# Reduced Complexity Projection-Aggregation Decoders for Reed-Muller Codes

Jiajie Li

Department of Electrical & Computer Engineering
McGill University
Montréal, Canada

February 2022

---

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Six Quatrains Done Playfully-II, by Du Fu

Yang, Wang, Lu, and Luo, the style of those times, "trivial in composition," the sneering does not cease.

Your persons and your names both will perish, yet it will not stop the rivers flowing for all time.

*[Translated by Stephen Owen]*

# Abstract

Reed-Muller (RM) codes are classical linear codes that share a closed relation with the capacity achieving polar codes. The RM code attracts attention recently due to its favorable characteristics, such as the universal code construction, capacity-achieving capability on binary memoryless symmetric channels, polarization effect, and the promising performance brought by the minimum distance under the maximum likelihood (ML) decoding.

The recursive projection-aggregation (RPA) decoder is a recently proposed near ML decoder for RM codes with low rates and short code lengths. However, the high computational complexity of RPA decoding is a major bottleneck for using RPA in applications that have a limited resource and energy budget. In this work, syndrome-based early stopping techniques as well as a scheduling scheme are proposed for the RPA decoder, which help in reducing the computational complexity while keeping similar decoding performance. Comparing to the baseline RPA decoder, the proposed techniques result in a $69-98\%$ reduction in the average computational complexity for a target frame error rate of $10^{-5}$. Additionally, this work introduces hardware-friendly approximation functions to replace the RPA's computationally expensive transcendental projection function.

The collapsed projection-aggregation (CPA) decoder reduces the computational complexity of the RPA decoder by removing the recursive structure. From simulations, the CPA decoder has similar error-correction performance as the RPA decoder, when decoding RM $(7, 3)$ and $(8, 2)$ codes. The computational complexity can be further reduced by only selecting a subset of sub-spaces, which is achieved by pruning CPA decoders. In this work, optimization methods are proposed to find the pruned CPA (PCPA) decoder with small performance loss. Furthermore, the min-sum approximation is used to replace non-linear projection and aggregation functions, and a simplified list decoder based on the syndrome check is proposed. Under the same complexity, the optimized PCPA decoder has less performance loss than randomly constructed PCPA decoders in most case. The min-sum approximation incurs less than 0.15 dB performance loss at a target frame error rate of $10^{-4}$, and the simplified list decoder does not have noticeable performance loss.

## Résumé

Les codes Reed-Muller (RM) sont des codes linéaires classiques qui partagent une relation étroite avec la capacité à réaliser des codes polaires. Le code RM attire l'attention récemment en raison de ses caractéristiques favorables, telles que la construction de code universel, la capacité d'atteindre la capacité sur les canaux symétriques binaires sans mémoire, l'effet de polarisation et les performances prometteuses apportées par la distance minimale sous le décodage du maximum de vraisemblance (ML).

Le décodeur récursif par projection-agrégation (RPA) est un décodeur proche ML récemment proposé pour les codes RM avec de faibles débits et des longueurs de code courtes. Cependant, la complexité de calcul élevée du décodage RPA est un goulot d'étranglement majeur pour l'utilisation de RPA dans des applications qui ont un budget de ressources et d'énergie limité. Dans ce travail, des techniques d'arrêt précoce basées sur le syndrome ainsi qu'un schéma d'ordonnancement sont proposés pour le décodeur RPA, ce qui aide à réduire la complexité de calcul tout en conservant des performances de décodage similaires. Par rapport au décodeur RPA de base, les techniques proposées entraînent une réduction de 69 à 98 % de la complexité de calcul moyenne pour un taux d'erreur de trame cible de $10^{-5}$. De plus, ce travail introduit des fonctions d'approximation compatibles avec le matériel pour remplacer la fonction de projection transcendantale coûteuse en calcul du RPA.

Le décodeur de projection-agrégation (CPA) réduit réduit la complexité de calcul du décodeur RPA en supprimant la structure récursive. D'après les simulations, le décodeur CPA a des performances de correction d'erreurs similaires à celles du décodeur RPA, lors du décodage des codes RM (7,3) et (8,2). La complexité de calcul peut être encore réduite en sélectionnant uniquement un sous-ensemble de sous-espaces, ce qui est obtenu en élaguant les décodeurs CPA. Dans ce travail, des méthodes d'optimisation sont proposées pour trouver le décodeur CPA élagué (PCPA) avec une faible perte de performance. De plus, l'approximation min-sum est utilisée pour remplacer les fonctions de projection et d'agrégation non linéaires, et un décodeur de liste simplifié basé sur la vérification du syndrome est proposé. Sous la même complexité, le décodeur PCPA optimisé a moins de perte de performance que les décodeurs PCPA construits de manière aléatoire dans la plupart des cas. L'approximation de la somme minimale entraîne une perte de performances inférieure à 0.15 dB à un taux d'erreur de trame cible de $10^{-4}$, et le décodeur de liste simplifié n'a

pas de perte de performances notable.

## Acknowledgments

Pursing this master degree is truly joyful, and it is also unforgettable due to this pandemic time. I am a truly lucky person because of having a chance to be in the graduate school and receiving many love and support. I would like to take this opportunity to express my sincere gratitude.

I would like to thank my supervisor, Professor Warren J. Gross, for offering me the opportunity of being a part of the ISIP lab and giving immense support during this master degree. Without this opportunity and support, this wonderful journey would not be possible. His brilliant mind, leadership, and charming personality always motivate me to achieve a higher goal in research and strive for a better me along the way.

I would like to thank Dr. Thibaud Tonnellier for helping me to kick start my research and guiding me through things in the ISIP lab. The process of pursuing this master degree would not be so smooth, without your help and support. I would like to thank Dr. Furkan Ercan for his guidance and feedback in the early stage of my research. I still remember how thrilled I was when you taught me how to appreciate the beauty of good research. I would like to thank Dr. Syed Mohsin Abbas for his generous help and patience and countless coffee. Your help and support mean a lot for getting things done in this challenging time. I would like to thank Nghia Doan for his selfless sharing of ideas and insightful discussions. Thank you for bring me to a whole new world in research. I would also like to thank all my colleagues in the ISIP lab for the warm and welcome environment you cultivate. The whole journey would be boring and dull, without all of you.

Many more love and support are from my friends and families. I am appreciated for all the love and support you gave, no matter where you are and how small the portion you give. I am too shy to express my gratitude, when we are talking either face-to-face or online. I would like to use this page to help me say "Thank you!" to all of you.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**5G** 5th Generation

**ADMM** Alternating direction method of multipliers

**AWGN** Additive white Gaussian nose

**BPSK** Binary phase shift key

**CPA** Collapsed projection-aggregation

**CRC** Cyclic rdundancy check

**CP** Cutting-plane

**ECC** Error correction code

**FER** Frame error rate

**FHT** Fast Hadamard transform

**IoT** Internet of things

**LDPC** Low-density parity-check

**LLR** Log-likelihood ratio

**MAP** Maximum-a-posteriori

**MIQP** Mixed-integer quadratic programming

**ML** Maximum likelihood

**NP** Non-deterministic polynomial-time

**PCPA** Pruned collapsed projection-aggregation

**RM** Reed-Muller

**RPA** Recursive projection-aggregation

**RREF** Reduced row echelon form

**SCL** Successive cancellation list

**SRPA** Sparse recursive projection-aggregation

**URLLC** Ultra-Reliable Low-Latency Communication

# Chapter 1

# Introduction

Modern communication systems are realized by the digital logic, and all messages (e.g., sounds, pictures) are interpreted as 0 and 1 in the communication system. To enhance the reliability of the communication system, redundancies are imposed to the binary message. These redundancies help to recover the message from the received noisy sequence, which is called the error correction code (ECC). Shannon's mathematically theory of communication builds the foundation of reliable communications over noise channels, and it shows that there is a limit, channel capacity, of how much information can be reliably transmitted through channels [2]. Many capacity achieving ECCs (e.g., low-density parity-check (LDPC) codes [3], [4], [5] and the polar codes [6]) have been proposed and adopted into the modern communication systems, such as the current 5G communication standard [7].

The Reed-Muller (RM) code is a classical linear block code with a close relation to polar codes [8], the first known class of codes that can asymptotically achieve the channel capacity. RM codes have received significant attention recently due to characteristics such as universal code construction, capacity-achieving capability on the binary erasure channel and the binary symmetric channel [9], [10], [11], polarization effect [12], and the promising performance brought by the minimum distance under the maximum likelihood (ML) decoding [13]. A most recent work posted on the open review platform shows that the RM code achieves capacity on binary memoryless symmetric channels under bitwise maximum-a-posteriori (MAP) decoding [14].

Several decoders for RM codes are available in the literature. Reed's majority vote decoder [15] is the first decoder for RM codes, and it can correct error patterns with

a Hamming weight less than $\frac{d}{2}$ [16], where $d$ is the minimum distance. Fast Hadamard transform (FHT) decoder [17], [18] is an efficient ML decoder for the first-order RM codes. The Sidel'nikov-Pershakov algorithm based decoders [19], [20] and the Dumer's recursive list decoder [21], [22], [23] are RM code decoders designed for second or higher-order RM codes.

For low-rate and short-length RM codes, a recently proposed recursive projection-aggregation (RPA) decoder achieves near ML decoding performance [1]. Moreover, for low-rate and short-length codes, RPA decoder for RM codes outperforms the successive cancellation list (SCL) decoder [24] (with a list size of 32) for polar codes [1]. Furthermore, for RM codes, RPA decoder outperforms the Sidel'nikov-Pershakov algorithm based decoders and the Dumer's recursive list decoder with a list size of 128 [1].

For 5G communication networks, there are emerging applications such as Ultra-Reliable Low-Latency Communication (URLLC), where short data packets and the low frame error rate (FER) ($\leq 10^{-5}$) are in high demand [25]. RM codes with RPA decoding can be a viable option for these applications. Moreover, the RPA decoder can be implemented in parallel to reduce the decoding latency.

However, due to its recursive structure, the RPA decoder has a high computational complexity $\mathcal{O}(n^r \log n)$ [1], where $r$ is the code order and $n$ is the code length. The high computational complexity is a major bottleneck for using the RPA decoder in applications with a limited computation resource and energy budget. A simplified RPA, which reduces the complexity by projecting a codeword to two-dimensional sub-spaces, is proposed by Ye and Abbe [1]. A collapsed projection-aggregation (CPA) decoder is proposed in [26], which projects the received codeword to order 1 RM sub-codes without going through recursions. Sparse RPA (SRPA) decoder [27], a multi-decoder variant of the RPA decoder, greatly reduces the complexity by deploying multiple sparse RPA decoders and each decoder chooses a subset of projections randomly. Cyclic redundancy check (CRC) is used to select the final candidate, but the effective code rate is reduced due to CRC. A pruning metric for the CPA decoder is proposed in [28] to reduce the decoding performance loss due to lowering the decoding complexity by pruning. In [28], pruned CPA (PCPA) decoders with minimal performance loss ($\approx 0.1$ dB) are shown, but construction methods are not reported. Given the current progress in reducing RPA's complexity, there is still room for improvement in reducing the complexity for RPA decoding.

## 1.1 Objectives

The objective of this work is to reduce the computational complexity of the RPA decoder and its variants while limiting the performance degradation.

## 1.2 Summary of Contributions

**Reduction of computational complexity**
Syndrome-based early stopping techniques as well as a scheduling scheme are proposed to reduce the RPA decoder's complexity. Simulation results show that the scheduling scheme can reduce the worst-case complexity by more than 50%, while having negligible performance loss. Simulation results also show that proposed techniques reduce the RPA's average complexity by $69 - 98\%$ at the target FER of $10^{-5}$, while maintaining similar decoding performance.

**Functions approximation**
Hardware-friendly approximation functions are used to replace the computationally expensive non-linear projection and aggregation functions used by the RPA and the CPA decoder. From simulations, approximation functions cause at most 0.2 dB loss of decoding performance at practical FERs ($10^{-4}$ and $10^{-5}$).

**Optimization for PCPA decoder**
Mixed-integer quadratic programming (MIQP) is used to find the PCPA decoder with less performance loss. According to simulation results, under the same complexity, the optimized PCPA decoder has less performance loss than randomly constructed PCPA decoders in most cases.

**Simplified list decoder for RPA and CPA decoders**
A simplified list decoder based on the syndrome check is proposed, which replaces the Reed's decoder in the regular list decoder with the syndrome check. It simplifies the design of the list decoder, and re-use the functional part proposed in this work. The simplified list decoder has negligible performance loss, compared to the regular list decoder.

## 1.3 Thesis Organization

In this thesis, Chapter 2 introduces the background of RM codes, methods to construct the generator matrix, decoding algorithms, and a breief introduction of the MIQP. Introduction of the RPA decoder and its variants is in Chapter 3. Chapter 4 presents the complexity reduction method and the approximation functions proposed in the this work to reduce the computational complexity of the RPA decoder, which is the content in [29]. Chapter 5 shows the optimization methods for PCPA decoders, the function approximation method for the CPA and PCPA decoder, and the simplified list decoder. The content of Chapter 5 is shown in [30].

## 1.4 Related Publications

This thesis results in the following list of papers.

1, J. Li, S. M. Abbas, T. Tonnellier, and W. J. Gross, "Reduced complexity RPA decoder for Reed-Muller codes," in 2021 International Symposium on Topics in Coding (ISTC), 2021. [29]

2, J. Li and W. J. Gross, "Optimization and simplification of PCPA decoder for Reed-Muller codes," unpublished. [30]

# Chapter 2

# Background

This chapter introduces the background of the RM code. Section 2.1 introduces notions used by this thesis. Section 2.2 gives an overview of basic parameters related to the RM code and the encoding process of RM codes. In Section 2.3, two methods of constructing the generator matrix are introduced. In Section 2.4, two basic RM code decoders are introduced, which are components of the RPA decoder and its variants. Section 2.4 also gives a brief review of other decoding algorithms for RM codes. Section 2.5 gives a brief introduction of the MIQP.

## 2.1 Notations

Bold upper-case letters ($\boldsymbol{M}$) denote matrices, while bold lower-case letters ($\boldsymbol{v}$) denote vectors unless explicitly specify. $^\top$ is the transpose operator. $\mathbb{R}$ denotes arbitrary real values. Elements of the vector are denoted by the regular lower-case letters with a subscript index in decimal ($v_3$) or bold lower-case letter with a binary vector ($\boldsymbol{v}(011)$), the binary representation of the decimal index, in round brackets $\boldsymbol{v}(\boldsymbol{z})$, where $\boldsymbol{z}$ is the binary vector. Notations used by this chapter (Chapter 2) might differ from notions used by other chapters, for the ease of explanations.

## 2.2 Overview and Encoding of RM Codes

This work focuses on RM codes for binary-input memoryless channels, and operations are restricted over the binary field $\mathbb{F}_2$. RM codes are defined by parameters $r$, $m$, and $k$, where

$r$ is the order, $n = 2^m$ is the code length, and $k = \sum_{i=0}^{i=r} \binom{m}{i}$ is the code dimension. For a RM$(m, r)$ code, the minimum distance is $d = 2^{m-r}$, and the minimum weight is $2^{m-r}$. The index of the length $n$ RM codeword is denoted as the binary representation of $[0, n-1]$, which is $\boldsymbol{z} \in \mathbb{E} := \mathbb{F}_2^m$, and $\mathbb{F}_2^m$ denotes the set of length $m$ binary vectors.

The encoding process of RM codes is

$$\boldsymbol{c} = \boldsymbol{m}\boldsymbol{G}, \tag{2.1}$$

where $\boldsymbol{c}$ is the codeword, $\boldsymbol{m}$ is the message, and $\boldsymbol{G}$ is the generator matrix of the RM$(m, r)$ code. For RM codes, another naming convention for the generator matrix and the encoding process is using an $m$-variate polynomial $f$ of degree $\leq r$, which is explained in [16] and the following.

A length $m$ binary vector $\boldsymbol{z} = (z_m, ..., z_1) \in \mathbb{F}_2^m$ is used to denote the index of the codeword. $\text{Eval}(f) := (\text{Eval}_{\boldsymbol{z}}(f) : \boldsymbol{z} \in \mathbb{F}_2^m)$ denotes the evaluation of the polynomial $f$ in all $2^m$ possible indexes $\boldsymbol{z}$. The RM$(m, r)$ code is defined by the set of vectors $\text{RM}(m, r) := \{\text{Eval}(f) : f \in \mathbb{F}_2(x_1, ..., x_m), \deg(f) \leq r\}$. Let a subset $A$ be $A \subseteq [m] := \{1, ..., m\}$, and let a shorthand notation for the monomials be $x_A = \prod_{i \in A} x_i$. As $x^n = x$ for arbitrary $n \geq 1$ in binary, so only $x_i$ with degree $\leq 1$ is considered in the polynomial $f$ of the RM code. Thus, all polynomials $f$ with degree $\leq r$ are the linear combination of the following set of monomials $\{x_A : A \subseteq [m], |A| \leq r\}$. Then $\text{Eval}(f)$ can be written as $\text{Eval}(f) = \sum_{A \subset [m], |A| \leq r} m_A x_A$, and the set of monomials can be used to denote the generator matrix for the RM code. An example of the generator matrix of the RM(3,3) code is the following:

$$\boldsymbol{G}(3,3) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_1x_2 \\ x_1x_3 \\ x_2x_3 \\ x_1x_2x_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The parity-check matrix of the RM$(m, r)$ code is the generator matrix of the RM$(m, m -$

$r-1)$ code [16].

## 2.3 Construction of the Generator Matrix for RM Codes

In this section, two recursive construction methods, the Plotkin construction and the Kronecker product, are introduced.

### 2.3.1 Plotkin Construction

For any given polynomial $f$, it can be decomposed into the following form:

$$f(x_1, x_2, ..., x_m) = g(x_1, x_2, ..., x_{m-1}) + x_m h(x_1, x_2, ..., x_{m-1}), \tag{2.2}$$

where $g \in \mathrm{RM}(m-1, r), h \in \mathrm{RM}(m-1, r-1)$. Besides decomposing the polynomial, the evaluation vector $\mathrm{Eval}(f)$ can be decomposed into two parts as well. The evaluation can be divided into two parts, $\mathrm{Eval}^{[z_m=0]}(f)$ and $\mathrm{Eval}^{[z_m=1]}(f)$. Using a $\mathrm{RM}(m=3, r)$ code as an example. $\mathrm{Eval}^{[z_m=0]}(f)$ refers to the code bits in indexes 0, 1, 2, 3, and $\mathrm{Eval}^{[z_m=1]}(f)$ refers to code bits in indexes 4, 5, 6, 7. The sum of two parts in binary is $\mathrm{Eval}^{[/z_m]} = \mathrm{Eval}^{[z_m=0]}(f) + \mathrm{Eval}^{[z_m=1]}(f)$. According to (2.2), $\mathrm{Eval}^{[/z_m]}$ is the evaluation vector of $h(x_1, x_2, ..., x_{m-1})$, and $\mathrm{Eval}^{[z_m=0]}(f)$ is the evaluation vector of $g(x_1, x_2, ..., x_{m-1})$. Thus, $\mathrm{Eval}^{[/z_m]} \in \mathrm{RM}(m-1, r-1)$ and $\mathrm{Eval}^{[z_m=0]}(f) \in \mathrm{RM}(m-1, r)$. This is the Plotkin construction $\boldsymbol{c} = (\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v})$ [16], where $\boldsymbol{c} \in \mathrm{RM}(m, r)$, $\boldsymbol{u} \in \mathrm{RM}(m-1, r)$, $\boldsymbol{v} \in \mathrm{RM}(m-1, r-1)$, and $(,)$ denotes the concatenation in this section. It could be check by $(\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v}) = (\mathrm{Eval}^{[z_m=0]}(f), \mathrm{Eval}^{[z_m=0]}(f) + \mathrm{Eval}^{[z_m=0]}(f) + \mathrm{Eval}^{[z_m=1]}(f)) = (\mathrm{Eval}^{[z_m=0]}(f), \mathrm{Eval}^{[z_m=1]}(f))$. The Plotkin construction infers a recursive construction method for the generator matrix,

$$\boldsymbol{G}(m, r) = \begin{bmatrix} \boldsymbol{G}(m-1, r) & \boldsymbol{G}(m-1, r) \\ \boldsymbol{0} & \boldsymbol{G}(m-1, r-1) \end{bmatrix}, \boldsymbol{G}(1, 1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \tag{2.3}$$

where the first column refers to the polynomial for $\boldsymbol{u}$, and the second column refers to the polynomial for $\boldsymbol{u}+\boldsymbol{v}$. Order 0 RM codes are the repetition code.

Here is an example showing that the addition of two parts of a codeword will produce a lower-order codeword. Considering the generator matrix for the RM(3,2) code,

$$\boldsymbol{G}(3,2) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_1x_2 \\ x_1x_3 \\ x_2x_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

As the coefficients for the polynomial (message bits) are the same for all column vectors, the effect of adding two parts of the codeword can be shown by adding the column vectors inside the generator matrix. The resulting matrix is the generator matrix for the lower-order codeword. Here, the columns vectors are divided into two parts, $z_3 = 0$ and $z_3 = 1$. The partition and the addition are the following:

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \boldsymbol{G}(2,1).$$

From above, the first matrix in the addition contains column vectors where $z_3 = 0$, and it refers to column vectors 0 to 3 in $\boldsymbol{G}(3,2)$ counting from left to right. The second matrix refers to column vectors 4 to 7 counting from left to right. All-zeros rows in the resulting matrix can be removed, as they do not contribute to the encoding process. The resulting matrix is the generator matrix for a $\mathrm{RM}(3-1, 2-1) = \mathrm{RM}(2,1)$ code.

### 2.3.2 Kronecker Product

The generator matrix can also be constructed using the Kronecker product [8],

$$\boldsymbol{G}(m,m) = \boldsymbol{F}^{\otimes m}, \tag{2.4}$$

where $\boldsymbol{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, and $\boldsymbol{F}^{\otimes m}$ denotes the $m$th order Kronecker product of $\boldsymbol{F}$. For a RM$(3, 3)$ code, the generator matrix constructed by the Kronecker product is

$$\boldsymbol{G}(3,3) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

and it is equivalent to the $\boldsymbol{G}(3, 3)$ in Section 2.2 with row permutations. The generator matrix for the RM code after row permutations is still the generator matrix for the RM code.

The generator matrix for RM$(m, r)$ code can be constructed by picking rows with weight $w \geq 2^{m-r}$ from the $\boldsymbol{G}(m, m)$, which is equivalent to place the message bits into positions corresponding to rows with a weight $\geq 2^{m-r}$, and these positions are are denoted as information bit positions. 0 is set to all other bit positions, and they are denoted as frozen bit positions. For example, consider sending an all-ones message, and it is encoded to a RM$(3, 2)$ code. Bit 0 is the frozen bit, and bit 1-7 are the information bits,

$$\boldsymbol{c} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Thus, the encoded codeword is equivalent to

$$\boldsymbol{c} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The RM code and the polar code use the same generator matrix, but following different rules for picking information bit positions. The RM code picks the information bit positions by the hamming weight of the row vector, and polar codes use other reliability measures [8].

## 2.4 Decoders for RM Codes

In this section, the Reed's/majority vote decoder and the fast Hadamard transform (FHT) decoder would be explained using examples from [31] and [32].

### 2.4.1 Reed's/Majority Vote Decoder

For RM codes, the first decoding algorithm is the Reed's/majority vote decoder with a complexity of $\mathcal{O}(n \log^r n)$ [16], and it can correct error patterns with a hamming weight less than half of the minimum distance [15], [16]. It can decode RM codes of arbitrary orders. It firstly decodes the order $r$ message bits, and then proceeds to the order $r-1$ message bits until reaching order 0. In this section, the Reed's decoder is explained by decoding a RM$(4, 2)$ code. The generator matrix of the RM$(4, 2)$ code is

$$\boldsymbol{G}(4,2) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_1x_2 \\ x_1x_3 \\ x_1x_4 \\ x_2x_3 \\ x_2x_4 \\ x_3x_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Let the corresponding message vector be $\boldsymbol{m} = (m_0,m_1,m_2,m_3,m_4,m_{12},m_{13},m_{14},m_{23},m_{24},m_{34})$. $m_{12}$, $m_{13}$, $m_{14}$, $m_{23}$, $m_{24}$, $m_{34}$ are message bits corresponding to order 2 monomials $x_1x_2$, $x_1x_3$, $x_1x_4$, $x_2x_3$, $x_2x_4$, $x_3x_4$. $m_1$, $m_2$, $m_3$, $m_4$ are message bits corresponding to order 1 monomials $x_1$, $x_2$, $x_3$, $x_4$. $m_0$ is the message bit for the order 0 monomial 1.

The decoding process is the following, which is demonstrated using the codeword $\boldsymbol{c}$. The same decoding process is applied on the received codeword $\boldsymbol{y} := \{y(\boldsymbol{z}), \boldsymbol{z} \in F_2^m\}$. From the generator matrix of the RM$(4,2)$ code, it can be seen that

$c_0 = m_0$, $c_1 = m_0 + m_1$, $c_2 = m_0 + m_2$, $c_3 = m_0 + m_1 + m_2 + m_{12}$, $c_4 = m_0 + m_3$,
$c_5 = m_0 + m_1 + m_3 + m_{13}$, $c_6 = m_0 + m_2 + m_3 + m_{23}$,
$c_7 = m_0 + m_1 + m_2 + m_3 + m_{12} + m_{13} + m_{23}$, $c_8 = m_0 + m_4$, $c_9 = m_0 + m_1 + m_4 + m_{14}$,
$c_8 = m_0 + m_4$, $c_9 = m_0 + m_1 + m_4 + m_{14}$, $c_{10} = m_0 + m_2 + m_4 + m_{24}$,
$c_{11} = m_0 + m_1 + m_2 + m_4 + m_{12} + m_{14} + m_{24}$, $c_{12} = m_0 + m_3 + m_4 + m_{34}$,
$c_{13} = m_0 + m_1 + m_3 + m_4 + m_{13} + m_{14} + m_{34}$, $c_{14} = m_0 + m_2 + m_3 + m_4 + m_{23} + m_{24} + m_{34}$,
$c_{15} = m_0 + m_1 + m_2 + m_3 + m_4 + m_{12} + m_{13} + m_{14} + m_{23} + m_{24} + m_{34}$.

Firstly, the order 2 coefficients/message bits are computed. From the above relation, $2^{4-2} = 4$ disjoint linear equations can be constructed to compute order 2 message bits, and then perform the majority vote among those results to recover order 2 coefficients:

$m_{12}^1 = c_0 + c_1 + c_2 + c_3$, $m_{12}^2 = c_4 + c_5 + c_6 + c_7$,
$m_{12}^3 = c_8 + c_9 + c_{10} + c_{11}$, $m_{12}^4 = c_{12} + c_{13} + c_{14} + c_{15}$, $m_{12} = \text{maj}(m_{12}^1, m_{12}^2, m_{12}^3, m_{12}^4)$,
$m_{13}^1 = c_0 + c_1 + c_4 + c_5$, $m_{13}^2 = c_2 + c_3 + c_6 + c_7$,
$m_{13}^3 = c_8 + c_9 + c_{12} + c_{13}$, $m_{13}^4 = c_{10} + c_{11} + c_{14} + c_{15}$, $m_{13} = \text{maj}(m_{13}^1, m_{13}^2, m_{13}^3, m_{13}^4)$,
$m_{14}^1 = c_0 + c_1 + c_8 + c_9$, $m_{14}^2 = c_2 + c_3 + c_{10} + c_{11}$,
$m_{14}^3 = c_4 + c_5 + c_{12} + c_{13}$, $m_{14}^4 = c_6 + c_7 + c_{14} + c_{15}$, $m_{14} = \text{maj}(m_{14}^1, m_{14}^2, m_{14}^3, m_{14}^4)$,
$m_{23}^1 = c_0 + c_2 + c_4 + c_6$, $m_{23}^2 = c_1 + c_3 + c_5 + c_7$,
$m_{23}^3 = c_8 + c_{10} + c_{12} + c_{14}$, $m_{23}^4 = c_9 + c_{11} + c_{13} + c_{15}$, $m_{23} = \text{maj}(m_{23}^1, m_{23}^2, m_{23}^3, m_{23}^4)$,
$m_{24}^1 = c_0 + c_2 + c_8 + c_{10}$, $m_{24}^2 = c_1 + c_3 + c_9 + c_{11}$,
$m_{24}^3 = c_4 + c_6 + c_{12} + c_{14}$, $m_{24}^4 = c_5 + c_7 + c_{13} + c_{15}$, $m_{24} = \text{maj}(m_{24}^1, m_{24}^2, m_{24}^3, m_{24}^4)$,
$m_{34}^1 = c_0 + c_4 + c_8 + c_{12}$, $m_{34}^2 = c_1 + c_5 + c_9 + c_{13}$,
$m_{34}^3 = c_2 + c_6 + c_{10} + c_{14}$, $m_{34}^4 = c_3 + c_7 + c_{11} + c_{15}$, $m_{34} = \text{maj}(m_{34}^1, m_{34}^2, m_{34}^3, m_{34}^4)$,

where maj stands for the majority vote. Those computed message bits are encoded back to the codeword using the order 2 monomials in the generator matrix, and then subtract from the codeword,

$$\boldsymbol{c} = \boldsymbol{c} \bigoplus \begin{bmatrix} m_{12} & m_{13} & m_{14} & m_{23} & m_{24} & m_{34} \end{bmatrix} * \begin{bmatrix} x_1x_2 \\ x_1x_3 \\ x_1x_4 \\ x_2x_3 \\ x_2x_4 \\ x_3x_4 \end{bmatrix},$$

where $\bigoplus$ is the summation over $\mathbb{F}_2$. Then, proceeding to compute the message bits for order 1 monomials, which has $2^{4-1} = 8$ disjoint linear equations.

$m_1^1 = c_0 + c_1$, $m_1^2 = c_2 + c_3$, $m_1^3 = c_4 + c_5$, $m_1^4 = c_6 + c_7$, $m_1^5 = c_8 + c_9$,
$m_2^6 = c_{10} + c_{11}$, $m_1^7 = c_{12} + c_{13}$, $m_1^8 = c_{14} + c_{15}$, $m_1 = \text{maj}(m_1^1, m_1^2, m_1^3, m_1^4, m_1^5, m_1^6, m_1^7, m_1^8)$,
$m_2^1 = c_0 + c_2, m_2^2 = c_1 + c_3, m_2^3 = c_4 + c_6, m_2^4 = c_5 + c_7, m_2^5 = c_8 + c_{10}$,
$m_2^6 = c_9 + c_{11}$, $m_2^7 = c_{12} + c_{14}$, $m_2^8 = c_{13} + c_{15}$, $m_2 = \text{maj}(m_2^1, m_2^2, m_2^3, m_2^4, m_2^5, m_2^6, m_2^7, m_2^8)$,

$m_3^1 = c_0 + c_4$, $m_3^2 = c_1 + c_5$, $m_3^3 = c_2 + c_6$, $m_3^4 = c_3 + c_7$, $m_3^5 = c_8 + c_{12}$,
$m_3^6 = c_9 + c_{13}$, $m_3^7 = c_{10} + c_{14}$, $m_3^8 = c_{11} + c_{15}$, $m_3 = \text{maj}(m_3^1, m_3^2, m_3^3, m_3^4, m_3^5, m_3^6, m_3^7, m_3^8)$,
$m_4^1 = c_0 + c_8$, $m_4^2 = c_1 + c_9$, $m_4^3 = c_2 + c_{10}$, $m_4^4 = c_3 + c_{11}$, $m_4^5 = c_4 + c_{12}$,
$m_4^6 = c_5 + c_{13}$, $m_4^7 = c_6 + c_{14}$, $m_4^8 = c_7 + c_{15}$, $m_4 = \text{maj}(m_4^1, m_4^2, m_4^3, m_4^4, m_4^5, m_4^6, m_4^7, m_4^8)$.

The order 1 message bits are encoded back to a codeword, and then subtract from the codeword $\boldsymbol{c}$,

$$\boldsymbol{c} = \boldsymbol{c} \bigoplus \begin{bmatrix} m_1 & m_2 & m_3 & m_4 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}.$$

The order 0 message bit is equal to $m_0 = \mathbb{1}[(\sum_{\boldsymbol{z} \in \mathbb{F}_2^m} \boldsymbol{c}(\boldsymbol{z})) > 2^{4-1}]$, where $\mathbb{1}$ is the indicator function.

### 2.4.2 FHT Decoder

The ML decoding relies on comparing the received codeword to the code book. The first-order RM code is a Hadamard code [32], so the ML decoding process for the first-order RM code can be implemented as a Hadamard transform [17], [18]. The Hardamard matrix for the RM(3,1) code is the following:

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

Columns counting from left to right are indexed as 0 to $2^3 - 1 = 7$, and they are corresponding to the message vectors $0000, 0001, ..., 0111$, where the first bit $m_0 = 0$. Also, columns 1, 2, 4 correspond to the Binary Phase-shift keying (BPSK) modulation of the

evaluation of $f = x_1$, $f = x_2$, and $f = x_3$ of the generator matrix in Section 2.2, and other columns are linear combinations of the column vectors 1, 2, and 4. Using the Hadamard transform, the process of performing the ML decoding for first-order RM codes is the following. The inner products between the received the sequence $\boldsymbol{y}$ and each column in the Hadamard matrix are computed, results of these inner products are called correlations, and they are stored in a vector $\boldsymbol{r}$. There are 8 correlations in total. The column index $\hat{\boldsymbol{z}}$ corresponding to the maximum absolute value of the correlation is selected. If $\boldsymbol{r}(\hat{\boldsymbol{z}}) > 0$, the column $\hat{\boldsymbol{z}}$ is the decoded codeword after the demodulation. If the $\boldsymbol{r}(\hat{\boldsymbol{z}}) \leq 0$, the decoded codeword is the bit-flipped version of the column $\hat{\boldsymbol{z}}$. $\boldsymbol{r}(\hat{\boldsymbol{z}}) \leq 0$ because the $f(1) = m_0$ complements all code bits if $m_0 = 1$.

The complexity of performing the Hadamard transform between the received sequence $y$ and the Hadamard matrix is $\mathcal{O}(n^2)$. FHT is an efficient implementation of the Hadamard transform, and it has a complexity of $\mathcal{O}(n \log n)$ [17], [18].

### 2.4.3 Overview of Other Decoding Algorithms

For RM codes, there are different decoding algorithms designed by different rules. For example, there are decoding algorithms based on the large automoprsim group (the Sidel'nikov-Pershakov algorithm based decoders [19], [20], [33], the permuted successive cancellation decoder [34], or other permuted low complexity decoders based on the automoprsim group [35]), the minimum weight parity check [36], the permutation of the factor graph [37], [38], [39], the recursive structure of RM codes (Dumer's recursive decoders [21], [22], [23] and RPA decoder [1]), and etc.

## 2.5 Mixed-Integer Quadratic Programming

The convex quadratic programming has the following form

$$\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{P}\boldsymbol{w} + \boldsymbol{q}^\top \boldsymbol{w} + r, \text{ s.t. } \boldsymbol{A}\boldsymbol{w} \leq a, \boldsymbol{B}\boldsymbol{w} = b, \tag{2.5}$$

where $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ is a positive semi-definite ($\boldsymbol{w}^\top \boldsymbol{P}\boldsymbol{w} \geq 0$, $\forall \boldsymbol{w} \in \mathbb{R}^n$) and symmetric matrix , $\boldsymbol{q} \in \mathbb{R}^n$, $r \in \mathbb{R}$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{B} \in \mathbb{R}^{p \times n}$ [40].

The mixed-integer quadratic programming (MIQP) has an extra integrity constraint,

$w_i \in \mathbb{Z}\ \forall i \in I$, and it has the following form:

$$\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{P}\boldsymbol{w} + \boldsymbol{q}^\top \boldsymbol{w} + r, \text{ s.t. } \boldsymbol{A}\boldsymbol{w} \leq a, \boldsymbol{B}\boldsymbol{w} = b, w_i \in \mathbb{Z}\ \forall i \in I, \boldsymbol{d} \leq \boldsymbol{w} \leq \boldsymbol{h}, \tag{2.6}$$

where $I$ is the set of indexes of variables that should be integer, and $\boldsymbol{d}$ and $\boldsymbol{h} \in \mathbb{R}^n$ are the lower and upper bound of the variable $\boldsymbol{w} \in \mathbb{R}^n$. The MIQP is NP-hard [41], [42], but the relaxed MIQP, dropping the integrity constraint, is convex if $\boldsymbol{P}$ is positive semi-definite and symmetric. The convex relaxed MIQP can be efficiently solved by existing solvers.

# Chapter 3

# RPA Decoder and Its Variants

In this Chapter, the background information of the RPA decoder and its variants is presented. This chapter will be divided into three sections. Section 3.1 introduces the concept of sub-spaces and cosets, which are used by the projection and aggregation function in the RPA decoder. The notion for the index emulation follows the projection-punctuation system in [43]. Section 3.2 introduces the RPA decoder, CPA decoder, and their list decoders. Section 3.3 introduces two low complexity variants of the RPA and CPA decoder.

For binary-input memoryless channels $W : \{0, 1\} \to \mathcal{W}$, the log-likelihood ratio (LLR) is:

$$\text{LLR} := \ln(\frac{W(x|0)}{W(x|1)}), \tag{3.1}$$

where $x \in \mathcal{W}$ is the channel output. The LLR vector is denoted by $\boldsymbol{L}$ in this thesis. Each codeword of the RM$(m, r)$ code can be represented by an $m$ variate polynomial of degree $\leq r$ in the vector space $\mathbb{E} := \mathbb{F}_2^m$, where $\mathbb{F}_2^m$ denotes the length $m$ binary vectors. Then, the coordinates of the RM codeword $\boldsymbol{c} \in \text{RM}(m, r)$ can be indexed by the binary vector $\boldsymbol{z} \in \mathbb{E}$ [1], [27].

## 3.1 Sub-space Emulation and Indexing

The basis vectors, $\{\boldsymbol{z}_i, i \in [1, d]\}$, of the arbitrary $d$-dimensional subspace are ordered in the reduced row echelon form (RREF). Each left-most 1 in the basis vector $\boldsymbol{z}_i$ indicates a corresponding one-dimensional subspace. In [43], the definition of the $d$-dimensional subspace is the following.

**Definition 1.** *The d-dimensional subspace is the span of the basis vectors,* $\mathbb{B} = \text{span}(\begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ ... \\ \boldsymbol{z}_d \end{bmatrix})$, *where* $\begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ ... \\ \boldsymbol{z}_d \end{bmatrix}$ *is organized in RREF, and the element* $\boldsymbol{z}_i$ *is the non-zero binary vector* $\boldsymbol{z}_i \in \mathbb{E} \setminus 0$, $i \in [1, d]$.

From the definition for the 1-dimensional subspace in [1], the 1-dimensional subspace contains a zero and the basis vector $\boldsymbol{z}_i \in \mathbb{E} \setminus 0$, which is the span of the basis vector. In the RPA decoder, the received codeword is projected $n-1$ 1-dimensional spaces, because there are $n-1$ non-zero element in $\mathbb{E}$ implying that there are $n-1$ sub-spaces. For each of the $n-1$ cosets, the quotient space $\mathbb{E}/\mathbb{B}$ contains all cosets $T = \boldsymbol{z} + \mathbb{B}$ for some $\boldsymbol{z}$ [1].

Taking the coset and indexes of projecting a RM$(4, 2)$ code to a 1-dimensional sub-spaces as an example. Let the sub-space be $\mathbb{B} = [0000, 0001]$. The left-most 1 is in position 1 counting from the right, and the position 1 is set as the masked bit position for $\mathbb{B} = [0000, 0001]$. All valid indexes $\boldsymbol{z}$ for this subspace are $[0000, 0010, 0100, 0110, 1000, 1010,\ 1100, 1110]$, where all those $\boldsymbol{z}$s have value 0 in position 1. $\mathbb{E}/\mathbb{B}$ indicates the quotient space that include all eight cosets $T = z + \mathbb{B}$ of $\mathbb{B} = [0000,0001]$, and these eight cosets are:

$$[0000 + 0000, 0000 + 0001] = [0000, 0001],\ [0010 + 0000, 0010 + 0001] = [0010, 0011],$$
$$[0100 + 0000, 0100 + 0001] = [0100, 0101],\ [0110 + 0000, 0110 + 0001] = [0110, 0111],$$
$$[1000 + 0000, 1000 + 0001] = [1000, 1001],\ [1010 + 0000, 1010 + 0001] = [1010, 1011],$$
$$[1100 + 0000, 1100 + 0001] = [1100, 1101],\ [1110 + 0000, 1110 + 0001] = [1110, 1111].$$

For projections to $d$-dimensional sub-spaces, masked positions are $d$ positions (counting from the right) corresponding to the left-most 1 in $\boldsymbol{z}_1$, .., and $\boldsymbol{z}_d$. Indexes and cosets are generated using the same procedure mentioned above.

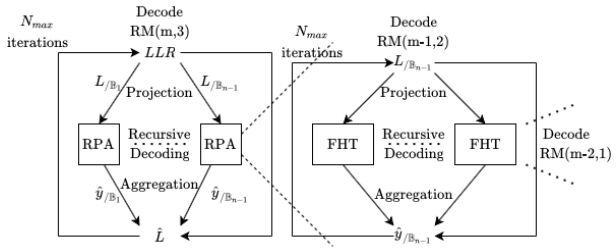



**Fig. 3.1** Workflow of the RPA decoding adapted from Fig. 1 in [1].

## 3.2 RPA and CPA Decoder

### 3.2.1 RPA Decoder

Fig. 3.1 describes the iterative RPA decoding procedure. Given $\boldsymbol{L}$, RPA is a three-step iterative decoding process namely projection, recursive decoding, and aggregation.

*1) Projection:* In the projection phase, LLRs for the lower-order codewords, RM$(m-1, r-1)$, are computed by projecting $\boldsymbol{L}$ onto one-dimensional subspaces, $\mathbb{B}_i = [0, z_i]$ and $z_i \in \mathbb{E} \setminus 0$, based on the cosets $T \in \mathbb{E}/\mathbb{B}_i = z + \mathbb{B}_i$ for some $z$ [1]. Given the received LLR vector $\boldsymbol{L}$, the projection for each coset $T$ is defined as [1]:

$$\boldsymbol{L}_{/\mathbb{B}_i}(T) = \ln\left(\exp\left(\sum_{z\in T} \boldsymbol{L}(z)\right) + 1\right) - \ln\left(\sum_{z\in T} \exp\left(\boldsymbol{L}(z)\right)\right). \tag{3.2}$$

*2) Recursive decoding:* In this phase, RM$(m, r)$ codes are recursively decoded, and the order of RM codes is recursively reduced until first-order codes, RM$(m-r+1, 1)$, are reached. The first-order RM codes can be efficiently decoded by using FHT [17], [18]. The input to the recursive decoding phase is $\boldsymbol{L}_{/\mathbb{B}_i}$, and the output is the decoded codeword $\hat{\boldsymbol{y}}_{/\mathbb{B}_i}$.

*3) Aggregation:* In this phase, $\boldsymbol{L}$ and the decoded codewords from recursive decoding ($\hat{\boldsymbol{y}}_{/\mathbb{B}_i}$) are aggregated to estimate the codeword for higher order RM codes, and the final estimation $\hat{\boldsymbol{L}}$ can be computed as the following:

$$\hat{\boldsymbol{L}}(z) = \frac{1}{n-1} \times \boldsymbol{L}_{\boldsymbol{cumu}}(z), \tag{3.3}$$

where $\boldsymbol{L}_{\boldsymbol{cumu}}(z) = \sum_{i=1}^{n-1}(1 - 2\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(z + \mathbb{B}_i))\boldsymbol{L}(z + z_i)$.

*4) Early stopping criteria:* During the RPA decoding, a saturation-based early stopping

criterion is used to check if

$$|\hat{\boldsymbol{L}}(z) - \boldsymbol{L}(z)| < \theta|\boldsymbol{L}(z)|, \forall z \in \mathbb{E}, \tag{3.4}$$

where $\theta$ is a small constant [1]. The RPA outputs the decoded codeword, if either the maximum number of iterations $N_{max} = \lceil \frac{m}{2} \rceil$ or the early stopping criterion (3.4) is met.

### 3.2.2 CPA Decoder

Similar to the RPA decoder, CPA is a three-step iterative decoder, namely projection, decoding RM$(m-r+1,1)$ sub-codes, and aggregation.

*1) Projection:* LLRs of RM$(m-r+1,1)$ sub-codes are computed by projecting the LLR vector $\boldsymbol{L}$ into $(r-1)$-dimensional sub-spaces, $\mathbb{B}_i := \text{span}(\boldsymbol{z}_1, ..., \boldsymbol{z}_{r-1})$ and $\boldsymbol{z}_1, ..., \boldsymbol{z}_{r-1} \in \mathbb{E} \setminus 0$, based on the cosets $T \in \mathbb{E}/\mathbb{B}_i = \boldsymbol{z} + \mathbb{B}_i$ for some $\boldsymbol{z}$ [1], [26], [43]. There are $n_B = \binom{m}{r-1}_2 = \prod_{i=0}^{r-2} \frac{2^{m-i}-1}{2^{r-1-i}-1}$ distinct sub-spaces [26]. Following assumptions are made for the projection [1]: I), code bits $\boldsymbol{c}(\boldsymbol{z})$s transmit through channels independently; II), $\boldsymbol{c}(\boldsymbol{z})$s are i.i.d Bernoulli$-\frac{1}{2}$ random variables. The projection is to find the LLR of observing $\boldsymbol{x}(\boldsymbol{z})$s in the same coset $T$ given the corresponding $\boldsymbol{c}(\boldsymbol{z})$s' parity-check [1]:

$$\begin{aligned}
\boldsymbol{L}_{/\mathbb{B}_i}(T) &= \ln \frac{P(\{\boldsymbol{x}(\boldsymbol{z}), \forall \boldsymbol{z} \in T\} | \bigoplus_{\boldsymbol{z}\in T} \boldsymbol{c}(\boldsymbol{z}) = 0)}{P(\{\boldsymbol{x}(\boldsymbol{z}), \forall \boldsymbol{z} \in T\} | \bigoplus_{\boldsymbol{z}\in T} \boldsymbol{c}(\boldsymbol{z}) = 1)} \\
&\overset{(a)}{=} \ln \frac{P(\bigoplus_{\boldsymbol{z}\in T} \boldsymbol{c}(\boldsymbol{z}) = 0 | \{\boldsymbol{x}(\boldsymbol{z}), \forall \boldsymbol{z} \in T\})}{P(\bigoplus_{\boldsymbol{z}\in T} \boldsymbol{c}(\boldsymbol{z}) = 1 | \{\boldsymbol{x}(\boldsymbol{z}), \forall \boldsymbol{z} \in T\})} \\
&\overset{(b)}{=} \ln \frac{1 + \prod_{\boldsymbol{z}\in T} (2P(\boldsymbol{c}(\boldsymbol{z}) = 0 | \boldsymbol{x}(\boldsymbol{z})) - 1)}{1 - \prod_{\boldsymbol{z}\in T} (1 - 2P(\boldsymbol{c}(\boldsymbol{z}) = 1 | \boldsymbol{x}(\boldsymbol{z})))} \\
&\overset{(c)}{=} \ln \frac{1 + \prod_{\boldsymbol{z}\in T} \tanh(\boldsymbol{L}(\boldsymbol{z})/2)}{1 - \prod_{\boldsymbol{z}\in T} \tanh(\boldsymbol{L}(\boldsymbol{z})/2)} \\
&= 2 \tanh^{-1}(\prod_{\boldsymbol{z}\in T} \tanh(\boldsymbol{L}(\boldsymbol{z})/2)),
\end{aligned} \tag{3.5}$$

where $\bigoplus$ is the summation over $\mathbb{F}_2$, (a) is the Bayes rule, (b) is according to the Lemma 1

in [3], and (c) is due to $2P(\boldsymbol{c}(\boldsymbol{z})=0|\boldsymbol{x}(\boldsymbol{z}))-1=1-2P(\boldsymbol{c}(\boldsymbol{z})=1|\boldsymbol{x}(\boldsymbol{z}))=\tanh(\boldsymbol{L}(\boldsymbol{z})/2)$ [44]. and (3.5) is the same as the statement in [26]. It is equivalent to equations (12) and (13) in [1] when projecting to one and two-dimensional sub-spaces, and the derivation is shown in Appendix A.1.

*2) Decoding RM*$(m-r+1,1)$ *sub-codes:* LLR vectors of RM$(m-r+1,1)$ sub-codes are decoded by the FHT decoder [17], [18], and the FHT decoder outputs the decoded order 1 RM codeword $\hat{\boldsymbol{y}}_{/\mathbb{B}_i}$. The code bits $\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)$ is the estimation of $\bigoplus_{\boldsymbol{z}\in T}\boldsymbol{c}(\boldsymbol{z})$.

*3) Aggregation:* In the aggregation phase, the LLR of each variable is updated according to all other variables in the same coset $T$ and the estimation $\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)$,

$$\boldsymbol{L}_{\boldsymbol{cumu}}(\boldsymbol{z})=\sum_{i=1}^{n_B}-1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)}(2\tanh^{-1}(\prod_{\boldsymbol{z}_i\in T\setminus\boldsymbol{z}}\tanh(\frac{L(\boldsymbol{z}_i)}{2}))). \tag{3.6}$$

It is equivalent to aggregation equations for the RPA and simplified RPA decoder (equation (14) in [1]), and the derivation is shown in Appendix A.1. Then, $\hat{\boldsymbol{L}}=\boldsymbol{L}_{\boldsymbol{cumu}}/n_B$ is used as the $\boldsymbol{L}$ for the next iteration.

*4) Early stopping criteria:* The following criteria is adopted in the CPA decoder,

$$||\hat{\boldsymbol{L}}-\boldsymbol{L}||_2<\theta||\hat{\boldsymbol{L}}||_2,\ \hat{\boldsymbol{c}}_{i-1}==\hat{\boldsymbol{c}}_i,\ i\in[1,N_{max}], \tag{3.7}$$

where $||\cdot||_2$ is the $l-2$ norm, $\theta$ is a small constant, $N_{max}=\lceil\frac{m}{2}\rceil$ is the maximum number of iterations, and $\hat{\boldsymbol{c}}_i$ is the hard decision of $\hat{\boldsymbol{L}}$ at iteration $i$. The CPA outputs the decoded codeword, if either $N_{max}$ or early stopping criteria (3.7) is met.

### 3.2.3 RPA and CPA's List Decoder

To reduce the FER, the RPA/CPA decoder can be extended to a Chase list decoder [1]. For list decoders with a list size of 8, the received LLR vector is sorted according to their absolute values $|\boldsymbol{L}(\boldsymbol{z})|$. Three LLRs with the smallest $|\boldsymbol{L}(\boldsymbol{z})|$ are replaced with $\pm\max(|\boldsymbol{L}(\boldsymbol{z})|)$ or $\pm 2\max(|\boldsymbol{L}(\boldsymbol{z})|)$, $\boldsymbol{z}\in\mathbb{E}$. Thus, there are 8 possible cases that are decoded by the RPA or CPA decoder. As decoded codewords from RPA and CPA are not always RM codes, Reed's decoder is used to correct the decoded codeword to be a RM

code. Among 8 possible cases, the one with the highest posterior probability,

$$\text{argmax}_{\hat{\boldsymbol{c}}_{\text{Reed, i}}} \sum_{\text{z}\in\{0,1\}^{\text{m}}} (-1^{\hat{\boldsymbol{c}}_{\text{Reed, i}}(\text{z})} \boldsymbol{L}(\boldsymbol{z})),\ i \in [1, 8], \tag{3.8}$$

is selected as the decoded codeword. The list decoder can return ML decoding performance, under various code lengths and rates.

## 3.3 RPA and CPA decoder's Low-Complexity Variants

### 3.3.1 Sparse RPA Decoder

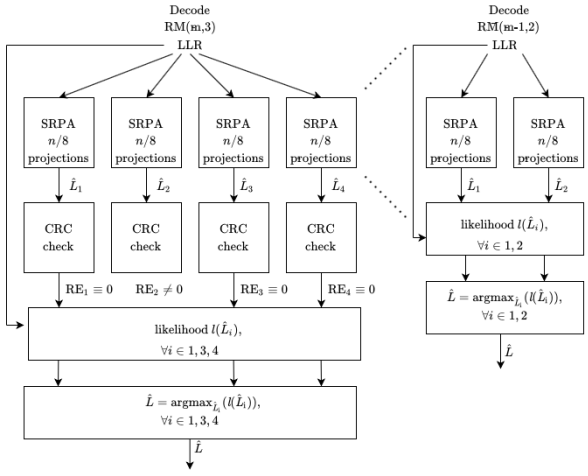



**Fig. 3.2** Workflow of the SRPA decoder.

To reduce the complexity of the RPA decoder, a multi-decoder technique named sparse RPA (SRPA) decoder is proposed in [27], and Fig. 3.2 shows the work flow of a SRPA decoder [27]. The proposed approach consists of multiple sparse RPAs that are generated by performing only a subset of projections in each decoder. The projections for each decoder are chosen randomly. Given $q$ sparse decoders, there are $q$ estimates, $\hat{\boldsymbol{L}}_1$, $\hat{\boldsymbol{L}}_2$, ..., $\hat{\boldsymbol{L}}_q$. At the end of the decoding process, the estimate that passes the CRC and yields the highest likelihood score $\text{argmax}_{\hat{\boldsymbol{L}}_i} < \boldsymbol{L}, \hat{\boldsymbol{L}}_i >$ is selected as the decoded codeword. Numerical results presented in [27] depict that SRPA reduces the complexity by $50 - 79\%$, while maintaining similar decoding performance as the RPA. However, due to CRC, the effective code rate is reduced.

### 3.3.2 Pruned CPA Decoder

Given the received error patter $\boldsymbol{e}$ and projections to arbitrary $(r-1)$-dimensional sub-spaces $\mathbb{B}_1$ and $\mathbb{B}_2$, [28] shows that the probability of projected error patterns $\boldsymbol{e}_{/\mathbb{B}_1} = \boldsymbol{e}_{/\mathbb{B}_2}$ is

$$P = \frac{1}{2}[1 + (1-2\epsilon)^{(2^{(r-1)+1} - 2|\mathbb{B}_1 \bigcap \mathbb{B}_2|)}], \tag{3.9}$$

where $\epsilon$ is the probability that an independent error occurs in $\boldsymbol{e}$. To reduce the chance of having similar projected error patterns, a subset, $S$, of $(r-1)$-dimensional sub-spaces is constructed to include sub-spaces with the least similarity to one another, which is measured by the set correlation $r_S$ [28]:

$$r_S := \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} r_{ij}, \; r_{ij} := \frac{\dim(\mathbb{B}_i \bigcap \mathbb{B}_j)}{r-1}, \tag{3.10}$$

where $r_{ij}$ is the pair-wised correlation of $\mathbb{B}_i$ and $\mathbb{B}_j$, and it is non-negative. For a set size $|S|$, the optimal subset has the smallest set correlation $r_S$, and it depends on the RM code that is decoded by the CPA decoder. So, the subset can be found off-line, and no additional decoding complexity is introduced.

The computation of $r_S$ can be rewritten into the matrix multiplication form

$$r_S = \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \tag{3.11}$$

where $u_i \in \boldsymbol{u}$, $\mathbb{B}_i \in S$ is indicated by $u_i = 1$, and $\mathbb{B}_i \notin S$ is indicated by $u_i = 0$. $\boldsymbol{R}$ is a size $n_B \times n_B$ matrix that stores all pair-wised correlations $r_{ij}$, and it is symmetric. (3.11) is a quadratic form, and it can be used as the objective function in the quadratic programming [40]. Quadratic programming is convex, if the matrix in the quadratic term is symmetric and positive semi-definite [40].

# Chapter 4

# Reduced Complexity RPA Decoder for Reed-Muller Codes

This chapter introduces complexity reduction methods for the RPA decoder. Section 4.1 presents the proposed complexity reduction techniques. Numerical simulation results are presented in Section 4.2. Section 4.3 presents the proposed hardware friendly approximation functions for the RPA decoding. Finally, in Section 4.4, the chapter summary is made.

## 4.1 Proposed Complexity Reduction Techniques

During the RPA decoding process, the existing saturation-based early stopping criterion (3.4) compares the results of two consecutive iterations in a sequence. In this paper, we propose using a syndrome-based approach to check for RPA decoder's results within the same iteration.

For RPA decoding, each projection and aggregation has the complexity $\mathcal{O}(n)$ and the decoding of first-order RM codes (FHT) has the complexity $\mathcal{O}(n \log n)$. The number of the FHT, involved during RPA decoding, is commonly used to measure the complexity of the RPA algorithm [1], [26], [27], [45]. For example, SRPA approach uses the average number of the FHT performed during the RPA's decoding process as the measure of the complexity [27].

For RM$(m, r)$ codes, the worst-case complexity of RPA decoding in terms of the number of FHT is given as $(N_{max})^{r-1} \prod_{i=0}^{r-2}(2^{m-i} - 1)$, where $N_{max}$ is the maximum number of iterations for RPA and $N_{max}$ is the same for all recursive layers of RPA decoding.

### 4.1.1 Proposed Syndrome-Based Early Stopping Criteria

For any $(n, k)$ linear block code $(\mathcal{C})$, where $n$ is the code length and $k$ is the code dimension, there exists a $k \times n$ matrix $\boldsymbol{G}$ called the generator matrix of the code and a $(n-k) \times n$ matrix $\boldsymbol{H}$ called the parity-check matrix such that

$$\forall\ \boldsymbol{c} \in \mathcal{C},\ \boldsymbol{H} \cdot \boldsymbol{c}^{\top} = \boldsymbol{0}. \tag{4.1}$$

Consider that $\boldsymbol{c}$ has been transmitted over a noisy channel and $\boldsymbol{x}$ is received at the output of the channel. Due to the noisy channel, $\boldsymbol{x}$ can differ from $\boldsymbol{c}$. Therefore, we can establish the relationship between $\boldsymbol{x}$ and $\boldsymbol{c}$ as: $\boldsymbol{x} = \boldsymbol{c} \oplus \boldsymbol{e}$, where $\boldsymbol{e}$ is the *error vector* caused by the channel noise. The *syndrome* is defined by $\boldsymbol{s} \triangleq \boldsymbol{H} \cdot \boldsymbol{x}^{\top}$. According to (4.1), $\boldsymbol{s}$ is zero if and only if $\boldsymbol{x}$ is a codeword. Thus, if $\boldsymbol{s}$ is zero, either there is no error or the error vector itself is a codeword [46].

For a RM$(m, r)$ code, the parity-check matrix is given as $\boldsymbol{H} = \boldsymbol{G}(m, m-r-1)$ [16], where $\boldsymbol{G}(m, m-r-1)$ is the generator matrix for the RM$(m, m-r-1)$ code. In this work, using the parity-check matrix for the RM$(m, r)$ code, we propose to use the syndrome check (4.1) for early stopping of the RPA decoder. Given the received LLR vector $(\boldsymbol{L})$, a syndrome check is performed to check if $\boldsymbol{H} \cdot \hat{\boldsymbol{l}}^{\top} == \boldsymbol{0}$, where $\hat{\boldsymbol{l}}$ is the hard decision vector of $\boldsymbol{L}$. If the syndrome check is satisfied, decoding is assumed to be successful. Otherwise, the RPA decoder proceeds with projecting the codewords to lower dimensions as explained in Section 3.2.1.

During the RPA decoding, in each iteration, the syndrome check is performed on the aggregated LLR $\boldsymbol{L_{cumu}}$. For the RM$(7, 2)$ code, Fig. 4.1 (a) plots the average number of the FHT decoding for the baseline RPA as well as the proposed RPA (RPA$_{\text{SYN}}$), where the syndrome check is applied after each RPA iteration. As depicted in Fig. 4.1 (a), the proposed syndrome-based early stopping technique reduces the average number of the FHT by 50.7%, however, it does not take the overhead of syndrome computations into account.

For a fair comparison, with the baseline RPA, the number of the FHT can be converted to the equivalent number of operations $(+/-)$ performed $(\mathcal{O}(n \log n))$. Similarly, for the proposed syndrome-based method, the number of operations for the syndrome computation $(\mathcal{O}(n \times (n-k)))$ can be added to the number of operations for the FHT. Fig. 4.1 (b) compares the complexity in terms of the number of operations, for the baseline RPA and the proposed syndrome-based RPA (RPA$_{\text{SYN}}$). From Fig. 4.1 (b), it can be seen that

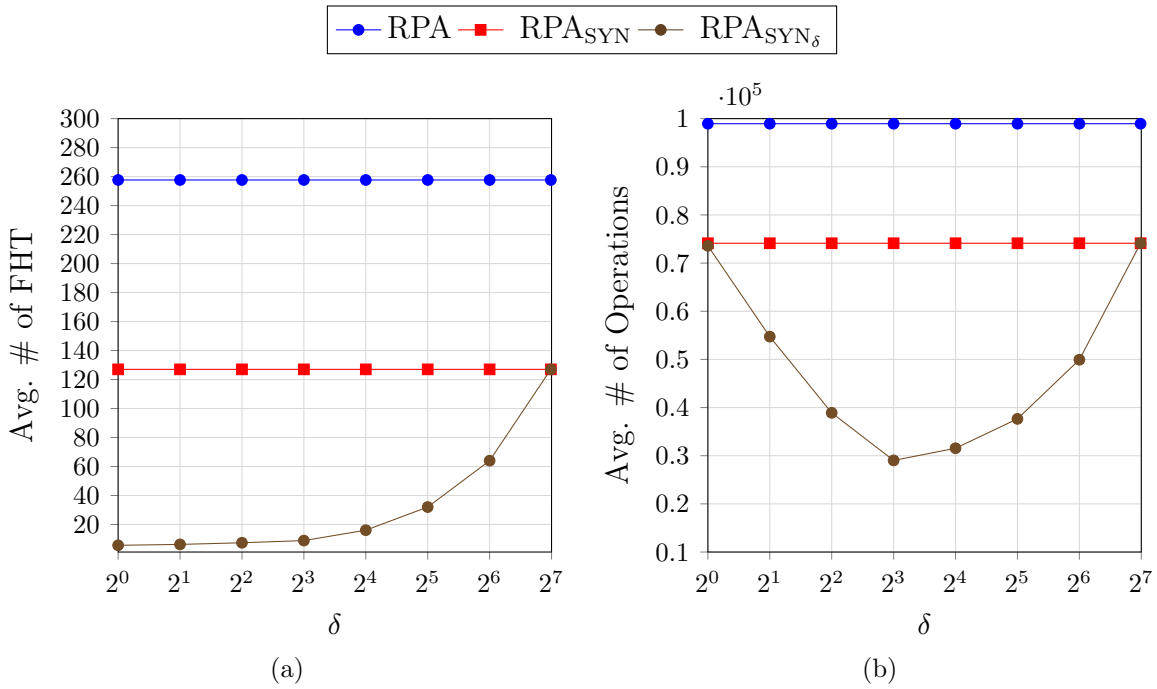



**Fig. 4.1** The average complexity for the RPA decoding of the RM$(7, 2)$ code at $E_b/N_0 = 4.25$ dB and $N_{max} = 4$.

RPA$_{\text{SYN}}$ reduces the complexity by 25.1% as compared to the baseline RPA.

To reduce the complexity further, at every iteration of RPA, we propose to perform the syndrome check after aggregating $\delta$ recursive decoding results, $\boldsymbol{L}_{\boldsymbol{cumu,1:k\times\delta}}$, where $\boldsymbol{L}_{\boldsymbol{cumu,1:k\times\delta}}$ denotes aggregated results from $\mathbb{B}_1$ to $\mathbb{B}_{k\times\delta}$, $\delta \leq n-1$ and $\forall k \in [1, \lfloor \frac{n-1}{\delta} \rfloor]$. Fig. 4.1 (a) plots the average number of the FHT of the proposed scheme, RPA$_{\text{SYN}_\delta}$, for different values of $\delta$. For example, for $\delta = 32$, the syndrome check is performed after aggregating every 32 recursive decoding results ($\boldsymbol{L}_{\boldsymbol{cumu,1:k\times 32}}$) at each iteration. From Fig. 4.1 (a) with parameter $\delta = 32$, the average number of FHT decoding is reduced by 87.6% as compared to the baseline RPA. In terms of the number of operations, for $\delta = 32$, the RPA$_{\text{SYN}_\delta}$ reduces the average complexity by 61.9% when compared to the baseline RPA as shown in Fig. 4.1 (b).

### 4.1.2 Proposed Scheduling for Reducing RPA's Complexity

From the numerical simulation results of RPA decoding of RM$(m, r)$ codes, we observed that the majority of the sign change occurs in the first iteration, implying that the majority

**Table 4.1** Worst-case complexity, measured by the number of FHT decoding, of SRPA, RPA , and $\text{RPA}_{\text{SCH}}$ with $N_{max} = \lceil \frac{m}{2} \rceil$ and $d = 2$.

| | RM(7,2) | RM(8,3) | RM(m,r) |
|---|---|---|---|
| RPA | 508 | 518160 | $(N_{max})^{r-1} \prod_{i=0}^{r-2} (2^{m-i} - 1)$ |
| $\text{RPA}_{\text{SCH}}$ | 239 | 114481 | $\prod_{i=0}^{r-2} \sum_{j=0}^{N_{max}-1} \lceil \frac{2^{m-i}-1}{d^j} \rceil$ |
| SRPA | 128 | 65536 | $((N_{max})^{r-1} \prod_{i=0}^{r-2} \frac{2^{m-i}}{8} q_{r-i})$ |

**Table 4.2** The average reduction in % of FHT used by different complexity reduction techniques at selective $E_b/N_0$.

| | RM(7,2) | | RM(8,3) | |
|---|---|---|---|---|
| | 1.5dB | 4.25dB | 1.0dB | 3.25dB |
| $\text{RPA}_{\text{SCH}}$ | 38.6% | 25.5% | 57.6% | 39.5% |
| $\text{RPA}_{\text{SYN}_\delta}$ | 88.0% | 96.5% | 83.7% | 99.8% |
| $\text{RPA}_{\text{SYN}_\delta+\text{SCH}}$ | 88.2% | 96.5% | 87.7% | 99.8% |
| SRPA | 64.6% | 50.3% | 77.7% | 64.3% |

of errors are corrected in the first iteration. The average sign change of decoding RM$(7, 2)$ is shown in Fig. 4.2. A similar observation is reported in [45].

Based on this observation, we propose a scheduling scheme (SCH) to reduce the number of projections in successive iterations during the RPA decoding process. This reduction in the number of projections depends on a user-defined decaying parameter, $d \geq 1$. In the proposed scheduling, given $d$, iteration $j$ ($j \in [1, N_{max}]$) uses a subset of projections ($\lceil \frac{n-1}{d^{j-1}} \rceil$ projections) chosen uniformly at random. The proposed scheduling is applied across all RPA's recursive layers. In general, for decoding a RM$(m, r)$ code, the worst-case complexity of proposed scheduling for RPA ($\text{RPA}_{\text{SCH}}$) is $\prod_{i=0}^{r-2} \sum_{j=1}^{N_{max}} \lceil \frac{2^{m-i}-1}{d^{j-1}} \rceil$ FHT. This scheduling technique can also be combined with the proposed syndrome-based early stopping criterion ($\text{RPA}_{\text{SYN}_\delta+\text{SCH}}$). The worst-case complexities of RPA decoder, SRPA decoder, and the proposed scheduling scheme with $d = 2$ are shown in Table 4.1. The proposed scheduling scheme reduces the worst-case complexity by more than 50%, compared to the baseline RPA decoder.

**Table 4.3** The average number of +/− used by FHT and syndrome check at selected $E_b/N_0$s of the RM(7,2) code.

| | 1.5 | 2.5 | 3.5 | 4.25 |
|---|---|---|---|---|
| RPA | $1.39 \times 10^5$ | $1.21 \times 10^5$ | $1.04 \times 10^5$ | $9.89 \times 10^4$ |
| $\text{RPA}_{\text{SYN}_\delta}$ | $9.60 \times 10^4$ | $4.72 \times 10^4$ | $3.29 \times 10^4$ | $3.02 \times 10^4$ |
| Reduction | 30.8% | 60.9% | 68.4% | 69.5% |

**Table 4.4** The average number of +/− used by FHT and syndrome check at selected $E_b/N_0$s of the RM(8,3) code.

| | 1 | 2 | 2.5 | 3.25 |
|---|---|---|---|---|
| RPA | $8.84 \times 10^7$ | $6.49 \times 10^7$ | $5.49 \times 10^7$ | $4.44 \times 10^7$ |
| $\text{RPA}_{\text{SYN}_\delta}$ | $7.44 \times 10^7$ | $1.08 \times 10^7$ | $2.86 \times 10^6$ | $5.80 \times 10^5$ |
| Reduction | 15.9% | 83.4% | 94.8% | 98.7% |

## 4.2 Proposed Reduced Complexity RPA

Algorithm 1 summarizes the pseudo code of the RPA decoder with the proposed complexity reduction techniques ($\text{RPA}_{\text{SYN}_\delta+\text{SCH}}$). The inputs of the algorithm are the channel LLR vector $\boldsymbol{L}$, $m$, $r$, $N_{max}$, the decaying parameter $d$, the frequency of performing syndrome checks $\delta$, and the saturation threshold $\theta$. The parity-check matrix is generated according to $m$ and $r$ (line 4). Then, the syndrome check is performed on the received LLR vector $\boldsymbol{L}$ (line 6). If the syndrome check is satisfied, the decoding process is skipped, and the hard decision of $\boldsymbol{L}$ is returned as the codeword. If not, the RPA decoding is performed (lines 12-14).

However, different from the regular RPA decoding, the proposed $\text{RPA}_{(\text{SYN}_\delta+\text{SCH})}$ performs the syndrome check after aggregating every $\delta$ decoding results (line 15). If any of the syndrome checks passes, the hard decision of $\boldsymbol{L_{cumu}}$ is returned as the codeword. Moreover, the maximum number of projections $P_{max}$ is reduced according to $d$ for the subsequent iteration (line 20). In each iteration, after decoding $P_{max}$ projections, if the early stopping criterion (3.4) is satisfied, the decoding process is stopped and the hard decision of the average of the aggregated LLR vector ($\hat{\boldsymbol{L}}$) is passed as the decoded result.

Fig. 4.3 (a-b) plot the FER performance for the RPA decoding of RM$(7, 2)$ and RM$(8, 3)$ codes, with the BPSK modulation over an AWGN channel. Moreover, the parameters

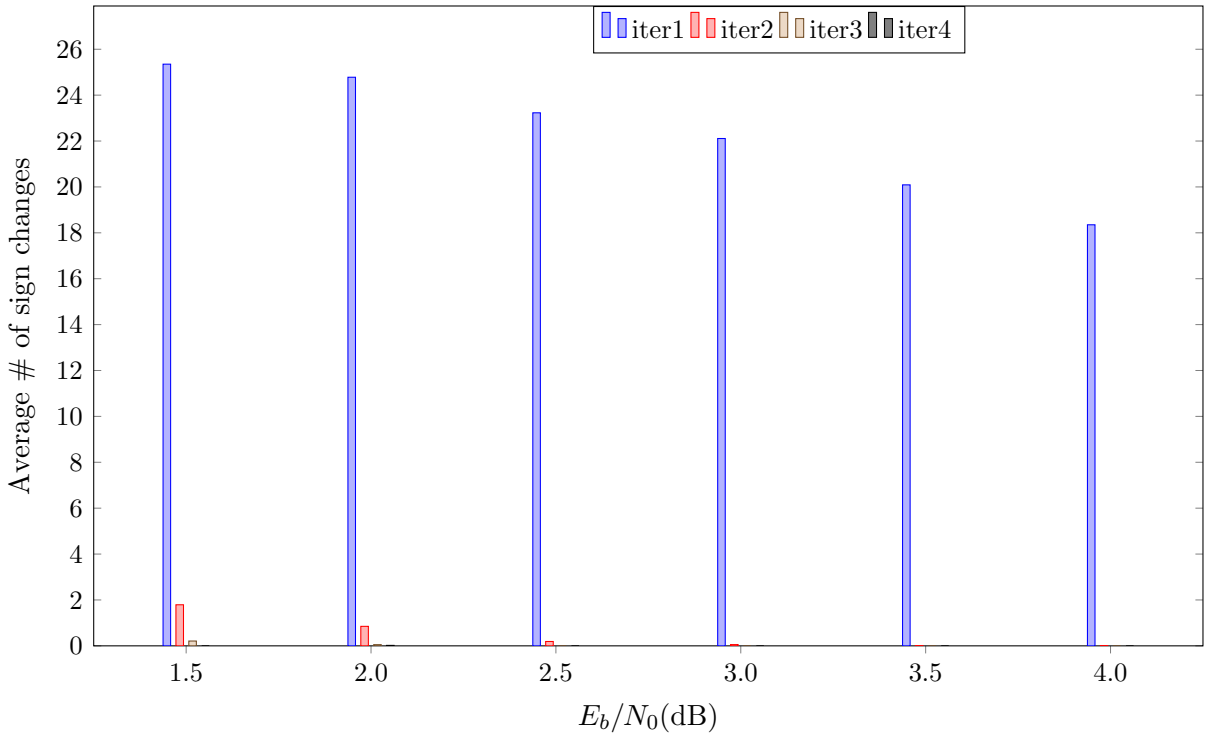



**Fig. 4.2** Average sign changes in each iteration when decoding the RM(7,2) code.

$N_{max} = \lceil \frac{m}{2} \rceil$, $\theta = 0.05$, $d = 2$ and $\delta = 8$ are used for RPA, SRPA, and proposed complexity reduction techniques.

The proposed techniques for RPA ($\text{RPA}_{\text{SCH}}$, $\text{RPA}_{\text{SYN}_\delta}$, $\text{RPA}_{\text{SYN}_\delta+\text{SCH}}$) are compared with the baseline RPA as well as previously proposed SRPA decoder. The proposed techniques have similar decoding performance to the baseline RPA as well as SRPA. However, the proposed techniques have a significant impact on reducing the complexity as shown in Fig. 4.3 (c-d).

For decoding the RM$(7, 2)$ code, $\text{RPA}_{\text{SYN}_\delta}$ reduces the complexity, in terms of the average number of operations required by FHT and syndrome computations for $\text{RPA}_{\text{SYN}_\delta}$, by 69.3% and 37.4% as compared to the baseline RPA and SRPA for a target FER of $10^{-5}$ as shown in Fig. 4.3 (c). Table 4.3 and Table 4.2 show the reduction of basic operations and FHT decoding attempts required by the RPA decoder and proposed complexity reduction techniques at selected $E_b/N_0$ points.

Furthermore, for decoding the RM$(8, 3)$ code, $\text{RPA}_{\text{SYN}_\delta}$ reduces the complexity by 98.2%

and 94.6% as compared to the baseline RPA and SRPA for a target FER of $10^{-5}$ as shown in Fig. 4.3 (d). Table 4.4 and Table 4.2 show the reduction of basic operations and FHT decoding attempts required by the RPA decoder and proposed complexity reduction techniques at selected $E_b/N_0$ points.

## 4.3 Hardware Friendly Approximation Functions for RPA

In this section, we propose to approximate the transcendental projection function (3.2) with the hardware friendly approximations. When projecting to the one-dimensional subspaces, the size of the coset is always 2 [1]. Let $X$ and $Y$ denote input LLRs ($\boldsymbol{L}(z)$) to (3.2), and then (3.2) is re-written as:

$$\begin{aligned}\boldsymbol{L}_{/\mathbb{B}_i}(T) &= \ln\left(\exp\left(X+Y\right)+1\right)\\ &- \ln\left(\exp\left(X\right)+\exp\left(Y\right)\right).\end{aligned} \tag{4.2}$$

According to the Jacobi logarithm [47],

$$\ln(\exp\left(X\right)+\exp\left(Y\right)) = \max(X,Y) + f(X,Y), \tag{4.3}$$

where $f(X,Y) = \ln(1+\exp\left(-|X-Y|\right))$ is the correction function. Thus, (4.2) can be approximated as

$$\begin{aligned}&\ln(\exp(X+Y)+\exp(0)) - \ln(\exp(X)+\exp(Y))\\ &= \max(X+Y,0) + f(X+Y,0) - \max(X,Y) - f(X,Y).\end{aligned} \tag{4.4}$$

In literature, there are three popular schemes to approximate $f(X,Y)$ namely the max-log-MAP ($f(X,Y)=0$), the linear-log-MAP [48] ($f(X,Y) = \max(0, 0.6925 - 0.25|X-Y|)$), and the constant-log-MAP [47],

$$f(X,Y) = \begin{cases}\frac{3}{8} & \text{if } |X-Y|<2\\ 0 & \text{otherwise}\end{cases}.$$

Additionally, (4.2) can also be re-organized and approximated using the min-sum approx-

imation [26], [49] as,

$$\begin{aligned} &\ln\left(\exp\left(X+Y\right)+1\right)-\ln\left(\exp\left(X\right)+\exp\left(Y\right)\right) \\ &= 2\tanh^{-1}(\tanh\frac{X}{2}\tanh\frac{Y}{2}) \\ &\approx \operatorname{sign}(X)\operatorname{sign}(Y)\min(|X|,|Y|). \end{aligned} \tag{4.5}$$

Fig. 4.4 plots the FER performance for RPA decoding of RM codes with various approximation functions. In comparison with the baseline RPA for the RM$(8,3)$ code, RPA decoders with max-log-MAP ($\text{RPA}_{\text{max-log-MAP}}$) and min-sum ($\text{RPA}_{\text{min-sum}}$) approximations result in performance degradations of 0.2 dB at the target FER of $10^{-5}$, whereas other approximations incur negligible decoding performance losses. In summary, different approximation schemes can be chosen, depending on the target FER requirement and available hardware resources.

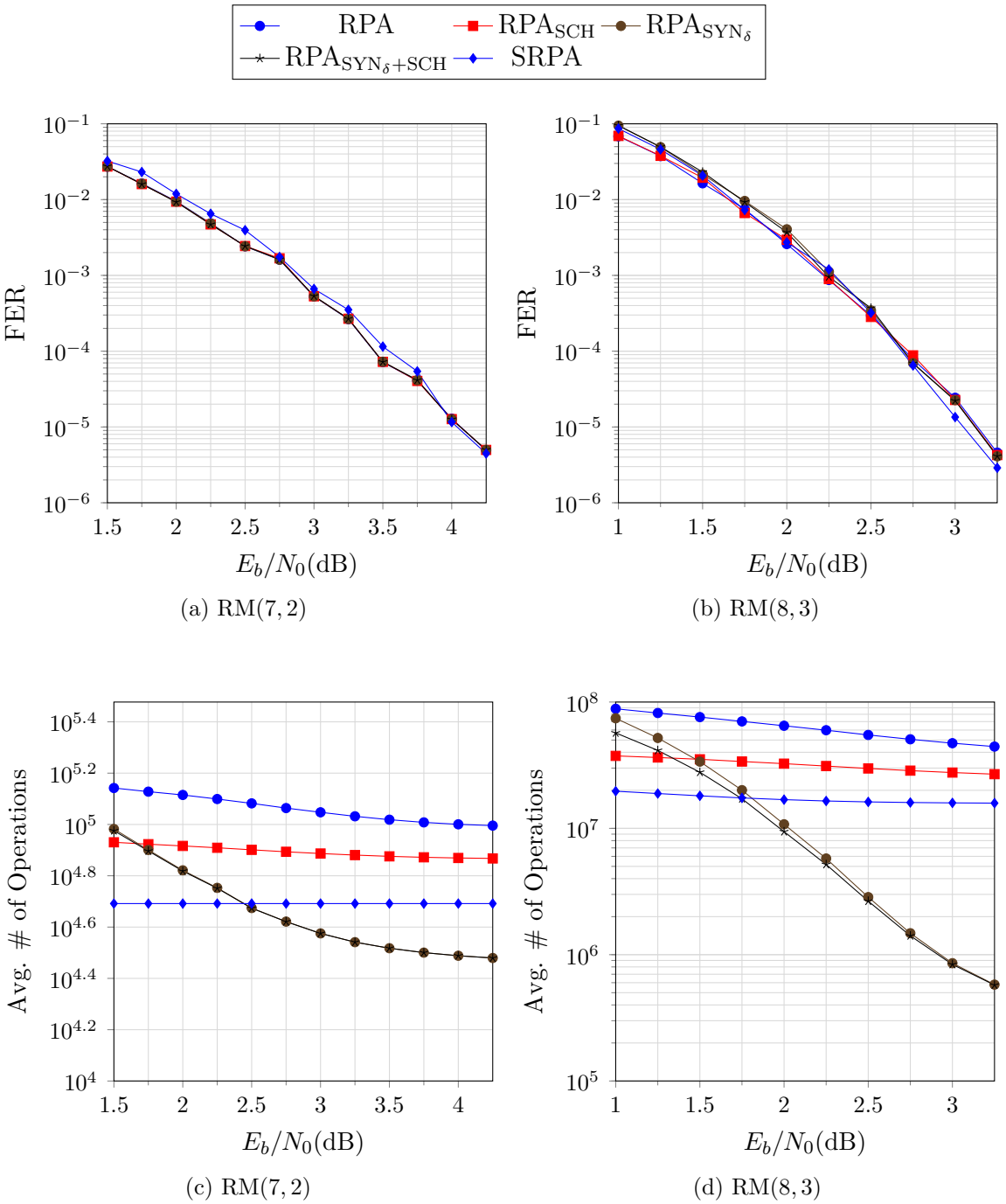
## 4.4 Chapter Summary

RPA decoder, a recently proposed near ML performing decoder for RM codes, suffers from the high computational complexity. In this work, we have proposed syndrome-based early stopping techniques and a scheduling scheme for reducing the computational complexity of the RPA decoder. The comparison with the baseline RPA reveals that proposed techniques result in a $69-98\%$ reduction in the average computational complexity for a target FER of $10^{-5}$. Similarly, in comparison with the previously proposed SRPA complexity reduction technique, the proposed RPA results in a $37-94\%$ complexity reduction. Moreover, this work introduces hardware-friendly approximation functions to replace the RPA's computationally expensive transcendental projection function.

**Algorithm 1:** Reduced complexity RPA

**Input:** $\boldsymbol{L}$, $m$, $r$, $N_{max}$, $d$, $\delta$, $\theta$
**Output:** $\hat{\boldsymbol{c}}$

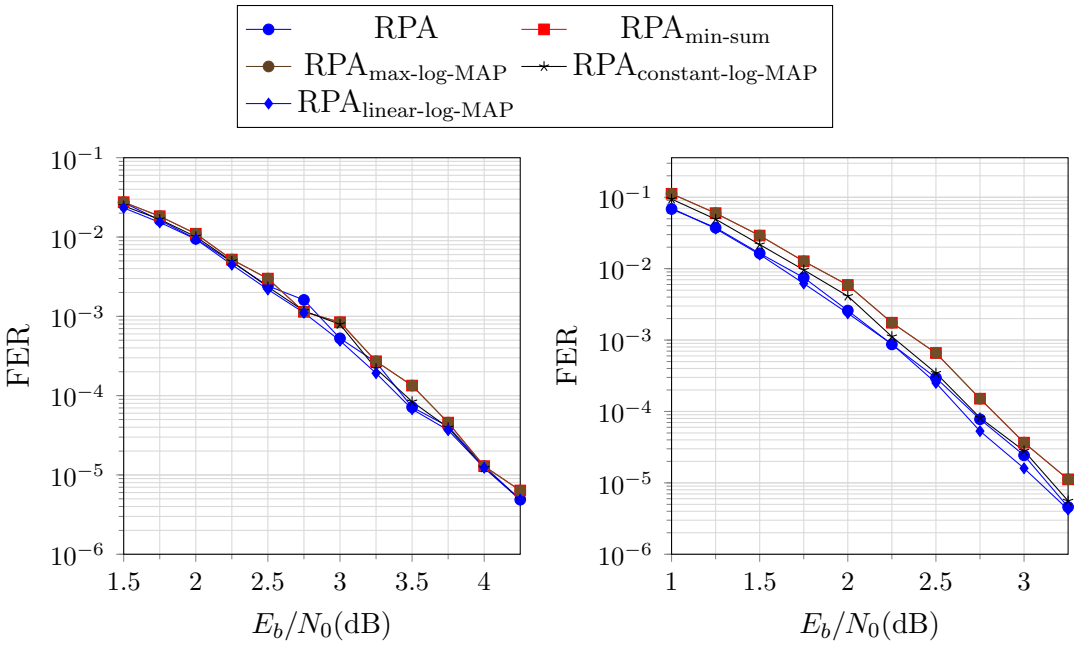
**if** $r == 1$ **then**
  **return** $\hat{\boldsymbol{c}} \leftarrow$ `FHT-Decoding`$(\boldsymbol{L})$
**else**
  $\boldsymbol{H} \leftarrow$ `GenerateParityMatrix`$(\boldsymbol{G}(m, m-r-1))$
  $P_{max} \leftarrow 2^m - 1$
  $\boldsymbol{s} \leftarrow$ `SyndromeCheck`$(\boldsymbol{L}, \boldsymbol{H})$
  **if** $\boldsymbol{s} == \boldsymbol{0}$ **then**
    **return** $\hat{\boldsymbol{c}} \leftarrow$ `HardDecision`$(\boldsymbol{L})$
  **for** $t = 1, ..., N_{max}$ **do**
    $\boldsymbol{L_{cumu}}(z) \leftarrow 0 \ \forall z \in \mathbb{E}$
    **for** $i = 1, ..., P_{max}$ **do**
      $\boldsymbol{L}_{/\mathbb{B}_i} \leftarrow$ `Projection`$(\boldsymbol{L}, \mathbb{B}_i)$
      $\hat{\boldsymbol{y}}_{/\mathbb{B}_i} \leftarrow$ `RPA`$(\boldsymbol{L}_{/\mathbb{B}_i}, m, r, N_{max}, d, \delta, \theta)$
      $\boldsymbol{L_{cumu}} +=$ `Aggregation`$(\boldsymbol{L}, \hat{\boldsymbol{y}}_{/\mathbb{B}_i})$
      **if** $i \pmod{\delta} == 0$ **then**
        $\boldsymbol{s} \leftarrow$ `SyndromeCheck`$(\boldsymbol{L_{cumu}}, \boldsymbol{H})$
        **if** $\boldsymbol{s} == \boldsymbol{0}$ **then**
          **return** $\hat{\boldsymbol{c}} \leftarrow$ `HardDecision`$(\boldsymbol{L_{cumu}})$
    $\hat{\boldsymbol{L}} \leftarrow \frac{\boldsymbol{L_{cumu}}}{P_{max}}$
    $P_{max} \leftarrow \lceil \frac{P_{max}}{d} \rceil$
    **if** $|\hat{\boldsymbol{L}}(z) - \boldsymbol{L}(z)| < \theta |\boldsymbol{L}(z)|, \forall z \in \mathbb{E}$ **then**
      **return** $\hat{\boldsymbol{c}} \leftarrow$ `HardDecision`$(\hat{\boldsymbol{L}})$
    $\boldsymbol{L} \leftarrow \hat{\boldsymbol{L}}$
**return** $\hat{\boldsymbol{c}} \leftarrow$ `HardDecision`$(\boldsymbol{L})$




(a) RM$(7, 2)$ (b) RM$(8, 3)$

(c) RM$(7, 2)$ (d) RM$(8, 3)$

**Fig. 4.3** Comparisons of decoding performance and average complexity for RPA decoding of RM codes.




(a) RM$(7, 2)$ (b) RM$(8, 3)$

**Fig. 4.4** Comparisons of RPA decoding performance using different approximation schemes.

# Chapter 5

# Optimization and Simplification of PCPA Decoder

In Section 5.1, finding the optimized subset is transformed into a mixed-integer quadratic programming problem, and optimization methods and results are presented. Section 5.2 discusses the CPA decoder with the min-sum approximation, the simplified list decoder, and their simulation results. Finally, the chapter summary is drawn in Section 5.3.

## 5.1 Optimization for PCPA Decoder

### 5.1.1 Mixed-Integer Quadratic Programming

Finding subsets with a small $r_s$ can be transformed into a MIQP problem performed off-line, and the objective function and constraints are

$$\min_{\boldsymbol{u}} \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u}, \text{s.t. } u_i \in \{0,1\}\ \forall u_i \in \boldsymbol{u},\ \mathbf{1}^\top \boldsymbol{u} = |S|, \tag{5.1}$$

where $\mathbf{1}$ is an all-ones vector.

MIQP is NP-hard [41], so relaxation or heuristic approaches should be used to solve MIQP efficiently. In this work, two applicable methods are used to efficiently solve the MIQP. The first method uses a heuristic solver called ADMM [41] to directly solve (5.1). It returns relatively good results, but the optimal solution is not guaranteed [41]. The MIQP can be relaxed by allowing $u_i \in \mathbb{R}\ \forall u_i \in \boldsymbol{u}$. After the relaxation, as the $r_{ij} \geq 0$,

$\boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u} \geq 0 \; \forall \boldsymbol{u} \in \mathbb{R}^{n_B}$ implies that $\boldsymbol{R}$ is symmetric and positive semi-definite. Thus, the relaxed MIQP is convex, and the second method, cutting-plane (CP) [50], can solve the MIQP as the following. CP replaces the quadratic objective function in MIQP with a linear objective function, the epigraph problem form [40], and following constraints [51],

$$\begin{aligned} \min_{w} w, \text{ s.t. } & \boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{v} - w \leq 0, \; w \geq 0, \\ & v_i \in [0, f_{max}] \; \forall v_i \in \boldsymbol{v}, \; \mathbf{1}^\top \boldsymbol{v} = 1, \\ & u_i = \mathbb{1}[v_i > 0] \; \forall u_i \in \boldsymbol{u}, \; \mathbf{1}^\top \boldsymbol{u} = |S|, \end{aligned} \tag{5.2}$$

where $w$ is the slack variable, $\boldsymbol{v}$ is bounded in a interval $[0, f_{max}]$ [51], and $\mathbb{1}$ is the indicator function. Then, in every iteration, a linear constraint is imposed, which approximates $\boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{v}$ by using order 1 Taylor expansion around $\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + \boldsymbol{\delta}$ [42], [51] and replacing $\boldsymbol{\delta}$ by $\boldsymbol{v}_i - \boldsymbol{v}_{i-1}$ [51],

$$\boldsymbol{v}_i^\top \boldsymbol{R}\boldsymbol{v}_i - w \approx -\boldsymbol{v}_{i-1}^\top \boldsymbol{R}\boldsymbol{v}_{i-1} + 2\boldsymbol{v}_{i-1}^\top \boldsymbol{R}\boldsymbol{v}_i - w \leq 0. \tag{5.3}$$

$\boldsymbol{v}_i$ is the variable vector that would be solved in every iteration, and $\boldsymbol{v}_{i-1}$ is the result from the previous iteration $i-1$ [51]. The CP does not guarantee the optimal result neither.

For medium-length and high-order RM codes, the number of distinct sub-spaces ($n_B$) for CPA is huge, so ADMM and CP would require many computation and memory resources. For example, in ADMM, the complexity of the matrix factorization is $\mathcal{O}(n_B^3)$ for a dense $\boldsymbol{R}$, the complexity of computing each subsequent iteration is $\mathcal{O}(n_B^2)$ [41], and the memory complexity of storing $\boldsymbol{R}$ is $\mathcal{O}(n_B^2)$. Thus, a greedy search method [52] is proposed. The workflow of this greedy method, summarized by the pseudo code shown in Algorithm 2, is the following. This greedy algorithm firstly constructs a set of sub-spaces that have zero correlations with each other, and then it progressively adds a new selection into the set $S$. The selected $\mathbb{B}_i$ is the one that would add the least set correlation to the current set $S$,

$$\underset{i}{\operatorname{argmin}} \sum_{j=1}^{|S|} \frac{\dim(\mathbb{B}_i \bigcap \mathbb{B}_j)}{r-1}, \; \forall \; \mathbb{B}_j \in S, \text{ and } \forall \; \mathbb{B}_i \notin S. \tag{5.4}$$

It computes the correlation (5.4) at most $\sum_{i=0}^{|S|} n_B - i$ times, and it only requires $n_B$ memories to store results, which is an alternative solution when the size of $\boldsymbol{R}$ is large.

### 5.1.2 Results and Analysis of Optimized Subsets

The CP is implemented using the existing solver in MATLAB [51], and the ADMM is implemented according to [41]. Parameters are tuned to generate the best result that our AMD Ryzen 5 2600 six-core processor with 15.6 GB of RAM simulation platform can return. Tuned parameters are shown in Appendix A.2. In this chapter, all simulations perform over the AWGN channel and use the BPSK modulation.

**Table 5.1** Minimum $r_S$ returned from different methods.

| | | | $r^*_{S,\text{CP}}$ | | | $r^*_{S,\text{ADMM}}$ | | | $r^*_{S,\text{Greedy}}$ | | | $r^*_S$ from [28] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n_B$ | $\lvert S\rvert$ | 64 | 128 | 256 | 64 | 128 | 256 | 64 | 128 | 256 | 64 |
| RM(7, 3) | 2667 | | 129 | 518 | 2197 | 130 | 525 | 2201 | 129 | 520 | 2197 | 69 |
| RM(7, 4) | 11811 | | 404.67 | 1710.67 | 7234 | 421.33 | 1804 | 7476.67 | 398 | 1708.67 | 7217.33 | |
| RM(7, 5) | 11811 | | 1322.50 | 5382.50 | 21754.50 | 1346 | 5456.50 | 21865 | 1322 | 5375 | 21771.50 | |
| RM(8, 3) | 10795 | | 65 | 271 | 1076 | 67 | 257 | 1036 | 64 | 257 | 1035 | |

To quantify the effect of obtaining a small set correlation, a sensitivity analysis is performed for different set sizes $|S|$. The sensitivity is defined as

$$\text{Sensitivity} = \frac{(\text{FER}^* - \text{FER})/\text{FER}^*}{(r^*_S - r_S)/r^*_S}. \tag{5.5}$$

$r^*_S$ and $\text{FER}^*$ are the minimum $r_S$ returned from the proposed method and the corresponding FER. $r_S$ and FER are the set correlation and the FER of randomly constructed subsets.

A positive sensitivity means finding $r^*_S$ helps reduce the FER, and a negative sensitivity means it does not. Subsets generated by the greedy search are used in the sensitivity analysis and FER simulations, in this work. To eliminate randomness effects of randomly constructed subsets, the average sensitivity is presented, and it is calculated using 10 randomly constructed subsets and the subset generated by the greedy search. For the fairness, the optimized and randomly constructed decoders with the same $|S|$ are compared in each $E_b/N_0$ point. The same $|S|$ means the optimized and randomly constructed decoders have the same worst-case complexity because they have the same number of FHTs per iteration and the same $N_{max}$.

Table 5.1 shows set correlations of optimized subsets generated by proposed methods, for RM codes with $n_B \lessapprox 10^4$. Results of all proposed methods are similar. From average sensitivities of RM(7, 3) and RM(7, 4) codes (Fig. 5.1 (a) and (c)), finding $r^*_S$ does reduce

the FER in most cases, so the optimized decoder has less performance loss than randomly constructed decoders in most cases. From the FER plot, Fig. 5.1 (b), of decoding the RM$(7, 3)$ code, the optimized PCPA decoders with $|S| = 64$ (PCPA64) has 0.1 dB performance loss at a target FER of $10^{-3}$, which matches the result of $r_S = 69$ in [28]. From results of the RM$(7, 3)$ code in Table 5.1, $r_S^* = 129$ is higher than the smallest $r_S = 69$ in [28], but they have similar performance loss. From Fig. 5.1 (b) and (d), optimized PCPA decoders with $|S| = 128$ (PCPA128) already return similar FERs as CPA decoders at a target FER of $10^{-4}$, when decoding RM$(7, 3)$ and RM$(7, 4)$ codes.

## 5.2 Min-sum Approximation and the Simplified List Decoder

Given (3.5), the min-sum approximation can be directly applied to the projection function of the CPA decoder [49],

$$\boldsymbol{L}_{/\mathbb{B}_i}(T) \approx (\prod_{\boldsymbol{z}\in T} \text{sign}(\boldsymbol{L}(\boldsymbol{z}))) \min(\{\boldsymbol{L}(\boldsymbol{z}), \forall\ \boldsymbol{z} \in T\}). \tag{5.6}$$

For the aggregation, the inverse hyperbolic tangent part in (3.6) can be replaced by the min-sum approximation as well.

Results returned from the RPA/CPA decoder are not necessary RM codes [1]. Reed's decoder is used to correct the decoded codeword returned from the RPA/CPA decoder to be a RM code [1], in the list decoder. In this work, the proposed simplified list decoder replaces the Reed's decoder with a syndrome check, which only computes the posterior probability of codewords that pass the syndrome check. Its pseudo code is shown in Algorithm 3.

Fig. 5.2 shows the FER of decoding RM$(7, 3)$ and RM$(7, 4)$ codes with the min-sum approximation, PCPA128 list decoders with list sizes of 16, 8, 4, 2 (L16, L8, L4, L2), and PCPA128's simplified list decoders with list sizes of 16, 8, 4, 2 (SL16, SL8, SL4, SL2). From Fig. 5.2 (a) and (b), CPA and PCPA decoders with the min-sum approximation have less than 0.15 dB performance degradation at a target FER of $10^{-4}$. From Fig. 5.2 (c) and (d), the PCPA's simplified list decoder returns similar decoding performance as the regular list decoder, which implies that the syndrome check can safely replace the Reed's decoder.

**Algorithm 2:** Greedy Search

**Input:** $m$, $r$, $|S|$
**Output:** $S$

$\{\mathbb{B}_i\} = \texttt{Shuffle}(\texttt{GenerateB}(m, r))$
$S(1) = \mathbb{B}_1$
$i$, $t = 2$
$n_B = \texttt{Size}(\{\mathbb{B}_i\})$
**while** $i \leq n_B$ & $t \leq |S|$ **do**
  **if** $(\mathbb{B}_i \bigcap \mathbb{B}_j) \setminus 0 == \emptyset$, $\forall \mathbb{B}_j \in S$ **then**
    $S(t) = \mathbb{B}_i$
    $t = t + 1$
  $i = i + 1$
Delete $\mathbb{B}_j$ from $\{\mathbb{B}_i\}$, $\forall \mathbb{B}_j \in S$
**while** $t \leq |S|$ **do**
  $n_B = \texttt{Size}(\{\mathbb{B}_i\})$
  $\boldsymbol{r}(i) \leftarrow 0$, $\forall i \in [1, n_B]$
  **for** $i = 1, ..., n_B$ **do**
    **for** *each* $\mathbb{B}_j \in S$ **do**
      $\boldsymbol{r}(i) = \boldsymbol{r}(i) + \texttt{dim}(\mathbb{B}_i \bigcap \mathbb{B}_j)/(r-1)$
  $index = \text{argmin}_i \boldsymbol{r}(i)$
  $S(t) = \mathbb{B}_{index}$
  $t = t + 1$
  Delete $\mathbb{B}_{index}$ from $\{\mathbb{B}_i\}$
**return** $S$

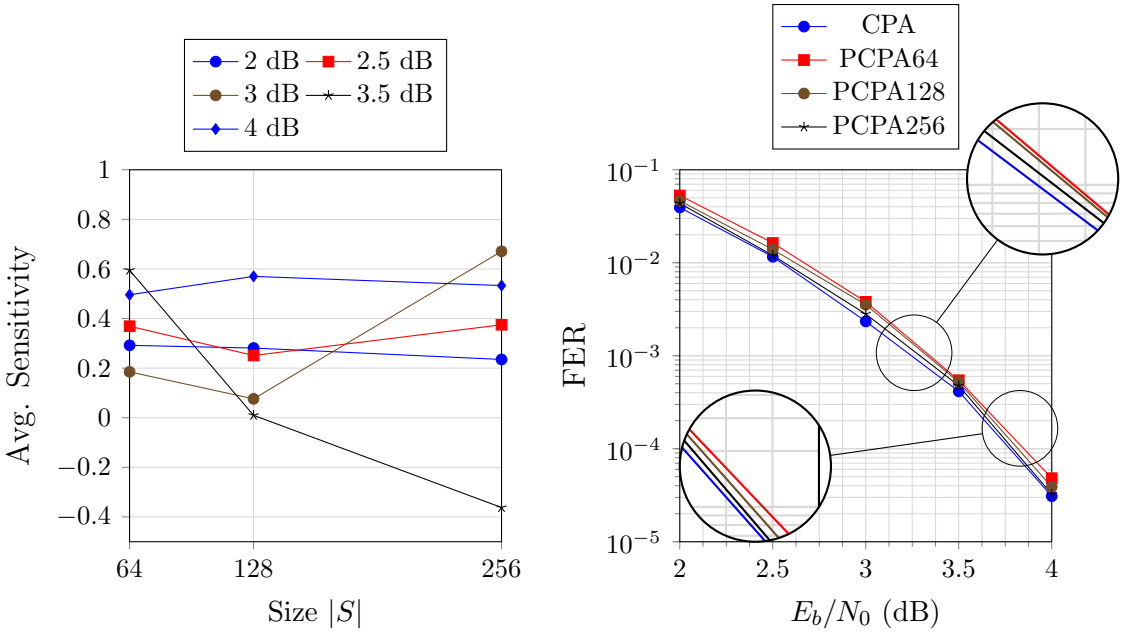
## 5.3 Chapter Summary

Finding a subset of sub-spaces with a small set correlation helps to achieve small performance loss for the PCPA decoder. In this work, finding the subset of sub-spaces with a small set correlation is viewed as a MIQP problem. Methods for solving this MIQP problem are proposed. Supporting by the sensitivity analysis and the corresponding FER, under the same complexity, the optimized subset does have less performance loss than randomly constructed subsets. Furthermore, the min-sum approximation is used to replace non-linear projection and aggregation functions in the CPA decoder, and it has less than 0.15 dB performance loss at a target FER of $10^{-4}$. Lastly, the proposed simplified list decoder does not have noticeable performance degradation, compared to the regular list decoder.

**Algorithm 3:** Simplified CPA list decoder

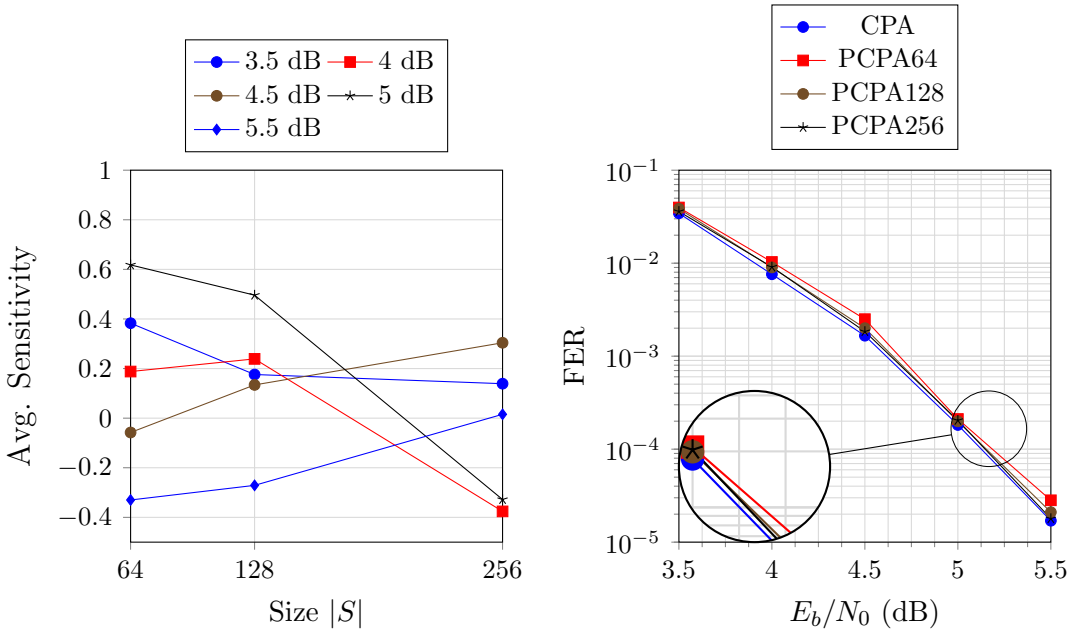
**Input:** $\boldsymbol{L}$, $m$, $r$, $N_{max}$, $\theta$, $t$
**Output:** $\hat{\boldsymbol{c}}$

$\boldsymbol{H} \leftarrow$ `GenerateParityMatrix`$(\boldsymbol{G}(m, m-r-1))$
$\tilde{\boldsymbol{L}} \leftarrow \boldsymbol{L}$
$(\boldsymbol{z}_1, ..., \boldsymbol{z}_t) \leftarrow$ indices of the $t$ smallest $|\boldsymbol{L}(\boldsymbol{z})|$, $\boldsymbol{z} \in \mathbb{E}$
$L_{max} \leftarrow 2\max(\{|\boldsymbol{L}(\boldsymbol{z})|, \forall \boldsymbol{z} \in \mathbb{E}\})$
$i = 1$
**for** $\boldsymbol{l} \in \{L_{max}, -L_{max}\}^t$ **do**
    $(\boldsymbol{L}(\boldsymbol{z}_1), \boldsymbol{L}(\boldsymbol{z}_2), ..., \boldsymbol{L}(\boldsymbol{z}_t)) \leftarrow \boldsymbol{l}$
    $\hat{\boldsymbol{c}} \leftarrow$ `CPA`$(\boldsymbol{L}, m, r, N_{max}, \theta)$
    **if** `SyndromeCheck`$(\hat{\boldsymbol{c}}, \boldsymbol{H}) == \boldsymbol{0}$ **then**
        $\tilde{\boldsymbol{C}}(i, :) \leftarrow \hat{\boldsymbol{c}}$
        $i = i + 1$
$index = \text{argmax}_i \sum_{\boldsymbol{z} \in \{0,1\}^m} ((-1)^{\tilde{\boldsymbol{C}}(i,\boldsymbol{z})} \tilde{\boldsymbol{L}}(\boldsymbol{z}))$
**return** $\hat{\boldsymbol{c}} \leftarrow \tilde{\boldsymbol{C}}(index, :)$




(a) Sensitivity RM(7, 3)

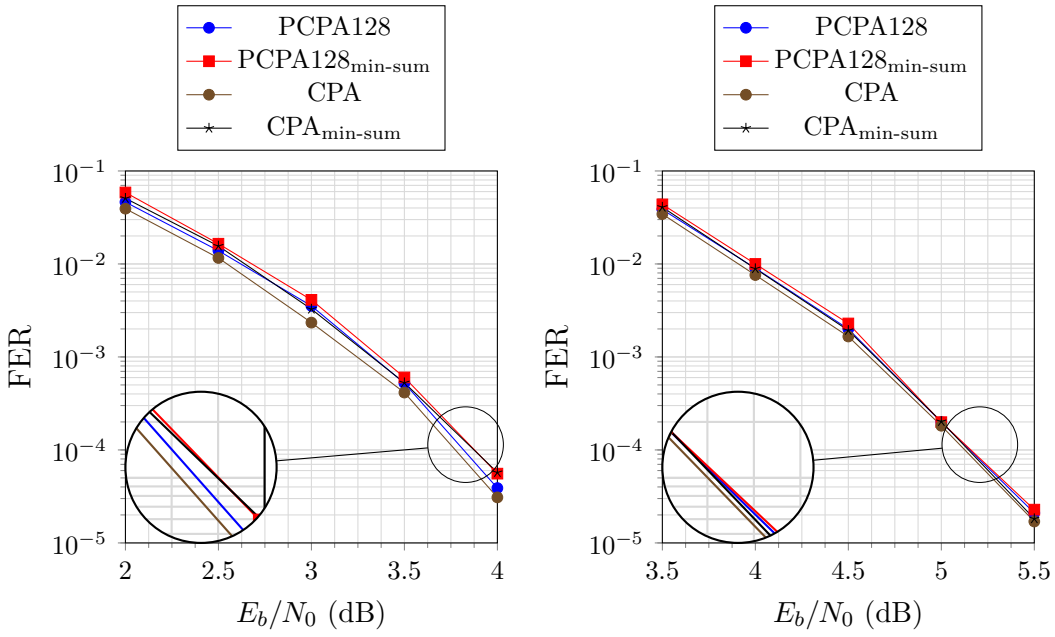
(b) RM(7, 3)



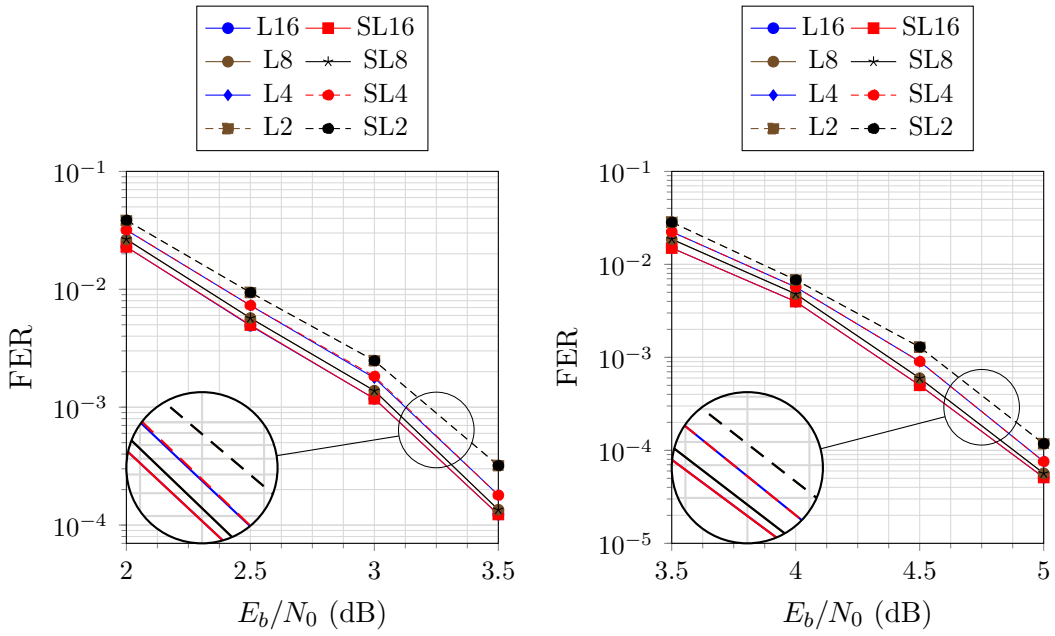

(c) Sensitivity RM(7, 4)

(d) RM(7, 4)

**Fig. 5.1** The sensitivity analysis of optimized PCAP decoders and their decoding performance.




(a) RM(7, 3) (b) RM(7, 4)




(c) RM(7, 3) (d) RM(7, 4)

**Fig. 5.2** Decoding performance of decoders (CPA and optimized PCPA128) using the min-sum approximation and the optimized PCPA128's list decoder.

# Chapter 6

# Conclusion

Emerging applications create the demand of low rate and short length channel codes with good decoding performance measured by the FER. RM codes attract attention due to their favorable characteristics and recent research progresses. A recently proposed RPA decoder achieves near ML decoding performance, when decoding low rate and short length RM codes. The RPA decoder can be implemented in parallel to reduce the decoding latency, which is friendly for hardware implementations. But its high computational complexity due to the recursive structure stops it from using in applications that have a limited computation resource and energy budget. Several modifications have been made in the RPA decoder and result in several low complexity variants for the RPA decoder.

This work proposed techniques to further reduce the complexity of the RPA decoder and optimized the low complexity variant of the CPA decoder. Also, approximations are applied to the projection and aggregation function in the RPA decoder and the CPA decoder, which makes these decoding algorithms more hardware friendly. Also, their list decoder is simplified by replacing the Reed's decoder with a simple syndrome check. This proposed simplification largely reduces the difficulties in implementations, and it reuses the functional part proposed in this work. So, the simplification might infer a reduction in the area for the hardware implementation while having little compromises in decoding performance. In conclusion, techniques proposed in this work help to go one step forward in closing the gap between the projection-aggregation decoders and their practical usage in emerging applications.

## 6.1 Suggestions for Future Work

To compete with other low complexity decoders for RM codes in emerging applications, more works should be done for the RPA and the CPA decoder. There are some suggestions for future works.

### 6.1.1 Near ML decoders with Reduced List Sizes

Due to the relatively high computational complexity of the RPA decoder and the CPA decoder, their list decoders are still expensive to compute. The list decoder generates the test patterns in a similar way used by the Chase list decoder, which is inefficient. Finding an efficient strategies for generating list patterns could be a possible research direction. Given the success of adopting CRC in the SCL decoder for polar codes, it is also interesting to investigate that if the CRC can help the list decoder to approach ML decoding performance with a small list size.

### 6.1.2 Hardware Implementations for the RPA Decoder and Its Variants

Efficient hardware implementations are needed for the soft decision RPA decoder and the CPA decoder. Proper selections of complexity reduction techniques for the hardware implementation should be investigated. Good hardware designs are also need to meet the performance requirement demanded by emerging applications.

# References


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

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, pp. 379–423, Jul. 1948.

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

[4] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electron. Letter*, vol. 32, no. 18, p. 1645, 1996.

[5] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[6] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[7] 3GPP, "NR; Multiplexing and channel coding," Tech. Rep. TS 38.212, Jun. 2018.

[8] E. Arikan, "A survey of Reed-Muller codes from polar coding perspective," in *2010 IEEE Information Theory Workshop on Information Theory (ITW 2010, Cairo)*, 2010, pp. 1–5.

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

[10] O. Sberlo and A. Shpilka, "On the performance of Reed-Muller codes with respect to random errors and erasures," in *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '20. USA: Society for Industrial and Applied Mathematics, 2020, p. 1357–1376.

[11] E. Abbe, A. Shpilka, and A. Wigderson, "Reed–Muller codes for random erasures and errors," *IEEE Transactions on Information Theory*, vol. 61, no. 10, pp. 5229–5252, 2015.

[12] E. Abbe and M. Ye, "Reed-Muller codes polarize," in *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 2019, pp. 273–286.

[13] E. Arikan, H. Kim, G. Markarian, U. Ozgiir, and E. Poyraz, "Performance of short polar codes under ML decoding," in *Proc. ICT Mobile Summit 2009, (Santander, Spain)*, 10-12 June 2009.

[14] G. Reeves and H. D. Pfister, "Reed–Muller codes achieve capacity on BMS channels," 2021, arXiv:2110.14631.

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

[16] E. Abbe, A. Shpilka, and M. Ye, "Reed-Muller codes: theory and algorithms," *IEEE Transactions on Information Theory*, pp. 1–1, 2020.

[17] R. R. Green, "A serial orthogonal decoder," *JPL Space Programs Summary*, vol. 37, pp. 247–253, 1966.

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

[19] V. M. Sidel'nikov and A. S. Pershakov, "Decoding of Reed-Muller codes with a large number of errors," *Problemy peredachi informatsii*, vol. 28, no. 3, pp. 80–94, 1992.

[20] Bassem Sakkour, "Decoding of second order Reed-Muller codes with a large number of errors," in *IEEE Information Theory Workshop, 2005.*, 2005, pp. 3 pp.–.

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

[22] ——, "Soft-decision decoding of Reed-Muller codes: a simplified algorithm," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 954–963, 2006.

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

[24] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[25] H. Chen, R. Abbas, P. Cheng, M. Shirvanimoghaddam, W. Hardjawana, W. Bao, Y. Li, and B. Vucetic, "Ultra-reliable low latency cellular networks: Use cases, challenges and approaches," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 119–125, 2018.

[26] M. Lian, C. Häger, and H. D. Pfister, "Decoding Reed–Muller codes using redundant code constraints," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 42–47.

[27] D. Fathollahi, N. Farsad, S. A. Hashemi, and M. Mondelli, "Sparse multi-decoder recursive projection aggregation for Reed-Muller codes," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1082–1087.

[28] Q. Huang and B. Zhang, "Pruned collapsed projection-aggregation decoding of Reed-Muller codes," 2021, arXiv:2105.11878.

[29] J. Li, S. M. Abbas, T. Tonnellier, and W. J. Gross, "Reduced complexity RPA decoder for Reed-Muller codes," in *2021 IEEE International Symposium on Topics in Coding(ISTC),*, 2021.

[30] J. Li and W. J. Gross, "Optimization and simplification of PCPA decoderfor Reed-Muller codes," unpublished.

[31] R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding.* West Sussex, England: John Wiley & Sons, 2006.

[32] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms.* USA: Wiley-Interscience, 2005.

[33] K. Ivanov and R. Urbanke, "Improved decoding of second-order Reed-Muller codes," in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[34] M. Kamenev, Y. Kameneva, O. Kurmaev, and A. Maevskiy, "A new permutation decoding method for Reed-Muller codes," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 26–30.

[35] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. t. Brink, "Automorphism ensemble decoding of Reed–Muller codes," *IEEE Transactions on Communications*, vol. 69, no. 10, pp. 6424–6438, 2021.

[36] E. Santi, C. Hager, and H. D. Pfister, "Decoding Reed-Muller codes using minimum-weight parity checks," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1296–1300.

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

[38] K. Ivanov and R. Urbanke, "Permutation-based decoding of Reed-Muller codes in binary erasure channel," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 21–25.

[39] N. Doan, S. A. Hashemi, M. Mondelli, and W. J. Gross, "Decoding Reed-Muller codes with successive factor-graph permutations," 2021, arXiv: 2109.02122.

[40] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[41] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad, "A simple effective heuristic for embedded mixed-integer quadratic programming," in *2016 American Control Conference (ACC)*, 2016, pp. 5619–5625.

[42] C. Bliek, P. Bonami, and A. Lodi, "Solving mixed-integer quadratic programming problems with IBM-CPLEX : a progress report," in *Proceedings of the Twenty-Sixth RAMP Symposium*, Hosei University, Tokyo, Oct. 16-17 2014.

[43] M. Lian, "Belief propagation with deep unfolding for high-dimensional inference in communication systems," Ph.D. dissertation, Duke Univ, Durham, North Carolina, 2019. [Online]. Available: https://dukespace.lib.duke.edu/dspace/handle/10161/20148

[44] A. Shokrollahi, "LDPC codes: An introduction. in: Feng K., Niederreiter H., Xing C. (eds) Coding, Cryptography and Combinatorics." *Progress in Computer Science and Applied Logic*, vol. 23, 2004.

[45] M. Hashemipour-Nazari, K. Goossens, and A. Balatsoukas-Stimming, "Hardware implementation of iterative projection-aggregation decoding of Reed-Muller codes," 2020, arXiv:2012.00581.

[46] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Elsevier, 1977.

[47] W. J. Gross and P. G. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders," *Electronics Letters*, vol. 34, no. 16, pp. 1577–1578, 1998.

[48] Jung-Fu Cheng and T. Ottosson, "Linearly approximated log-MAP algorithms for turbo decoding," in *VTC2000-Spring. 2000 IEEE 51st Vehicular Technology Conference Proceedings (Cat. No.00CH37026)*, vol. 3, 2000, pp. 2252–2256 vol.3.

[49] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[50] J. J. E. Kelley, "The cutting-plane method for solving convex programs," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, Dec. 1960.

[51] MATLAB, "Mixed-integer quadratic programming portfolio optimization: Problem-based." [Online]. Available: https://www.mathworks.com/help/optim/ug/miqp-portfolio-problem-based.html

[52] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

# Appendix A

## A.1 Equivalent Forms for the Projection and the Aggregation Function

**1-dimensional sub-spaces**

*Projection*: $\boldsymbol{L}(\boldsymbol{z}_1)$, $\boldsymbol{L}(\boldsymbol{z}_2)$ denote the 2 elements in the same coset $T$.

$$\begin{aligned} 2\tanh^{-1}(\prod_{i=1}^{2}\tanh(\frac{\boldsymbol{L}(\boldsymbol{z}_i)}{2})) &= 2\tanh^{-1}(\prod_{i=1}^{2}\frac{(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}{(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)}) \\ &= \ln(\frac{\prod_{i=1}^{2}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)+\prod_{i=1}^{2}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}{\prod_{i=1}^{2}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)-\prod_{i=1}^{2}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}), \end{aligned} \tag{A.1}$$

where

$$\prod_{i=1}^{2}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1) = \exp(\sum_{i=1}^{2}\boldsymbol{L}(\boldsymbol{z}_i)) + \sum_{i=1}^{2}\exp(\boldsymbol{L}(\boldsymbol{z}_i)) + 1, \tag{A.2}$$

and

$$\prod_{i=1}^{2}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1) = \exp(\sum_{i=1}^{2}\boldsymbol{L}(\boldsymbol{z}_i)) - \sum_{i=1}^{2}\exp(\boldsymbol{L}(\boldsymbol{z}_i)) + 1. \tag{A.3}$$

Thus,

$$2\tanh^{-1}(\prod_{i=1}^{2}\tanh(\frac{\boldsymbol{L}(\boldsymbol{z}_i)}{2})) = \ln(\exp(\sum_{i=1}^{2}\boldsymbol{L}(\boldsymbol{z}_i))+1) - \ln(\sum_{i=1}^{2}\exp(\boldsymbol{L}(\boldsymbol{z}_i))), \tag{A.4}$$

which is the equation (12) in [1].

*Aggregation*: For arbitrary $\boldsymbol{L}(\boldsymbol{z})$, $\boldsymbol{L}(\boldsymbol{z}_1)$ denote the other element $\in T$.

$$-1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)}(2\tanh^{-1}(\tanh(\frac{\boldsymbol{L}(\boldsymbol{z}_1)}{2}))) = -1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)}\boldsymbol{L}(\boldsymbol{z}_1), \tag{A.5}$$

which is the aggregation function for the RPA decoder [1].

**2-dimensional sub-spaces**

*Projection*: $\boldsymbol{L}(\boldsymbol{z}_1)$, $\boldsymbol{L}(\boldsymbol{z}_2)$, $\boldsymbol{L}(\boldsymbol{z}_3)$, $\boldsymbol{L}(\boldsymbol{z}_4)$ denote the 4 elements in the same coset $T$.

$$\begin{aligned} 2\tanh^{-1}(\prod_{i=1}^{4}\tanh(\frac{\boldsymbol{L}(\boldsymbol{z}_i)}{2})) &= 2\tanh^{-1}(\prod_{i=1}^{4}\frac{(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}{(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)}) \\ &= \ln(\frac{\prod_{i=1}^{4}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)+\prod_{i=1}^{4}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}{\prod_{i=1}^{4}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)-\prod_{i=1}^{4}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}), \end{aligned} \tag{A.6}$$

where

$$\begin{aligned} \prod_{i=1}^{4}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1) = \exp(\sum_{i=1}^{4}\boldsymbol{L}(\boldsymbol{z}_i)) + \sum_{i=1}^{4}\exp(\sum_{j\in[4]\setminus\{i\}}\boldsymbol{L}(\boldsymbol{z}_j)) + \sum_{1\le i\le j\le 4}\exp(\boldsymbol{L}(\boldsymbol{z}_i)+\boldsymbol{L}(\boldsymbol{z}_j)) \\ + \sum_{i=1}^{4}\exp(\boldsymbol{L}(\boldsymbol{z}_i)) + 1, \end{aligned} \tag{A.7}$$

and

$$\begin{aligned} \prod_{i=1}^{4}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1) = \exp(\sum_{i=1}^{4}\boldsymbol{L}(\boldsymbol{z}_i)) - \sum_{i=1}^{4}\exp(\sum_{j\in[4]\setminus\{i\}}\boldsymbol{L}(\boldsymbol{z}_j)) + \sum_{1\le i\le j\le 4}\exp(\boldsymbol{L}(\boldsymbol{z}_i)+\boldsymbol{L}(\boldsymbol{z}_j)) \\ - \sum_{i=1}^{4}\exp(\boldsymbol{L}(\boldsymbol{z}_i)) + 1. \end{aligned} \tag{A.8}$$

Thus,

$$2\tanh^{-1}(\prod_{i=1}^{4}\tanh(\frac{\boldsymbol{L}(\boldsymbol{z}_i)}{2})) = \ln(\exp(\sum_{i=1}^{4}\boldsymbol{L}(\boldsymbol{z}_i)) + \sum_{1\leq i\leq j\leq 4}\exp(\boldsymbol{L}(\boldsymbol{z}_i)+\boldsymbol{L}(\boldsymbol{z}_j))+1) \\ - \ln(\sum_{i=1}^{4}\exp(\sum_{j\in[4]\setminus\{i\}}\boldsymbol{L}(\boldsymbol{z}_j)) + \sum_{i=1}^{4}\exp(\boldsymbol{L}(\boldsymbol{z}_i))), \tag{A.9}$$

which is the equation (13) in [1].

*Aggregation*: For arbitrary $\boldsymbol{L}(\boldsymbol{z})$, $\boldsymbol{L}(\boldsymbol{z}_1)$, $\boldsymbol{L}(\boldsymbol{z}_2)$, $\boldsymbol{L}(\boldsymbol{z}_3)$ denote the other 3 elements $\in T$.

$$-1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)}(2\tanh^{-1}(\prod_{i=1}^{3}\tanh(\frac{L(\boldsymbol{z}_i)}{2}))) = -1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)}(2\tanh^{-1}(\prod_{i=1}^{3}\frac{(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)}{(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)})) \\ = -1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i}(T)}(\ln(\prod_{i=1}^{3}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)+\prod_{i=1}^{3}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1)) \\ - \ln(\prod_{i=1}^{3}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1)-\prod_{i=1}^{3}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1))), \tag{A.10}$$

where

$$\prod_{i=1}^{3}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1) = \exp(\sum_{i=1}^{3}\boldsymbol{L}(\boldsymbol{z}_i)) + \sum_{i=1}^{3}\exp(\sum_{j\in[3]\setminus\{i\}}\boldsymbol{L}(\boldsymbol{z}_j)) + \sum_{i=1}^{3}\exp(\boldsymbol{L}(\boldsymbol{z}_i))+1, \tag{A.11}$$

and

$$\prod_{i=1}^{3}(\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1) = \exp(\sum_{i=1}^{3}\boldsymbol{L}(\boldsymbol{z}_i)) - \sum_{i=1}^{3}\exp(\sum_{j\in[3]\setminus\{i\}}\boldsymbol{L}(\boldsymbol{z}_j)) + \sum_{i=1}^{3}\exp(\boldsymbol{L}(\boldsymbol{z}_i))-1. \tag{A.12}$$

Thus,

$$-1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i(T)}}(2\tanh^{-1}(\prod_{i=1}^{3}\tanh(\frac{L(\boldsymbol{z}_i)}{2}))) = -1^{\hat{\boldsymbol{y}}_{/\mathbb{B}_i(T)}}(\ln(\exp(\sum_{i=1}^{3}\boldsymbol{L}(\boldsymbol{z}_i)) + \sum_{i=1}^{3}\exp(\boldsymbol{L}(\boldsymbol{z}_i))) - \ln(\sum_{i=1}^{3}\exp(\sum_{j\in[3]\setminus\{i\}}\boldsymbol{L}(\boldsymbol{z}_j)) + 1)), \tag{A.13}$$

which is equivalent to the equation (14) in [1].

## A.2 Parameters for Optimization Methods

**Table A.1** CP parameters with rng(1).

| | RM(7, 3) | | | RM(7, 4) | | | RM(7, 5) | | | RM(8, 3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert S\rvert$ | 64 | 128 | 256 | 64 | 128 | 256 | 64 | 128 | 256 | 64 | 128 | 256 |
| $f_{\min}$ | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.001 | 0.0001 | 0.0001 | 0.0001 | 0.001 | 0.0001 | 0.001 |
| $f_{\max}$ | $\frac{1}{\lvert S\rvert-10}$ | $\frac{1}{\lvert S\rvert-10}$ | $\frac{1}{\lvert S\rvert-10}$ | $\frac{1}{\lvert S\rvert-10}$ | $\frac{1}{\lvert S\rvert-10}$ | $\frac{1}{\lvert S\rvert-10}$ | $\frac{1}{\lvert S\rvert-5}$ | $\frac{1}{\lvert S\rvert-5}$ | $\frac{1}{\lvert S\rvert-5}$ | $\frac{1}{\lvert S\rvert-15}$ | $\frac{1}{\lvert S\rvert-5}$ | $\frac{1}{\lvert S\rvert-10}$ |
| diff | $5e^{-2}$ | $1e^{-3}$ | $1e^{-4}$ | $5e^{-4}$ | $5e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $5e^{-2}$ | $5e^{-2}$ | $5e^{-2}$ |
| iteration | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 40 | 40 | 40 |

**Table A.2** ADMM parameters with rng(1).

| | RM(7, 3) | | | RM(7, 4) | | | RM(7, 5) | | | RM(8, 3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert S\rvert$ | 64 | 128 | 256 | 64 | 128 | 256 | 64 | 128 | 256 | 64 | 128 | 256 |
| maxiter | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 |
| repeat | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 | 200 |
| $\rho$ | 0.2 | 0.5 | 0.6 | 0.2 | 0.1 | 0.3 | 0.05 | 0.01 | 0.006 | 0.006 | 0.006 | 0.007 |
| resthr | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ | $1e^{-4}$ |