfully decoded and which were not. This list differs from the list of original

frame-error flags $\boldsymbol{E}$ obtained from the simulation of the ideal system. A decoding error is declared when the ideal system fails to decode the word or when there is an early decoder stopping ($F_{\text{res}}(c) < F_{\text{req}}(c)$).

6.2.5.3 State of the Buffer Over the Course of Simulation

Figure 6.7 depicts the number of used buffer slots over the course of the simulation at E_b/N_0 of 2.25 dB, where the words come from the channel at a fixed rate that corresponds to a production coefficient $\upsilon_{\text{pr}} = 1.125$. The codeword-dropping mechanism is depicted in blue while the multi-threshold is in red. The results in Figure 6.7 show that both mechanisms effectively prevent buffer overflow, i.e., buffer occupied slots never reach $B_{\text{tot}} = 100$ slots.

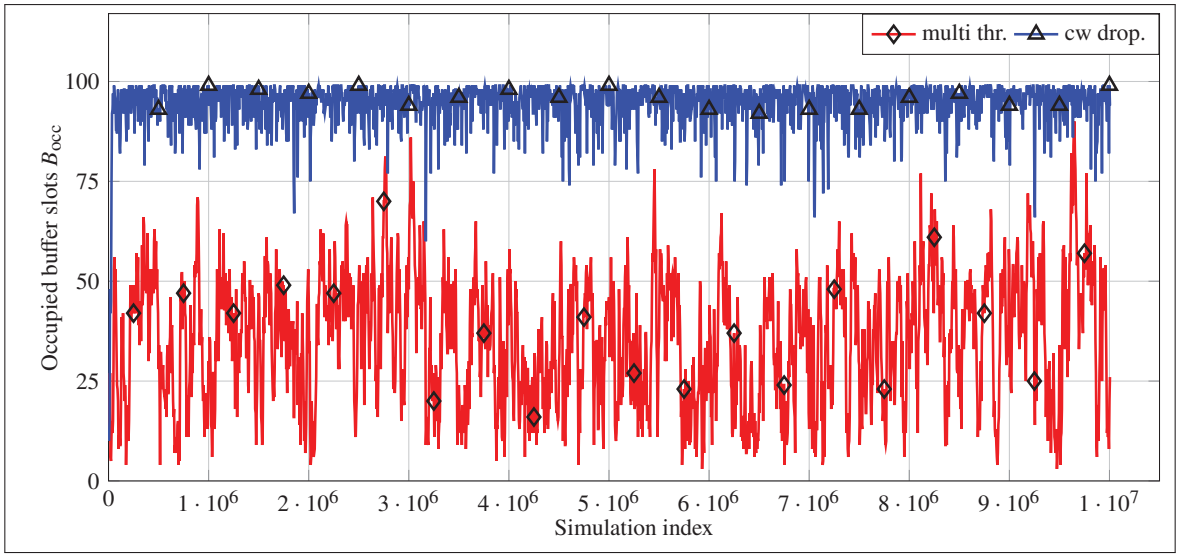


Figure 6.7 Number of occupied buffer slots over the course of a simulation of the codeword-dropping and the multi-threshold mechanisms for E_b/N_0 point of 2.25 dB and $\upsilon_{\text{pr}} = 1.125$

6.2.5.4 Error-correction performance

Figure 6.8 depicts the FER of the model, where the controller implements the codeword-dropping (blue) and multi-threshold mechanisms (red). Simulations are conducted for various values of E_b/N_0, but for a fixed channel-production rate corresponding to $\upsilon_{\text{pr}} = 1.125$. As such, the channel-production interval is close to the delay of a single DSCF-1 trial. The black curve is the ideal performance, which is provided for reference. The results show that at low channel E_b/N_0 both considered control mechanisms experience a degradation of the error-correction performance compared to the ideal case. This gap is reduced as the channel improves; the loss is virtually nonexistent at a E_b/N_0 of 2.375 dB. Across the range, we see that the multi-threshold mechanism either matches or outperforms the codeword-dropping mechanism. At the FER point of 10^{-2}, the DSCF-1 decoder within the ideal

system achieves the FER of 10^{-2} at approximately $E_b/N_0 = 2.25$ dB. The codeword-dropping and the multi-threshold mechanisms show performance losses of approximately 0.1 dB and 0.0625 dB respectively.

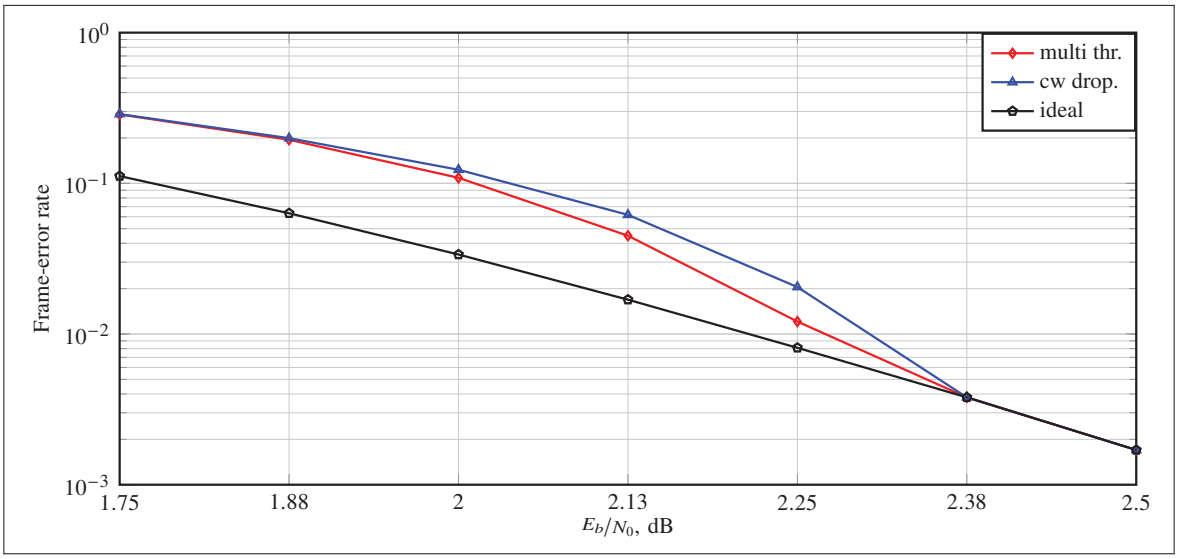


Figure 6.8 FER of the codeword-dropping and the multi-threshold mechanisms for the range of E_b/N_0 and $\upsilon_{pr} = 1.125$

Figure 6.9 shows the FER of the model for both mechanisms for a fixed $E_b/N_0 = 2.25$ dB and various υ_{pr}. The DSCF-1 decoder with $T_{max} = 11$ is applied to both mechanisms. Although it cannot be sustained, the ideal performances for various T_{max} values are shown as horizontal lines. The results indicate that at lower production coefficients, both codeword-dropping and multi-threshold mechanisms have an error-correction loss compared to the ideal case with $T_{max} = 11$. At the $\upsilon_{pr} = 1.091$, the FER is even worse than the ideal case with $T_{max} = 3$. The gap reduces as the production coefficient increases. The multi-threshold mechanism fares better than codeword dropping across the whole range. At the $\upsilon_{pr} = 1.125$, the FER of the multi-threshold mechanism reaches the ideal case for $T_{max} = 5$. Both mechanisms match the ideal FER for $T_{max} = 11$ at $\upsilon_{pr} = 1.2$.

6.2.5.5 Conclusion on Flip Decoding Under Fixed Channel-Production Rate

This section has described the proposed multi-threshold mechanism, which adjusts the execution time of the flip decoder in real time, allowing it to sustain operation without buffer overflow. The proposed mechanism is implemented in the controller of the system model, which contains the channel, buffer, controller, and decoder. This system model allows the flip decoder to operate with a fixed channel-production rate, meaning that incoming data blocks are delivered from the channel at a fixed time interval. When applied to DSCF-1 decoder, the proposed multi-threshold mechanism allows to operate in a system with a fixed channel-production rate approximately 1.13 times lower than a single decoding trial while preventing the buffer from overflowing. Compared to the ideal

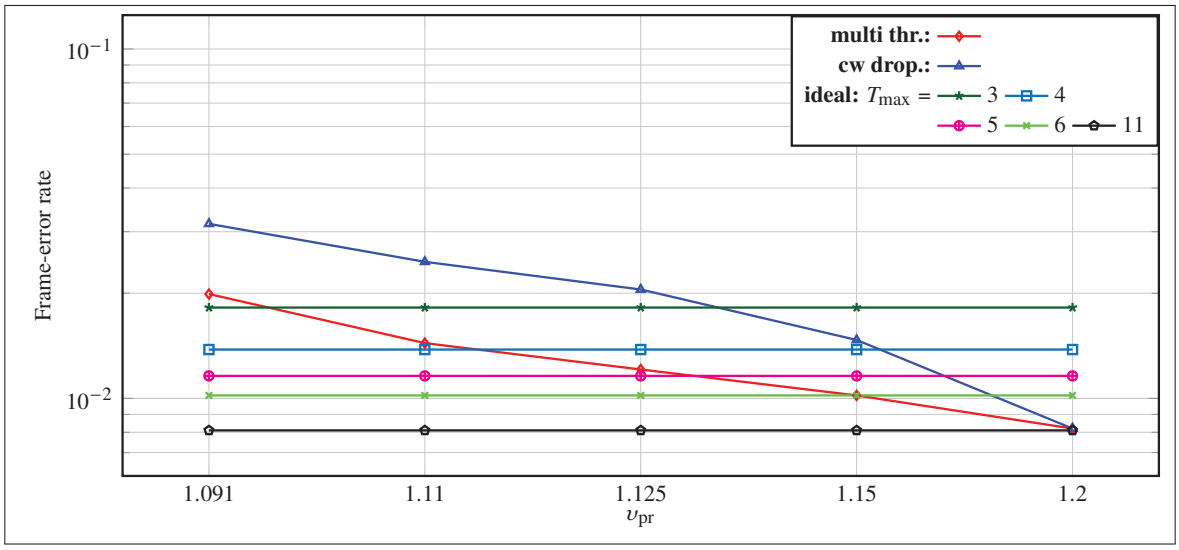


Figure 6.9 FER of the codeword-dropping and the multi-threshold mechanisms for the range of the production coefficient υ_{pr} and E_b/N_0 of 2.25 dB

scenario, the proposed mechanism has an error-correction loss 0.06 dB at the target FER of 10^{-2}. The ideal scenario corresponds to the unsustainable case, as the buffer quickly overflows.

6.3 Conclusion

This chapter described two proposed execution-time reduction mechanisms for flip decoders based on SCF. The reduction mechanisms are adapted to specific channel conditions.

First, our proposed early-stopping mechanism is presented. The mechanism attempts to distinguish undecodable codewords from decodable ones by using the proposed early-stopping metric with the metric threshold, both of which are contributions of this work. The metric threshold is found by simulation under specific channel conditions. Based on that metric, a codeword may be classified as likely undecodable, in which case the decoder attempts a reduced maximum number of trials, much smaller than the initial maximal number of trials. After those trials, if decoding is not successful, the decoder stops. When applied to DSCF-1 decoder, the proposed early-stopping mechanism achieves the reduction of the average execution time by around 22% and execution-time variance by approximately 45%. These reductions resulted in a minor error-correction loss of approximately 0.05 dB at the target FER of 10^{-2}.

Second, we described our proposed multi-threshold mechanism that restrains the delay of a flip decoder depending on the state of the buffer to avoid overflowing. This mechanism is implemented in the system where the channel produces data at a fixed rate, which closely approaches that of a single decoding trial. When applied to DSCF-1

decoder, our proposed multi-threshold mechanism allows to operate in a system with a fixed channel-production rate approximately 1.13 times lower than the rate associated with a single decoding trial, while preventing buffer overflow. Compared to the ideal scenario, our proposed mechanism has a minor error-correction loss of approximately 0.06 dB at the target FER of 10^{-2}. The ideal scenario, where the decoder operates without constraints, corresponds to the unsustainable case, as the buffer quickly overflows.

The two mechanisms described in this chapter are compatible with each other. When combined, the multi-threshold mechanism will restrain the maximum number of trials of the flip decoder with the early-stopping mechanism to prevent buffer overflow. The average execution time, reduced by applying the early-stopping mechanism, will also decrease the balanced number of trials. As a result, the channel-production rate much closer to a single decoding trial may be achieved compared to the standard flip decoder. However, preliminary results have indicated only a small increase in this rate. Furthermore, as both mechanisms negatively affect the error-correction performance, their combined impact becomes significant. Therefore, combining these two mechanisms may lead to more negative than positive effects.

CONCLUSION AND RECOMMENDATIONS

This PhD study contributes to the design of energy-efficient flip decoders for polar codes. In particular, the focus is on mechanisms that improve execution-time characteristics of the flip decoders with minimal to no impact on error-correction performance. The proposed mechanisms require only a small amount of additional hardware and computational resources, much less than the resources of the original decoder.

Summary of Contributions

1. Simplified and Generalized Restart Mechanisms for Flip Decoders

Chapter 3 presented the proposed simplified restart mechanism (SRM) and generalized restart mechanism (GRM) for the flip decoders based on SCF. Both mechanisms can reduce the average execution time of flip decoders by skipping parts of tree traversal, which do not negatively affect the error-correction performance. In the SRM, a conditional restart is performed from the second half of the codeword if that is where the bit-flipping candidate is located.

The proposed GRM is an extension of the SRM. The parts of the tree in the GRM are skipped to estimate a bit-flipping candidate and all of the prior bits of each additional decoding trial. The decoding tree is traversed from its root along the restart path, enabling a direct estimation of the restart bit. To perform a restart, the partial-sum (PS) bits are restored from the bit estimates that are stored in memory after the initial unsuccessful decoding attempt.

The SRM and GRM are effective and can be used with any variation of flip decoders. The proposed SRM and GRM require only a small amount of additional memory, which is a fractional part of the memory of the full decoder. Therefore, the proposed mechanisms result in efficient hardware implementations of flip decoders for polar codes.

Impact: The proposed mechanisms in Chapter 3 contribute to improving the execution-time characteristics of flip decoders based on SCF. The SRM reduced the average execution time by 7% to 30% when applied to DSCF-3 decoder. This decoder achieves the error-correction performance of the state-of-the-art SCL decoder. The GRM reduced the average execution time of the DSCF-3 decoder by 26% to 60%. The PS restoration required by the GRM is shown to be made by low-complexity encoding operations. It can be noted that the execution-time reduction of the SRM and GRM resulted in approximately 4% of additional memory.

2. Modified Generalized Restart Mechanism for Fast Decoding Techniques

Chapter 4 presented the modified GRM to flip decoders with fast decoding techniques. Fast decoding techniques, such as Fast Simplified SC (Fast-SSC) and latency-reducing technique (LRT), are existing techniques that reduce the execution time by recognizing special nodes that can be fast decoded. These decoders substantially reduce the execution-time characteristics when applied to flip decoders, also known as Fast-SCF and SCF with LRT. This study shows that the proposed GRM can be combined with flip decoders where Fast-SSC and LRT are used as baseline decoding algorithms.

Impact: The proposed mechanism in Chapter 4 contributes to improving the execution time characteristics of flip decoders with state-of-the-art fast decoding techniques. The modified GRM reduced the average execution time by 15% to 22% when applied to Fast-DSCF-3 decoder. This reduction is in addition to the reduction already attained by fast decoding. In contrast to the original decoding, the modified GRM requires approximately 4% of additional memory.

3. Restart Mechanisms for List-Flip Decoders

Chapter 5 described the proposed restart mechanisms for the list-flip decoders based on SCLF. This chapter explained the challenges of applying the previously proposed GRM to the list-flip decoder. In the GRM, the part of the tree to estimate a path-flipping candidate and all the previous bits is skipped for each additional decoding trial. To perform a restart, the state memory for each information bit must be stored after the initial SCL pass. However, this resulted in a significant increase in memory usage that exceeds the total memory of the standard list-flip decoder.

The LLRM is proposed to overcome the issue of memory overhead. In the LLRM, an additional trial is resumed from a limited set of restart locations. The state memories of only these locations must be stored, which significantly reduces the memory requirements. Alongside this, three design methods were proposed to determine these restart locations. The first two methods are based on code construction, where the locations equally divide the codeword or the information set of the code. The third method is our proposed probability-based method, where the locations equally divide a codeword in a way that each segment has an equal path-flipping probability distribution.

The proposed GRM and LLRM can reduce the average execution time of list-flip decoders without any negative effect on error-correction performance. The probability-based method to design restart locations in LLRM aims to maximize the execution-time reduction while minimizing the memory overhead.

Impact: The proposed LLRM in Chapter 5 contributes to improving the execution-time characteristics of list-flip decoders. The LLRM reduced the average execution time by 10% to 40% when applied to DSCLF-3 decoder. This decoder provides the strongest error-correction performance compared to other list-flip decoders. Similarly, the GRM reduced the average execution time by 13% to 43%. The LLRM resulted in approximately 2% memory overhead, whereas the GRM incurred 170% to 1500% memory overhead. According to the results, the proposed LLRM is the more appropriate algorithm for hardware implementations of list-flip decoders.

4. Early-Termination Mechanisms for Flip Decoders

Chapter 6 described the two proposed execution-time reduction mechanisms for flip decoders based on SCF. The first is the early-stopping mechanism and the second is the multi-threshold mechanism. The central idea of these mechanisms is early termination depending on specific channel conditions or input data rates.

The proposed early-stopping mechanism attempts to differentiate undecodable codewords from decodable ones using the proposed early-stopping metric with a metric threshold. If the metric suggests a codeword is likely undecodable, the decoder attempts a reduced maximum number of trials, much smaller than the initial maximum number of trials. However, if decoding is unsuccessful after these reduced trials, the decoder stops.

The proposed multi-threshold mechanism restrains the delay of a flip decoder depending on the state of the buffer to avoid overflowing. This mechanism is implemented in the system where the channel produces data at a fixed rate, which closely approaches that of a single decoding trial.

Impact: The mechanisms proposed in Chapter 6 contribute to improving the execution-time characteristics of flip decoders based on SCF. The early-stopping mechanism reduced the average execution time of the DSCF-1 decoder by approximately 22% and execution-time variance by approximately 45%. This resulted in a minor error-correction loss of approximately 0.05 dB. The multi-threshold mechanism allowed to operate in a system with a fixed channel-production rate approximately 1.13 times lower than the rate associated with a single decoding trial while preventing buffer overflow. Compared to the ideal scenario, our proposed mechanism resulted in a minor error-correction loss of approximately 0.06 dB at the target FER of 10^{-2}. The ideal scenario, where the decoder operates without constraints, corresponds to the unsustainable case, as the buffer quickly overflows.

Academic Achievements

The following articles were produced during the PhD study:

- I. Sagitov and P. Giard, "An Early-Stopping Mechanism for DSCF Decoding of Polar Codes", *IEEE Workshop on Signal Processing Systems (SiPS),* 2020, Coimbra, Portugal.
- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Successive-Cancellation Flip Decoding of Polar Codes with a Simplified Restart Mechanism." *IEEE Wireless Communications and Networking Conference (WCNC),* 2023, Glasgow, United Kingdom.
- C. Pillet, I. Sagitov, G. Domer, and P. Giard, "Partitioned Successive-Cancellation List Flip Decoding of Polar Codes." *IEEE Workshop on Signal Processing Systems (SiPS),* 2024, Cambridge, USA.
- I. Sagitov, C. Pillet, and P. Giard, "Successive-Cancellation Flip Decoding of Polar Codes Under Fixed Channel-Production Rate." *arXiv preprint arXiv:2409.03051,* 2024.
- C. Pillet, I. Sagitov, D. Deslandes, and P. Giard, "Successive-Cancellation Flip and Perturbation Decoder of Polar Codes." *IEEE Wireless Communications and Networking Conference (WCNC),* 2025, Milan, Italy (*under review*).
- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Generalized Restart Mechanism for Successive-Cancellation Flip Decoding of Polar Codes." *J. Signal Process. Syst.,* 2024 (*under review*).
- I. Sagitov, C. Pillet, A. Balatsoukas-Stimming, and P. Giard, "Restart Mechanisms for the Successive-Cancellation List-Flip Decoding of Polar Codes." *J. Signal Process. Syst.,* 2024 (*under review*).

Limitations and Future Work

This doctoral study contributes to the design of energy-efficient flip decoders for polar codes, based on the SCF, SCLF algorithms, and their variations. This section discusses the limitations of the proposed mechanisms for flip decoders and the future research directions.

Execution-Time Reduction for High-Rate Codes: The proposed restart mechanisms – SRM, GRM, and LLRM – improve execution-time characteristics of the flip decoders. These mechanisms have no negative effect on the original error-correction performance and require only a small amount of additional memory, which is a fraction of the total decoder memory. However, the results show that the proposed restart mechanisms perform more effectively when used with low-rate polar codes. This can be explained by the distribution of the bit-flipping locations, which indicates a predominant placement of the locations at the RHS of the codeword. Thus, the restart mechanisms perform the simplified restart from the RHS more frequently, skipping larger parts of the decoding tree

and effectively reducing execution time. The GRM and LLRM provide a larger reduction than the SRM for polar codes with medium-high rates, although the effectiveness is still better for low-rate codes. Therefore, improving the effectiveness of execution-time reduction for higher-rate codes can be a future study direction.

Analytical Generation of the Set of Restart Locations for LLRM: The proposed LLRM allows to restart the additional trials of a list-flip decoder by using a small set of restart locations. These locations have to be carefully selected to achieve the execution-time reduction of the GRM. Compared to the other two proposed methods, the probability-based method reduces average execution time the most and has the smallest memory overhead. However, this design requires the probability-mass function (PMF) of path-flipping locations, which is obtained by prior simulations. Additionally, the simulations must be re-run when the code length or rate changes. This creates unwanted complexity in practical applications. Instead, the restart locations could be derived from the pre-calculated sequence, which is unique for code of a certain length. The same motivation led to the construction of the universal reliability sequence in 5G, which is used to assign information and frozen bits in a codeword. Therefore, this sequence for restart locations in LLRM is another potential future research direction.

Improvements for Early-Stopping Mechanisms: The proposed early-termination mechanisms restrain the maximum number of trials using thresholds derived from prior simulations. These simulations must be re-run when the code length or rate changes and when channel conditions change. This creates additional complexity in practical applications. Instead, it would be advantageous to derive these thresholds using low-complexity offline methods or to generate a universal sequence of thresholds. This can be an area for further investigation. The proposed early-termination mechanisms result in a minor error-correction loss but greatly improve the execution-time characteristics. Nevertheless, it would be better to further minimize this error-correction gap. This can be further investigated in future research works.

Integration of Execution-Time Reduction Mechanisms to Partitioned Flip Decoders: In their study, Ercan *et al.* (2018) have proposed another variation of SCF, PSCF decoding. In PSCF, a codeword is divided into partitions. Each partition has its own CRC, and the SCF decoding is applied to each partition separately. This technique allows to potentially exit decoding trials earlier without decoding the full codeword. Therefore, the average execution time is reduced. Also, the PSCF decoder improves the error-correction performance compared to the SCF decoder. In our recently published article of Pillet, Sagitov, Domer & Giard (2024), we proposed the partitioned SCLF (PSCLF) decoding. In the PSCLF algorithm, a code is divided into partitions. Each partition is decoded with a CRC-aided SCLF decoder. As a result, the PSCLF improves the error-correction performance and reduces the average execution time compared to the SCLF decoder. The restart mechanisms that are proposed in this thesis

can also be applied to PSCF and PSCLF decoders. As a result, an additional reduction in the average execution time can be achieved. However, for the PSCLF, the memory requirements of the restart mechanism have to be carefully analyzed. The partitions at the RHS of the codeword that store path information for each restart location will require a large memory size. Therefore, in future research work, the proposed restart mechanisms can be applied to the PSCF and PSCLF decoders.

Hardware Implementation: The architectural model applied in this research work is based on estimations of a realistic hardware implementation. By using this model, we estimate the execution time and memory requirements of the flip decoders. These metrics are used to analyze the impact of our proposed mechanisms. However, the actual hardware implementation will allow us to obtain precise estimates of our algorithm modifications to the chip area, energy efficiency, latency, and decoding throughput. Also, there are no implementations of list-flip decoders reported in the literature. Therefore, hardware implementation of flip and list-flip decoders with the restart mechanism, along with a comparative analysis of existing works, can be considered a potential research direction.

APPENDIX I

ADDITIONAL ANALYSIS FOR LIST-FLIP DECODERS WITH PROPOSED MECHANISMS

This appendix provides the simulation results for SCLF and DSCLF decoders. Chapter 5 of this thesis presents results for the list-flip decoders with the list size $L = 2$. In this Appendix, the results and discussion are provided for $L = 4$. The results demonstrate the effectiveness of the proposed LLRM to the list-flip decoders for different list sizes. The notation SCL-L is used in this Appendix to indicate the list size of the SCL decoder.

Simulation Setup

The effects of the proposed LLRM to list-flip decoders are observed through the simulations. A simulation setup similar to that described in Subsection 2.4.1 of Chapter 2 is applied. The 5G polar codes (3GPP, 2018) of length $N = 1024$ and a rate $R = 1/2$ is used. The number of processing elements is set to $P = 64$, as recommended by Leroux *et al.* (2013) for a code with $N = 1024$. The BPSK modulation is used over an AWGN channel. Simulations are conducted by generating a minimum of $C = 2 \cdot 10^5$ codewords or continuing until 2000 frame errors are observed. For all simulations, the number of restart locations for the LLRM is set to $Y = 4$.

The DSCLF-2 and DSCLF-3 decoders are analyzed with the list sizes of $L = 2$ and $L = 4$. The values for the maximum number of trials $T_{\max}$ are selected to closely achieve performance of the SCL decoder, as discussed in Subsection 2.4.1 of Chapter 2. For the DSCLF-2 decoder, $T_{\max} = 51$ is set, whereas for the DSCLF-3, $T_{\max} = 301$ is set. The metric in DSCLF decoder is calculated according to (2.5) and with the step-approximation function (2.6).

According to simulation results in Figure 2.7 of Chapter 2, the DSCLF-3 decoder with $L = 2$ and DSCLF-2 with $L = 4$ achieve the same error-correction performance as the SCL-16. Also, the DSCLF-3 decoder with $L = 4$ achieves the same performance as the SCL-32 decoder.

Memory Estimates and Average Execution Time

Memory requirements are estimated by (5.3)–(5.9) for the list-flip decoders with LLRM and GRM. The blocks of $\boldsymbol{\alpha}_{\text{ch}}$, $\boldsymbol{\alpha}_{\text{int}}$ and $\boldsymbol{\mathcal{M}}_{\text{flip}}$ are quantized by $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$, $Q_{\text{flip}} = 7$ bits according to Ercan *et al.* (2020a). All memory blocks storing the path metrics **PM** are quantized by $Q_{\text{pm}} = 8$ bits according to Hashemi *et al.* (2017).

Table-A I-1 provides the memory overhead, expressed percent, induced by embedding the GRM and LLRM to the DSCLF-2 and DSCLF-3 decoders. For the LLRM, the restart locations $\boldsymbol{\Upsilon}_{\text{prob}}$, $\boldsymbol{\Upsilon}_{\text{divK}}$ and $\boldsymbol{\Upsilon}_{\text{divN}}$ are applied. In addition to overhead values, memory estimates of the original decoders in bits Λ_{SCLF} (5.4) are also provided. The results show that the original decoder memory Λ_{SCLF} highly changes by increasing L. However, the overhead

of the proposed LLRM is almost identical with varying L. Thus, applying the LLRM to DSCLF-2 with $L = 2$ results in $\{2.16\%, 10.03\%, 5.98\%\}$ overhead, while applying the LLRM to DSCLF-2 with $L = 4$ results in $\{2.63\%, 10.35\%, 6.14\%\}$ overhead. The overhead of applying the GRM is also similar when L changes. However, this overhead is enormous regardless of L, which makes the GRM unfeasible for the list-flip decoders.

Table-A I-1 Memory estimates and overhead for DSCLF-2 and DSCLF-3 decoders with $L \in \{2, 4\}$ with LLRM, with GRM and original decoders. All results are for code rate $R = 1/2$

Decoders	L	$T_{\max}$	Λ_{SCLF} (5.4)	Overh. w. LLRM			Overh. w. GRM
				Υ_{prob}	Υ_{divK}	Υ_{divN}	
			bits	%			%
DSCLF-2	2	51	33110	2.16	10.03	5.98	853.00
DSCLF-3	2	301	42860	1.38	7.75	4.62	658.90
DSCLF-2	4	51	63846	2.63	10.35	6.14	884.57
DSCLF-3	4	301	73596	1.62	9.00	5.33	767.38

Table-A I-2 provides numerical results of the average execution-time reduction $\Delta\overline{\mathcal{L}}_{\text{lflip}}$ estimated by (5.24) for the DSCLF-2 and DSCLF-3 decoders with $L = 2$ and $L = 4$. The reductions are provided by applying the GRM and LLRM to the original decoders at the FER of 10^{-2}, in percent. The results show that reductions are almost identical when the list size L changes. Thus, applying the LLRM to DSCLF-2 with $L = 2$ results in reductions of $\{11.63\%, 9.15\%, 9.73\%\}$, while applying the LLRM to DSCLF-2 with $L = 4$ results in reductions of $\{19.23\%, 14.00\%, 15.10\%\}$. The results also show that higher reduction is achieved for the DSCLF-3 decoder. This decoder also provides stronger error-correction performance compared to the DSCLF-2 with the same L but has a higher average execution time.

Table-A I-2 Reduction $\Delta\overline{\mathcal{L}}_{\text{lflip}}$ (5.24) in % by applying the LLRM and GRM to DSCLF-ω decoders with $L \in \{2, 4\}$ at FER 10^{-2}

Decoders	L	$T_{\max}$	E_b/N_0 (dB)	$\Delta\overline{\mathcal{L}}_{\text{lflip}}$, GRM	$\Delta\overline{\mathcal{L}}_{\text{lflip}}$, LLRM		
					Υ_{prob}	Υ_{divN}	Υ_{divK}
DSCLF-2	2	51	1.750	14.88	11.63	9.15	9.73
DSCLF-3	2	301	1.625	23.70	19.71	13.13	15.15
DSCLF-2	4	51	1.625	14.04	10.79	9.00	9.06
DSCLF-3	4	301	1.500	24.00	19.23	14.00	15.10

Impact of List Sizes and Maximum Number of Trials

Table-A I-3 provides memory estimations and average execution time of the original (non-GRM) decoders. This table presents the total memory and the memories of components of the decoder. The average execution time is presented at the FER of 10^{-2}. The SCL-16 decoder is provided for reference. The results show that DSCLF-3 decoder with $L = 2$ has the smallest total memory compared to other decoders, and the SCL decoder has the largest memory. On the other hand, the average execution time of DSCLF-3 decoder is the largest. Note that the SCL decoder has a constant execution time, which is calculated by (5.13).

Table-A I-3 Memory estimates and average execution time of DSCLF-3, DSCLF-2, and SCL-16 decoders at FER 10^{-2}

Decoders	L	$T_{\max}$	E_b/N_0 dB	Λ_{SCL} bits	Λ_{flip} bits	Mem. tot. bits	Avg. exec. time CCs
DSCLF-3	2	301	1.625	31760	11100	42860	17984
DSCLF-2	4	51	1.625	62496	1350	63846	6072
SCL	16	–	1.625	246912	–	246912	3622

Table-A I-4 provides the memory increase and the reduction of the average execution time for each of the DSCLF-ω decoders compared to the SCL-16 decoder. The results show that compared to the DSCLF-3 decoder, the SCL-16 requires 476.09% more memory, and compared to the DSCLF-2 decoder, it requires 286.73% more memory. On the other hand, compared to the DSCLF-3 decoder with $L = 2$, the SCL-16 results in 79.86% reduction of the average execution time. Compared to the DSCLF-2 decoder with $L = 2$, this reduction is 40.35%. These results align with the statement made in Chapter 5 that choosing the DSCLF-3 decoder with $L = 2$ over the DSCLF-2 decoder with $L = 4$ is preferable. This choice allows us to achieve area- and energy-efficient decoder implementation with strong error-correction performance. However, this decoder has a higher average execution time. Therefore, considering both area and execution-time characteristics, the DSCLF-2 decoder can also be a good choice for hardware implementation.

Table-A I-4 Memory increase and reduction of the average execution time of DSCLF-3 and DSCLF-2 decoders compared to SCL-16 decoder at FER 10^{-2}

Decoders	L	$T_{\max}$	E_b/N_0 dB	Memory incr. %	Avg. exec. time red. %
DSCLF-3	2	301	1.625	476.09	79.86
DSCLF-2	4	51	1.625	286.73	40.35

Conclusion

The results provided in this Appendix show that the proposed LLRM has the same effectiveness for the list-flip decoders with different list sizes L. For the list sizes $L = 2$ and $L = 4$, the DSCLF-3 decoder with the LLRM resulted in the same memory overhead, which is approximately 2% in both cases. The LLRM reduced the average execution time by approximately 19% in both cases. Applying the GRM resulted in a memory overhead of approximately 700% in both cases, making it unfeasible for hardware implementation.

The results also showed that the SCL component has the largest part in the total memory of the list-flip decoders. The DSCLF-3 decoder with $L = 2$ and DSCLF-2 with $L = 4$ achieve the same error-correction performance as the SCL-16 decoder. Compared to the DSCLF-3 decoder, the SCL-16 resulted in approximately 500% memory overhead, and compared to the DSCLF-2 decoder, it resulted in approximately 300% memory overhead. Therefore, choosing the DSCLF-3 decoder with $L = 2$ over the DSCLF-2 decoder with $L = 4$ is preferable to achieve area- and energy-efficient decoder implementation with strong error-correction performance.

LIST OF REFERENCES

3GPP. (2018). *NR; Multiplexing and channel coding* (Report n°TS 38.212). Consulted at http://www.3gpp.org/DynaReport/38-series.htm.

Afisiadis, O., Balatsoukas-Stimming, A. & Burg, A. (2014). A low-complexity improved successive cancellation decoder for polar codes. *Asilomar Conf. on Signals, Syst., and Comput. (ACSSC)*, pp. 2116-2120. doi: 10.1109/ACSSC.2014.7094848.

Alamdar-Yazdi, A. & Kschischang, F. R. (2011). A Simplified Successive-Cancellation Decoder for Polar Codes. *IEEE Commun. Lett.*, 15(12), 1378-1380. doi: 10.1109/LCOMM.2011.101811.111480.

Arıkan, E. (2009). Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. *IEEE Trans. Inf. Theory*, 55(7), 3051-3073. doi: 10.1109/TIT.2009.2021379.

Balatsoukas-Stimming, A., Giard, P. & Burg, A. (2017). Comparison of Polar Decoders with Existing Low-Density Parity-Check and Turbo Decoders. *IEEE Wireless Commun. and Netw. Conf. Workshops (WCNCW)*, pp. 1-6. doi: 10.1109/WCNCW.2017.7919106.

Chandesris, L., Savin, V. & Declercq, D. (2018). Dynamic-SCFlip Decoding of Polar Codes. *IEEE Trans. Commun.*, 66(6), 2333-2345. doi: 10.1109/TCOMM.2018.2793887.

Cheng, F., Liu, A., Zhang, Y. & Ren, J. (2019). Bit-Flip Algorithm for Successive Cancellation List Decoder of Polar Codes. *IEEE Access*, 7, 58346-58352. doi: 10.1109/ACCESS.2019.2914691.

ChenYang, X., Ji, C., YouZhe, F., Chi-ying, T., Jie, J., Hui, S. & Bin, L. (2018). A High-Throughput Architecture of List Successive Cancellation Polar Codes Decoder With Large List Size. *IEEE Trans. Signal Process.*, 66(14), 3859-3874. doi: 10.1109/TSP.2018.2838554.

Condo, C., Hashemi, S. & Gross, W. J. (2017). Efficient bit-channel reliability computation for multi-mode polar code encoders and decoders. *IEEE Int. Workshop on Signal Process. Syst. (SiPS)*, pp. 1-6. doi: 10.1109/SiPS.2017.8109987.

Condo, C., Ercan, F. & Gross, W. J. (2018). Improved successive cancellation flip decoding of polar codes based on error distribution. *IEEE Wireless Commun. and Netw. Conf. Workshops (WCNCW)*. doi: 10.1109/WCNCW.2018.8368991.

Ercan, F., Condo, C. & Hashemi, S. A. (2018). Partitioned Successive-Cancellation Flip Decoding of Polar Codes. *IEEE Int. Conf. on Commun. (ICC)*, pp. 1-6. doi: 10.1109/ICC.2018.8422464.

Ercan, F., Condo, C. & Gross, W. J. (2019). Improved Bit-Flipping Algorithm for Successive Cancellation Decoding of Polar Codes. *IEEE Trans. Commun.*, 67(1), 61-72. doi: 10.1109/TCOMM.2018.2873322.

Ercan, F., Tonnellier, T., Doan, N. & Gross, W. J. (2020a). Practical Dynamic SC-Flip Polar Decoders: Algorithm and Implementation. *IEEE Trans. Signal Process.*, 68, 5441-5456. doi: 10.1109/TSP.2020.3023582.

Ercan, F., Tonnellier, T., Doan, N. & Gross, W. J. (2020b). Simplified Dynamic SC-Flip Polar Decoding. *IEEE Int. Conf. on Acoustics, Speech, and Signal Process. (ICASSP)*, pp. 1733-1737. doi: 10.1109/ICASSP40776.2020.9052925.

Ercan, F., Tonnellier, T. & Gross, W. J. (2020c). Energy-Efficient Hardware Architectures for Fast Polar Decoders. *IEEE Trans. Circuits Syst. I: Reg. Papers*, 67(1), 322-335. doi: 10.1109/TCSI.2019.2942833.

Fossorier, M., Mihaljevic, M. & Imai, H. (1999). Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Trans. Commun.*, 47(5), 673-680. doi: 10.1109/26.768759.

Giard, P. & Burg, A. (2018). Fast-SSC-Flip Decoding of Polar Codes. *IEEE Wireless Commun. and Netw. Conf. Workshops (WCNCW)*, pp. 73-77. doi: 10.1109/WCNCW.2018.8369026.

Giard, P., Balatsoukas-Stimming, A., Müller, T. C., Bonetti, A., Thibeault, C., Gross, W. J., Flatresse, P. & Burg, A. (2017). PolarBear: A 28-nm FD-SOI ASIC for Decoding of Polar Codes. *IEEE J. Emerg. Sel. Topics Circuits Syst.*, 7(4). doi: 10.1109/JETCAS.2017.2745704.

Hashemi, S. A., Condo, C. & Gross, W. J. (2017). Fast and Flexible Successive-Cancellation List Decoders for Polar Codes. *IEEE Trans. Signal Process.*, 65(21), 5756-5769. doi: 10.1109/TSP.2017.2740204.

Ivanov, F., Morishnik, V. & Krouk, E. (2021). Improved generalized successive cancellation list flip decoder of polar codes with fast decoding of special nodes. *Journal of Commun. and Netw.*, 23(6), 417-432. doi: 10.23919/JCN.2021.000038.

Lee, U., Roh, J. H., Hwangbo, C. & Sunwoo, M. H. (2021). A Uniformly Segmented SC-Flip Decoder for Polar Codes with Memory Reduction Methods. *IEEE Int. Symp. on Circuits and Syst. (ISCAS)*. doi: 10.1109/ISCAS51556.2021.9401189.

Leroux, C., Tal, I., Vardy, A. & Gross, W. J. (2011). Hardware architectures for successive cancellation decoding of polar codes. *IEEE Trans. Acoust., Speech, Signal Process.*, 1665-1668. doi: 10.1109/ICASSP.2011.5946819.

Leroux, C., Raymond, A., Sarkis, G. & Gross, W. J. (2013). A Semi-Parallel Successive-Cancellation Decoder for Polar Codes. *IEEE Trans. Signal Process.*, 61(2), 289-299. doi: 10.1109/TSP.2012.2223693.

Mishra, A., Raymond, A., Amaru, L. G., Sarkis, G., Leroux, C., Meinerzhagen, P., Burg, A. & Gross, W. J. (2012). A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS. *2012 IEEE Asian Solid State Circuits Conf. (A-SSCC)*, pp. 205-208. doi: 10.1109/IPEC.2012.6522661.

Mondelli, M., Hassani, S. H. & Urbanke, R. (2017). Construction of polar codes with sublinear complexity. *2017 IEEE Int. Symp. on Inf. Theory (ISIT)*, pp. 1853-1857. doi: 10.1109/ISIT.2017.8006850.

Pan, Y.-H., Wang, C.-H. & Ueng, Y.-L. (2020). Generalized SCL-Flip Decoding of Polar Codes. *IEEE Global Telecommun. Conf. (GLOBECOM)*, pp. 1-6. doi: 10.1109/GLOBECOM42002.2020.9321982.

Pillet, C., Sagitov, I., Domer, G. & Giard, P. (2024). Partitioned Successive-Cancellation List Flip Decoding of Polar Codes. *IEEE Int. Workshop on Signal Process. Syst. (SiPS)*. doi: arXiv:2409.00266.

Sagitov, I. & Giard, P. (2020). An Early-Stopping Mechanism for DSCF Decoding of Polar Codes. *IEEE Int. Workshop on Signal Process. Syst. (SiPS)*, pp. 1-6. doi: 10.1109/SiPS50750.2020.9195213.

Sagitov, I., Pillet, C., Balatsoukas-Stimming, A. & Giard, P. (2023). Successive-Cancellation Flip Decoding of Polar Code with a Simplified Restart Mechanism. *IEEE Wireless Commun. and Netw. Conf. (WCNC)*, pp. 1-6. doi: 10.1109/WCNC55385.2023.10119097.

Sarkis, G., Giard, P., Vardy, A., Thibeault, C. & Gross, W. J. (2014). Fast Polar Decoders: Algorithm and Implementation. *IEEE J. Sel. Areas Commun.*, 32(5), 946-957. doi: 10.1109/JSAC.2014.140514.

Shen, Y., Balatsoukas-Stimming, A., You, X., Zhang, C. & Burg, A. P. (2022). Dynamic SCL Decoder With Path-Flipping for 5G Polar Codes. *IEEE Wireless Commun. Lett.*, 11(2), 391-395. doi: 10.1109/LWC.2021.3129470.

Stillmaker, A. & Baas, B. (2017). Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm. *Integration*, 58, 74-81. doi: 10.1016/j.vlsi.2017.02.002.

Stimming, A., Parizi, M. & Burg, A. (2014). LLR-Based Successive Cancellation List Decoding of Polar Codes. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3903-3907. doi: 10.1109/ICASSP.2014.6854333.

Stimming, A., Parizi, M. & Burg, A. (2015). LLR-Based Successive Cancellation List Decoding of Polar Codes. *IEEE Trans. Signal Process.*, 63(19), 5165-5179. doi: 10.1109/TSP.2015.2439211.

Tal, I. & Vardy, A. (2011). List decoding of polar codes. *IEEE Int. Symp. on Inf. Theory (ISIT)*, pp. 1-5. doi: 10.1109/ISIT.2011.6033904.

Tal, I. & Vardy, A. (2013). How to Construct Polar Codes. *IEEE Trans. Inf. Theory*, 59(10), 6562-6582. doi: 10.1109/TIT.2013.2272694.

Tal, I. & Vardy, A. (2015). List decoding of polar codes. *IEEE Trans. Inf. Theory*, 61(5), 2213-2226. doi: 10.1109/TIT.2015.2410251.

Trifonov, P. (2012). Efficient Design and Decoding of Polar Codes. *IEEE Trans. Commun.*, 60(11), 3221-3227. doi: 10.1109/TCOMM.2012.081512.110872.

Yongrun, Y., Zhiwen, P., Nan, L. & Xiaohu, Y. (2018). Successive Cancellation List Bit-flip Decoder for Polar Codes. *2018 10th Int. Conf. on Wireless Commun. and Signal Process. (WCSP)*, pp. 1-6. doi: 10.1109/WCSP.2018.8555688.

Zhou, Y., Miao, Y., Liu, M. & Liu, W. (2021). Evaluation of 5G channel coding technology: for the mMTC scenario. *6th Int. Symp. on Comp. and Inf. Process. Technnology (ISCIPT)*, pp. 567-572. doi: 10.1109/ISCIPT53667.2021.00121.