Title

Low-Complexity Decoding of Low-Density Parity Check Codes Through Optimal Quantization and Machine Learning and Optimal Modulation and Coding for Short Block-Length Transmissions

Author

Wang, Linfang

Publication Date

2023

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Low-Complexity Decoding of Low-Density Parity Check Codes Through Optimal Quantization and Machine Learning and Optimal Modulation and Coding for Short Block-Length Transmissions

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical & Computer Engineering

by

Linfang Wang

2023

ABSTRACT OF THE DISSERTATION

Low-Complexity Decoding of Low-Density Parity Check Codes
Through Optimal Quantization and Machine Learning
and Optimal Modulation and Coding for Short Block-Length Transmissions

by

Linfang Wang
Doctor of Philosophy in Electrical & Computer Engineering
University of California, Los Angeles, 2023
Professor Richard D. Wesel, Chair

This dissertation investigates two topics in channel coding theory: low-complexity decoder design for low-density parity-check (LDPC) codes and reliable communication in the short blocklength regime.

For the first topic, we propose a finite-precision decoding method that features the three steps of Reconstruction, Computation, and Quantization (RCQ). The parameters of the RCQ decoder, for both the flooding-scheduled and the layered-scheduled, can be designed efficiently using discrete density evolution featuring hierarchical dynamic quantization (HDQ). To further reduce the hardware usage of the RCQ decoder, we propose a second RCQ framework called weighted RCQ (W-RCQ). Unlike the RCQ decoder, whose quantization and reconstruction parameters change in each layer and iteration, the W-RCQ decoder limits the number of quantization and reconstruction functions to a very small number during the decoding process, for example, three or four. However, the W-RCQ decoder weights check-to-variable node messages using dynamic parameters optimized by a quantized neural

network. The proposed W-RCQ decoder uses fewer parameters than the RCQ decoder, thus requiring much fewer resources such as lookup tables.

For the second topic, we apply probabilistic amplitude shaping (PAS) to cyclic redundancy check (CRC)-aided tail-biting trellis-coded modulation (TCM). CRC-TCM-PAS produces practical codes for short block lengths on the additive white Gaussian noise (AWGN) channel. In the transmitter, equally likely message bits are encoded by a distribution matcher (DM), generating amplitude symbols with a desired distribution. A CRC is appended to the sequence of amplitude symbols, and this sequence is then encoded and modulated by TCM to produce real-valued channel input signals. We prove that the sign values produced by the TCM are asymptotically equally likely to be positive or negative. The CRC-TCM-PAS scheme can thus generate channel input symbols with a symmetric capacity-approaching probability mass function. We also provide an analytical upper bound on the frame error rate of the CRC-TCM-PAS system over the AWGN channel. This FER upper bound is the objective function for jointly optimizing the CRC and convolutional code. This paper also proposes a multi-composition DM, a collection of multiple constant-composition DMs. The optimized CRC-TCM-PAS systems achieve frame error rates below the random coding union (RCU) bound in AWGN and outperform the short-blocklength PAS systems with various other forward error correction codes.

The dissertation of Linfang Wang is approved.

Lara Dolecek

Gregory J. Pottie

Dariush Divsalar

Christina Fragouli

Richard D. Wesel, Committee Chair

University of California, Los Angeles

2023

To my parents, Genqi and Xiangqun.

To my wife, my dear Hanzhi (Stephanie).

To my cat, Ona

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am filled with utmost gratitude and appreciation towards my advisor, Prof. Richard Wesel. Our initial meeting, where you interviewed me as your doctoral student, still holds a special place in my heart. Thank you for introducing me to the world of coding theory and information theory. Your approach of never pressuring me to do any work allowed me to truly enjoy the research work and the entire doctoral student experience. Your unwavering support and patience have helped me build my confidence in this field and even in my life. Your passion for research, work, and life has also influenced my attitude positively.

I would like to thank all faculties and staff members of the Electrical & Computer Engineering Department. I would like to Prof. Richard Wesel for the channel coding class, I took your class in my first quarter, and your class was so informative and helpful, even though I could not understand all the class contents because of my poor English at the time. I would like to thank Prof. Suhas Diggavi for your amazing information theory class. I had taken an information class twice before I came to UCLA, but I didn't learn the beauty of the information theory class until I had your class. I would like to thank Deeona Columbia, Ryo Arreola. Thank you for helping me numerous times.

I would like to thank Prof. Dariush Divsalar, your expertise in LDPC code construction is invaluable. I would like to thank Dr. Maximilian Stark, whose collaboration on the IB decoder was an enjoyable journey. His ideas and insights have been truly inspiring, and I am grateful for the opportunity to have learned from him. I would like to thank Thomas Wiegart of the Technical University of Munich for his passion, persistence, and optimism in research. It is always a pleasure to work with you, from the probabilistic permutation shaping to shaping using the generator matrix. Finally, I would like to thank Dr. Gianluigi Liva, Prof. Giuseppe Durisi, and Prof. Gonzalo Vazquez-Vilar, thanks for your support on the finite-length bound calculation.

I would like to thank my lab mates, Hengjie Yang, Amaael Antonini, and Semira Galija-

sevic, for their support throughout my Ph.D. Hengjie, your meticulous attitude to research influenced me all the way. I would like to thank all of my undergraduate collaborators, Jonathan Nguyen, Caleb Terrill, Sean Chen, Felipe Areces, and Chester Hulse. Caleb, thanks for helping me with building the FPGA of the RCQ decoder. You helped me realize my dream. I would like to thank Fan Zhang at SK Hynix for giving me my first internship opportunity. This internship shaped part of my Ph.D. goal to do something real rather than only fancy ideas. I would like to thank Rekha Pitchumani at Samsung Semiconductor for providing me with the internship opportunity to explore the application of RCQ decoder in storage systems. I would like to thank Zongwang Li for providing excellent LDPC codes. My third internship was with Jung Bae, Hamid Saber, Homayoon Hatami, and Vahid Jamali at Samsung Semiconductor. I have wanted to do deep-learning-based channel codes. Thanks for providing me the opportunity to study the turbo autoencoder, and I do enjoy the working atmosphere. You guys are really passionate and devoted to the research. Finally, I would like to thank Ganning Yang, Qiuliang Xie, Yabo Li, and Chia-Ming Lou at MediaTek, thank you for bringing me to the world of industry, and I am really excited about our current research project.

I would like to thank my friends in Los Angeles, Yixin Chen, Jiawei Zhang, Dezhan Tu, Hengjie Yang, Bijie Bai, and several others who have been a constant pillar of strength in my life. Their presence in my life has been a source of immense comfort and motivation, and I feel blessed to have them by my side.

I am incredibly grateful to my fiancee, Hanzhi (Stephanie) Xia, for all the support and comfort she provided me during a particularly stressful time. Stephanie, thank you for bringing so much happiness to my life. I could not have completed this dissertation without your invaluable assistance.

Finally, I would like to thank my parents and my brother, without whom I can not complete my Ph.D. degree. Thank you for supporting and accompanying me all the way through.

VITA

2018-2023	Graduate Research Assitant, University of California, Los Angeles
2019	Research Engineering Intern, SK Hynix.
2020	Teaching Assitant, University of California, Los Angeles
2020	Research Engineering Intern, Samsung Semiconductor, Inc.
2021	Teaching Assitant, University of California, Los Angeles
2022	Research Engineering Intern, Samsung Semiconductor, Inc.
2023	Teaching Assitant, University of California, Los Angeles
2023	Research Engineering Intern, Mediatek.

PUBLICATIONS

L. Wang, D. Song, F. Areces, T. Wiegart and R. D. Wesel, "Probabilistic Shaping for Trellis-Coded Modulation With CRC-Aided List Decoding," in *IEEE Transactions on Communications*, vol. 71, no. 3, pp. 1271-1283, March 2023.

L. Wang, C. Terrill, M. Stark, Z. Li, S. Chen, C. Hulse, C. Kuo, R. Wesel, G. Bauch, R. Pitchumani, "Reconstruction-Computation-Quantization (RCQ): A Paradigm for Low Bit Width LDPC Decoding," in *IEEE Transactions on Communications*, vol. 70, no. 4, pp. 2213-2226, April 2022.

L. Wang, S. Dan, F. Areces, and R. D. Wesel. "Achieving Short-Blocklength RCU bound via CRC List Decoding of TCM with Probabilistic Shaping." *IEEE International Conference on Communications (ICC)*, Seoul, South Korea, May 16–20, 2022.

L. Wang, S. Chen, J. Nguyen, D. Dariush, and R. Wesel, "Neural-Network-Optimized Degree-Specific Weights for LDPC MinSum Decoding", *IEEE 11th International Symposium on Topics in Coding (ISTC)*, Aug.30 - Sept. 3, virtual conference, 2021.

L. Wang, R. D. Wesel, M. Stark, and G. Bauch, "A Reconstruction-Computation-Quantization (RCQ) Approach to Node Operations in LDPC Decoding", *IEEE Global Communications Conference (GLOBECOM)*, Taipei, Taiwan, Dec. 8-10, 2020.

M. Stark, **L. Wang**, G. Bauch, and R. D. Wesel, (2020). "Decoding Rate-Compatible 5G-LDPC Codes with Coarse Quantization Using the Information Bottleneck Method", *IEEE Open Journal of the Communications Society*, vol. 1, pp. 646-660, 2020.

CHAPTER 1

Introduction

Channel codes, which correct errors in data, are essential for our networked world. Wi-Fi, 5G cell phone communication, flash memories, and hard disk drives rely on channel codes for reliability. This dissertation studies the following channel coding problems:

1. The low-complexity decoder design for low-density parity check (LDPC) code.

2. The reliable communication system in the short-blocklength regime.

LDPC [Gal62a] codes have been implemented broadly, including the NAND flash and wireless communication systems. The sum-product decoder, which is considered as the most powerful decoder for the LDPC code, is kept from practical use due to its high computation complexity. The low-complexity approximations of sum-product decoders, such as the normalized Min Sum (NMS) decoder and offset Min Sum (OMS) decoder, suffer from the error correction performance loss when using small bit width for the messages of the decoder, i.e., less than 5 bits. As a result, LDPC decoders with low complexity and excellent decoding performance are desired in practical communication systems with limited hardware resources such as area and routing capacity.

In recent years, extensive research on non-uniformly quantized decoders has shown that, by quantizing the messages of the decoders smartly, the decoders can deliver excellent error correction performances with a low bit width of less than 5 bits. There are two popular ideas in the research of non-uniformly quantized decoders. The first type of decoders, such as Vasic's finite alphabet iterative decoders (FAID) [PDD13b] and Lewandowsky's Information-

Bottleneck (IB) [LB18a] decoder, convert all the arithmetic operations to lookup operations by designing the look-up tables (LUTs) with low-bitwidth input and output messages. The second type of non-uniformly quantized decoder designs the non-uniform quantizers and dequantizers for the messages in the decoder, such that the messages have a fine resolution when used for computation and have a coarse resolution when transmitted between the check node units and the variable node units. One famous decoder that uses the second idea is the mutual-information-maximization (MIM) quantized belief propagation (QBP) decoder in [LT05a].

chapter 2 generalizes the framework in [LT05b] and proposes a finite-precision LDPC decoding method that features the three steps of Reconstruction, Computation, and Quantization (RCQ). Unlike the MIM-QBP, which is an approximation of the sum-product decoder, RCQ decoders can be an approximation of either a box-plus decoder (*bp-RCQ*) or a Min-Sum decoder (*ms-RCQ*). As an iterative message-passing decoder, the RCQ decoder can be flooding-scheduled or layered-scheduled. In the flooding-scheduled decoder, the reconstruction and quantization modules are updated in each iteration; In the layered-scheduled decoder, reconstruction and quantization modules are updated in each iteration and layer. This chapter also presents using discrete density evolution featuring hierarchical dynamic quantization (HDQ) to design the parameters of RCQ decoder efficiently.

chapter 3 studies how to reduce further the number of parameters required by the RCQ decoder with the help of degree-specific weights optimized by a neural network. The layered-RCQ decoder needs to update the quantization and reconstruction function parameters in each layer at each iteration. For the LDPC codes with many layers or requiring many iterations, the extra resources for the RCQ decoder parameters may offset the benefit of its low-bitwidth messages. Hence, reducing the number of quantization and reconstruction pairs of the RCQ decoder is desirable.

The conventional NMS decoder and OMS decoder use a single parameter (or weight) throughout the decoding process. Recent research has shown that the decoding perfor-

mance of those decoders can be boosted by assigning the weight dynamically to each edge in each iteration [NBB16b]. Such decoders are called neural NMS (N-NMS) or OMS (N-OMS) decoders, as the weights can be optimized by training the neural network obtained by unfolding the NMS or OMS decoders. N-NMS and N-OMS decoders are impractical for long-blocklength LDPC codes due to the huge number of weights. Chapter 3 shows that the neural decoders can be significantly simplified by assigning the iteration-specific weights based on the check and variable node degree with the same decoding performance as the decoders that assign distinct weights to each edge. The simplified decoder is named the neural 2-dimensional (2D) neural NMS (N-2D-NMS) or OMS (N-2D-OMS) decoder.

With the help of the neural 2D decoder, **chapter 3** proposes a novel RCQ framework called weighted-RCQ (W-RCQ). Unlike the RCQ decoder, whose quantization and reconstruction parameters change in each layer and iteration, the W-RCQ decoder limits the number of quantizer/quantizer pairs to a very small number, for example, four or fewer. However, the W-RCQ decoder weights check-to-variable messages using dynamic parameters optimized via a quantized NN (QNN). The proposed W-RCQ decoder uses fewer parameters than the RCQ decoder, thus requiring much fewer LUTs. Simulations in chapter 3 for a (9472,8192) LDPC code on a field-programmable gate array (FPGA) device show that the 4-bit W-OMS-RCQ decoder delivers comparable decoding performance but with much fewer hardware resources, compared with the 4-bit RCQ decoder and the 5-bit OMS decoder.

The second topic of the dissertation studies reliable communications over the additive white Gaussian noise (AWGN) channel with high spectral efficiency for short block lengths. To closely approach theoretical limits, it is helpful to use shaping so that signal points are not equally likely, not equally spaced, or both [Gal68, FGL84, For92, KP93, LFT94, FWS01, XWS21]. Recently, a new transmission framework called probabilistic amplitude shaping (PAS) [BSS15, BSS19] is proposed. PAS employs a distribution matcher (DM) [SB15] before a forward error correction (FEC) encoder and channel-signaling mapping function to accomplish optimal or almost optimal shaping.

Chapter 4 presents a PAS system for the AWGN channel in the short-blocklength regime. In the proposed PAS system, a DM takes the binary information bits and generates the magnitude symbols with the desired probability mass function. The popular distribution matcher, constant composition DM, performs excellently in the long blocklength regime but not in the short blocklength regime. This dissertation proposes a multi-composition DM, which delivers a satisfying performance at the short block length. Then, the output of the DM is encoded by a CRC-aided, rate-$\frac{k_0}{k_0+1}$, systematic, recursive tail-biting convolutional code (TBCC) and modulated via a channel-signal mapping function. The TBCC and the channel-signal mapping function constitute the TCM [Ung82]. The proposed PAS system is also referred to as the CRC-TCM-PAS system. The optimized CRC-TCM-PAS systems achieve FERs below the random-coding union (RCU) bound in AWGN and outperform the short-blocklength PAS systems with various other forward error correction codes studied in [CDJ19].

Finally, **Chapter 5** summarizes the dissertation and points out promising future directions.

CHAPTER 2

Reconstruction-Computation-Quantization (RCQ): A Paradigm for Low Bit Width LDPC Decoding

2.1 Introduction

Low-Density Parity-Check (LDPC) codes [Gal62b] have been implemented broadly, including in NAND flash systems and wireless communication systems. Message-passing algorithms such as belief propagation (BP) and Min Sumare utilized in LDPC decoders. In practice, decoders with low message bit widths are desired when considering the limited hardware resources such as area, routing capabilities, and power utilization of FPGAs or ASICs. Unfortunately, low bit width decoders with uniform quantizers typically suffer a large degradation in decoding performance [LT05a]. On the other hand, the iterative decoders that allow for the dynamic growth of message magnitudes can achieve improved performance [ZS14].

LDPC decoders that quantize messages non-uniformly have gained attention because they provide excellent decoding performance with low bit width message representations. One family of non-uniform LDPC decoders use lookup tables (LUTs) to replace the mathematical operations in the check node (CN) unit and/or the variable node (VN) unit. S. K. Planjery *et al.* propose finite alphabet iterative decoders (FAIDs) for regular LDPC codes in [PDD13a, DVP13], which optimize a *single* LUT to describe VN input/output behavior. In [PDD13a] a FAID is designed to tackle certain trapping sets and hence achieves a lower error floor than BP on the binary symmetric channel (BSC). Xiao *et al.* optimize the parameters of FAID using a recurrent quantized neural network (RQNN) [XVT19a, XVT20a], and the simulation

results show that RQNN-aided linear FAIDs are capable of surpassing floating-point BP in the waterfall region for regular LDPC codes.

Note that the size of the LUTs in [PDD13a, DVP13, XVT19a, XVT20a] describing VN behavior are an exponential function with respect to node degree. Therefore, these FAIDs can only handle regular LDPC codes with small node degrees. For codes with large node degrees, Kurkoski *et al.* develop a mutual-information-maximization LUT (MIM-LUT) decoder in [RK16], which decomposes a single LUT with multiple inputs into a series of concatenated 2×1 LUTs, each with two inputs and one output. This decomposition makes the number of LUTs linear with respect to node degree, thus significantly reducing the required memory. The MIM-LUT decoder performs lookup operations at both the CNs and VNs. The 3-bit MIM-LUT decoder shows a better FER than floating-point BP over the additive white Gaussian noise (AWGN) channel. As the name suggests, the individual 2×1 LUTs are designed to maximize mutual information [KY14].

Lewandowsky *et al.* use the information bottleneck (IB) machine learning method to design LUTs and propose an IB decoder for regular LDPC codes. As with MIM-LUT, IB decoders also use 2×1 LUTs at both CNs and VNs. Stark *et al.* extend the IB decoding structure to support irregular LDPC codes through the technique of message alignment [SLB18, Sta21]. The IB decoder shows an excellent performance on a 5G LDPC code [SBW20a, SWB20]. In order to reduce the memory requirement for LUTs, Meidlinger *et al.* propose the Min-IB decoder, which replaces the LUTs at CNs with label-based min operation [MBB15, MM17, MMB20, GBM18].

Because the decoding requires only simple lookup operations, the LUT-based decoders deliver high throughput. However, the LUT-based decoders require significant memory resources when the LDPC code has large degree nodes and/or the decoder has a large predefined maximum decoding iteration time, where each iteration requires its own LUTs. The huge memory requirement for numerous large LUTs prevents these decoders from being viable options when hardware resources are constrained to a limited number of LUTs.

Lee *et al.* [LT05a] propose the mutual information maximization quantized belief propagation (MIM-QBP) decoder which circumvents the memory problem by designing non-uniform quantizers and reconstruction mappings at the nodes. Both VN and CN operations are simple mappings and fixed point additions in MIM-QBP. He *et al.* in [HCM19a] show how to systematically design the MIM-QBP parameters for quantizers and reconstruction modules. Wang *et al.* further generalize the MIM-QBP structure and propose a reconstruction-computation-quantization (RCQ) paradigm [WWS20a] which allows CNs to implement either the min or boxplus operation.

All of the papers discussed above focus on decoders that use the flooding schedule. The flooding schedule can be preferable when the code length is short. However, in many practical settings such as coding for storage devices where LDPC codes with long block lengths are selected, the flooding schedule requires an unrealistic amount of parallel computation for some typical hardware implementations. Layered decoding [JF05], on the other hand, balances parallel computations and resource utilization for a hardware-friendly implementation that also reduces the number of iterations as compared to a flooding implementation for the same LDPC code.

2.1.1 Contributions

As a primary contribution, this work extends our previous work on RCQ [WWS20a] to provide dynamic quantization that changes with each layer of a layered LDPC decoder, as is commonly used with a protograph-based LDPC code. The original RCQ approach [WWS20a], which uses the same quantizers and reconstructions for all layers of an iteration, suffers from FER degradation and a high average number of iterations when applied to a layered decoding structure. The novelty and contributions in this chapter are summarized as follows:

- *Layer-specific RCQ Decoding structure.* This chapter proposes the layer-specific RCQ

decoding structure. The main difference between the original RCQ of [WWS20a] and the layer-specific RCQ decoder is that layer-specific RCQ designs quantizers and reconstructions for each layer of each iteration. The layer-specific RCQ decoder provides better FER performance and requires a smaller number of iterations than the original RCQ structure with the same bit width. This improvement comes at the cost of an increase in the number of parameters that need to be stored in the hardware.

- *layer-specific RCQ Parameter Design.* This work uses layer-specific discrete density evolution featuring hierarchical dynamic quantization (HDQ) to design the layer-specific RCQ parameters. We refer to this design approach as layer-specific HDQ discrete density evolution. For each layer of each iteration, layer-specific HDQ discrete density evolution separately computes the PMF of the messages. HDQ designs distinct quantizers and reconstructions for each layer of each iteration.

- *FPGA-based RCQ Implementations.* This chapter presents the Lookup Method, the Broadcast Method and the Dribble Method, as alternatives to distribute RCQ parameters efficiently in an FPGA. This chapter verifies the practical resource needs of RCQ through an FPGA implementation of an RCQ decoder using the Broadcast method. Simulation results for a (9472, 8192) quasi-cyclic (QC) LDPC code show that a layer-specific Min SumRCQ decoder with 3-bit messages achieves a more than 10% reduction in LUTs and routed nets and more than a 6% reduction in register usage while maintaining comparable decoding performance, compared to a standard offset Min Sumdecoder with 5-bit messages.

2.1.2 Organization

The remainder of this chapter is organized as follows: Sec. 2.2 introduces the RCQ decoding structure and presents an FPGA implementation of an RCQ decoder. Sec. 2.3 describes HDQ, which is used for channel observation quantization and RCQ parameter design. Sec.

2.5 shows the design of the layer-specific RCQ decoder. Sec. 2.6 presents simulation results including FER and hardware resource requirements. Sec. 2.7 concludes our work.

2.2 The RCQ Decoding Structure

Message passing algorithms update messages between variable nodes and check nodes in an iterative manner either until a valid codeword is found or the maximum number of iterations I_T is reached. The updating procedure of message passing algorithms contains two steps: 1) computation of the output message, 2) communication of the message to the neighboring node. To reduce the complexity of message passing, the computed message is often quantized before being passed to the neighboring node. We refer to the computed messages as the *internal messages*, and communicated messages passed over the edges of the Tanner graph as *external messages*.

For the uniform quantization decoder, the external messages are simply clipped internal messages, in order for a lower routing complexity. However, When external messages are produced by a uniform quantizer, low bit width external messages can result in an early error floor [ZS14]. Non-uniform quantizers, on the other hand, address error floor issue by providing larger message magnitude range. Zhang *et al.* design a $q+1$ quasi-uniform LDPC decoder, where 2^q messages are allocated to uniform quantization, and the other 2^q messages correspond to exponentially growing quantization interval lengths [ZS14]. Thorpe *et al.* introduced a non-uniform quantizer in [LT05a]. Their decoder adds a non-uniform quantizer and a reconstruction mapping to the output and input of the hardware implementation of each node unit. This approach delivers excellent decoding performance even with a low external bit width. The RCQ decoder [WWS20a] can be seen as a generalization of the decoder introduced in [LT05a].

In this section, we provide detailed descriptions of the RCQ decoding structure. Three FPGA implementation methods for realizing the RCQ functionality are also presented.

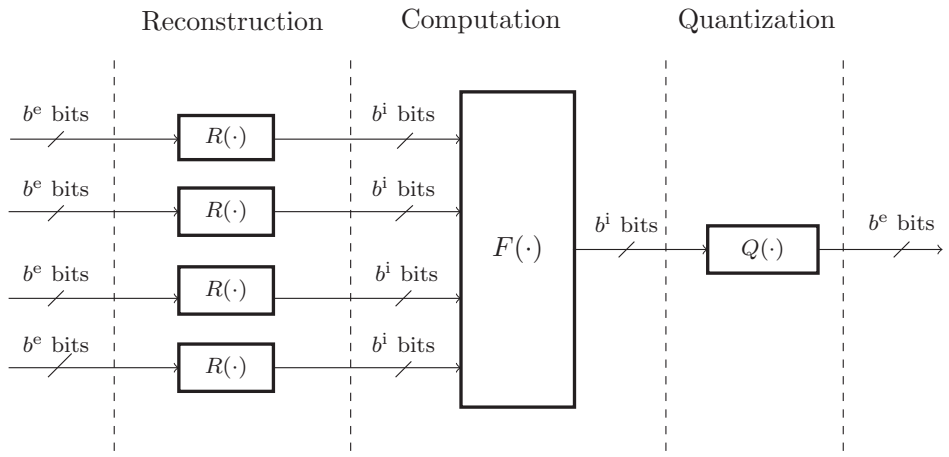


Figure 2.1: Illustration of a generalized RCQ unit which consists of three modules: *Reconstruction* that maps a b^{e}-bit value to a b^{i}-bit value, *Computation* that performs arithmetic operations, and *Quantization* that quantizes a b^{i}-bit value to a b^{e}-bit value.

2.2.1 Generalized RCQ Unit

A generalized RCQ unit as shown in Fig. 2.1 consists of the following three modules:

2.2.1.1 Reconstruction Module

The reconstruction module applies a reconstruction function $R(\cdot)$ to each incoming b^{e}-bit external message to produce a b^{i}-bit internal message, where $b^{\mathrm{i}} > b^{\mathrm{e}}$. We denote the bit width of CN and VN internal message by $b^{\mathrm{i,c}}$ and $b^{\mathrm{i,v}}$, respectively. For the flooding-scheduled RCQ decoder, $R(\cdot)$ is iteration-specific and we use $R_{\mathrm{c}}^{(t)}(\cdot)$ and $R_{\mathrm{v}}^{(t)}(\cdot)$ to represent the reconstruction of check and variable node messages at iteration t, respectively. In the layer-specific RCQ decoder, $R(\cdot)$ uses distinct parameters for each layer in each iteration. We use $R_{\mathrm{c}}^{(t,r)}(\cdot)$ and $R_{\mathrm{v}}^{(t,r)}(\cdot)$ to represent the the reconstruction of check and variable node messages at layer r of iteration t, respectively. The reconstruction functions are mappings of the input external messages to log-likelihood ratios (LLR) that will be used by the node. In this paper, these mappings are systematically designed by HDQ discrete density evolution, which will be

introduced in a later section.

For a quantizer $Q(\cdot)$ that is symmetric, an external message $d \in \mathbb{F}_2^{b^{\mathrm{e}}}$ can be represented as $[d^{\mathrm{MSB}}\ \tilde{d}]$, where $d^{\mathrm{MSB}} \in \{0,1\}$ indicates sign and $\tilde{d} \in \mathbb{F}_2^{b^{\mathrm{e}}-1}$ corresponds to magnitude. We define the magnitude reconstruction function $R^*(\cdot) : \mathbb{F}_2^{b^{\mathrm{e}}-1} \to \mathbb{F}_2^{b^{\mathrm{i}}-1}$, which maps the magnitude of external message, $\tilde{d}$, to the magnitude of internal message. Without loss of generality, we restrict our attention to monotonic reconstruction functions so that

$$R^*(\tilde{d}_1) > R^*(\tilde{d}_2) > 0, \quad \text{for } \tilde{d}_1 > \tilde{d}_2, \tag{2.1}$$

where $\tilde{d}_1$, $\tilde{d}_2 \in \mathbb{F}_2^{b^{\mathrm{e}}-1}$. The reconstruction $R(d)$ can be expressed by $R(d) = \left[d^{\mathrm{MSB}}\ \ R^*(\tilde{d})\right]$. Under the assumption of a symmetric channel, we have $R([0\ \tilde{d}]) = -R([1\ \tilde{d}])$.

2.2.1.2 Computation Module

The computation module $F(\cdot)$ uses the b^{i}-bit outputs of the reconstruction module to compute a b^{i}-bit internal message for the CN or VN output. We denote the computation module implemented in CNs and VNs by F_{c} and F_{v}, respectively. An RCQ decoder implementing the min operation at the CN yields a Min Sum(ms) RCQ decoder. If an RCQ decoder implements belief propagation (bp) via the *boxplus* operation, the decoder is called *bpRCQ*. The computation module, F_{v}, in the VNs is addition for both bpRCQ and msRCQ decoders.

If the RCQ decoder implements the *Min* operation at the check node yielding a MinSum (ms) decoder, i.e.:

$$F_c(h_1, \ldots, h_J) = \prod_j \operatorname{sign}(h_j) \times \min_j |h_j|, \tag{2.2}$$

where $h_j \in \mathbb{F}_2^{b^i}$, $j = 1, ..., J$ are internal messages, then we call the decoder an *msRCQ* decoder.

If an RCQ decoder implements belief propagation (bp) via the *boxplus* operation :

$$F_c(h_1, \ldots, h_J) = h_1 \boxplus h_2 \boxplus ... \boxplus h_J, \tag{2.3}$$

the decoder is called *bpRCQ*. The operator $\boxplus$ is defined as:

$$h_1 \boxplus h_2 = \log\left(\frac{1+e^{h_1+h_2}}{e^{h_1}+e^{h_2}}\right). \tag{2.4}$$

At variable node unit, both msRCQ and bpRCQ decoder sum up all incoming messages:

$$F_v(r_1,\ldots,r_J) = \sum_{j=1}^{J} r_j. \tag{2.5}$$

2.2.1.3 Quantization Module

The quantization module $Q(\cdot)$ quantizes the b^{i}-bit internal message to produce a b^{e}-bit external message. Under the assumption of a symmetric channel, we use a symmetric quantizer that features sign information and a magnitude quantizer $Q^*(\cdot)$. The magnitude quantizer selects one of $2^{b^{\mathrm{e}}-1}-1$ possible indexes using the threshold values $\{\tau_0,\tau_1,...,\tau_{\max}\}$, where $\tau_j \in \mathbb{F}_2^{b^{\mathrm{i}}}$ for $j \in \{0,1,...,2^{b^{\mathrm{e}}-1}-2\}$ and $\tau_{\max}$ is $\tau_{j_{\max}}$ for $j_{\max} = 2^{b^{\mathrm{e}}-1}-2$. We also require

$$\tau_i > \tau_j > 0, \quad i > j. \tag{2.6}$$

Given an internal message $h \in \mathbb{F}_2^{b^{\mathrm{i}}}$, which can be decomposed into sign part h^{MSB} and magnitude part $\tilde{h}$, $Q^*(\tilde{h}) \in \mathbb{F}_2^{b^{\mathrm{e}}-1}$ is defined by:

$$Q^*(\tilde{h}) = \begin{cases} 0, & \tilde{h} \le \tau_0 \\ j, & \tau_{j-1} < \tilde{h} \le \tau_j \\ 2^{b^{\mathrm{e}}-1}-1, & \tilde{h} > \tau_{\max} \end{cases}, \tag{2.7}$$

where $0 < j \le j_{\max}$. Therefore, $Q(h)$ is defined by $Q(h) = [h^{\mathrm{MSB}}\ Q^*(\tilde{h})]$. The super/subscripts introduced for $R(\cdot)$ also apply to $Q(\cdot)$.

2.2.2 Bit Width of RCQ decoder

The three tuple $(b^{\mathrm{e}}, b^{\mathrm{i,c}}, b^{\mathrm{i,v}})$ represents the precision of messages in a RCQ decoder. For the *msRCQ* decoder, it is sufficient to use only the pair $(b^{\mathrm{e}}, b^{\mathrm{i,v}})$ because $b^{\mathrm{i,c}} = b^{\mathrm{e}}$, we simply

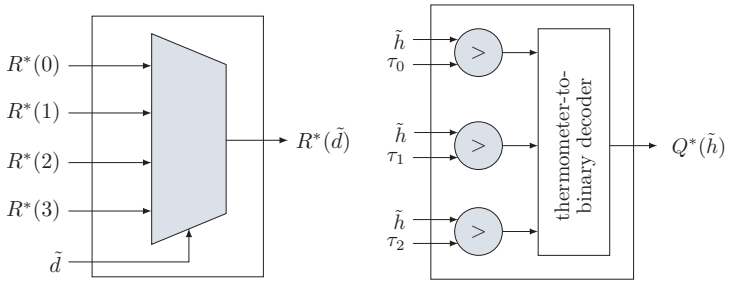
denote $b^{\mathrm{i,v}}$ by b^{v}. The CN min operation computes the XOR of the sign bits and finds the minimum of the extrinsic magnitudes. For a symmetric channel, the min operation can be computed by manipulating the external messages, because the external message delivers the *relative LLR meaning* of reconstructed values. Since we only use external messages to perform the min operation, $R^{\mathrm{c}}(\cdot)$ and $Q^{\mathrm{c}}(\cdot)$ are not needed for the msRCQ decoder. Finally, we use ∞ to denote a floating point representation.

2.2.3 FPGA Implementation for RCQ

The RCQ FPGA decoder may be viewed as a modification to existing hardware decoders based on the BP or MS decoder algorithms, which have been studied extensively [ZDN06, SH16, LZS17, AKK19]. The RCQ decoders require extra $Q(\cdot)$ and $R(\cdot)$ functions to quantize and reconstruct message magnitudes. To implement $Q(\cdot)$ and $R(\cdot)$ functions, we have devised the *Lookup*, *Broadcast*, and *Dribble* methods. These three approaches are functionally identical, but differ in the way that the parameters needed for the $Q(\cdot)$ and $R(\cdot)$ operations are communicated to the nodes.

2.2.3.1 Lookup Method

The quantization and reconstruction functions simply map an input message to an output message. Thus, a simple implementation uses lookup tables implemented using read-only memories (ROMs) to implement all these mappings. As an example, for the iteration-specific magnitude quantizer $Q^{*(t)}(\cdot)$, all iterations can be implemented by a single table indexed by the pair $(\tilde{x}, t)$, where $\tilde{x}$ is the internal message magnitude and t is the current iteration. This index forms an address into a ROM to produce an output $\tilde{y}$. The $Q(\cdot)$ and $R(\cdot)$ functions in every VN require their own ROMs, implemented using block RAMs. If block RAMs with multiple ports are available, then they can be shared by different VN banks to reduce the total amount required. If no ROM sharing occurs, then L VN unit with two ROMs each



(a) (b)

thermometer code	binary form
000	00
001	01
011	10
111	11

(c)

Figure 2.2: msRCQ magnitude reconstruction module (a) and magnitude quantization module (b). In FPGA, magnitude reconstruction module is realized by a multiplexer, and magnitude quantization is realized by comparison functions and a thermometer-to-binary decoder which realizes the mapping relationship shown in (c).

results in a total of $2L$ additional block RAMs used. This amount can be reduced with ROM sharing and other synthesis techniques. Because $Q(\cdot)$ and $R(\cdot)$ change with respect to different iterations and/or layers, one potential drawback of the Lookup method is a large block RAM requirement.

2.2.3.2 Broadcast Method

The Broadcast method provides a scheme where all RCQ parameters are stored centrally in a control unit, instead of being stored in each VN. As an example, for the layered RCQ decoder

whose parameters update every layer and iteration, the pair (t, m), which corresponds to the current iteration and current layer, is used to index into ROMs in the control unit These ROMs output quantization thresholds $\{\tau_0^{(t,m)}, \tau_1^{(t,m)}, \ldots, \tau_{\max}^{(t,m)}\}$ and reconstruction values $\{R^{(i,l)}(0), R^{(t,m)}(1), \ldots, R^{(t,m)}(2^{b^c-1}-1)\}$, which are wired to the VN units. The $Q(\cdot)$ and $R(\cdot)$ blocks in the VN units only take in the parameters for each decoding iteration and layer, and use logic to perform their respective operations. Each VN only takes in the $Q(\cdot)$ and $R(\cdot)$ parameters necessary for decoding the current iteration and layer, and use logic to perform their respective operations. Fig. 2.2 shows an implementation for a 3-bit RCQ, which uses mere 2 bits for magnitude reconstruction and quantization. The 2-bit magnitude reconstruction module is realized by a 4×1 multiplexer. The 2-bit magnitude quantization consists of two steps, first a thermometer code [AV18], where the contiguous ones are analogous to mercury in a thermometer, is generated by comparing the input with all thresholds, and then the thermometer code is converted to the 2-bit binary form by using a thermometer-to-binary decoder, which realizes the mapping relationship in Fig. 2.2c. Two block RAMS are required in the control unit for the thresholds and reconstruction values. Small LUTs in each VN implement the $Q(\cdot)$ and $R(\cdot)$ functions. The main penalty of the Broadcast method is the additional wiring necessary to route the RCQ parameters from the central control unit to the VNs.

The main penalty of the Broadcast is the additional wiring necessary to route the L-msRCQ parameters from the control unit to VN banks. If w bits are used for each of the thresholds and reconstruction values of 3-bit L-msRCQ, a total of $7w$ additional wires need to be routed to each VN unit, w wires for each of the three thresholds and each of the four reconstruction values. With L VN units, the total amount of added routes is $7wL$. For a 4-bit L-msRCQ decoder, the total increase is $15wL$. The same parameters are routed to all the VN units. Thus shared wiring may be used in some cases.

2.2.3.3 Dribble Method

The Dribble method attempts to reduce the number of long wires required by the Broadcast method. Registers in the VNs save the current thresholds and reconstruction values necessary for the $Q(\cdot)$ and $R(\cdot)$ functions. Once again, quantization and reconstruction can be implemented using the logic in Fig. 2.2. When a new set of parameters is required, the bits are transferred (dribbled) one by one or in small batches from the control unit to the VN unit registers. Just as in the Broadcast method, two extra block RAMs and logic for the $Q(\cdot)$ and $R(\cdot)$ functions are required. But where the Broadcast method needs $7w$ additional wires routed to each VN bank for 3-bit L-msRCQ, the Dribble method requires only as many wires as the transfer batch size. The penalty of the Dribble method comes with the extra usage of registers in the VN units. A total of $7w$ bits stored in registers would be necessary in each VN bank to save the current threshold and reconstruction values for 3-bit L-msRCQ. In total, $7wL$ bits of register storage would be used for 3-bit L-msRCQ, and $15wL$ bits would be necessary for 4-bit L-msRCQ. This total can be reduced by having multiple VN units share sets of registers. We have implemented all methods and explored their resource utilization in [TWC21a].

2.3 Hierarchical Dynamic Quantization (HDQ)

This section introduces the HDQ algorithm, a non-uniform quantization scheme that this paper uses both for quantization of channel observations and for quantization of internal messages by RCQ. Our results show, for example, that HDQ quantization of AWGN channel observations achieves performance similar to the optimal dynamic programming quantizer of [KY14] for the binary input AWGN channel, with much lower computational complexity.

2.3.1 Motivation

The quantizer plays an important role in RCQ decoder design. First, the channel observation is quantized as the input to the decoder. This section explores how to use HDQ to quantize the channel observations. Second, the parameters of $R(\cdot)$ and $Q(\cdot)$ are also designed by quantizing external messages according to their probability mass function (PMF) as determined by discrete density evolution. The use of HDQ to quantize internal messages is described in Section 2.5.

The HDQ approach designs a quantizer that maximizes mutual information in a greedy or progressive fashion. Quantizers aiming to maximize mutual information are widely used in non-uniform quantization design [HCM19a, WWS20a, LB18b, SLB18, SBW20a, MBB15, MMB20, MM17, SWB20, GBM18, WLW19, WCS11, WVC14]. Due to the interest of this paper, the cardinality of quantizer output is restricted to 2^b, i.e., this paper seeks b-bit quantizers. Kurkoski and Yagi [TV13] proposed a dynamic programming method to find an optimal quantizer that maximizes mutual information for a binary input discrete memoryless channel (BI-DMC) whose outputs are from an alphabet with cardinality B, with complexity $\mathcal{O}(B^3)$. The dynamic programming method of [KY14] finds the optimal quantization, but the approach becomes impractical when B is large.

In order to quantize the outputs for a channel with large cardinality B when constructing polar codes, Tal and Vardy devised a sub-optimal greedy quantization algorithm with complexity $\mathcal{O}(B\log(B))$ [TV13]. In [LB18b], Lewandowsky *et al.* proposed the modified Sequential Information Bottleneck (mSIB) algorithm to design the channel quantizer and LUTs for LDPC decoders. mSIB is also a sub-optimal quantization technique with complexity $\mathcal{O}(aB)$, where a is the number of trials. As a machine learning algorithm, multiple trials are required for good results with mSIB. Typical values of a range, for example, from 15 to 70.

HDQ is proposed in [WWS20a] as an efficient b-bit quantization algorithm for the sym-

metric BI-DMC with complexity $\mathcal{O}\left(\frac{2^b}{\log(\gamma)}\log(B)\right)$. HDQ has less complexity than mSIB and also the Tal-Vardy algorithm. This section reviews the HDQ using symmetric binary input AWGN channel as an example. As an improvement to the HDQ of [WWS20a], sequential threshold search is replaced with golden section search [Kie53].

2.3.2 The HDQ Algorithm

Let the encoded bit $x \in \{0, 1\}$ be modulated by Binary Phase Shift Keying (BPSK) and transmitted over an AWGN channel. The modulated BPSK signal is represented as $s(x) = -2x + 1$. We denote the channel observation at the receiver by y where

$$y = s(x) + z, \tag{2.8}$$

and $z \sim \mathcal{N}(0, \sigma^2)$. The joint probability density function of x and y, $f(x, y; \sigma)$, is:

$$f(x, y; \sigma) = \frac{1}{2\sqrt{2\pi\sigma^2}} e^{-\frac{(y-s(x))^2}{2\sigma^2}}. \tag{2.9}$$

HDQ seeks an b-bit quantization of the continuous channel output y, as in [WCS11]. In practice, often y is first quantized into B values using high-precision uniform quantization where $B \gg 2^b$, i.e., analog-to-digital (A/D) conversion. Let W be the result of the A/D output, where $W \in \mathcal{W}$ and $\mathcal{W} = \{0, 1, ..., B-1\}$. The alphabet of B channel outputs from the A/D converter is then subjected to further non-uniform quantization resulting in a quantization alphabet of 2^b values. We use D to represent the non-uniform quantizer output, which is comprised of the b bits $D = [D_1, ..., D_b]$. HDQ aims to maximize the mutual information between X and D.

For the symmetric binary input AWGN channel, a larger index w implies a larger LLR, i.e.:

$$\log \frac{P_{W|X}(i|0)}{P_{W|X}(i|1)} < \log \frac{P_{W|X}(j|0)}{P_{W|X}(j|1)}, \ \forall i < j. \tag{2.10}$$

Based on Lemma 3 in [KY14], any binary-input discrete memoryless channel that satisfies (2.10) has an optimal b-bit quantizer that is determined by $2^b - 1$ boundaries, which can be

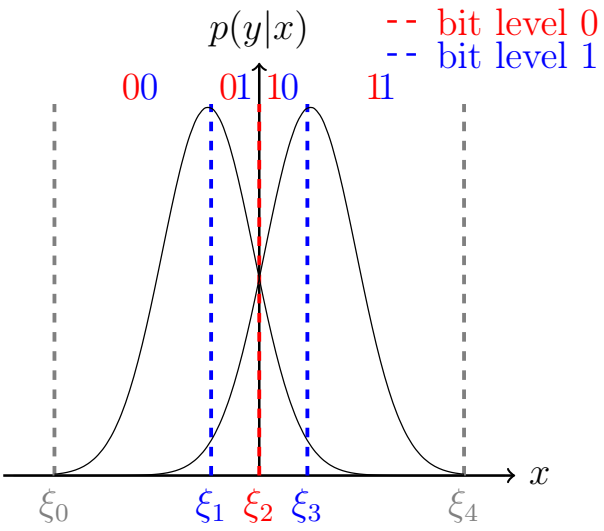


Figure 2.3: Given the conditional probability $p(y|x)$ of symmetric BI-AWGN channel, HDQ sequentially quantizing A/D output w into a 2-bit message by first finding the index ξ_2, then the indices ξ_1 and ξ_3.

identified by their corresponding index values. Denote the $2^b - 1$ index thresholds by $\{\xi_1, \xi_2, ..., \xi_{2^b-1}\} \subset \mathcal{W}$. Unlike the dynamic programming algorithm [KY14], which optimizes boundaries jointly, HDQ *sequentially* finds thresholds according to *bit level*, similar to the progressive quantization in [WLW19].

HDQ quantizes the symmetric BI-AWGN channel output using a progressive [WLW19] or greedy approach. The general b-bit HDQ approach is as follows:

1. We assume an initial high-precision uniform quantizer. For this case, set the extreme index thresholds $\xi_0 = 0$ and $\xi_{2^b} = B - 1$, which are the minimum and maximum outputs of the uniform quantization.

2. The index threshold $\xi_{2^{(b-1)}}$ is selected as follows to determine the bit level 0:

$$\xi_{2^{(b-1)}} = \arg \max_{\xi_0 < \xi < \xi_{2^b}} I(X; D_1)\,, \tag{2.11}$$

Algorithm 1: Hierarchical Dynamic Quantization

input : $P(X,W), X \in \{0,1\}, W \in \{0,...,B-1\}$; b

output: $\{\xi_0, \xi_1, ..., \xi_{2^b-1}\}$, $P(X,T)$

$\xi_0 \leftarrow 0$, $\xi_{2^b} \leftarrow B-1$

for $i \leftarrow 0$ **to** $b-1$ **do**

  for $j \leftarrow 0$ **to** $2^i - 1$ **do**

    $\xi_{(j+0.5)2^{b-i}} = \texttt{GSS}(\xi_{j2^{b-i}}, \xi_{(j+1)2^{b-i}})$

  end

end

$P_{XT}(x,t) = \sum_{w=\xi_t}^{\xi_t+1} P_{XW}(x,w)$, $X \in \{0,1\}$, $T \in \{0,...,2^{b-1}\}$

where

$$D_1 = \mathbb{1}(W \geq \xi_2^{(b-1)}). \tag{2.12}$$

3. The index thresholds $\xi_{2^{(b-2)}}$ and $\xi_{3*2^{(b-2)}}$ are selected as follows to determine bit level 1:

$$\xi_{2^{(b-2)}} = \arg \max_{\xi_0 < \xi < \xi_{2^b-1}} I(X; D_2 | D_1 = 0), \tag{2.13}$$

$$\xi_{3*2^{(b-2)}} = \arg \max_{\xi_{2^b-1} < \xi < \xi_{2^b}} I(X; D_2 | D_1 = 1), \tag{2.14}$$

and

$$D_2 = \begin{cases} \mathbb{1}(W \geq \xi_{2^{(b-2)}}) & \text{if } D_1 = 0 \\ \mathbb{1}(W \geq \xi_{3*2^{(b-2)}}) & \text{if } D_1 = 1 \end{cases}. \tag{2.15}$$

4. In the general case, when the thresholds for k previous quantization bits have been determined, 2^k thresholds $\{\xi_{(j+0.5)2^{b-k}}, j = 0,..,2^k-1\}$ must be selected to determine the next quantization bit. Each threshold maximizes $I(X; D_{k+1} | D_k = d_k, \ldots, D_1 = d_1)$ for a specific result for the k previous quantization bits.

Fig. 2.3 illustrates how HDQ quantizes the symmetric binary input AWGN for the case where $b = 2$. First, the indices ξ_0 and ξ_4 of the extreme points are set. Then the

index ξ_2 is set to maximize $I(X;D_1)$. Finally, the indices ξ_1 and ξ_3 are set to maximize $I(X;D_2|D_1)$ by independently selecting ξ_1 to maximize $I(X;D_2|D_1=0)$ and ξ_3 to maximize $I(X;D_2|D_1=1)$.

Alg. 1 provides a full description of HDQ algorithm. The function $\texttt{GSS}(\xi_\ell,\xi_r)$ uses the golden section search algorithm described in Sec.2.3.3 for thresholds search.

HDQ provides the 2^b-1 index thresholds $\{\xi_1,\ldots,\xi_{2^b-1}\}$. For channel quantization, the index thresholds can be mapped to channel outputs. For the RCQ decoding, the messages are LLR values, the LLR magnitude thresholds $\{\tau_0,\ldots,\tau_{2^{b-1}-2}\}$ are calculated from the index thresholds $\{\xi_{2^{b-1}+1},\ldots,\xi_{2^b-1}\}$ as follows:

$$\tau_i = \log\frac{P_{W|X}(\xi_{1+i+2^{b-1}}|0)}{P_{W|X}(\xi_{1+i+2^{b-1}}|1)}, i=0,1,\ldots,2^{b-1}-2. \tag{2.16}$$

HDQ also provides the joint probability between code bit X and quantized message D, $P(X,D)$. The magnitude reconstruction function $R^*(\cdot)$ is computed as follows:

$$R^*(d)=\log\frac{P_{XT}(0,d+2^{b-1})}{P_{XT}(1,d+2^{b-1})},\ d=0,1,...,2^{b-1}-1. \tag{2.17}$$

2.3.3 Golden-Section Search and Complexity Analysis

After k stages of HDQ, there are 2^k quantization regions each specified by their leftmost and rightmost indices ξ_ℓ and ξ_r. The next stage finds a new threshold ξ^* for each of these 2^k regions. Each ξ^* is selected to maximize a conditional mutual information as follows:

$$\xi^* = \arg\max_{\xi_\ell<\xi<\xi_r} I(\xi), \tag{2.18}$$

where

$$I(\xi) = I\left(X; D_{k+1}(\xi)|D_1=d_1,\ldots,D_k=d_k\right) \tag{2.19}$$

$$= \sum_{x,d_{k+1}} P\left(x,d_{k+1}(\xi)|d_1^k\right)\log\frac{P(d_{k+1}(\xi)|x,d_1^k)}{P(d_{k+1}(\xi)|d_1^k)} \tag{2.20}$$

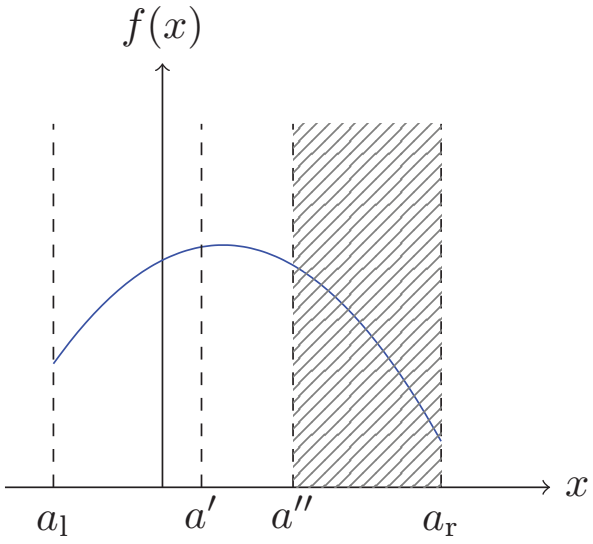


Figure 2.4: Illustration of one iteration of golden-section search for finding maximum point of $f(x)$ in the interval $[a_{\mathrm{l}}, a_{\mathrm{r}}]$. $a' = a_{\mathrm{r}} - \frac{a_{\mathrm{r}}-a_{\mathrm{l}}}{\gamma}$ and $a'' = a_{\mathrm{l}} + \frac{a_{\mathrm{r}}-a_{\mathrm{l}}}{\gamma}$. Because $f(a'') < f(a')$, $[a'', a_{\mathrm{r}}]$ is truncated and $[a_{\mathrm{l}}, a'']$ becomes the new search interval for the next iteration.

for the binary k-tuple $d_1^k = d_1, \ldots, d_k$ that defines (ξ_ℓ, ξ_r). The probability $P\left(x, d_{k+1}(\xi)|d_1^k\right)$ is defined as follows:

$$P\left(x, d_{k+1}(\xi)|d_1^k\right) = \begin{cases} \frac{\sum_{w=\xi_l}^{\xi} P_{XW}(x,w)}{\sum_{w=\xi_l}^{\xi_r} P_W(w)} & d_{k+1} = 0 \\ \frac{\sum_{w=\xi+1}^{\xi_r} P_{XW}(x,w)}{\sum_{w=\xi_l}^{\xi_r} P_W(w)} & d_{k+1} = 1 \end{cases}. \tag{2.21}$$

Because $I(\xi)$ is concave in ξ, the local maximum can be found using the golden section search [Kie53], a simple but robust technique to find extreme point of a unimodal function by successively narrowing the range of values on a specified interval. Specifically, Fig. 2.4 illustrates one iteration of golden-section search for finding maximum point of $f(x)$ in the interval $[a_{\mathrm{l}}, a_{\mathrm{r}}]$. First, find $a' = a_{\mathrm{r}} - \frac{a_{\mathrm{r}}-a_{\mathrm{l}}}{\gamma}$ and $a'' = a_{\mathrm{l}} + \frac{a_{\mathrm{r}}-a_{\mathrm{l}}}{\gamma}$, where $\gamma = \frac{\sqrt{5}+1}{2}$. Because $f(a'') < f(a')$, which suggests that the maximum point lies in $[a_{\mathrm{l}}, a'']$, the interval $[a'', a_{\mathrm{r}}]$ is truncated and $[a_{\mathrm{l}}, a'']$ is updated as the next round search interval. Further details of

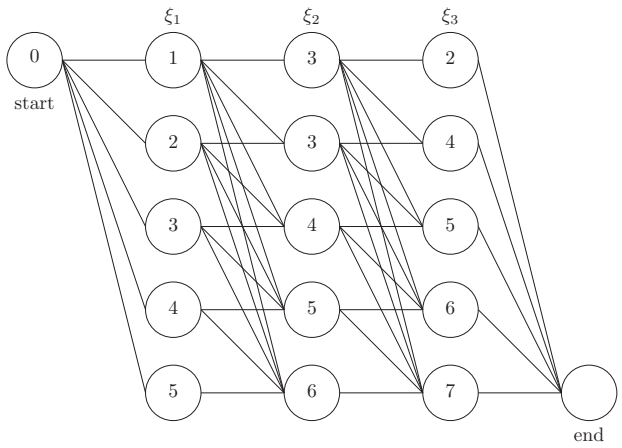


Figure 2.5: A trellis whose paths represent all 2-bit quantizers for a BI-DMC with 8 outputs. The vertices in column i are possible values for i^{th} threshold ξ_i. Each branch in the trellis identifies a quantization region.

golden-section search can be found in [Kie53]. When using the golden-section search to find all $2^b - 1$ thresholds for the b-bit HDQ, $I(\xi)$ will be computed using (2.19) a number of times that is proportional to:

$$\log_\gamma(B) + \sum_{i=1}^{2^1} \log_\gamma(B_{2,i}) + \ldots + \sum_{i=1}^{2^{b-1}} \log_\gamma(B_{b,i}), \tag{2.22}$$

$$= \log_\gamma(B) + \log_\gamma \prod_{i=1}^{2^1} B_{2,i} + \ldots + \log_\gamma \prod_{i=1}^{2^{b-1}} B_{b,i}, \tag{2.23}$$

$$\leq \log_\gamma(B) + 2\log_\gamma\left(\frac{B}{2}\right) + \ldots + 2^{b-1}\log_\gamma\left(\frac{B}{2^{b-1}}\right) \tag{2.24}$$

$$= \frac{2^b}{\log(\gamma)}\log(B). \tag{2.25}$$

$B_{j,i}$ is the i^{th} interval length in $j-1$ bit level quantization and $\sum_{i=1}^{2^{j-1}} B_{j,i} = B$. Therefore, a b-bit quantization on a B-output channel using HDQ can be designed in $\mathcal{O}\left(\frac{2^b}{\log(\gamma)}\log(B)\right)$ time.

2.3.4 Comparing HDQ with Optimal Dynamic Programming

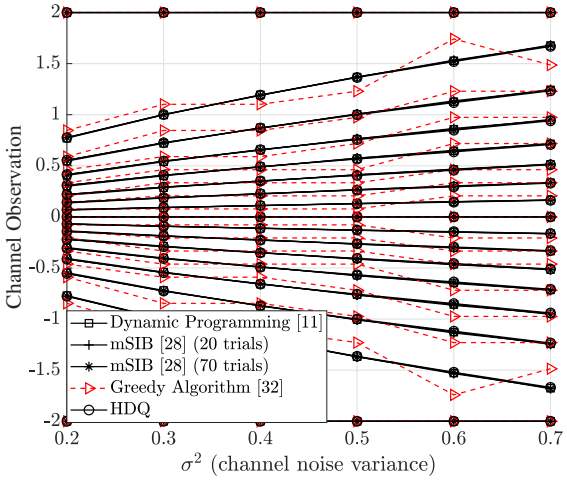
Unlike the dynamic programming approach of Kuskoski and Yagi [KY14], HDQ does not always provide the optimal solution. This subsection provides an example contrasting HDQ with the dynamic programming solution. Following [KY14], Fig. 2.5 gives a trellis whose paths represent all 2-bit quantizers for a binary input DMC with 8 outputs. The outputs are indexed from 0 to 7 and satisfy (2.10). The vertices in column i are possible values for ξ_i, and each path represents a valid quantizer whose thresholds are determined by the vertices in each column. Each branch in the trellis identifies a quantization region. For example, the branch connecting vertex $\xi_0 = 0$ to vertex $\xi_1 = 2$ specifies the leftmost quantization region as $\{0,1\}$, i.e., $\xi_\ell = 0$ and $\xi_r = 1$.

The dynamic programming algorithm determines vertices of all columns jointly, whereas HDQ identifies the vertices in a greedy way, by first finding the vertex in column 2 to maximize $I(X; D_1)$ and then vertices in column 1 and 4 to maximize $I(X; D_2|D_1 = d_1)$. Hence, the greedy approach of HDQ only searches part of trellis and therefore is sub-optimal. However, our simulations show that HDQ finds the quantizer that perform closely to the optimal one.

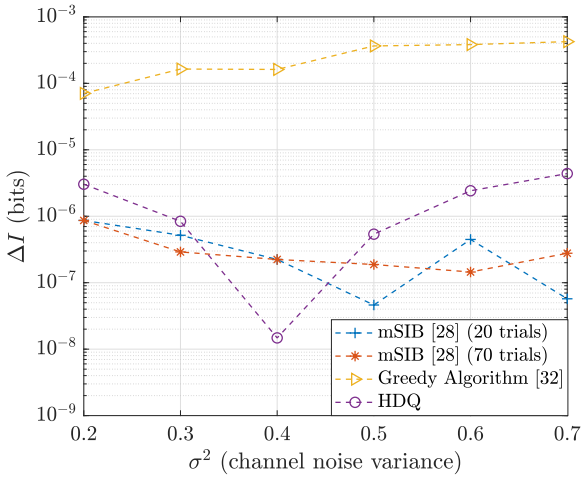
2.3.5 Simulation Result

This section provides simulation results for quantizing symmetric binary input AWGN channel observations. The simulations compare HDQ to the optimal dynamic programming result as well as to two sub-optimal approaches: mSIB with 20 and 70 trials and the greedy quantization algorithm describe in [LB18b]. For all the quantization approaches, the channel observations are first quantized uniformly into $B = 2000$ points between -2 and 2.

Fig. 2.6a gives the thresholds as a function of σ^2 for HDQ, dynamic programming, mSIB with 20 and 70 trials, and greedy quantization. The quantization thresholds for HDQ, dynamic programming, and mSIB are indistinguishable in Fig. 2.6a. HDQ has significantly



(a)



(b)

Figure 2.6: Fig. (a): Quantization thresholds for dynamic programming, msIB, and HDQ on the BI-AWGNC as a function of σ^2 for $B = 2000$. Fig. (b): Mutual information loss between each sub-optimal quantizer and optimal quantizer for BI-AWGNC as a function of σ^2 for $B = 2000$.

lower complexity than both dynamic programming and mSIB. The thresholds for greedy quantization algorithm of [TV13] deviate noticeably from the thresholds found by the other approaches.

In order to quantify the performance of sub-optimal quantizers, we define ΔI as follows:

$$\Delta I = I^{\text{dp}}(X; D) - I^{\text{sub}}(X; D), \tag{2.26}$$

where $I^{\text{dp}}(X; D)$ and $I^{\text{sub}}(X; D)$ are the mutual information between code bit X and quantized value D as obtained by dynamic programming and sub-optimal quantizers, respectively. Fig. 2.6b plots ΔI as a function of σ^2 for each sub-optimal quantizer. All three sub-optimal quantizers perform quite well with $\Delta I < 10^{-3}$ bits. However, HDQ and mSIB achieve $\Delta I < 10^{-6}$, significantly outperforming the greedy approach of [TV13].

2.4 Flooding-scheduled RCQ Decoder

RCQ decoder is a result of quantized density evolution: In the t^{th} iteration, the quantization functions and the reconstruction functions associated with that iteration, i.e., Q_t^c, R_t^v, Q_t^v, R_{t+1}^v are constructed by quantizing the joint p.m.f. between code bits and the message from either the variable node or check node. These functions are also the parameters of the flooding-scheduled RCQ decoder. To differentiate our discrete density evolution from the one using uniform quantization [SFR01], we name our density evolution *HDQ Discrete Density Evolution* (HDQ-DDE). Specifically, this section describes the HDQ-DDE when the check node uses box-plus operation. The decoder generated by such HDQ-DDE is a flooding-scheduled bpRCQ decoder.

2.4.1 MIM-DDE at check node

Denote the joint p.m.f between the *external message* from the i^{th} variable node and corresponding code bit by $P^{v,i}(X, T)$, $X = \{0, 1\}$, $T = \{0, \ldots, 2^m - 1\}$. Based on the independence

assumption in density evolution [RU01], all incoming messages have same distribution:

$$P^{v,i}(X,T) = P^{v}(X,T), \quad i = 0, ..., d_c - 2 \tag{2.27}$$

where d_c is check node degree. At check node, the code bit corresponding to output is the XOR sum of code bits corresponding to all inputs. By denoting:

$$P^{v,a}(X,T) \circledast P^{v,b}(X,T) \triangleq \sum_{\substack{m,n: \\ m \bigoplus n = k}} P^{v,a}(X_m,T) P^{v,b}(X_n,T), \tag{2.28}$$

where $m, n, k \in \{0, 1\}$, the joint p.m.f between code bit corresponding to output and input messages, $P^{c}_{out}(X, \mathbf{T})$, can be represented by:

$$P^{c}_{out}(X,\mathbf{T}) = P^{v,0}(X,T) \circledast ... \circledast P^{v,d_c-2}(X,T) \tag{2.29}$$

$$= P^{v}(X,T) \circledast ... \circledast P^{v}(X,T) \tag{2.30}$$

$$\triangleq P^{v}(X,T)^{\circledast(d_c-1)}, \tag{2.31}$$

where $\mathbf{T}$ is a vector containing all incoming $d_c - 1$ messages.

In order to keep the cardinality of external message the same, $P^{c}_{out}(X, \mathbf{T})$ needs to be quantized to 2^m levels. As pointed in [LB18b], $|\mathbf{T}| = 2^{m(d_c-1)}$ will be very large when m and d_c is large. For an example, if $d_c = 8$ and $m = 4$, $|\mathbf{T}| = 2.68 * 10^8$. Hence, directly quantizing $P^{c}_{out}(X, \mathbf{T})$ is impossible. To mitigate the problem of *cardinality bombing*, we propose an intermediate coarse quantization algorithm called One-Step-Annealing (OSA) quantization without sacrificing mutual information. Note that Eq. (2.31) can be calculated in a recursive way and each step takes two inputs:

$$P^{c}_{out}(X,\mathbf{T})^{\circledast i} = P^{v}(X,T)^{\circledast(i-1)} \circledast P^{v}(X,T) \tag{2.32}$$

We observe that, in each step, the output of Eq.(2.32) has some entries with very close log likelihood ratio (LLR) values. By merging entries whose LLR difference is small enough, mutual information loss is negligible. Hence, OSA simply merges entries whose LLR values

Algorithm 2: One Step Annealing Algorithm (OSA)

input : $\Pr(X,Y), X \in \{0,1\}, Y \in \{0,...,N-1\}$; l_s

output: $\Pr(X,T)$

$j \leftarrow 0$, $\Pr(X_0,T_j) \leftarrow P(X_0,Y_0)$, $\Pr(X_1,T_j) \leftarrow P(X_1,Y_0)$, $l \leftarrow \log\frac{\Pr(X_0,Y_0)}{\Pr(X_1,Y_0)}$

for $i \leftarrow 1$ **to** $N-1$ **do**

  if $(\log\frac{P(X_0,T_i)}{P(X_1,T_i)} - l) \leq l_s$ **then**

    $P(X_0,T_j) \leftarrow \Pr(X_0,T_j) + \Pr(X_0,Y_i)$, $P(X_1,T_j) \leftarrow \Pr(X_1,T_j) + \Pr(X_1,Y_i)$

  else

    $j \leftarrow j+1$

    $\Pr(X_0,T_j) \leftarrow \Pr(X_0,Y_i)$, $\Pr(X_1,T_j) \leftarrow \Pr(X_1,Y_i)$

    $l \leftarrow \log\frac{\Pr(X_0,Y_i)}{\Pr(X_1,Y_i)}$

  end

end

Figure 2.7: OSA illustration: points are ordered w.r.t. LLR values. Each color represents a cluster and LLR value difference in each cluster is less than l_s.

difference is less than a threshold l_s, and the output of OSA will be the input of the next p.m.f calculation step, i.e.:

$$P^v(X,T)^{\circledast i} = \texttt{OSA}(P^v(X,T)^{\circledast(i-1)}, l_s) \circledast P^v(X,T). \tag{2.33}$$

We use $l_s \in [10^{-4}, 10^{-3}]$ in our simulation. Fig. 2.7 shows an illustration of OSA and a full description of the OSA algorithm is given in Algorithm 2. The following table shows $|\mathbf{T}|$ after we implement OSA and choose different l_s. The example we show has the parameter $m = 4$, $d_c = 8$. The result shows that OSA greatly decreases the output cardinality, and based on our simulation, mutual information losses under these three l_s are all less than 10^{-7} bits.

l_s	0	10^{-4}	$5 * 10^{-4}$	10^{-3}
$\lvert\mathbf{T}\rvert$	$2.68 * 10^8$	$3.3 * 10^4$	$1.7 * 10^3$	$1.3 * 10^3$

2.4.2 MIM-DDE at variable node

Each variable node sums the LLR messages from its channel observation and neighboring check nodes. By denoting:

$$P^{c,a}(X,T) \boxdot P^{c,b}(X,T) = \frac{1}{P(X)} P^{c,a}(X,T) P^{c,b}(X,T), \tag{2.34}$$

the joint p.m.f between code bit X and incoming message combination $\mathbf{T}$, $P^v_{out}(X,\mathbf{T})$, given variable node degree d_v, can be expressed by:

$$P^v_{out}(X,\mathbf{T}) = P^{ch}(X,T) \boxdot P^c(X,T)^{\boxdot(d_v-1)}, \tag{2.35}$$

Similarly, for irregular LDPC codes with variable edge degree distribution

$$\lambda(x) = \sum_{i=2}^{d_{v,max}} \lambda_i x^{i-1}, \tag{2.36}$$

$P^v_{out}(X,\mathbf{T})$ is given by:

$$P^v_{out}(X,\mathbf{T}) = P^{ch}(X,T) \boxdot \sum_{i=2}^{d_{v,max}} \lambda_i P^c(X,T)^{\boxdot(d_v-1)}. \tag{2.37}$$

$P^v_{out}(X,\mathbf{T})$ is then quantized to 2^m levels by HDQ. Also, as a result of HDQ, and joint p.m.f between code bit X and quantized messages T, $P^v(X,T)$, is updated. Q^v in this iteration and R^c in the next iteration are built correspondingly. Note that variable nodes also face the *cardinality bombing* problem, hence `OSA` is needed in each recursive step.

Thus, by implementing MIM-DDE, we can iteratively update $P^c(X,T)$, $P^v(X,T)$ and build Q^c_i, Q^v_i, R^c_i and R^v_i, $i = \{0,...,I_T-1\}$.

In MIM-DDE, we only limit the precision of external messages, i.e. m, and keep internal messages, n^c (only for *bp-RCQ*) and n^v, full precision. To make internal message precision finite, a uniform n^c (or n^v) quantizer is required when implementing F^c(or F^v).

2.4.3 Threshold

At any specified $\frac{E_b}{N_o}$, flooding-scheduled HDQ discrete density evolution constructs the $R^{(t)}(\cdot)$ and $Q^{(t)}(\cdot)$ functions at each iteration t and also computes the mutual information $I^{(t)}\left(\frac{E_b}{N_o}\right)$ between a code bit and its corresponding variable node message in each layer r at each iteration t. An important design question is which value of $\frac{E_b}{N_o}$ to use to construct the $R^{(t)}(\cdot)$ and $Q^{(t)}(\cdot)$ functions implemented at the decoder, which necessarily will work over a range of $\frac{E_b}{N_o}$ values in practice. Define the threshold of a flooding RCQ decoder given a maximum number of decoding iterations I_T as:

$$\frac{E_b}{N_o}^* = \inf\left\{\frac{E_b}{N_o} : I^{(I_T)}\left(\frac{E_b}{N_o}\right) > 1-\epsilon\right\}, \tag{2.38}$$

i.e., $\frac{E_b}{N_o}^*$ is the smallest $\frac{E_b}{N_o}$ that achieves a mutual information between the code bit and the external message that is greater that $1-\epsilon$. Our simulation results show that $\frac{E_b}{N_o}^*$ for $\epsilon = 10^{-4}$ produced $R^{(t)}(\cdot)$ and $Q^{(t)}(\cdot)$ functions that deliver excellent FER performance across a wide $\frac{E_b}{N_o}$ range.

2.5 Layered-scheduled RCQ Decoder

This section is focused on HDQ discrete density evolution for LDPC decoders with a layered schedule. Specifically, this section considers layer-specific msRCQ decoding on QC-LDPC codes.

2.5.1 Decoding a Quasi-Cyclic LDPC Code with a Layered Schedule

QC-LDPC codes are structured LDPC codes characterized by a parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ which consists of square sub-matrices with size S, which are either the all-zeros matrix or a cyclic permutation of the identity matrix. These cyclic permutations are also called circulants that are represented by σ^i to indicate that the rows of the identity matrix are cyclically shifted by i positions. Thus an $M \times U$ *base matrix* H_p can concisely define

a QC-LDPC code, where each element in H_{p} is either $\mathbf{0}$ (the all-zeros matrix) or σ^i (a circulant). QC-LDPC codes are perfectly compatible with horizontal layered decoding by partitioning CNs into M layers with each layer containing S consecutive rows. This ensures that each VN connects to at most one CN in each layer.

Denote the i^{th} CN and j^{th} VN by c_i and v_j respectively. Let $u^{(t)}_{c_i \to v_j}$ be the LLR message from c_i to its neighbor v_j in t^{th} iteration and l_{v_j} be the posterior of v_j. In the t^{th} iteration, a horizontal-layered Min Sumdecoder calculates the messages $u^{(t)}_{c_i \to v_{j'}}$ and updates the posteriors $l_{v_{j'}}$ as follows:

$$l_{v_{j'}} \leftarrow l_{v_{j'}} - u^{(t-1)}_{c_i \to v_{j'}} \quad \forall j' \in \mathcal{N}(c_i), \tag{2.39}$$

$$u^{(t)}_{c_i \to v_{j'}} = \left(\prod_{\tilde{j} \in \mathcal{N}(c_i)/\{j'\}} \operatorname{sign}(l_{v_{\tilde{j}}}) \right) \times \min_{\tilde{j} \in \mathcal{N}(c_i)/\{j'\}} |l_{v_{\tilde{j}}}|, \quad \forall j' \in \mathcal{N}(c_i), \tag{2.40}$$

$$l_{v_{j'}} \leftarrow l_{v_{j'}} + u^{(t)}_{c_i \to v_{j'}} \quad \forall j' \in \mathcal{N}(c_i). \tag{2.41}$$

$\mathcal{N}(c_i)$ denotes the set of VNs that are neighbors of c_i. For a QC-LDPC code with a long block length, layered decoding is preferable for hardware implementations because parallel computations of each of (2.39), (2.40), and (2.41) exploit the QC-LDPC structure.

2.5.2 Representation Mismatch Problem

The RCQ decoding structure in [WWS20a] can be used with a layered schedule as discussed in Sec. 2.5.1. Fig. 2.8a illustrates the paradigm for an msRCQ decoder with a layered schedule. The $Q_{\mathrm{v}}^{(t)}$ and $R_{\mathrm{v}}^{(t)}$ are designed by the HDQ discrete density evolution as in [WWS20a]. Even though the msRCQ decoder has better FER performance than the standard Min Sumdecoder under a flooding schedule [WWS20a], under a layered schedule, msRCQ has worse FER performance than standard Min Sumand also requires more iterations. These performance differences are shown below in Fig. 2.11 of Sec. 2.6. This subsection explains how the

performance degradation of the RCQ decoder under the layered schedule is caused by the representation mismatch problem.

Consider a regular LDPC code defined by a parity check matrix H. In iteration t, define the PMF between code bit x and external CN messages $u^{(t)}_{c_i \to v_j}$ as $P^{(t)}_{(c_i,v_j)}(X, D)$, where $X = \{0, 1\}$ and $D = \{0, ..., 2^{b^e} - 1\}$. One underlying assumption of HDQ discrete density evolution is that all CN messages have the same PMF in each iteration, i.e., for any (c_i, v_j) and $(c_{i'}, v_{j'})$ that satisfy $H_{i,j} = H_{i',j'} = 1$:

$$P^{(t)}_{(c_i,v_j)}(X, D) = P^{(t)}_{(c_{i'},v_{j'})}(X, D). \tag{2.42}$$

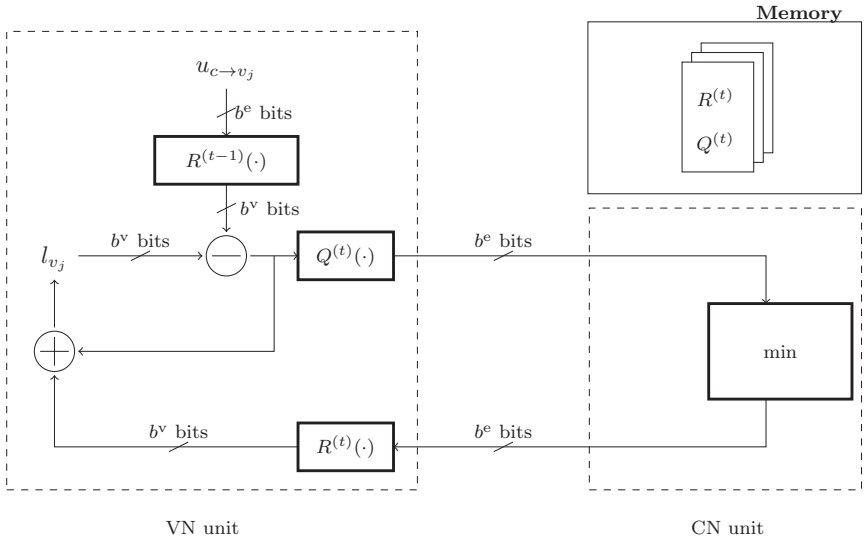
(2.42) implies that the message indices of different CN have the same LLR representation, i.e.:

$$\log \frac{P^{(t)}_{(c_i,v_j)}(0, d)}{P^{(t)}_{(c_i,v_j)}(1, d)} = \log \frac{P^{(t)}_{(c_{i'},v_{j'})}(0, d)}{P^{(t)}_{(c_{i'},v_{j'})}(1, d)},\ d \in \{0, ..., 2^{b^e} - 1\}. \tag{2.43}$$

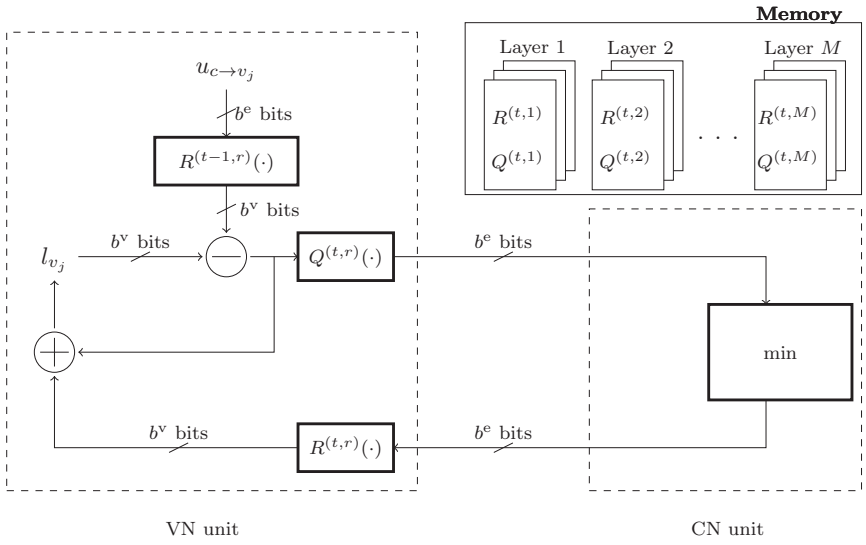
The msRCQ decoder with a flooding schedule obeys (2.42) and (2.43) because the VN messages to calculate different CN messages have the same distribution. Therefore, it is sufficient for a decoder with a flooding schedule to use the iteration-specific reconstruction function $R^{(t)}$ for all external CN messages. However, for a decoder with a layered schedule, the VN messages to calculate CN messages from different layers have different distributions. For the decoder with a layered schedule, $l^{(t)}_{v_j \to c_i}$ is calculated by:

$$l^{(t)}_{v_j \to c_i} = l^{(ch)}_{v_j} + \sum_{\{i' | i' \in \mathcal{N}(v_j), i' < i\}} u^{(t)}_{c_{i'} \to v_j} + \sum_{\{i' | i' \in \mathcal{N}(v_j), i' > i\}} u^{(t-1)}_{c_{i'} \to v_j}, \tag{2.44}$$

Unlike a decoder using a flooding schedule, which updates $l^{(t)}_{v_j \to c_i}$ only using CN messages in iteration $t-1$, decoders using a layered schedule use messages from both iteration $t-1$ and iteration t. The VN messages computed in different layers utilize different proportions of check-to-variable node messages from iterations $t-1$ and t. Since the check-to-variable node messages from different iterations have different reliability distributions, the VN messages from different layers also have different distributions. Therefore (2.42) and (2.43) no longer



(a)



(b)

Figure 2.8: Two layered decoders. Fig. (a) uses the same RCQ parameters for each layer as with the *msRCQ* design for a flooding decoding in [WWS20a]. Fig. (b) shows the proposed *layer-specific msRCQ* decoder in [TWC21a], which features separate RCQ parameters for each layer. The main difference is that msRCQ decoder uses iteration specific parameters while L-msRCQ decoder considers layer-and-iteration parameters.

hold true, and a single $R^{(t)}(\cdot)$ is insufficient to accurately describe CN messages from different layers.

In conclusion, the *Representation Mismatch Problem* refers to inappropriately using a single $R^{(t)}$ and single $Q^{(t)}$ for all layers in iteration t of a layered decoding schedule. This issue degrades the decoding performance of layer-scheduled RCQ decoder. On the other hand, the conventional fixed-point decoders that do not perform coarse non-uniform quantization, such as standard Min Sumdecoder, are not affected by the changing the distribution of messages in different layers and hence don't have representation mismatch problem.

2.5.3 Layer-Specific RCQ Design

Based on the analysis in the previous subsection, R and Q should adapt for the PMF of messages in each layer, in order to solve the representation mismatch problem. This motivates us to propose the layer-specific RCQ decoding structure in this paper, as illustrated in Fig. 2.8b. The key difference between the RCQ decoder and layer-specific RCQ decoder is that layer-specific RCQ designs quantizers and reconstruction mappings for each layer in each iteration. We use $R^{(t,r)}$ and $Q^{(t,r)}$ to denote the reconstruction mapping and quantizer for decoding iteration t and layer r, respectively. As illustrated in Fig. 2.8b, layer-specific RCQ specifies R and Q for each layer to handle the issue that messages in different layers have different PMFs. This leads to a significant increase in the required memory because the memory required to store $R^{(t,r)}$ and $Q^{(t,r)}$ is proportional to the product of the number of layers and the number of iterations required for decoding the QC-LDPC code.

Designing $Q^{(t,r)}(\cdot)$ and $R^{(t,r)}(\cdot)$ for layer-specific msRCQ requires the message PMF for each layer in each iteration. However, HDQ discrete density evolution [WWS20a], which performs density evolution based on ensemble, fails to capture layer-specific information. In this section, we propose a layer-specific HDQ discrete density evolution based on base matrix H_{p} of QC-LDPC code. In layer-specific HDQ discrete density evolution, the joint PMF between code bit X and external message D from check/variable nodes are tracked

in each layer in each iteration. We use $P^{(t,r)}(X, D^{\mathrm{c}}),\ X \in \{0,1\},\ D^{\mathrm{c}} \in \{0,\ldots,2^{b^{\mathrm{e}}}-1\}$ to represent the joint PMF between code bit and CN message in layer m and iteration t. Similarly, VN messages are denoted by $P^{(t,r)}(X, D^{\mathrm{v}})$.

2.5.3.1 Initialization

For an AWGN channel with noise variance σ^2, the LLR of channel observation y is $l = \frac{2}{\sigma^2}y$. For the msRCQ decoder with bit width $(b^{\mathrm{e}}, b^{\mathrm{v}})$, the continuous channel LLR input is uniformly quantized into $2^{b^{\mathrm{v}}}$ regions. Each quantization region has a true log likelihood ratio, which we refer to as l_d, so that we have an alphabet of b^{v} real-valued log likelihood ratios $\mathcal{D}^{\mathrm{ch}} = \{l_0, ..., l_{2^{b^{\mathrm{v}}}-1}\}$. Using these values, the joint PMF between the code bit X and channel LLR message $D^{\mathrm{ch}} \in \{0, ..., 2^{b^{\mathrm{v}}}-1\}$ is:

$$P_{XD^{\mathrm{ch}}}(x,d) = P_D(d)\frac{e^{(1-x)l_d}}{e^{l_d}+1},\ X \in \{0,1\},\ l_d \in \mathcal{D}^{\mathrm{ch}}. \tag{2.45}$$

The distribution $P_{XD^{\mathrm{ch}}}(x,d)$ is used for the HDQ discrete density evolution design. The actual decoder does not use the real-valued likelihoods l_d but rather uses b^{v}-bit channel LLRs obtained by uniformly quantizing continuous channel LLR values.

2.5.3.2 Variable Nodes PMF Calculation

Given a base matrix H_{p}, with entry $H_{\mathrm{p}}(r,c)$ at row r and column c, define the sets of active rows $\mathcal{R}(c)$ for a specified column c and active columns $\mathcal{C}(r)$ for a specified row r as follows:

$$\mathcal{R}(c) = \{r | H_{\mathrm{p}}(r,c) \neq 0\}, \quad \mathcal{C}(r) = \{c | H_{\mathrm{p}}(r,c) \neq 0\}. \tag{2.46}$$

In iteration t and layer r, consider the joint PMF between a code bit X corresponding to a VN in the circulant $H_{\mathrm{p}}(r,c)$ and the vector $\mathbf{D}$, which includes the channel message D^{ch} for

X and the check node messages D^{c} incident to that VN. This PMF is calculated by:

$$\begin{aligned} P_{\mathrm{v}}^{(t,r,c)}(X,\mathbf{D}) = P(X,D^{\mathrm{ch}}) \boxdot \left(\boxdot_{\substack{k\in\mathcal{R}(c)\\ k<r}} P^{(t,k)}(X,D^{\mathrm{c}}) \right) \\ \boxdot \left(\boxdot_{\substack{k\in\mathcal{R}(c)\\ k>r}} P^{(t-1,k)}(X,D^{\mathrm{c}}) \right), \end{aligned} \tag{2.47}$$

$\boxdot$ is defined as follows:

$$P(x,[d_1,d_2]) = P(X_1,D_1) \boxdot P(X_2,D_2) \tag{2.48}$$

$$\triangleq \frac{1}{P_X(x)} P_{X_1D_1}(x,d_1) P_{X_2D_2}(x,d_2), \tag{2.49}$$

$x \in \{0,1\},\ d_1,d_2 \in \{0,\ldots,2^{b^{\mathrm{e}}}-1\}$. When $|\mathcal{R}(c)|$ is large, the alphabet $\mathcal{D}$ of possible input message vectors $\mathbf{D}$ is large with $|\mathcal{D}| = 2^{b^{\mathrm{v}}+(|\mathcal{R}(c)|-1)b^{\mathrm{e}}}$. To manage the complexity of HDQ discrete density evolution, message vectors $\mathbf{D}$ with similar log likelihoods are clustered via one-step-annealing as in [WWS20a] for (2.47).

The layer-specific msRCQ decoder uses layer-specific parameters, and for each layer the marginal distribution on the computed variable node messages will be distinct. The marginal distribution used by HDQ at layer r is computed as follows:

$$\tilde{P}_{\mathrm{v}}^{(t,r)} = \left\{ \frac{1}{|\mathcal{C}(r)|} P_{\mathrm{v}}^{(t,r,c)}(X,\mathbf{D}) \mid c \in \mathcal{C}(r) \right\} \tag{2.50}$$

where $P^{(t,r)}(X,D^v)$ and $Q^{(t,r)}(\cdot)$ can be obtained by quantizing $\tilde{P}_{\mathrm{v}}^{(t,r)}$ using HDQ:

$$\left[P^{(t,r)}(X,D^{\mathrm{v}}), Q^{(t,r)}(\cdot)\right] = \texttt{HDQ}\left(\tilde{P}_{\mathrm{v}}^{(t,r)}, 2^{b^{\mathrm{e}}}\right), \tag{2.51}$$

where `HDQ` is defined as a function that realizes b^e-bit HDQ on $\tilde{P}_{\mathrm{v}}^{(t,r)}$ and generates $P^{(t,r)}(X,D^{\mathrm{v}})$ and $Q^{(t,r)}$ as outputs. Note that (33) and (34) realize implicit message alignment in [Sta21] such that the internal messages from any $c \in \mathcal{C}(r)$ use same set of thresholds for quantization and the same external messages from any $c \in \mathcal{C}(r)$ have same LLR interpretations, regardless of node degree.

2.5.3.3 Check Nodes PMF Calculation

Let $l_{\mathrm{v}}^{(t,r)}(d)$ be the LLR of external VN message d in layer r and iteration t. As an LLR, this CN input $l_{\mathrm{v}}^{(t,r)}(d)$ has the following meaning:

$$l_{\mathrm{v}}^{(t,r)}(d) = \log \frac{P_{XD^{\mathrm{v}}}^{(t,r)}(0,d)}{P_{XD^{\mathrm{v}}}^{(t,r)}(1,d)}, \quad d = 0, ..., 2^{b^e} - 1. \tag{2.52}$$

Given input messages $d_1, d_2 \in \mathcal{D}^{\mathrm{v}}$, the CN min operation produces the following output:

$$\begin{aligned} l_{\mathrm{MS}}^{\mathrm{out}} = &\min\left(|l_{\mathrm{v}}^{(t,r)}(d_1)|, |l_{\mathrm{v}}^{(t,r)}(d_2))|\right) \\ &\times \mathrm{sgn}(l_{\mathrm{v}}^{(t,r)}(d_1)) \times \mathrm{sgn}(l_{\mathrm{v}}^{(t,r)}(d_2)). \end{aligned} \tag{2.53}$$

Under the symmetry assumption, there is a $d^{\mathrm{out}} \in \mathcal{D}^{\mathrm{v}}$ that has the LLR computed as $l_{\mathrm{MS}}^{\mathrm{out}}$:

$$l_{\mathrm{MS}}^{\mathrm{out}} = \log \frac{P_{XD^{\mathrm{v}}}^{(t,r)}(0,d^{\mathrm{out}})}{P_{XD^{\mathrm{v}}}^{(t,r)}(1,d^{\mathrm{out}})}. \tag{2.54}$$

At the check node output, l_{MS}^{out} will be assigned the label $d^{out} \in \mathcal{D}^{v}$ that satisfies (2.54). However, the LLR meaning associated with that d^{out} must be adjusted.

Define the follow function:

$$d^{\mathrm{out}} = \mathtt{MS}(d_1, d_2), \tag{2.55}$$

where $d^{\mathrm{out}}, d_1, d_2 \in \mathcal{D}^{\mathrm{v}}$. (2.55) holds if and only if (2.53) and (2.54) and are both satisfied.

Define the binary operation $\circledast$ by:

$$\tilde{P}_{XD}(x,d) = P(X_1, D_1) \circledast P(X_2, D_2) \tag{2.56}$$

$$\triangleq \sum_{\substack{d_1,d_2:\mathtt{MS}(d_1,d_2)=d \\ x_1,x_2:x_1 \bigoplus x_2 = x}} P_{X_1D_1}(x_1,d_1) P_{X_2D_2}(x_2,d_2). \tag{2.57}$$

The joint PMF between code bit and external CN message in layer r and iteration t can be updated by:

$$P^{(t,r)}(X, D^{\mathrm{c}}) = P^{(t,r)}(X, D^{\mathrm{v}}) \circledast ... \circledast P^{(t,r)}(X, D^{\mathrm{v}}) \tag{2.58}$$

$$\triangleq P^{(t,r)}(X, D^{\mathrm{v}})^{\circledast(|\mathcal{C}(r)|-1)}. \tag{2.59}$$

$R^{(t,r)}(\cdot)$ can be directly computed using $P^{(t,r)}(X, D^c)$:

$$R^{(t,r)}(d) = \log \frac{P^{(t,r)}_{XD^c}(0,d)}{p^{(t,r)}_{XD^c}(1,d)},\ d \in \{0, ..., 2^{b^e} - 1\}. \tag{2.60}$$

2.5.4 Threshold

The threshold of a layer-specific RCQ decoder given a base matrix with M layers and maximum number of decoding iterations I_T is defined as:

$$\frac{E_b}{N_o}^* = \inf \left\{ \frac{E_b}{N_o} : I^{(I_T,r)} \left(\frac{E_b}{N_o} \right) > 1 - \epsilon, \forall r \in [1, M] \right\}. \tag{2.61}$$

2.6 Simulation Result and Discussion

This section presents RCQ and layer-specific RCQ decoder designs for two example LDPC codes and compares their FER performance with existing conventional decoders such as BP, Min Sum, and state-of-the-art non-uniform decoders, such as an IB decoder. All decoders are simulated using the AWGN channel, and at least 100 frame errors are collected for each point. We also compare hardware requirements for an example LDPC code.

2.6.1 IEEE 802.11 Standard LDPC Code

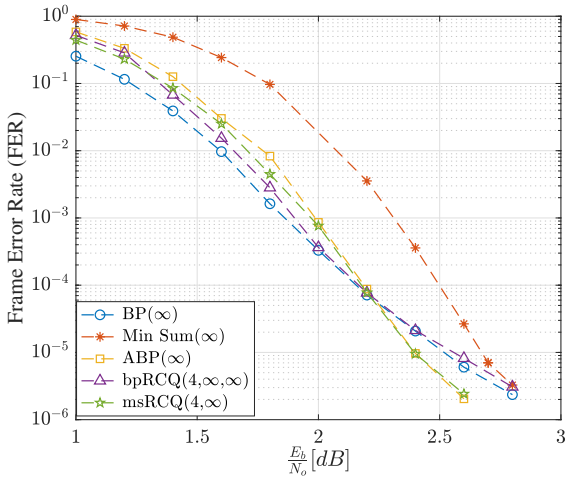
We first investigate the FER performance of RCQ decoders with a flooding schedule using an IEEE 802.11n standard LDPC code taken from [80212]. This code has $n = 1296$, $k = 648$, and the edge distribution is:

$$\lambda(x) = 0.2588x + 0.3140x^2 + 0.0465x^3 + 0.3837x^{10}, \tag{2.62}$$

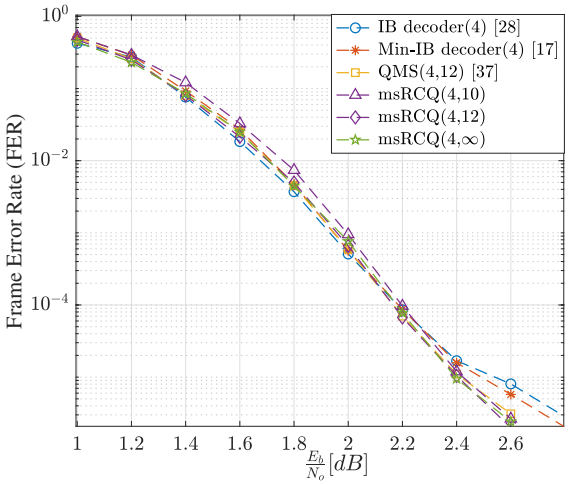
$$\rho(x) = 0.8140x^6 + 0.1860x^7. \tag{2.63}$$

The maximum number of decoding iterations was set to 50.

Fig. 2.9a shows the FER curves of 4-bit bpRCQ and msRCQ decoder with floating-point internal messages, i.e., bpRCQ(4,∞,∞) and msRCQ(4,∞), respectively . The notation of



(a) Decoders with floating point messages



(b) Decoders with fixed point messages

Figure 2.9: Fig. (a): FER performance of 4-bit msRCQ and bpRCQ decoders with floating point message representations use at the VNs. Fig. (b):FER performance of fixed point 4-bit msRCQ decoders, compared with other non-uniform quantization decoders.

∞ represents floating-point message representation. Denote floating point BP nad Min Sum by BP(∞) and Min Sum(∞), respectively. The 4-bit bpRCQ decoder has at most 0.1 dB degradation compared with the floating-point BP decoder, and outperforms floating-point BP at high $\frac{E_b}{N_o}$. The 4-bit msRCQ performs better than conventional Min Sumand even surpasses BP at high $\frac{E_b}{N_o}$. The lower error floor of msRCQ decoder as compared to standard BP follows from the slower message magnitude convergence rate as compared to standard BP. This is similar to improved error floors achieved by the averaged BP (ABP) [35], which decreases the rate of increase of message magnitudes by averaging the posteriors $l_v^{(t)}$ in consecutive iterations. As shown in Fig. 2.9a, ABP also delivers a lower error floor than standard BP.

The slow magnitude convergence rate of msRCQ decoder can be explained as follows. For conventional Min Sumdecoder, the magnitude of each check node message is always equal to the magnitude of an input variable node message for that CN. This is not true for the msRCQ decoder. msRCQ compares the relative LLR meanings of input messages and returns an external message by implementing the min operation. However, the external message is then reconstructed at the VN to an internal message magnitude that is in general different from the message magnitudes that were received by the neighboring CN.

For the example of a degree-3 CN, (2.64) computes the likelihood associated with a message l_t that is outputted from the min operation applied to the other two input messages indexed by i and j:

$$l_t = \log \frac{\sum_{\{(i,j)|t=MS(i,j)\}} P(0,i)P(0,j) + P(1,i)P(1,j)}{\sum_{\{(i,j)|t=MS(i,j)\}} P(1,i)P(0,j) + P(0,i)P(1,j)}. \tag{2.64}$$

Note that the boxplus operation is computed as follows :

$$l_i \boxplus l_j = \log \frac{P(0,i)P(0,j) + P(1,i)P(1,j)}{P(0,i)P(1,j) + P(1,i)P(0,j)}. \tag{2.65}$$

MinSum is an approximation to the boxplus operation, and boxplus produces a range of message values for edges that would share the same MinSum value $MS(i,j)$. Comparing

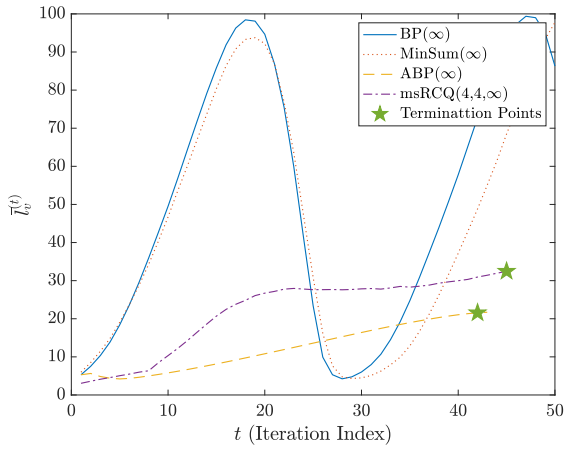


Figure 2.10: Average magnitudes of $l_v^{(t)}$ vs. iteration for BP, ABP, Min Sumand msRCQ for Fig. 6a simulation at $\frac{E_b}{N_o} = 2.6$ dB.

with (2.65), it can be seen that (2.64) applies the boxplus operation to the probability of the group of messages that share same value for $MS(i, j)$. Applying the boxplus operation to the *group* of messages produces a value that lies between the extremes of the messages produced by individual boxplus operations. This grouping process lowers the maximum output magnitude and therefore decreases the message magnitude growth rate in an iterative decoding process. As noted in [LM05], a possible indicator of the emergence of error trapping sets may be a sudden magnitude change in the values of certain variable node messages, or fast convergence to an unreliable estimate. Therefore, slowing down the convergence rate of VN messages can decrease the frequency of trapping set events. Both msRCQ decoder and A-BP in [LM05] reduce the the convergence rate of VN messages and hence deliver a lower error floor. However, A-BP requires extra computations to calculate the average message. On the other hand, the averaging process of msRCQ (i.e., 2.64) is inherent in $R(\cdot)$ and does not require additional complexity.

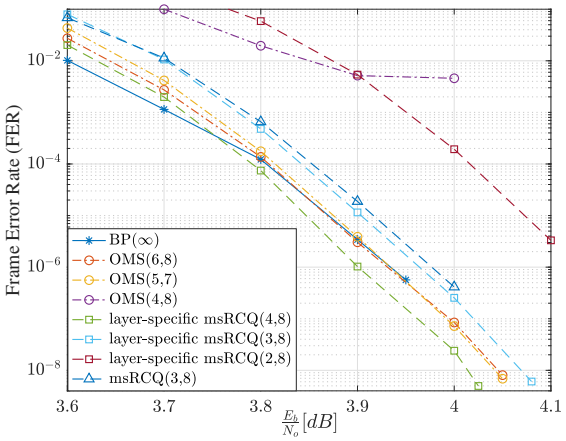
The effect of averaging can be seen in Fig. 2.10, which gives the average magnitude of $l_v^{(t)}$ for four decoders with a noise-corrupted all-zero codeword at $\frac{E_b}{N_o} = 2.6$ dB as the input. The oscillation pattern of the BP decoder has been reported and discussed in [LM05]. As shown

in Fig. 2.9a, ABP also outperforms belief propagation when $\frac{E_b}{N_o}$ is high. The oscillation occurs as errors alternate between the variable nodes that comprise the trapping set and their complement. Note that ABP requires extra computations to calculate the average message. However, the implicit averaging process of msRCQ (i.e., (2.64)) is inherent in $R(\cdot)$.

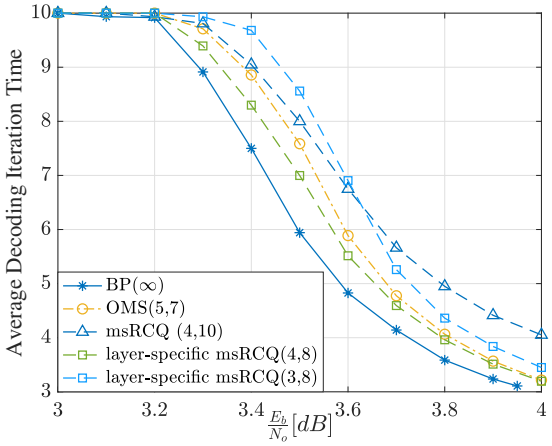
Fig. 2.9b compares msRCQ(4,10) with other non-uniform quantization LDPC decoders. Simulation results show that both IB [LB18b] and Min-IB [MM17] decoders exhibit an error floor after $2.40dB$. The MIM-QMS [KCH20] decoder has a similar decoding structure to msRCQ. Note that MIM-QMS requires the determination of the internal bit width used by the VNs before designing quantization and reconstruction parameters, so reducing the bit width of VNs requires another design cycle. In contrast, for the purposes of HDQ discrete density evolution design process, msRCQ assumes that the internal VN messages are real-valued. This assumption is an approximation since the internal VN messages will have finite precision in practical implementations. During actual decoding, the reconstruction operation $R(\cdot)$ produces a high-precision representation for use in computations at the VN. We found that assuming real-valued internal messages in the design process introduces negligible loss for practical internal message sizes while greatly simplifying the design. Our simulation results in 2.9b confirm that high precision internal messages have FER performance that is very close to real-valued internal messages. Actually, for the msRCQ, it is sufficient to have a simple clipping module at variable node, because all reconstructed values are fixed point messages. The RCQ decoder has more efficient memory usage than LUT-based decoders. For the investigated non-uniform LDPC code, 4-bit IB and 4-bit Min-IB require 14.43k and 10.24k bits, respectively, for storing LUTs per iteration, whereas msRCQ(4,12) and msRCQ(4,10) require 165 bits and 135 bits only.

2.6.2 (9472, 8192) QC-LDPC code

In this subsection we consider a rate-0.8649 quasi-regular LDPC code, with all VNs having degree 4 and CNs having degree 29 and 30, as might be used in a flash memory controller.



(a)



(b)

Figure 2.11: Fig. (a): FER performance of fixed point L-msRCQ decoders for (9472, 8192) LDPC code. Fig. (b): FER performance of fixed point L-msRCQ decoders for (9472, 8192) LDPC code.

Table 2.1: Hardware Usage of Various Decoding Structure for (9472,8192) QC-LDPC Code

Decoding Structure	LUTs	Registers	BRAMS	Routed Nets
OMS(5,7) (baseline)	21127	12966	17	29202
layer-specific RCQ(4,8)	20355(↓ 3.6%)	13967(↑ 7.0%)	17.5(↑ .03%)	28916(↓ 1%)
layer-specific RCQ(3,8)	17865(↓ 15.4%)	12098(↓ 6.7%)	17(−)	25332(↓ 13.3%)

We study this (9472, 8192) QC-LDPC code using various decoders with a *layered schedule*. The layer number of the investigated LDPC code is 10.

Fig. 2.11a shows the FER curves of various decoders. The maximum number of decoding iterations of all studied decoders is 10. The layer-specific msRCQ(4,8) outperforms msRCQ(4,10) by 0.04 dB, which shows the benefit of optimizing layer and iteration specific RCQ parameters. The layer-specific msRCQ(3,8) delivers similar decoding performance to msRCQ(4,10). The decoding performance of 2-bit layer-specific msRCQ has a 0.2 dB degradation compared with the 4-bit layer-specific msRCQ decoder. Given that $I_T = 10$, the thresholds of the investigated LDPC code under 4-bits msRCQ and 2-4 bit LS-msRCQ decoders are 3.58 dB, 3.67 dN, 3.46 dB and 3.40 dB, respectively. Fig. 2.11a also shows a fixed point offset Min Sum(OMS) decoder with offset factor 0.5. At a FER of 10^{-8}, OMS(6,8) and OMS(5,7) outperform layer-specific msRCQ(3,8) by 0.02 dB, yet are inferior to layer-specific msRCQ(4,8) by 0.02 dB. Fig. 2.11b shows the average decoding iteration times for some of the decoders studied in Fig. 2.11a. At high $\frac{E_b}{N_o}$, the msRCQ(4,10) decoder requires the largest average number of iterations to complete decoding. On the other hand, layer-specific msRCQ(4,8) has a similar decoding iteration time to OMS(5,7) and BP(∞) in this region. Layer-specific msRCQ(3,8) requires a slightly higher average number of iterations than layer-specific msRCQ(4,8) and OMS(5,7).

We implemented OMS and layer-specific msRCQ decoders with different bit widths on the programmable logic of a Xilinx Zynq UltraScale+ MPSoC device for comparison. Each design meets timing with a 500 MHz clock. The broadcast method described in [TWC21a] is

used for RCQ design. Table 2.1 summarizes the hardware usage of each decoder. Simulation result shows that layer-specific msRCQ(4,8) has a similar hardware usage with OMS(5,7), and layer-specific msRCQ(3,8) has more than a 10% reduction in LUTs and routed nets and more than a 6% reduction in registers, compared with OMS(5,7).

2.7 Conclusion

This chapter investigates the decoding performance and resource usage of RCQ decoders. For decoders using the flooding schedule, simulation results on an IEEE 802.11 LDPC code show that a 4-bit msRCQ decoder has a better decoding performance than LUT based decoders, such as IB decoders or Min-IB decoders, with significantly fewer parameters to be stored. It also surpasses belief propagation in the high $\frac{E_b}{N_o}$ region because a slower message convergence rate avoids trapping sets. For decoders using the layered schedule, conventional RCQ design leads to a degradation of FER performance and higher average decoding iteration time. Designing a layer-specific RCQ decoder, which updates parameters in each layer and iteration, improves the performance of a conventional RCQ decoder under a layered schedule. Layer-specific HDQ discrete density evolution is proposed to design parameters for RCQ decoders with a layered schedule. FPGA implementations of RCQ decoders are used to compare the resource requirements of the decoders studied in this paper. Simulation results for a (9472, 8192) QC LDPC code show that a layer-specific Min SumRCQ decoder with 3-bit messages achieves a more than 10% reduction in LUTs and routed nets and a more than 6% register reduction while maintaining comparable decoding performance, compared to a 5-bit offset Min Sum decoder.

CHAPTER 3

RCQ LDPC Decoding with Degree-Specific Neural Edge Weights

3.1 Introduction

Low-Density Parity-Check (LDPC) codes [Gal62b] has been implemented broadly, including in NAND flash systems and wireless communication systems. In practice, decoders for LDPC codes with low message bit widths are desired when considering the limited hardware resources on the field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs), such as area, routing capabilities, and power utilization. Unfortunately, low-bitwidth decoders with uniform quantizers typically suffer a large degradation in decoding performance [LT05b]. Recently, the non-uniformly quantized decoders [PDD13b, XVT20b, LB18a, SLB18, SBW20b, LT05b, HCM19b, WWS20b, TWC21b, WTS22] have shown to deliver excellent performance with very low message precision. One promising decoding paradigm is called reconstruction-computation-quantization (RCQ) decoder [WWS20b, TWC21b, WTS22].

The node operation in an RCQ decoder involves a reconstruction function that allows high-precision message computation and a quantization function that allows low-precision message passing between nodes. Specifically, the reconstruction function, equivalent to a dequantizer, maps the low-bitwidth messages received by a node to high-bitwidth messages for computation. The quantization function quantizes the calculated high-bitwidth messages to low-bitwidth messages that will be sent to its neighbor nodes. As shown in [TWC21b],

the 4-bit layer-scheduled RCQ decoder can have a better decoding performance than the 6-bit uniformly quantized offset MinSum (OMS) decoder.

The excellent decoding performance of RCQ decoder comes from its dynamic quantizers and dequantizers that are updated in each layer and each iteration. However, for practical consideration, the dynamic quantizers and dequantizers mean more look-up tables (LUTs). What's worse, the LUTs required for storing quantizers and dequantizers may offset the LUTs saved by using low bit width to pass messages. As reported in [WTS22], the 4-bit RCQ decoder has a similar LUT usage to the 5-bit OMS decoder for a (9472,8192) LDPC code.

Recently, numerous works have been focused on enhancing the performance of message-passing decoders with the help of neural networks (NNs) [NBB16a, LG17, NMB17, NML18, LSW18, WJZ18, LG18, LZJ18, XVT19b, DB19, ABS19, BHP20, WWF20, LCH19, NWH21]. Nachmani *et al.* in [NBB16a] propose a neural belief propagation (N-BP) decoder that assigns NN-learned multiplicative weights to the BP messages. Nachmani *et al.* and Lugosch *et al.* in [NML18, LG17, NBB16a] assign dynamic weights to the messages in normalized MinSum (NMS) and OMS decoder and propose the Neural NMS (N-NMS) and Neural OMS (N-OMS) decoder, respectively.

For the above neural decoders, each check-to-variable message and/or each variable-to-check message is assigned a distinct weight in each iteration. These neural decoders are impractical for long-blocklength LDPC codes because the number of required parameters is proportional to the number of edges in the Tanner graph corresponding to the parity check matrix. One solution is to share the weights across iterations or edges in the Tanner graph, like in [NMB17, WWF20, ABS19, LCH19]. However, these simple weight-sharing methods sacrifice decoding performance in different ways. Besides, the precursor works of literature are mainly focused on the short-blocklength codes ($n < 2000$), which may have resulted from the fact that the required memory for training neural decoders with long block lengths by using popular deep learning research platforms, such as Pytorch, is larger than the

computation resources of the researchers. However, as demonstrated in [ABS19, WCN21], it is possible to train neural decoders by only using CPUs on personal computers for very long-blocklength codes if resources are handled more efficiently.

3.1.1 Contribution

This paper combines the recent neural decoding with RCQ decoding paradigm and proposes a weighted RCQ (W-RCQ) decoder. Unlike RCQ decoder, whose quantizers/dequantizers change in each layer and iteration, the W-RCQ decoder limits the number of quantizer and dequantizer pairs to a very small number, for example, three. However, the W-RCQ decoder weights check-to-variable node messages using dynamic parameters optimized via a quantized NN (QNN). The proposed W-RCQ decoder uses fewer parameters than the RCQ decoder, requiring much fewer LUTs.

The novelties and contributions of this paper are summarized as follows:

- *Posterior Joint Training Method.* This paper identifies the gradient explosion issue when training neural LDPC decoders. A posterior joint training method is proposed in this paper to address the gradient explosion problem. Simulation results show that posterior joint training delivers better decoding performance than simply clipping large-magnitude gradients to some threshold value.

- *Node-Degree-Based Weight Sharing.* This paper illustrates that the weight values of the N-NMS decoder are strongly related to check node degree, variable node degree, and iteration index. As a result, this paper proposes node-degree-based weight-sharing schemes that assign the same weight to the edges with the same check and/or variable node degree.

- *Neural-2D-MinSum decoder.* By employing the node-degree-based weight sharing on the N-NMS and N-OMS decoder, this paper proposes the N-2D-NMS decoder and N-2D-OMS decoder. *2D* represents for 2-dimensional and implies that the weights

are shared based on two dimensions, i.e., check node degree and variable node degree. Simulation results on the (16200,7200) DVBS-2 LDPC code show that the N-2D-NMS decoder can achieve same decoding performance as N-NMS decoder.

- *W-RCQ Decoder.* This paper proposes the W-RCQ decoding paradigm. Our simulation result for a (9472,8192) LDPC code on a field-programmable gate array (FPGA) device shows that the 4-bit W-RCQ decoder delivers comparable FER performance but with much fewer hardware resources, compared with the 4-bit RCQ decoder and the 5-bit offset MinSum decoder.

3.1.2 Organization

The remainder of this paper is organized as follows: Section 3.2 derives the gradients for a flooding-scheduled N-NMS decoder and shows that the memory to calculate the gradients can be saved by storing the forward messages compactly. The derivations enable one to build and train the NNs for the neural decoders using programmable languages such as C++. The compact message format saves the required memory for training NNs. This section also describes the posterior joint training method that addresses the gradient explosion issue. Section 3.3 illustrates the relationship between neural weights of N-NMS decoder and node degree. The node-degree-based weight-sharing scheme is presented in this section. This section also gives neural-2D-MinSum decoders. Section 3.4 gives the W-RCQ decoding structure and describes how to train W-RCQ parameters via a QNN. The simulation results are presented in Section 3.5, and Section 3.6 concludes our work.

3.2 Training Neural MinSum Decoders for Long Blocklength Codes

For the neural network corresponding to a neural LDPC decoder, the number of neurons in each hidden layer equals the number of edges in the Tanner graph corresponded to the

parity check matrix [NBB16a]. For the popular NN platforms, such as PyTorch, each neuron requires a data structure that stores the value of the neuron, the gradient of the neuron, the connection of this neuron with other neurons, and so on. Hence, there is a huge memory requirement for PyTorch to train the neural decoders for long-blocklength LDPC codes. (difficult for researchers with limited resources usage)

The data structure used in PyTorch is useful and convenient for conventional deep neural network tasks but redundant to the neural LDPC decoders. One reason is that the neuron connections between hidden layers are repetitive and can be interpreted by the parity check matrix. This immediately reduces the required memory. This section uses N-NMS decoder as an example to show that the memory required to calculate gradients of neural MinSum decoders can be further reduced by storing the messages in forward propagation compactly.

3.2.1 Forward Propagation of N-NMS Decoder

Let $H \in \mathbb{F}_2^{n\times k}$ be the parity check matrix of an (n, k) binary LDPC code, where n is the codeword length and k is dataword length. Denote i^{th} variable node and j^{th} check node by v_i and c_j, respectively. For the flooding-scheduled decoder, in the t^{th} decoding iteration, N-NMS decoder updates the check-to-variable (C2V) message, $u^{(t)}_{c_j\to v_i}$, by:

$$u^{(t)}_{c_i\to v_j} = \beta^{(t)}_{(c_i,v_j)} \times \prod_{v_{j'}\in\mathcal{N}(c_i)\backslash\{v_j\}} \operatorname{sgn}\left(l^{(t-1)}_{v_{j'}\to c_i}\right) \times \min_{v_{j'}\in\mathcal{N}(c_i)\backslash\{v_j\}} \left|l^{(t-1)}_{v_{j'}\to c_i}\right|, \tag{3.1}$$

$\mathcal{N}(c_i)$ is the set of variable nodes that connect c_i and $\left\{\beta^{(t)}_{(c_i,v_j)} | i \in \{1,\dots k\}, j \in \{1,\dots n\}, H(i,j) = 1, t \in \{1,\dots,I_T\}\right\}$ is the set of trainable parameters. I_T represents the maximum iteration. The variable-to-check (V2C) message, $l^{(t)}_{v_i\to c_j}$, and posterior of each variable node, $l^{(t)}_{v_i}$, of N-NMS decoder in iteration t are calculated by:

$$l^{(t)}_{v_j\to c_i} = l^{ch}_{v_j} + \sum_{c_{i'}\in\mathcal{N}(v_j)\backslash\{c_i\}} u^{(t)}_{c_{i'}\to v_j}, \tag{3.2}$$

$$l^{(t)}_{v_j} = l^{ch}_{v_j} + \sum_{c_{i'}\in\mathcal{N}(v_j)} u^{(t)}_{c_{i'}\to v_j}. \tag{3.3}$$

$\mathcal{N}(v_j)$ represents the set of the check nodes that are connected to v_j. $l_{v_j}^{ch}$ is the log likelihood ratio (LLR) of channel observation of v_j. The decoding process stops when all parity checks are satisfied or I_T is reached.

3.2.2 Backward Propagation of N-NMS

Before performing back propagation to calculate the gradients, define $\texttt{min1}_{c_i}^t$, $\texttt{pos1}_{c_i}^t$, $\texttt{min2}_{c_i}^t$ and $\texttt{pos2}_{c_i}^t$ as follows:

$$\texttt{min1}_{c_i}^t = \min_{v_{j'} \in \mathcal{N}(c_i)} |l_{v_{j'} \to c_i}^{(t)}|, \qquad \texttt{pos1}_{c_i}^t = \operatorname*{argmin}_{v_{j'} \in \mathcal{N}(c_i)} |l_{v_{j'} \to c_i}^{(t)}|. \tag{3.4}$$

$$\texttt{min2}_{c_i}^t = \min_{v_{j'} \in \mathcal{N}(c_i)/\{\texttt{pos1}_{c_i}^t\}} |l_{v_{j'} \to c_i}^{(t)}|, \qquad \texttt{pos2}_{c_i}^t = \operatorname*{argmin}_{v_{j'} \in \mathcal{N}(c_i)/\{\texttt{pos1}_{c_i}^t\}} |l_{v_{j'} \to c_i}^{(t)}|. \tag{3.5}$$

$\texttt{min1}_{c_i}^t$ is the minimum magnitude that c_i receives in iteration t, and the minimum magnitude is provided by the variable node $\texttt{pos1}_{c_i}^t$. Similarly, $\texttt{min2}_{c_i}^t$ is the second minimum magnitude that c_i receives in iteration t, and the second minimum magnitude is provided by $\texttt{pos2}_{c_i}^t$.

Let J be some loss function for N-NMS neural network, for example, the multi-loss cross entropy in [NBB16a]. Denote the gradients of loss J with respect to (w.r.t.) the trainable weights, the C2V message and V2C message by $\frac{\partial J}{\partial \beta_{(c_i,v_j)}^{(t)}}$, $\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}$, and $\frac{\partial J}{\partial l_{v_j \to c_i}^{(t)}}$, respectively.

In iteration t, $\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}$ is updated as follows:

$$\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}} = \frac{\partial J}{\partial l_{v_j}^{(t)}} + \sum_{c_{i'} \in \mathcal{N}(v_j) \setminus \{c_i\}} \frac{\partial J}{\partial l_{v_j \to c_{i'}}^{(t)}}. \tag{3.6}$$

$\frac{\partial J}{\partial \beta_{(c_i,v_j)}^{(t)}}$ is calculated using $\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}$ by:

$$\frac{\partial J}{\partial \beta_{(c_i,v_j)}^{(t)}} = \frac{u_{c_i \to v_j}^{(t)}}{\beta_{(c_i,v_j)}^{(t)}} \frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}. \tag{3.7}$$

Let $u_{c_i \to v_j}^{(t)*} = \frac{u_{c_i \to v_j}^{(t)}}{\beta_{(c_i,v_j)}^{(t)}}$. Note that $u_{c_i \to v_j}^{(t)*}$ is the output of check node Min operation, and hence can be calculated efficiently by knowing $\operatorname{sgn}(l_{v_j \to c_i}^{(t)})$, $\texttt{min1}_{c_i}^t$, $\texttt{min2}_{c_i}^t$, $\texttt{pos1}_{c_i}^t$. To see this,

$$\operatorname{sgn}(u_{c_i \to v_j}^{(t)*}) = \prod_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \operatorname{sgn}(l_{v_{j'} \to c_i}^{(t-1)}), \tag{3.8}$$

$$|u_{c_i\to v_j}^{(t)*}| = \begin{cases} \texttt{min2}_{c_i}^t, & \text{if } v_j = \texttt{pos1}_{c_i}^t \\ \texttt{min1}_{c_i}^t, & \text{otherwise} \end{cases}. \tag{3.9}$$

For all variable nodes connected to check node c_i, in iteration t, only $\texttt{pos1}_{c_i}^{(t)}$ and $\texttt{pos2}_{c_i}^{(t)}$ receive backward information. Hence, $\frac{\partial J}{\partial l_{v_j\to c_i}^{(t-1)}}$ is computed as follows:

$$\frac{\partial J}{\partial l_{v_j\to c_i}^{(t-1)}} = \begin{cases} \text{sgn}\left(l_{v_j\to c_i}^{(t-1)}\right) \sum_{v_{j'}\in\mathcal{N}(c_i)\setminus\{v_j\}} \frac{\partial J}{\partial |u_{c_i\to v_{j'}}^{(t)*}|} & , v_j = \texttt{pos1}_{c_i}^{(t)} \\ \text{sgn}\left(l_{v_j\to c_i}^{(t-1)}\right) \frac{\partial J}{\partial \left|u_{c_i\to \texttt{pos1}_{c_i}^{(t)}}^{(t)*}\right|} & , v_j = \texttt{pos2}_{c_i}^{(t)} \\ 0 & , \text{otherwise.} \end{cases} \tag{3.10}$$

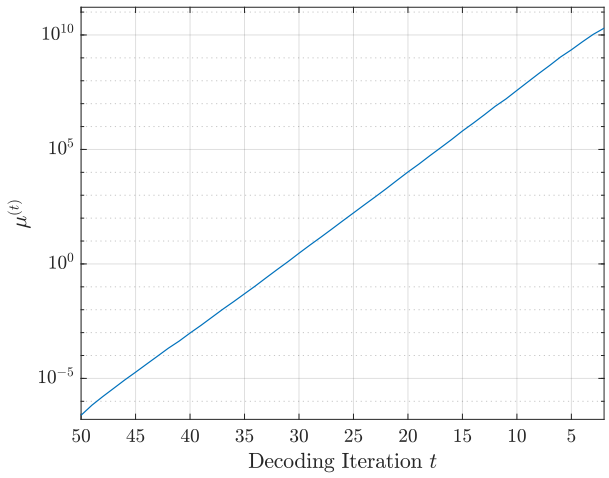
The term $\frac{\partial J}{\partial |u_{c_i\to v_j}^{(t)*}|}$ is calculated by:

$$\frac{\partial J}{\partial |u_{c_i\to v_j}^{(t)*}|} = \text{sgn}(u_{c_i\to v_j}^{(t)*})\beta_{(c_i,v_j)}^{(t)} \frac{\partial J}{\partial u_{c_i\to v_j}^{(t)}}. \tag{3.11}$$

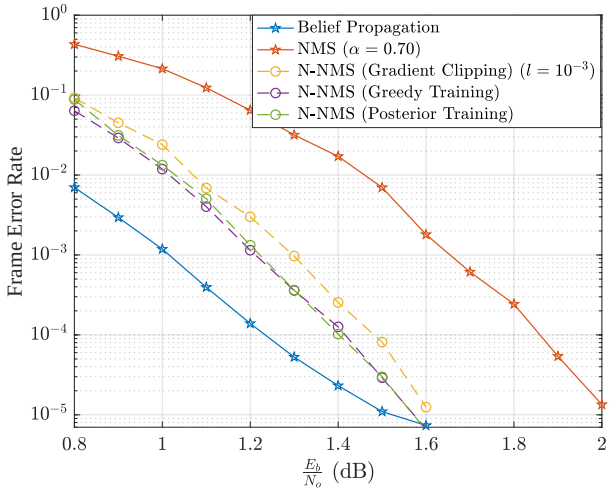
(3.6)-(3.11) indicate that the neuron values in each hidden layer can be stored compactly with $\text{sgn}\left(l_{v_j\to c_i}^{(t)}\right)$, $\texttt{min1}_{c_i}^t$, $\texttt{min2}_{c_i}^t$, $\texttt{pos1}_{c_i}^t$ and $\texttt{pos1}_{c_i}^t$. The compactly-stored neural values in the hidden layers leads to a significant memory reduction. Besides, (3.6), (3.10) and (3.11) imply that $\frac{\partial J}{\partial u_{c_i\to v_j}^{(t)}}$ and $\frac{\partial J}{\partial l_{v_j\to c_i}^{(t)}}$ can be calculated iteratively. Hence, the memory to store the gradients in two consecutive hidden layers, rather than all hidden layers, is sufficient to perform back propagation and calculate $\frac{\partial J}{\partial \beta_{(c_i,v_j)}^{(t)}}$. Note that the compact message format (3.4)-(3.5) is also used in the hardware implementation for the MinSum decoders.

3.2.3 Posterior Jointly Training

Equation (3.10) implies that in iteration t, for all variable nodes that connect check node c, only $\texttt{pos1}_c^t$ and $\texttt{pos2}_c^t$ receive gradients from c. Besides, $|\mathcal{N}(c)| - 1$ gradient terms flow to $\texttt{pos1}_c^1$. Hence, if check node c has a large degree, the gradient of J w.r.t. $\texttt{pos1}_c^t$ can have a large magnitude, and this large-magnitude gradient will be propagated to the neurons in the preceding layer that corresponded to the C2V messages whose check nodes (other than c)



(a)



(b)

Figure 3.1: Fig. (a): The average magnitude of gradients of loss J w.r.t. C2V messages in each decoding iteration. The gradients are calculated by feeding the flooding-scheduled (3096,1032) N-NMS decoder with an input sample and performing backward propagation. Fig. (b): FER curves of the flooding-scheduled N-NMS decoders for a (3096,1032) LDPC code. Gradient clipping, greedy training and posterior jointly training are used to address gradient explosion issue. The maximum decoding iteration is 50. The belief propagation decoder and NMS decoder with factor 0.7 are presented as comparison.

connect $\texttt{pos1}_c^t$. As a result, the large-magnitude gradients are accumulated and propagated as back propagation proceeds, which results in gradient explosion.

Fig. 3.1a shows the gradient explosion phenomenon when training a flooding-scheduled N-NMS decoder for a (3096,1032) LDPC code. Define $\mu^{(t)}$ as the average magnitude of the gradients of J w.r.t. all C2V messages in iteration t. The gradients are calculated by feeding the N-NMS decoder with some input sample and then performing backward propagation. Fig. 3.1a plots $\mu^{(t)}$ in each decoding iteration. The maximum check node degree and variable node degree of the code are 19 and 27, respectively. The maximum decoding iteration of the decoder is 50. It can be seen that the $\mu^{(t)}$ increases exponentially with the decrease of decoding iteration t.

(3.7) indicates that large magnitude of $\frac{\partial J}{\partial u_{(c,v)}^{(t)}}$ leads to large magnitude of $\frac{\partial J}{\partial \beta_{(c,v)}^{(t)}}$ and hence prevents the neural network from optimizing weights effectively. To the best of our knowledge, this paper is the first one to report the gradient explosion issue for neural LDPC decoder training. However, there have been several techniques that solve the gradient explosion problem:

1. *Gradient Clipping.* Gradient explosion is a common problem in the deep learning field such as recurrent neural network, and one way to solve this problem is gradient clipping [GBC16]. There are various methods for gradient clipping [Mik12, PMB13]. This paper considers to simply limit the maximum gradient magnitude to be some threshold l.

2. *Greedy Training.* Dai *et al* in [DTS21] proposed greedy training. Greedy training trains the parameters in t^{th} decoding iteration by fixing the pre-trained parameters in the first $t-1$ iterations. Greedy training solves the gradient explosion problem because the large magnitude gradients won't be accumulated and propagated to the preceding hidden layers, i.e, decoding iterations. However, greedy training requires a time complexity that is proportional to I_T^2, because one must have trained the $(t-1)$-iterations decoder

in order to train a t-iterations decoder.

(3.6) indicates that the gradient of J w.r.t. $u^{(t)}_{c_i \to v_j}$ comes from two parts: the first part is from the posterior $l^{(t)}_{v_j}$, and the second part is from the V2C messages $l^{(t)}_{v_j \to c'_i}$, $c_{i'} \in \mathcal{N}(v_j) \backslash \{c_i\}$. Based on the previous analysis, if any $l^{(t)}_{v_j \to c'_i}$, $c_{i'} \in \mathcal{N}(v_j) \setminus \{c_i\}$ has a large magnitude gradient, the neuron $u^{(t)}_{c_i \to v_j}$ can also have a large magnitude gradient. This will result in a large magnitude to the gradient of J w.r.t. $\beta^{(t)}_{(c_i, v_j)}$, as indicated by (3.7). In this paper, we propose posterior jointly training which calculates the gradient of J w.r.t. $u^{(t)}_{c_i \to v_j}$ only using the posterior $l^{(t)}_{v_j}$. More explicitly, for the flooding-scheduled N-NMS neural network, $\frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}}$ is calculated by:

$$\frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}} = \frac{\partial J}{\partial l^{(t)}_{v_j}}. \tag{3.12}$$

Hence, the gradient of J w.r.t. $\beta^{(t)}_{(c_i, v_j)}$ is calculated as:

$$\frac{\partial J}{\partial \beta^{(t)}_{c_i \to v_j}} = \frac{u^{(t)}_{c_i \to v_j}}{\beta^{(t)}_{c_i \to v_j}} \frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}} = \frac{u^{(t)}_{c_i \to v_j}}{\beta^{(t)}_{c_i \to v_j}} \frac{\partial J}{\partial l^{(t)}_{v_j}}. \tag{3.13}$$

By calculating the gradients of neurons in the t^{th} decoding iteration only using $l^{(t)}$, i.e., the posteriors in the t^{th} decoding iteration, (3.12) and (3.13) prevent the large magnitudes that are due to $\frac{\partial J}{\partial l^{(t)}_{v_j \to c_{i'}}}$ from propagating to the preceding hidden layers. This idea resembles the greedy training method. However, the posterior jointly training optimizes parameters of all decoding iterations jointly, hence it requires a time complexity that is proportional to I_T.

For the layer-scheduled N-NMS decoder, the conventional back propagation calculates the gradient of J w.r.t. $u^{(t)}_{c_i \to v_j}$ by:

$$\frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}} = \frac{\partial J}{\partial l^{(t)}_{v_j}} + \sum_{\{i' | c_{i'} \in \mathcal{N}(v_j), i' > i\}} \frac{\partial J}{\partial l^{(t)}_{v_j \to c_{i'}}} + \sum_{\{i' | c_{i'} \in \mathcal{N}(v_j), i' < i\}} \frac{\partial J}{\partial l^{(t+1)}_{v_j \to c_{i'}}}. \tag{3.14}$$

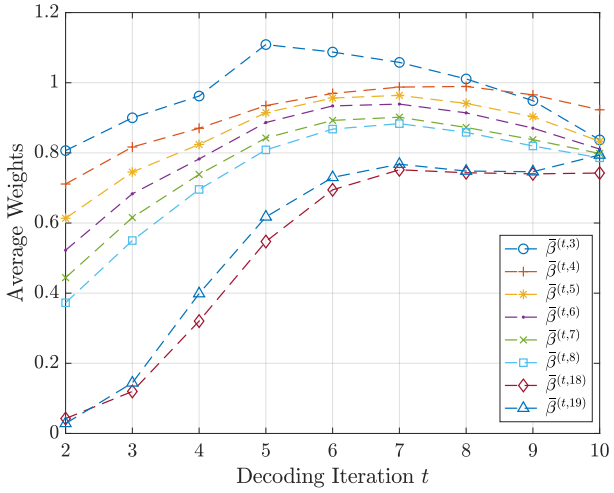
Posterior jointly training abandons the last term in (3.14) and calculates $\frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}}$ as follows:

$$\frac{\partial J}{\partial u^{(t)}_{c_i \to v_j}} = \frac{\partial J}{\partial l^{(t)}_{v_j}} + \sum_{\{i' | c_{i'} \in \mathcal{N}(v_j), i' > i\}} \frac{\partial J}{\partial l^{(t)}_{v_j \to c_{i'}}}. \tag{3.15}$$

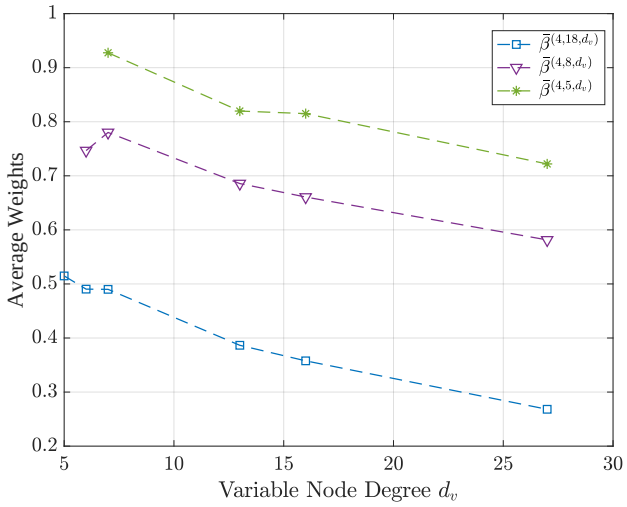
Fig. 3.1b shows the frame error rate (FER) of flooding-scheduled N-NMS decoders for a (3096,1032) LDPC code. The maximum decoding iteration time is 50. All three methods are implemented for preventing gradient explosion. Especially, for the gradient clipping, the threshold for gradient magnitude is $l = 10^{-3}$. The performance of BP and NMS with the same decoding schedule and maximum decoding iteration are also given for comparison. The NMS decoder uses multiplicative factor 0.7. The simulation result shows that greedy training and posterior jointly training deliver a better better performance than simple gradient clipping method. Greedy training and posterior jointly training deliver a similar performance, both of which are 0.4 dB better than conventional NMS decoder and have a better performance than BP at 1.6 dB. However, posterior jointly training has a lower time complexity than greedy training.

3.3 Node-Degree-Based Weight Sharing

N-NMS and N-OMS decoder for the long-blocklength LDPC codes are impractical, because the number of parameters of these decoders is proportional to the number of edges in the corresponding Tanner graph. Weight sharing [XCB21] solves this problem by assigning one weight to different neurons in the NN. Different weight sharing schemes have been proposed to reduce the number of neural weights in N-NMS and N-OMS decoder. However, simple weight sharing schemes, such as across iterations or edges in [WWF20, LCH19], degrade the decoding performance in different degrees. This section proposes node-degree-based weight sharing schemes which assign the same weights to the edges that have same check and/or variable node degree. We call the N-NMS and N-OMS decoder with node-degree-based weight sharing schemes by neural 2-dimensional NMS (N-2D-NMS) and neural 2-dimensional OMS (N-2D-OMS) decoder, respectively, because they are similar to the 2D-MS decoders in [JFD05, ZFG06]. Simulation results in Section 3.5 show that N-2D-NMS decoder can deliver the same decoding performance with N-NMS decoder.



(a)



(b)

Figure 3.2: Mean values of messages of a flooding-scheduled N-NMS decoder for a (3096,1032) LDPC code in each iteration show strong correlations to check and variable node degree.

3.3.1 Motivation

In this subsection, we investigate the relationship between the neural weights of a flooding-scheduled N-NMS decoder and node degrees. The N-NMS decoder is trained for a (3096, 1032) LDPC code, the same one used in Section 3.2.3. The maximum decoding iteration is 10.

Define the set of neural weights of N-NMS decoder that are associated to check node degree d_c in the t^{th} decoding iteration by $\mathcal{B}^{(t,d_c)}$, and $\mathcal{B}^{(t,d_c)} = \{\beta^{(t)}_{(c_i,v_j)} | \deg(c_i) = d_c\}$. Let $\bar{\beta}^{(t,d_c)}$ be the mean value of $\mathcal{B}^{(t,d_c)}$. Fig.3.2a shows $\bar{\beta}^{(t,d_c)}$ versus decoding iteration t with all possible check node degrees. The simulation result shows a clear relationship between check node degree and $\bar{\beta}^{(t,d_c)}$, i.e. a larger check node degree corresponds to a smaller $\bar{\beta}^{(t,d_c)}$. This difference is significant in the first few iterations. Additionally, $\bar{\beta}^{(t,d_c)}$ changes drastically in first few iterations for all check node degrees.

In order to explore the relationship between the weights and variable node degrees given a check node degree d_c and decoding iteration index t, we further define $\mathcal{B}^{(t,d_c,d_v)} = \{\beta^{(t)}_{(c_i,v_j)}$ $|\deg(c_i) = d_c, \deg(v_i) = d_v\}$. We denote the average value of $\mathcal{B}^{(t,d_c,d_v)}$ by $\bar{\beta}^{(t,d_c,d_v)}$. Fig.3.2b gives the average weights corresponding to various check and variable node degrees at iteration 4. Statistical results show that, given a specific iteration t and check node degree d_c, a larger d_v indicates a smaller $\bar{\beta}^{(t,d_c,d_v)}$.

In conclusion, the weights of N-NMS decoder are correlated with check node degree, variable node degree, and decoding iteration index. Thus, node degrees should affect the weighting of messages on their incident edges when decoding LDPC codes. This observation motivates us to propose a family of N-2D-MS decoders in this paper.

Table 3.1: Various Node-Degree-Based Weight Sharing Schemes and Required Number of Parameters per Iteration for Two Example Codes

Type	$\beta_*^{(t)}$	$\alpha_*^{(t)}$	The Number of Required Parameters per Iteration	
			(16200,7200) DVBS-2 code	(3096,1032) PBRL code
No Weight Sharing [NBB16a]				
0	$\beta_{(c_i,v_j)}^{(t)}$	1	$4.8 * 10^5$	$1.60 * 10^4$
Weight Sharing Based on Node Degree				
1	$\beta_{(\deg(c_i),\deg(v_j))}^{(t)}$	1	13	41
2	$\beta_{(\deg(c_i))}^{(t)}$	$\alpha_{(\deg(v_j))}^{(t)}$	8	15
3	$\beta_{(\deg(c_i))}^{(t)}$	1	4	8
4	1	$\alpha_{(\deg(v_j))}^{(t)}$	4	7
Weight Sharing Based on Protomatrix				
5 [DTS21]	$\beta_{\left(\left\lfloor \frac{i}{f} \right\rfloor,\left\lfloor \frac{j}{f} \right\rfloor\right)}^{(t)}$	1	–	101
6	$\beta_{\left(\left\lfloor \frac{i}{f} \right\rfloor\right)}^{(t)}$	1	–	17
7	1	$\alpha_{\left(\left\lfloor \frac{j}{f} \right\rfloor\right)}^{(t)}$	–	25
Weight sharing based on Iteration [LCH19, ABS19]				
8	$\beta^{(t)}$	1	1	1

3.3.2 Neural 2D Normalized MinSum Decoders

Based on the previous discussion, it is intuitive to consider assigning the same weights to messages with same check node degree and/or variable node degree. In this subsection, we propose a family node-degree-based weight sharing schemes. These weight sharing schemes can be used on the N-NMS decoder, which gives N-2D-NMS decoder.

In the t^{th} iteration, a flooding-scheduled N-2D-NMS decoder update $u_{c_i \to v_j}^{(t)}$ as follows:

$$u_{c_i \to v_j}^{(t)} = \beta_*^{(t)} \times \prod_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \operatorname{sgn}\left(l_{v_{j'} \to c_i}^{(t-1)}\right) \times \min_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \left| l_{v_{j'} \to c_i}^{(t-1)} \right|. \tag{3.16}$$

$$l_{v_j \to c_i}^{(t)} = l_{v_i}^{ch} + \alpha_*^{(t)} \sum_{c_{i'} \in \mathcal{N}(v_j)/\{c_i\}} u_{c_{i'} \to v_j}^{(t)}, \tag{3.17}$$

$$l_{v_j}^{(t)} = l_{v_i}^{ch} + \alpha_*^{(t)} \sum_{c_{i'} \in \mathcal{N}(v_j)} u_{c_{i'} \to v_j}^{(t)}. \tag{3.18}$$

$\beta_*^{(t)}$ and $\alpha_*^{(t)}$ are the learnable weights. The subscript * is replaced in Table 3.1 with the information needed to identify the specific weight depending on the weight sharing methodology. Table 3.1 lists different weight sharing types, each identified in the first column by a type number. As a special case, we denote type 0 by assigning distinct weights to each edge, i.e., N-NMS decoder. Columns 2 and 3 describe how each type assigns $\beta_*^{(t)}$ and $\alpha_*^{(t)}$, respectively. In this paper, we refer to a decoder that uses a type-x weight sharing scheme as a type-x decoder.

Types 1-4 assign the same weights based on node degree. In particular, Type 1 assigns the same weight to the edges that have same check node *and* variable node degree. Type 2 considers the check node degree and variable node degree separately. As a simplification, type 3 and type 4 only consider check node degree and variable node degree, respectively.

Dai. *et. al* in [DTS21] studied weight sharing based on the edge type of multi-edge-type (MET)-LDPC codes, or protograph-based codes. We also consider this metric for types 5, 6, and 7. Type 5 assigns the same weight to the edges with the same edge type, i.e., the edges that belong to the same position in protomatrix. In Table. 3.1, f is the lifting factor. Types 6 and 7 assign parameters based only on the horizontal (protomatrix row) and vertical layers (protomatrix column), respectively. Finally, type 8 assigns a single weight to all edges in each decoding iteration, as in [LCH19, ABS19].

A (3096,1032) LDPC code and the (16200,7200) DVBS-2 [ETS] standard LDPC code are considered in this section, and the number of parameters per iteration required for various

weight sharing schemes of these two codes are listed in column 4 and 5 in Table. 3.1, respectively. It is shown that the number of parameters required by the node-degree-based weight sharing is less than that required by the protomatrix-based weight sharing.

3.3.3 Neural 2D Offset MinSum Decoder

The node-degree-based weight sharing schemes can be applied to N-OMS decoder in a similar way and lead to neural 2D OMS (N-2D-OMS) decoder. Specifically, a flooding N-2D-OMS decoder updates $u_{c_i \to v_j}^{(t)}$ by:

$$u_{c_i \to v_j}^{(t)} = \prod_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \operatorname{sgn}\left(l_{v_{j'} \to c_i}^{(t-1)}\right) \times \operatorname{ReLu}\left(\min_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \left|(l_{v_{j'} \to c_i}^{(t-1)})\right| - \beta_*^{(t)} - \alpha_*^{(t)}\right). \quad (3.19)$$

$\operatorname{ReLu}(x) = \max(0, x)$. The $l_{v_j \to c_i}^{(t)}$ and $l_{v_j}^{(t)}$ are updated using (3.2) and (3.3). For the N-2D-OMS decoders, the constant value 1 in Table 3.1 should be replaced by 0.

3.3.4 Hybrid Neural Decoder

To further reduce the number of parameters, we consider a hybrid training structure that utilizes a neural network combining a feed forward module with a recurrent module. The corresponding decoder uses distinct neural weights for each of the first I' decoding iterations and uses the same weights for the remaining $I_T - I'$ iterations. The motivation for the hybrid decoder is from the observation that the neural weights of N-NMS decoder change drastically in the first few iterations, but negligibly during the last few iterations, as illustrated in Fig. 3.2. Therefore, using the same parameters for the last few iterations doesn't cause a large performance degradation.

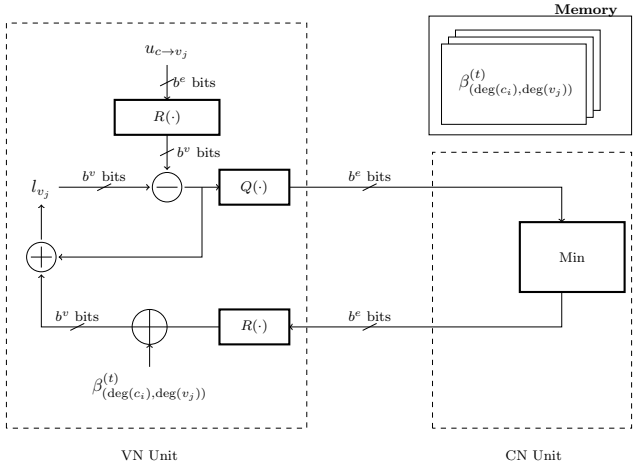


Figure 3.3: Layer-scheduled Neural Offset RCQ Decoder Structure

3.4 Weighted RCQ Decoder

This section combines the N-2D-NMS or N-2D-OMS decoder with RCQ decoding paradigm and proposes a weighted RCQ (W-RCQ) decoder. Unlike the RCQ decoder, whose quantizers and de-quantizers are updated in each iteration (and each layer, if layer-scheduled decoding is considered), W-RCQ decoder only uses a small number of quantizers and dequantizers during the decoding process. However, the C2V messages of W-RCQ decoder will be weighted by dynamic node-degree-based parameters that that are trained by a QNN.

3.4.1 Structure

Fig. 3.3 gives the decoding paradigm of a layer-scheduled weighted OMS-RCQ decoder (W-OMS-RCQ). The offset parameters in Fig. 3.3, $\beta^{(t)}_{(\deg(c_i),\deg(v_j))}$, use type-1 weight sharing scheme in the Table 3.1. b^c denotes the bit width of C2V message, and b^v denotes the bitwidth for V2C message and variable node posterior. l_{v_j} is the posterior of variable node v_j. In the t^{th} iteration, a layer-scheduled W-OMS-RCQ decoder calculates the messages

$u^{(t)}_{c_i \to v_{j'}}$ and updates the posteriors $l_{v_{j'}}$ as follows:

$$\tilde{l}_{v_{j'} \to c_i} \leftarrow l_{v_{j'}} - \text{Relu}\left(R\left(u^{(t-1)}_{c_i \to v_{j'}}\right) - \beta^{(t-1)}_{(\deg(c_i),\deg(v_{j'}))}\right), \quad \forall j' \in \mathcal{N}(c_i), \tag{3.20}$$

$$l^{(t)}_{v_{j'} \to c_i} = Q\left(\tilde{l}^{(t)}_{v_{j'} \to c_i}\right), \quad \forall j' \in \mathcal{N}(c_i), \tag{3.21}$$

$$u^{(t)}_{c_i \to v_{j'}} = \left(\prod_{\tilde{j} \in \mathcal{N}(c_i) \setminus \{j'\}} \text{sgn}\left(l_{v_{\tilde{j}} \to c_i}\right)\right) \times \min_{\tilde{j} \in \mathcal{N}(c_i)/\{j'\}} \left|l_{v_{\tilde{j}} \to c_i}\right|, \quad \forall j' \in \mathcal{N}(c_i), \tag{3.22}$$

$$l_{v_{j'}} \leftarrow \tilde{l}_{v_{j'} \to c_i} + \text{Relu}\left(R\left(u^{(t)}_{c_i \to v_{j'}}\right) - \beta^{(t)}_{(\deg(c_i),\deg(v_{j'}))}\right), \quad \forall j' \in \mathcal{N}(c_i). \tag{3.23}$$

The differences between W-RCQ decoder and RCQ decoder are:

- *Reconstruction and Quantization.* The reconstruction and quantization functions in a layer-scheduled RCQ decoder are dynamic, which means that the decoder updates $R(\cdot)$ and $Q(\cdot)$ in each decoding layer and iteration. Storing the quantizers and dequantizers of all layers and iterations in the local variable node units (VNUs) will cost a large number of LUTs. Hence a central control unit is considered for storing and distributing the parameters to each VNU [TWC21b]. On the other hand, the neural RCQ decoder only uses very few number of $R(\cdot)$ and $Q(\cdot)$ across all decoding iterations, for example, three or less. Besides, as will be seen in the next subsection, we require that the thresholds of quantizers and mapping values of dequantizers have the same values. Each $R(\cdot)$ and $Q(\cdot)$ are used for several iterations. Hence, $R(\cdot)$ and $Q(\cdot)$ are possible to be stored locally in VNUs.

- *Message adjustment.* W-RCQ decoder weights the reconstructed C2V messages with additive or multiplicative parameters, which result in W-OMS-RCQ and W-NMS-RCQ, respectively. As shown in Fig. 2.8b, a central control unit is used for storing and distributing the weights to VNUs.

3.4.2 Non-Uniform Quantizer

An important design choice for a W-RCQ decoder is the selection of quantization and reconstruction (dequantization) function. The authors in [WTS22] use discrete density evolution to design dynamic quantizers and dequantizers. In [ZS14], Zhang *et. al.* point that the message magnitude of iterative LDPC decoders can exhibit exponential behavior as a function of the number of decoding iterations, and the decoding performance of a quantized decoder can be improved by allowing exponential growth magnitude. For example, the authors in [ZS14] propose a $(q+1)$-bit quasi-uniform quantizer that uses one extra bit to efficiently increase the dynamic range of messages. For the W-RCQ decoder, this paper considers the quantizer and dequantizer that can be parameterized by a power function.

Let $Q(x)$ be a symmetric b^c-bit quantizer that features sign information and a magnitude quantizer $Q^*(|x|)$. The magnitude quantizer selects one of 2^{b^c-1} possible indices using the threshold values $\{\tau_0, \tau_1, \ldots, \tau_{\max}\}$, where $\tau_j = C\left(\frac{j}{2^{b^c-1}}\right)^\gamma$ for $j \in \{0, \ldots, 2^{b^c-1}-1\}$ and $\tau_{\max}$ is $\tau_{j_{\max}}$ for $j_{\max} = 2^{b^c-1}-1$. Given an input x, which can be decomposed into sign part $\mathrm{sgn}(x)$ and magnitude part $|x|$, $Q^*(|x|) \in \mathbb{F}_2^{b^c-1}$ is defined by:

$$Q^*(|x|) = \begin{cases} j, & \tau_j \le |x| < \tau_{j+1} \\ 2^{b_c-1}-1, & |x| \ge \tau_{\max} \end{cases}, \tag{3.24}$$

where $0 \le j \le j_{\max}$. Let $s(x)$ be the sign bit of x, which is defined as $s(x) = \mathbb{1}(x < 0)$, $Q(x)$ is defined as $Q(x) = [s(x)\ Q^*(|x|)]$. The set of thresholds of $Q^*(|x|)$ has a power-function form and is controlled by two parameters. The parameter C confines the maximum magnitudes the quantizer can take, and γ manipulates the non-uniformity of the quantizer. Specifically, if $\gamma = 1$, $Q(x)$ becomes a uniform quantizer.

Let $d \in \mathbb{F}_2^{b^c}$ be a b^c-bit message. d can be represented as $[d^{\mathrm{MSB}}\ \tilde{d}]$, where $d^{\mathrm{MSB}} \in \{0,1\}$ indicates sign and $\tilde{d} \in \mathbb{F}_2^{b^c-1}$ corresponds to magnitude. The magnitude reconstruction function $R^*(\tilde{d}) = C\left(\frac{\tilde{d}}{2^{b_c-1}}\right)^\gamma$, and $R(d) = (-2d^{\mathrm{MSB}}+1)R^*(\tilde{d})$. Note that both the magnitude quantization function and magnitude reconstruction function use $\{\tau_1, \ldots, \tau_{\max}\}$ as their pa-

rameters.

The number of required quantizer/dequantizer pairs for W-RCQ decoder can vary under different circumstances. If the code has a small variable node degree and the bit width of the quantizer is not too low (for example, 4 bits), one quantizer/dequantizer pair is sufficient through all decoding iterations. However, if the variable node degree of the LDPC code is high, or the bit width of quantizer is very small, using one quantizer/dequantizer pair is not able to accommodate the range of messages in the decoding process while providing a fine enough resolution, and is likely to degrade decoding performance. Therefore, we consider to use multiple quantizer/dequantizer pairs, and each pair is used for several iterations.

3.4.3 Training Quantized Neural Network

In this paper, we use the multi-loss cross entropy as the loss function and use posterior jointly training to train the QNN that is associated to the W-RCQ decoder. The parameters of the quantizers and dequantizers are fixed before training the neural network. One problem of QNN is that quantization functions results in zeros derivatives almost everywhere. In this work, we use a straight through estimator (STE) [BLC13, XVT20b] in the backward propagation.

3.4.4 Fixed-Point W-RCQ decoder

This paper uses the pair (b^c,b^v) to denote the bitwidth for fixed-point decoders, where b^c is the bitwidth of C2V messages and b^v is the bitwith of V2C messages and the posteriors of variable nodes. For the W-RCQ decoders, the learnable parameters are first trained under a floating point message representation and then quantized to b^v bits.

Table 3.2: LDPC Codes used for Simulation

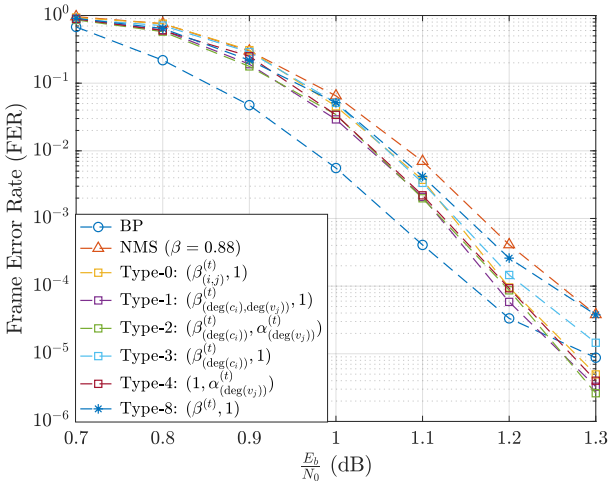
Code	Rate	Edge distribution
(16200,7200) DVBS-2 LDPC code [ETS]	$\frac{4}{9}$	$\lambda(x) = 2.06 * 10^{-5} + 0.3703x +$ $0.3333x^2 + 0.2963x^7$ $\rho(x) = 0.1186x^3 + 0.3332x^4 +$ $0.4445x^5 + 0.1037x^6$
(9472,8192) QC-LDPC code [WTS22]	$\frac{8}{9}$	$\lambda(x) = x^3$ $\rho(x) = 0.3919x^{28} + 0.6081x^{29}$
$k = 1032$ PBRL LDPC code [cls]	$\frac{8}{9}, \frac{8}{10}, \ldots, \frac{8}{24}$	$\lambda(x) = 0.1190 + 0.7940x^4 + 0.0952x^5 +$ $0.0556x^6 + 0.3095x^{12} + 0.1270x^{16} + 0.2143x^{26}$ $\rho(x) = 0.0238x^2 + 0.0635x^3 + 0.0794x^4 +$ $0.1905x^5 + 0.2222x^6 + 0.1270x^7 + 0.1429x^{17} +$ $0.1508x^{18}$

3.5 Simulation Result and Discussion

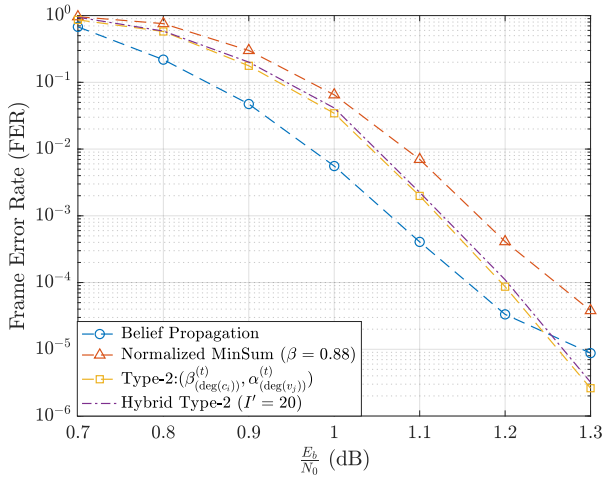
This section evaluates the performance of the N-2D-NMS decoder and the W-RCQ decoder for LDPC codes with different block lengths and code rates. The LDPC codes used in this section are listed in Table 3.2. All the encoded bits are modulated by binary phase-shift keying (BPSK) and transmitted through a Additive White Gaussian Noise (AWGN) channel.

3.5.1 (16200,7200) DVBS-2 LDPC code

Fig. 3.4a shows the FER performances of N-2D-NMS decoders with various weight sharing types for the (16200, 7200) DVBS-2 LDPC code. The FER performance of BP and NMS decoders are also given for comparison. The single multiplicative weight of NMS decoder is 0.88. All of the decoders are flooding-scheduled and maximum decoding iteration is 50. It

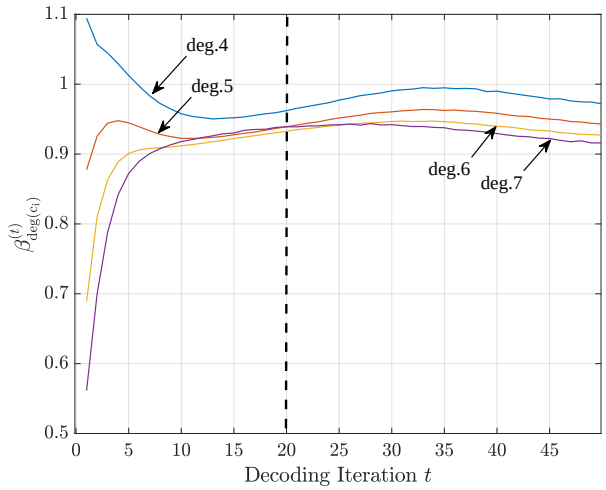


(a)

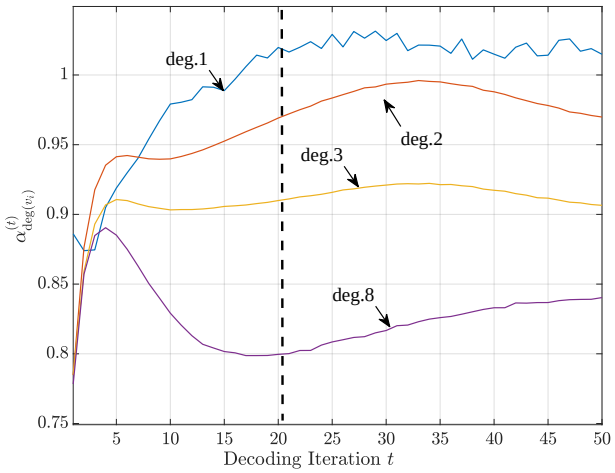


(b)

Figure 3.4: Fig. (a): The FER performance of the N-2D-NMS decoders with various weight sharing types for the (16200,7200) DVBS-2 LDPC code. Fig. (b): The FER performance of the hybrid type-2 N-2D-NMS decoder that uses distinct weights in the first 20 iterations and same weights in the remaining 30 iterations. Simulation result shows that the hybrid type-2 N-2D-NMS decoder has comparable decoding performance with the type-2 N-2D-NMS decoder that assigns distinct weights in each iteration.



(a)



(b)

Figure 3.5: The change of weights of the type-2 N-2D-NMS decoder for (16200, 7200) DVBS-2 LDPC code w.r.t. check node degree, variable node degree and iteration index. Specifically, Fig. (a) gives $\beta^{(t)}_{(\deg(c_i))}$ for all possible check node degrees in each decoding iteration t, Fig. (b) gives $\alpha^{(t)}_{(\deg(v_j))}$ for all possible variable node degrees in each decoding iteration t.

Table 3.3: The Quantizer/Dequantizer pairs of W-OMS-RCQ decoder for (9472,8192) LDPC code

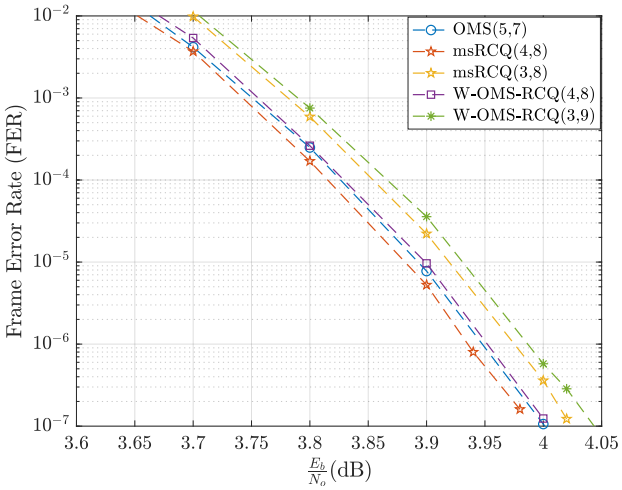
CN bitwidth	Quantizer/Reconstruction Parameter	Corresponding Decoder Iteration
4 bits	$C = 7$, $\gamma = 1.7$	$1 \sim 10$
3 bits	$C_1 = 3$, $\gamma_1 = 1.3$	$1 \sim 6$
	$C_2 = 5$, $\gamma_2 = 1.3$	$7 \sim 8$
	$C_3 = 7$, $\gamma_3 = 1.3$	$9 \sim 10$

is shown that the N-NMS decoder (i.e., type-0 decoder) outperforms BP at 1.3 dB with a lower error floor. The type-1 and 2 decoders, which share weights based on the check node and variable node degree, deliver even a slightly better decoding performance than N-NMS decoder.

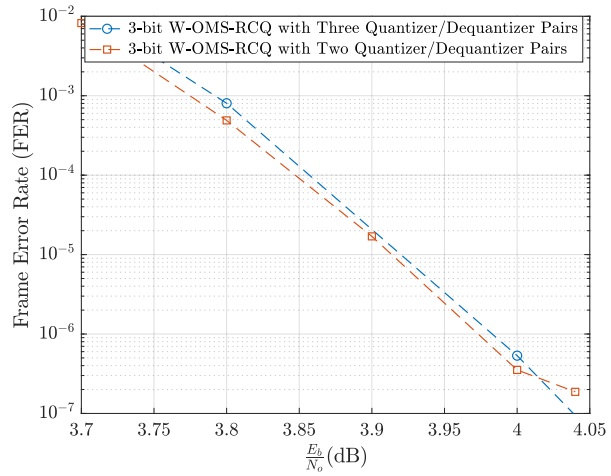
Fig. 3.4a also shows that the FER performance degrades if only considering to sharing weights w.r.t. check node degree (type-3) or variable node degree (type-4). Specifically, in this example, type-4 N-2D-NMS decoder outperforms type-3 N-2D-NMS decoder, because the variable node weights of investigated code have a larger dynamic range than check node weights, as shown in Fig. 3.5a, and 3.5b. Fig. 3.5a and 3.5b give the $\beta^{(t)}_{(\deg(c_i))}$ and $\alpha^{(t)}_{(\deg(v_j))}$ of type-2 N-2D-NMS decoder, which agree with our observation in the previous section; i.e., in each decoding iteration, larger degree node corresponds to a smaller value. Besides, as shown in Fig. 3.5a and 3.5b, the weights change negligibly after 20^{th} iteration. Thus, the hybrid type-2 N-2D-NMS decoder with $I' = 20$ delivers similar performance to the full feed forward decoding structure, as shown in Fig. 3.4b.

3.5.2 (9472,8192) Quasi-Cyclic LDPC code

This subsection designs 3-bit and 4-bit W-OMS-RCQ decoders for a (9742,8192) quasi-cyclic (QC) LDPC code and compares them with the fixed-point OMS decoder and RCQ decoders. All decoders in this subsection are layer-scheduled with maximum iteration 10.



(a)



(b)

Figure 3.6: Fig. (a): FER performance of W-OMS-RCQ decoders, RCQ decoders, and 5-bit OMS decoder for a (9472, 8192) QC LDPC code. Fig. (b): FER performance of 3-bit W-OMS-RCQ decoders with two and three quantizer/dequantizer pairs. Simulation result shows that the W-OMS-RCQ decoder with two quantizer/dequantizer pairs has an error error floor at FER of 10^{-7}.

Table 3.4: Hardware Usage of Various Decoding Structure for (9472,8192) QC-LDPC Code

Decoding Structure	LUTs	Registers	BRAMS	Routed Nets
OMS(5,7) (baseline)	21127	12966	17	29202
RCQ(4,8)	20355(↓ 3.6%)	13967(↑ 7.0%)	17.5(↑ .03%)	28916(↓ 1%)
RCQ(3,8)	17865(↓ 15.4%)	12098(↓ 6.7%)	17(−)	25332(↓ 13.3%)
W-OMS-RCQ(4,8)	17645(↓ 16.5%)	13297(↑ 2.6%)	17(−)	25361(↓ 13.2%)
W-OMS-RCQ(3,8)	16306(↓ 22.82%)	12104(↓ 6.65%)	17(−)	23252(↓ 20.38%)

The quantizer/reconstruction parameters for the 3-bit and 4-bit W-OMS-RCQ decoder are given in Table. 3.3.

Fig. 3.6a compares the FER performances of W-OMS-RCQ decoders with RCQ decoders and a 5-bit OMS decoder. The decoders in Fig. 3.6a are also implemented using an FPGA device (Xilinx Zynq UltraScale+ MPSoC) the study of resource usage. Table 3.4 lists the usage of lookup tables (LUTs), registers, block RAM (BRAM), and routed nets of various decoders. For the details of FPGA implementations of the decoders, we refer the readers to [TWC21b].

The simulation result shows the 4-bit RCQ decoder has the best FER performance. The 4-bit W-OMS-RCQ decoder and 5-bit OMS decoder have similar FER performance, which is inferior to the 4-bit RCQ decoder by 0.01 dB. However, as shown in Table 3.4, the 4-bit W-OMS-RCQ decoder requires much fewer resources than the 4-bit RCQ decoder and the 5-bit OMS decoder. Compared to the 5-bit OMS decoder, the 3-bit W-OMS-RCQ and 3-bit RCQ decoder have a 0.025 and 0.043 dB gap, respectively. Specifically, the 3-bit RCQ decoder has a similar LUT, BRAM, and routed net usage to the 4-bit W-OMS-RCQ decoder. On the other hand, the 3-bit W-OMS-RCQ uses much fewer resources than the 4-bit W-OMS-RCQ decoder.

The 3-bit W-OMS-RCQ decoder in Fig. 3.6a uses three quantizers for three decoding phases. In the first 3 iteration, most messages have low magnitudes, hence a quantizer with

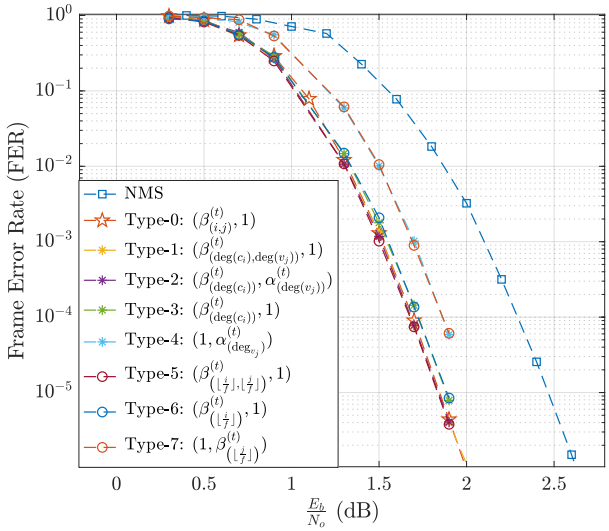


Figure 3.7: FER performance of N-2D-NMS decoders with various weight sharing types for a (3096,1032) PBRL LDPC code compared with N-NMS (type 0) and NMS.

small C is required for a finer resolution to the low-magnitude values. However, the message magnitudes increase with the increase of decoding iteration. As a result, the quantizers with larger C should be used correspondingly. Fewer quantizers may not accommodate the message magnitude growth in the decoding process and will result in performance degradation. For example, Fig. 3.6b considers a 3-bit W-OMS-RCQ decoder that uses two quantizer/dequantizer pairs, the first pair has $C_1 = 3$, $\gamma_1 = 1.3$ and is used for iteration $1 \sim 7$, the second pair has $C_2 = 5$, $\gamma_2 = 1.3$ and is used for iteration $8 \sim 10$. Simulation result shows that the 3-bit W-OMS-RCQ decoder that uses 2 quantizer/dequantizer pairs has an early error floor at FER of 10^{-7}.

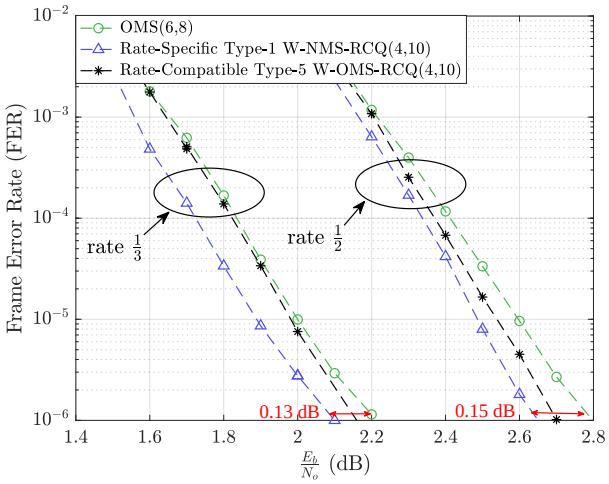
3.5.3 $k = 1032$ Protograph-Based Raptor-Like code

5G LDPC codes have the protograph-based raptor-like (PBRL) [CVD15] structure which offers inherent rate-compatibility and excellent decoding performance. In this subsection, we examine the performance of N-2D-NMS decoders and W-RCQ decoders for a $k = 1032$

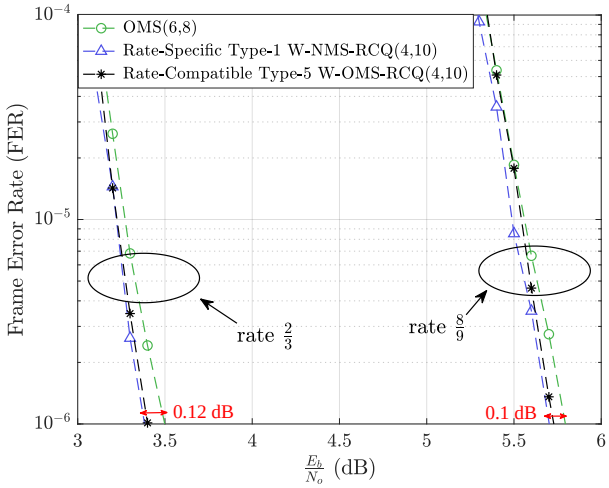
PBRL LDPC code, whose supported rates are listed in Table 3.1. The edge distribution of the lowest-rate code, which corresponds to the full parity check matrix, is also given in Table 3.1. All the decoders in this subsection are layer-scheduled with maximum 10 decoding iterations.

Fig. 3.7 shows the FER performance of N-2D-NMS decoder with various weight sharing types for the PBRL code with lowest code rates $\frac{1}{3}$. As a comparison, the decoding performance of the N-NMS (type 0) decoder and the NMS decoder are also given. All of the decoders use floating-point message representation. The simulation results show that N-NMS decoder has a more than 0.5 dB improvement over the NMS decoder. N-2D-NMS decoders with weight sharing types 1-7 are also simulated. Simulation result shows that the N-2D-NMS decoders with weight-sharing metrics based on check and variable node degree (i.e., type 1 and 2), or based on horizontal and vertical layer (i.e., type 5) deliver lossless performance w.r.t. N-NMS decoder. N-2D-NMS decoders with weight sharing types 4 and 6 have a degradation of around 0.05 dB compared with the N-NMS decoder. N-2D-NMS decoders with weight sharing types 5 and 7 have a degradation of around 0.2 dB compared with the N-NMS decoder. Thus, for this (3096,1032) PBRL LDPC code of Fig. 3.7, assigning weights based only on check nodes can gain more benefit than assigning weights only based on variable nodes.

Fig. 3.8 gives the FER performance of fixed-point W-NMS-RCQ decoders for the $k = 1032$ PBRL code with rate $\frac{1}{3}$, $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{8}{9}$. The W-NMS-RCQ decoder assigns 4 bits to C2V message and 10 bits to V2C message. Two quantizer/dequantizer pairs are used for W-NMS-RCQ decoder across all investigated rates. The first quantizer has $C_1 = 7$, $\gamma_2 = 1.7$ and is used for the first 7 iterations. The second quantizer has $C_2 = 10$, $\gamma_2 = 2.3$ and is used for last 3 iterations. We use a 6-bit OMS decoder as benchmark, because we find it delivers a better decoding performance than the NMS decoder with same bit width. Additionally, we didn't consider W-OMS-RCQ decoder for this code because the 4-bit W-OMS-RCQ decoder doesn't perform as well as the 4-bit W-NMS-RCQ decoder.



(a)



(b)

Figure 3.8: FER performance of 4-bit W-RCQ decoders for $k = 1032$ PBRL code with different code rates. The term "rate-specific" means to design distinct decoders for each code rate; The term "rate-compatible" means to train one decoder that matches all code rates. The 6-bit OMS decoder is given as comparison.

We first consider the 4-bit W-NMS-RCQ decoder with type-1 weight sharing that assigns the same weight to the edges with same check and variable node degree. The decoder is rate-specific; i.e., for each considered rate, a W-NMS-RCQ decoder is trained separately. The simulation results shows that, targeting a FER of 10^{-6}, the 4-bit rate-specific W-NMS-RCQ decoder outperforms the 6-bit OMS decoder with $0.1\sim 0.15$ dB for all considered code rates.

For the PBRL code, the proto-matrix of each possible rate is a sub-matrix of a base proto-matrix [CVD15]. As shown in Table. 3.1, the type-5 weight sharing assigns the same weight to the edges that correspond to the same element in the proto-matirx. Hence, it is possible to use *one* trained type-5 neural decoder to match different code rates. We refer to such decoder as a rate-compatible decoder. In [DTS21], the authors propose to training the rate-compatible decoder by using the samples from different code rates.

Fig. 3.8 shows the decoding performance of rate-compatible type-5 W-NMS-RCQ decoder. Simulation result shows that for the higher rate such as $\frac{2}{3}$ and $\frac{8}{9}$, the rate-compatible type-5 W-NMS-RCQ decoder has a similar decoding performance to the rate-specific type-1 W-NMS-RCQ decoder whose parameters for each rate are separately designed. However, for the lower rate such as $\frac{1}{3}$ and $\frac{1}{2}$, the rate-compatible type-5 W-NMS-RCQ decoder method doesn't deliver a decoding performance as well as rate-specific type-1 W-NMS-RCQ decoder. Besides, considering the four rates in Fig. 3.8, the number of neural weights for rate-specific type-1 and rate-compatible type-5 W-NMS-RCQ decoder are 96 and 101, respectively.

3.6 Conclusion

This chapter proposes the W-RCQ decoder, a non-uniformly quantized decoder that delivers excellent decoding performance in the low-bitwidth regime. Unlike RCQ decoder, which designs quantizer/dequantizer pairs for each layer and iteration, W-RCQ decoder only uses a small number of quantizer/dequantizer pairs and each one is responsible for several iterations. The W-RCQ decoder uses Min operation at check node, and the C2V messages are weighted

by multiplicative or additive parameters, which induce W-NMS-RCQ and W-OMS-RCQ, respectively.

For the neural decoders such as W-RCQ decoder and N-NMS decoder, assigning distinct weights to each edge in each decoding iteration is impractical for long-blocklength codes because of huge number of neural weights. This paper proposes various node-degree-based weight sharing schemes with lossless or lossy performance for the neural decoders, depending on whether the weight sharing considers both check and variable node degree or only one of them.

Additionally, this paper discusses the issues when training neural LDPC decoders. First, training the neural LDPC decoders for long blocklength code using Pytorch or TensorFlow could raise memory issue. This paper shows that the memory for training neural MinSum decoders can be saved by storing feed-forward messages compactly. Second, this paper identifies gradient explosion problem in the neural decoder training and proposes a posterior jointly training method that addresses this problem.

CHAPTER 4

Probabilistic Shaping for Trellis-Coded Modulation with CRC-Aided List Decoding

4.1 Introduction

This paper explores reliable communications over the additive white Gaussian noise (AWGN) channel with high spectral efficiency for short block lengths. To closely approach theoretical limits, it is helpful to use shaping so that signal points are not equally likely, not equally spaced, or both [Gal68, FGL84, For92, KP93, LFT94, FWS01, XWS21]. Recently, a new technique called probabilistic amplitude shaping (PAS) [BSS15, BSS19] employs a distribution matcher (DM) [SB15] before the forward error correction (FEC) encoder and channel-signaling mapping function to accomplish optimal or almost optimal shaping.

A PAS system as in [BSS15, BSS19] decomposes a channel input sequence into a magnitude symbol sequence and a sign sequence. The magnitude symbol sequence is generated by a DM. The output of the DM is provided as input to a systematic FEC code where the parity check bits indicate the signs of the channel inputs. A channel-signaling mapping function maps the amplitude symbol sequence and the sign-bit sequence to the corresponding sequence of transmitted signal points.

A distribution matcher [SB15, AB13, SS19, Sch20, PX19a, FMK18] maps a binary input sequence onto a symbol sequence that determines the magnitudes of the transmitted symbols. The binary input sequence typically has equally likely ones and zeros. However, the output symbols from the distribution matcher are not equally likely. Specifically, the distribution

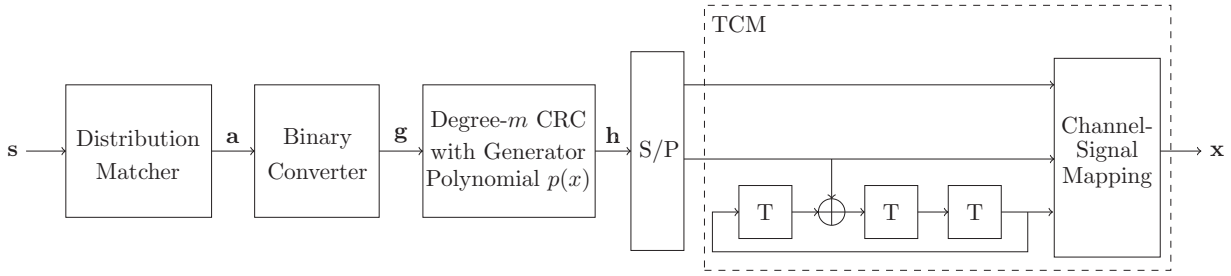


Figure 4.1: Diagram of the CRC-TCM-PAS transmitter. In the diagram, $\mathbf{s} \in \mathbb{F}_2^k$, $\mathbf{a} \in \mathcal{C}_{\mathrm{DM}} \subseteq \mathcal{A}^l$, $\mathbf{g} \in \mathbb{F}_2^{k_0 l}$, $\mathbf{h} \in \mathbb{F}_2^{k_0 l+m}$, $\mathbf{x} \in \mathcal{X}^n$, and $n = l + \frac{m}{k_0}$. The transmission rate of the system is $\frac{k}{n}$ bits/real channel use. The TCM in this figure uses a rate-$\frac{2}{3}$ TBCC.

matcher is designed such that the PAS system can generate channel inputs with a capacity-approaching probability mass function (PMF).

Even though it is well-known that a continuous Gaussian probability density function (PDF) is a capacity-achieving distribution for the power-constrained additive Gaussian white noise (AWGN) channel, a carefully designed finite-cardinality PMF can deliver performance that is almost indistinguishable from that of a Gaussian PDF and facilitates practical implementation. In [KP93], Kschischang *et. al.* use Maxwell-Boltzmann distribution to optimize the PMF of equally-spaced pulse-amplitude modulation (PAM) or quadrature amplitude modulation (QAM) constellations. Xiao *et. al.* use dynamic Blahut-Arimoto (DAB) to identify minimum-cardinality capacity-approaching input PMFs for PAM constellations [XWS21].

The empirical distribution of the output symbols of a good distribution matcher will closely resemble the target PMF as determined, for example, according to [XWS21]. The shell-mapping (SM) DM [AB13, SS19] is optimal in termes of normalized Kullback-Leibler (KL) divergence. Schulte *et al.* in [SB15] propose an asymptotically optimal distribution matcher, the constant composition (CC) DM. One advantage of CCDM is that it supports online encoding. Some other distribution matchers include those of [Sch20, PX19a, FMK18].

As in [BSS15, BSS19], the output of the distribution matcher is provided as input to a

systematic FEC code where the parity check bits indicate the signs of the channel inputs. The channel-signaling mapping function maps the amplitude-bit sequence of the DM and the sign-bit sequence of the FEC to the corresponding sequence of transmitted signal points.

The second module in the PAS transmitter is a coded modulator. Coded modulation is a combination of error correction code and modulation. A well know coded modulation is Underboeck's trellis-coded modulation (TCM) [Ung82]. The coded modulation in the PAS transmitter [Boc17] comprises a systematic error correction code and a channel-signal mapping function.

An important design choice for a PAS system is the selection of an FEC code. In the long blocklength regime, Böcherer *et. al.* in [BSS15] use low-density parity-check (LDPC) codes for the PAS system. In the short blocklength regime, Coşkun *et. al.* in [CDJ19] investigate PAS systems with various FEC choices including binary LDPC codes, non-binary LDPC codes and polar codes.

Recently, convolutional codes with cyclic redundancy code (CRC)-aided list decoding have shown excellent performance in the short blocklength regime [LYD19, YRW18, YLY19, YLP22]. Yang *et al.* in [LYD19] show that a tail-biting convolutional code (TBCC) with CRC-aided list decoding can achieve frame error rate (FER) performance very close to the short-blocklength random coding union (RCU) bound [PPV10]. King. *et al.* in [KKY22] provide an example where a TBCC outperforms a polar code in the AWGN channel when both are decoded using CRC-aided list decoding.

4.1.1 Contributions

In this paper, we propose a PAS system designed for the AWGN channel in the short-blocklength regime. The proposed PAS system uses a CRC-aided, rate-$\frac{k_0}{k_0+1}$, systematic, recursive TBCC as the FEC code. The TBCC and the channel-signal mapping function constitute the TCM [Ung82]. We refer to the proposed PAS system as CRC-TCM-PAS. Fig.

4.1 describes the transmitter of the CRC-TCM-PAS system. A CRC-TCM-PAS system can be designed as follows:

1. Using [KP93] or [XWS21], identify the capacity-approaching PMF for the PAM constellation under AWGN, which induces the PMF for the corresponding magnitudes.

2. Assuming an ideal distribution matcher that generates magnitude sequences whose symbols are independent and identically distributed (i.i.d.) according to the distribution calculated in 1), optimize the CRC and TBCC using the FER upper bound developed in Section 4.4.

3. Replace the ideal distribution matcher with a practical one.

The contributions of this chapter are summarized as follows:

- *CRC-TCM-PAS transmission system.* This chpter presents the paradigm of the CRC-TCM-PAS system.

- *Multi-composition distribution matcher (MCDM).* MCDM, which can be seen as a collection of CCDMs, is proposed in this chapter. We note that the proposed distribution matcher is a generalization of the MCDM in [PX19b], which limits the cardinality of the output alphabet to 2. We investigate two rules to select the CCDMs, which are related to high-probability sets and typical sets in information theory.

- *CRC-TCM-PAS Decoder.* We propose automorphism enabled decoding [GEE22] to achieve near-maximum-likelihoood performance with low time complexity.

- *Properties of CRC-TCM-PAS transmission system.* This chapter proves that, asymptotically, the sign values produced by the TCM are equally likely to be positive or negative. This yields channel input symbols that have a symmetric capacity-approaching distribution.

- *Optimization of CRC-TCM-PAS parameters.* This chapter derives an upper bound on the FER of CRC-TCM-PAS systems and uses this bound as an objective function to jointly optimize the CRC and TBCC. The optimized CRC-TCM-PAS systems achieve FERs below the random coding union (RCU) bound in AWGN and outperform the short-blocklength PAS systems with various other forward error correction codes studied in [CDJ19]. Simulation results show that the optimized CRC-TCM-PAS systems can exceed RCU bound and outperforms PAS systems with various other FEC codes explored in [CDJ19].

4.1.2 Organization

The remainder of this chapter is organized as follows: Section **??** reviews CCDM and presents MCDM. The performance of these distribution matchers are compared in this section. Section 4.2 presents CRC-TCM-PAS system architecture. Section 4.3 proves the symmetric capacity-approaching distribution of the output of the CRC-TCM-PAS system. Section 4.4 derives the FER upper bound, and Section 4.5 presents the simulation results of CRC-TCM-PAS systems with different input lengths and transmission rates. Section 4.6 concludes our work.

4.1.3 Notation

In this chapter, we use the italic upper case letter A to denote a random variable. We use $A^l = [A_1, \ldots, A_l]$ to denote a random vector. We use the italic lowercase letter a to denote a realization of A or a variable. We use the straight bold lowercase letter $\mathbf{a}$ to denote either a realization of A^l or a column vector. Specifically, $[\mathbf{a}]_m$ is a vector that contains last m elements in $\mathbf{a}$. Finally, we use the straight, bold upper case letter $\mathbf{A}$ to denote a matrix.

4.1.4 Preliminaries

Let S be a random variable that obeys Bernoulli $\left(\frac{1}{2}\right)$, and Denote the random sequence of S with length k by S^k, and the random sequence of A with length l by A^l. Specifically, $S^k = [S_1, \ldots, S_k]$ and $A^l = [A_1, \ \ldots, A_l]$. Let A be a random variable with alphabet $\mathcal{A} = \{0, 1, \ldots, |\mathcal{A}| - 1\}$.

A fixed-to-fixed distribution matcher is an injective function f_{DM} that maps a binary length-k source sequence $\mathbf{s} \in \mathbb{F}_2^k$ to a length-l symbol sequence $\mathbf{a} \in \mathcal{A}^l$, i.e.,

$$f_{DM} : \{0, 1\}^k \to \mathcal{A}^l. \tag{4.1}$$

$\mathcal{A} = \{0, 1, \ldots, |\mathcal{A}| - 1\}$ is the output symbol set. In this chapter, we limit $\log_2 |\mathcal{A}| = k$ to be some integer. The range of f_{DM} is the codebook of the distribution matcher, which is denoted by $\mathcal{C}_{\mathrm{DM}}$. Note that $\mathcal{C}_{\mathrm{DM}} \subseteq \mathcal{A}^l$. Because f_{DM} is an one-to-one mapping, it has $|\mathcal{C}_{\mathrm{DM}}| = 2^k$. Additionally, because the input bits of the DM are equally likely, it has $P_{A^l}(\mathbf{a}) = 2^{-k}$, for $\mathbf{a} \in \mathcal{C}_{\mathrm{DM}}$. Let $P(\bar{A})$ be the empirical distribution of a DM with codebook $\mathcal{C}_{\mathrm{DM}}$. $P(\bar{A})$ is calculated as follows:

$$P_{\bar{A}}(i) = \frac{1}{2^k l} \sum_{\mathbf{a} \in \mathcal{C}_{\mathrm{DM}}} \sum_{j=0}^{l-1} \mathbb{1}(a_j = a), \ a = 0, ..., |\mathcal{A}| - 1. \tag{4.2}$$

A good distribution matcher has a $P(\bar{A})$ that is close to the desired distribution, $P(\hat{A})$.

The quality of a DM can be measured as its KL divergence with a *theoretically* optimal DM, which is referred to as a random DM. The random DM uses the construction method of Shannon's random code [TJ06]. Given the desired probability $P(\hat{A})$, in each transmission, the random DM randomly generates a codebook that contains 2^k codewords of length l according to the distribution $P_{\hat{A}^l}(\mathbf{a}) = \prod_{i=1}^{l} P_{\hat{A}}(a_i)$. The KL divergence between a practical DM with $\mathcal{C}_{\mathrm{DM}}$ and a random DM is calculated by [SB15]:

$$D_{\mathrm{KL}}\left(P(A^l) || P(\hat{A}^l)\right) = \frac{1}{2^k} \log_2 \left(\sum_{\mathbf{a} \in \mathcal{C}_{\mathrm{DM}}} \frac{1}{P_{\hat{A}^l}(\mathbf{a})} \right) - k, \tag{4.3}$$

In this paper, we follow the convention in [SB15], and use the normalized KL divergence, $\frac{1}{l}D_{\mathrm{KL}}\left(P(A^l)||P(\hat{A}^l)\right)$, as the metric to evaluate the distribution matcher.

A DM with a small normalized KL divergence is desired. One well-known DM with simple encoding and decoding algorithm is CCDM, whose codebook, $\mathcal{C}_{\mathrm{CCDM}}$, contains the sequences that have the same *type*, which is defined as follows [TJ06, Chapter 11]:

Definition 1. *The type (or empirical distribution) $P_{\mathbf{a}}$ of a sequence $\mathbf{a} = [a_0, a_1, \ldots, a_{l-1}]$ is the relative proportion of occurrence of each symbol in $\mathcal{A}$, i.e., $P_{\mathbf{a}}(i) = \frac{\sum_{j=0}^{l-1} \mathbb{1}(a_j=i)}{l}$, $i \in \mathcal{A}$. Define the set of sequences of length l and type P as set class of P, denoted by $\mathcal{T}_P^l$:*

$$\mathcal{T}_P^l = \{\mathbf{a} \in \mathcal{A}^l : P_{\mathbf{a}} = P\}. \tag{4.4}$$

Based on Definition 1, the codebook of CCDM is a subset of a set class of some type P. The type P is chosen such that $2^k \leq |\mathcal{T}_P^l|$, and normalized KL divergence is minimized in the meanwhile. Because all codewords in $\mathcal{C}_{\mathrm{CCDM}}$ have the same type P, the empirical distribution of CCDM $P(\bar{A}) = P$. There are two major advantages to CCDM. First, the CCDM is asymptotically optimal, i.e., $\lim_{l\to\infty} \frac{1}{l}D_{\mathrm{KL}}\left(P(A^l)||P(\hat{A}^l)\right) = 0$. Second, a CCDM can use arithmetic coding to sequentially generate the codewords in $\mathcal{C}_{\mathrm{CCDM}}$ [SB15]. However, the normalized KL-divergence of CCDM is large in the short-blocklength regime [SB15].

4.1.5 Multi-Composition Distribution Matcher

In this section, we propose a multi-composition distribution matcher (MCDM) that delivers a small normalized KL divergence in the short blocklength regime. The MCDM codebook can been seen as a union of multiple CCDM codebooks. The codebook of an MCDM, $\mathcal{C}_{\mathrm{MCDM}}$, has the following properties:

1. $\mathcal{C}_{\mathrm{MCDM}}$ is a union of τ disjoint children codebooks, i.e., $\mathcal{C}_{\mathrm{MCDM}} = \bigcup_{i=1}^{\tau} \mathcal{C}_i$, and $\mathcal{C}_i \bigcap \mathcal{C}_j = \emptyset$, for $i \neq j$.

2. The codewords in a child codebook have the same type, i.e., $\mathcal{C}_i \subseteq \mathcal{T}^l_{P_{A_i}}$, $i = 1, 2, \ldots, \tau$. No two different children codebooks share the same type.

3. Let $k_i = \left\lfloor \log_2(\mathcal{T}_{P_{A_i}}) \right\rfloor$ for $i = 1, \ldots, \tau$, then $|\mathcal{C}_i| = 2^{k_i}$, for $i = 1, 2, \ldots, \tau - 1$, and $|\mathcal{C}_\tau| = 2^k - \sum_{i=1}^{\tau-1} 2^{k_i}$.

Hence, the MCDM encoding consists of two steps: choose $\mathcal{C}_i$ and perform arithmetic encoding with type P_{A_i}. Let b_i be the cardinality of the union of the first i codebooks, i.e., $b_i = \sum_{m=1}^{i} |\mathcal{C}_i|$, where $i = 1, \ldots, \tau$. Specifically, define $b_0 = 0$. Given a binary input $\mathbf{s}$, the encoding algorithm for MCDM is summarized as follows. First, choose the child CCDM $\mathcal{C}_i$ associated with input sequence $\mathbf{s}$, i is selected such that $b_{i-1} \leq s < b_i$, where s is the decimal representation of $\mathbf{s}$. Second, Calculate the child CCDM input as $\mathbf{c} = [\mathbf{s} - \mathbf{b}_{i-1}]_{k_i}$, where $\mathbf{b}_i$ to denote the binary representation of b_i and the operator $[\cdot]_{k_i}$ returns last k_i bits. Finally, Perform CCDM encoding with the child CCDM $\mathcal{C}_i$ using input $\mathbf{c}$, and generate the output sequence.

The MCDM decoding process is as follows: For any $\mathbf{a} \in \mathcal{A}^l$, the decoder first checks whether the type of $\mathbf{a}$ is one of the types in $\mathcal{C}_{\text{MCDM}}$. If so, the decoder checks whether $\mathbf{a}$ is in $\mathcal{C}_{\text{MCDM}}$. Otherwise, the decoder declares that $\mathbf{a} \notin \mathcal{C}_{\text{MCDM}}$.

An important design question regarding MCDM is the selection of children codebooks $\mathcal{C}_i$, $i = 1, ..., \tau$. Given a target distribution $P(\hat{A})$, we investigate two rules for choosing $\mathcal{C}_i$, namely, high-probability rule and typical-set rule:

Rule 1: High-probability Rule:

$$P(A_i) = \underset{P(A^*) \in \mathcal{P} \setminus \{P(A_1), .., P(A_{i-1})\}}{\operatorname{argmax}} \sum_{a=1}^{|\mathcal{A}|} P_{A^*}(a) \log P_{\hat{A}}(a). \tag{4.5}$$

Rule 2: Typical-set Rule:

$$P(A_i) = \underset{P(A^*) \in \mathcal{P} \setminus \{P(A_1), .., P(A_{i-1})\}}{\operatorname{argmin}} D_{\text{KL}}(P(A^*) || P(\hat{A})). \tag{4.6}$$

Table 4.1: Comparion of various DMs targeting for distribution $P(\hat{A}) = (0.072, 0.165, 0.321, 0.442)$.All DMs have 96 input bits and 63 output symbols.

	ESS	MCDM with $\mathcal{C}_{\text{HP}}$	MCDM with $\mathcal{C}_{\text{TS}}$	CCDM
normalized KL divergence	0.074	0.077	0.096	0.213
required storage (bits)	3.6e5	3e5	3e4	24

$\mathcal{P}$ is the set of all possible types of length-l symbol sequences. Rule 1 chooses the types whose sequences occur with the highest probability according to $P(\hat{A})$. On the other hand, rule 2 chooses the types that are most similar to $P(\hat{A})$. The codebooks built using rules 1 and 2 are related to the concept of high-probability set and typical set in information theory [TJ06, Chapter 3.3], respectively. We use $\mathcal{C}_{\text{HP}}$ and $\mathcal{C}_{\text{TS}}$ to denote the codebooks built using high-probability and typical-set rules, respectively.

4.1.6 Comparison

In this subsection, we compare the performance of various distribution matchers in terms of the normalized KL divergence and required memory. We design the distribution matcher with 96 input bits and 63 output symbols from an 4-ary alphabet. The target distribution is $P(\hat{A}) = (0.072, 0.165, 0.321, 0.442)$.

Additional to the MCDM and CCDM, we also consider a DM called enumerative sphere shaping (ESS) [AGG19]. ESS has an excellent performance in the short block length regime. Given a symbol sequence $\mathbf{a} = [a_1 \dots a_l]$, the energy of $\mathbf{a}$ is defined as $\sum_{i=1}^{l} a_i^2$. ESS considers the sequences whose energies are less than or equal to a threshold $E_{\max}$ as codeword candidates of the distribution matcher. Given an $E_{\max}$, ESS indexes the qualified sequences lexicographically, and an energy-bounded trellis is built to index the sequences.

Table 4.1 gives the normalized KL divergence of CCDM, MCDM, and ESS. CCDM delivers the largest normalized KL divergence, while ESS delivers the smallest normalized KL divergence. The MCDM with $\mathcal{C}_{\text{HP}}$ delivers a comparable normalized KL divergence with

ESS, and the MCDM with $\mathcal{C}_{\text{TS}}$ is slightly larger than that of MCDM with $\mathcal{C}_{\text{HP}}$.

We also compare the required memories for these four DMs. For the CCDM, it suffices to only store the type of codewords. For the ESS, the node values in the trellises are needed [AGG19]. The MCDM needs to store all of the types of children CCDMs and the binary thresholds $\mathbf{b}$. As shown in Table 4.1, CCDM only needs 24 bits for storing the codeword type. The MCDM with $\mathcal{C}_{\text{HP}}$ requires a little bit less memory than ESS. The memory for the MCDM with $\mathcal{C}_{\text{TS}}$ is an order of magnitude smaller than the memory for the MCDM with $\mathcal{C}_{\text{HP}}$, because it uses fewer children CCDMs. In this example, $\mathcal{C}_{\text{HP}}$ requires 2535 children CCDMs and $\mathcal{C}_{\text{TS}}$ requires 327.

4.2 CRC-TCM-PAS System

This section presents the transmitter structure and the decoding algorithms for the proposed CRC-TCM-PAS transmission system.

4.2.1 CRC-TCM-PAS Transmission System Structure

Fig. 4.1 illustrates the diagram of the proposed CRC-TCM-PAS transmitter. As shown in Fig. 4.1, CRC-TCM-PAS takes k bits and generates n real-valued channel input symbols. Hence, the transmission rate is k/n bits per channel use. The CRC-TCM-PAS system consists of three encoding procedures. First, a length-k binary source sequence $\mathbf{s} \in \mathbb{F}_2^k$ is encoded to a length-l symbol sequence $\mathbf{a} \in \mathcal{C}_{\text{DM}}$ by a distribution matcher. Then, the binary representation $\mathbf{g}$ of $\mathbf{a}$ with k_0 bits per symbol, $\mathbf{g} \in \mathbb{F}_2^{k_0 l}$, is encoded by a systematic m-bit CRC with generator polynomial $p(x)$. The proposed system implicitly requires that k_0 divides m. Finally, the TCM module encodes the CRC output and maps the encoded bits to a length-n channel input sequence $\mathbf{x} \in \mathcal{X}^n$, where $\mathcal{X}$ denotes the AM constellation set and $n = l + \frac{m}{k_0}$. The TCM module includes a systematic, rate-$\frac{k_0}{k_0+1}$ TBCC, and a channel-signal mapping function which maps each $k_0 + 1$ encoded bits onto one of 2^{k_0+1} symbols in the AM

constellation set $\mathcal{X}$.

The transmission rate of the CRC-TCM-PAS system is $\frac{k}{n}$ bits/real channel use. The remainder of this subsection introduces TBCC and the channel-signal mapping function for TCM.

4.2.1.1 Tail Biting Convolutional Code

A convolutional code with ν memory elements that takes a k_0-bit input symbol and generates a γ_0-bit output symbol in one stage is denoted by an (γ_0, k_0, ν) convolutional code. We refer to each input symbol as a *data frame*, and each output symbol as a *code frame*. This paper is focused on $(k_0 + 1, k_0, \nu)$ convolutional code. The convolutional code in Fig. 4.1 has $k_0 = 2$. Let $\mathcal{U} = \{0, 1, \ldots, 2^{k_0} - 1\}$ be the set of input symbols and $\mathcal{O} = \{0, 1, \ldots, 2^{\gamma_0} - 1\}$ be the set of output symbols. Denote the input symbol and output symbol in stage t by u_t and o_t, respectively. A convolutional code with n data frames can be described as an n-stage trellis. Denote the set of vertices (or states) at time instant t by $\mathcal{V}_t$. Let v_t be the state at time t. Denote an edge that starts with v_t, ends at v_{t+1} and has an output o_t by a 3-tuple (v_t, o_t, v_{t+1}). Let $\mathcal{E}_t$ be the set of edges in stage t. In this paper, we let $\mathcal{V}_t = \mathcal{V} = \{0, 1, \ldots, 2^{\nu} - 1\}$, and $\mathcal{E}_t = \mathcal{E}$. Let the sequence ($v_0$, o_0, v_1, o_1, $\ldots$, o_{n-1}, v_n) be a valid path in the trellis, a tail-biting path requires $v_0 = v_n$. Denote the TBCC trellis by $\mathcal{T}$, and denote the TBCC sub-trellises whose starting and ending state are i, $i \in \mathcal{V}$, by $\mathcal{T}_i$. This chapter considers recursive, rate-$\frac{k_0}{k_0+1}$, systematic TBCCs.

4.2.1.2 Mapping Rule

In order to maximize free Euclidean distance (ED) of TCM, Ungerboeck in [Ung82] proposed a mapping rule called "mapping by set partitioning". Ungerboeck's set partitioning mapping rule follows from the successive partitioning of a channel-signal set into subsets with increasing minimum distance between the signals in these subsets. With set partitioning,

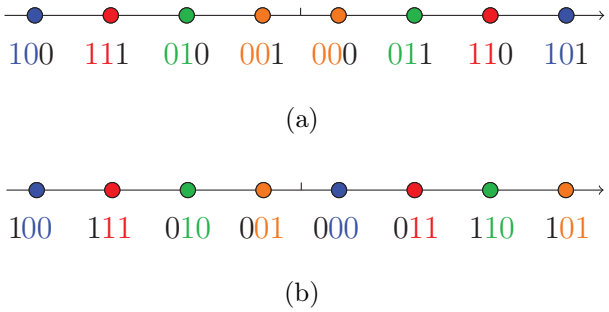


Figure 4.2: Labeling of 8-AM channel signals from (a) magnitude perspective and (b) coset perspective. The least significant two bits identify the coset. The most significant two bits indicate the magnitude. The exclusive-or of all three bits indicates the sign.

the coded bits serve as coset labels so that "uncoded errors" are guaranteed to have at least minimum distance between elements in the same coset.

Our design has an additional requirement that the systematic bits identify the magnitude of the symbol as produced by the distribution matcher. Fig. 4.2 gives binary labels for the equidistant 8-AM constellation set using a labeling that achieves both of these objectives. In this labeling, the sign is negative when the exclusive-or of all three bits is one. The two most significant bits are the systematic bits that identify the magnitude, and one may view the least significant bit as selecting the sign. The two least significant bits identify the coset, and one may view the most significant bit as selecting the sign.

4.2.2 Decoding Algorithms

The channel observation at the receiver over an AWGN channel is $\mathbf{y} = \mathbf{x} + \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ is the noise vector and σ^2 is the noise variance. This subsection introduces various decoding algorithms with varied complexity and error correction performance. We first give the definition of the codeword of a CRC-TCM-PAS system:

Definition 2. $\mathbf{x} \in \mathcal{X}^n$ *is a CRC-TCM-PAS codeword if it satisfies all of the following conditions:*

1. $\mathbf{x}$ *is a codeword of TCM.*

2. *The dataword of TCM that generates* $\mathbf{x}$*,* $\mathbf{h}$*, passes the CRC check.*

3. *The information bits* $\mathbf{g}$ *of the CRC codeword* $\mathbf{h}$*, are the binary representation of a codeword in* $\mathcal{C}_{\mathrm{DM}}$*.*

Denote the codebook of CRC-TCM-PAS by $\mathcal{C}_{\mathrm{CTP}}$, which has cardinality $|\mathcal{C}_{\mathrm{CTP}}| = 2^k$.

4.2.2.1 Maximum Likelihood (ML) Decoder

For AWGN, the ML decoder finds $\hat{\mathbf{x}} \in \mathcal{C}_{\mathrm{CTP}}$ that has smallest Euclidean distance with $\mathbf{y}$, i.e.:

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \in \mathcal{C}_{\mathrm{CTP}}}{\operatorname{argmin}} ||\mathbf{x} - \mathbf{y}||_2^2. \tag{4.7}$$

The ML decoder minimizes the FER, i.e., the probability of a codeword error, in AWGN. The decoding rule of (4.7) can be realized by serial list Viterbi decoding (SLVD) [SS94]. SLVD first finds the most likely path in tail-biting trellis $\mathcal{T}$. If the constellation point sequence corresponding to this path is not a codeword in $\mathcal{C}_{CTP}$, then SLVD is used again to find the next most likely path. If a path belongs to the sub-trellis $\mathcal{T}_i$, the trellis-tree algorithm (TTA) [SH90] for $\mathcal{T}_i$ is used for tracing back that path.

The ML decoding complexity can be decomposed into two parts. First, the initialization step calculates the metrics of local best paths in each of 2^ν sub-trellises. Second, if a path in $\mathcal{T}_i$ needs to be traced back, a data set of TTA for $\mathcal{T}_i$ needs to be constructed and maintained [SH90].

4.2.2.2 β-States Decoder

One solution to reduce the complexity of ML decoder is to consider only a subset of 2^ν states as the possible start/end states. We denote the subset by $\tilde{\mathcal{V}} \subseteq \mathcal{V}$ and the cardinality of $\tilde{\mathcal{V}}$

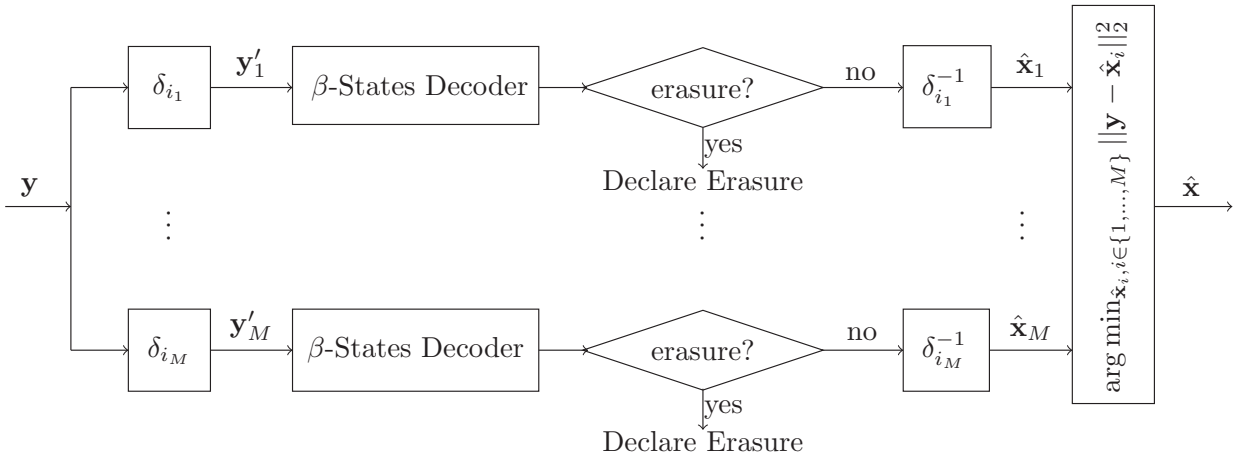


Figure 4.3: The diagram of an AE decoder with M parallel β-States decoders, i.e., AED(M,β).

by $|\tilde{\mathcal{V}}|$. In this paper, we refer to a β-States decoder as a decoder that considers β states as start/end states, i.e., $|\tilde{\mathcal{V}}| = \beta$. Let $v(\mathbf{x})$ be the TBCC initial state of the codeword $\mathbf{x}$. The β-States decoder solves the following problem:

$$\hat{\mathbf{x}} = \underset{\substack{\mathbf{x}\in\mathcal{C}_{\text{CTP}} \\ v(\mathbf{x})\in\tilde{\mathcal{V}}}}{\operatorname{argmin}} ||\mathbf{x}-\mathbf{y}||_2^2. \tag{4.8}$$

The set $\tilde{\mathcal{V}}$ is identified using one iteration of the wrap-around Viterbi algorithm (WAVA) [SLF03].

4.2.2.3 Automorphism Ensemble (AE) Decoder

Ensemble decoding algorithms [GEE22] employ M parallel independent and identical sub-optimal decoders, with each proposing a codeword estimate. From among these M proposed codewords, the ensemble decoder selects the most likely candidate as the decoder output [GEE22]. One category of ensemble decoding utilizes automorphism groups. An automorphism group is a set of permutations such that the permuted sequence of any codeword is still a codeword. When an automorphism group of the codes is known, identical

constituent decoders decoding permuted versions of the channel output may be used, yielding the so-called Automorphism Ensemble (AE) decoding [GEE22].

The cyclic shifts δ_i, $i = 0, ..., n-1$, are elements of an automorphism group of the TBCC, where δ_i indicates the cyclic shift of a sequence by i positions. Hence, as illustrated in Fig. 4.3, an AE decoder for the CRC-TCM-PAS system is constructed by employing M parallel β-States decoders for the channel observations that are cyclic-shifted by $\{\delta_{i_1}, \ldots, \delta_{i_M}\}$. The i^{th} β-States decoder either provides a shifted estimation candidate or declares an erasure. The final decoding result of the AE decoder is the candidate that has the smallest Euclidean distance from the channel observation. We denote an AE decoder with M parallel decoders with cyclic shifts $\{i_1, \ldots, i_M\}$, where each decoder utilizes β starting states obtained by WAVA as the decoder AED(M, β). In this paper, the cyclic shifts $\{i_1, \ldots, i_M\}$ are uniformly sampled from $\{0, \ldots, n-1\}$.

The M independent β-States decoders of AED(M, β) can be run in parallel, so the AED(M, β) has the same time complexity with a single β-States decoder but provides more potential codewords. However, the AED(M, β) requires more hardware resources than a single β-States decoder.

4.3 Channel Input Distribution of CRC-TCM-PAS System

This section proves that the distribution of the channel input X of the CRC-TCM-PAS system is symmetric, i.e., $P_X(x) = P_X(-x)$ for $x \in \mathcal{X}$, where $\mathcal{X}$ is the PAM constellation set. We begin the proof with a theorem that shows the CRC check bits in the CRC-TCM-PAS system are asymptotically uniform, even though the input bits of the CRC encoder are not.

4.3.1 Uniformity of CRC bits

Denote the random variable that represents a DM output symbol by $\bar{A}$ with PMF $P(\bar{A})$. Because the cardinality of output symbol set is 2^{k_0}, $\bar{A}$ can be represented by k_0 bits, which are denoted by B_i, $i = 0, \ldots, k_0-1$. Since $\bar{A}$ is not uniform, B_i, $i = 0, \ldots, k_0-1$, may have different distributions. Let $a \in \mathcal{A}$ be a realization of $\bar{A}$, and let $\mathbf{b}(a) = [b_{k_0-1}(a), \ldots, b_1(a),\ b_0(a)] \in \mathbb{F}_2^{k_0}$ be the binary representation of a. The PMF of B_i is calculated by:

$$P_{B_i}(b) = \sum_{a=0}^{|\mathcal{A}|-1} P_{\bar{A}}(a)\mathbb{1}\left(b_i(a) = b\right), \tag{4.9}$$

$b = 0, 1,\ i = 0, 1, ..., k_0 - 1$. $\mathbb{1}(\cdot)$ is the indicator function. As shown in Fig. 4.1, the binary converter maps a length-l symbol sequence to a length-$k_0 l$ binary sequence. Let $G^{k_0 l} = [G_0, \ldots, G_{k_0 l-1}]$ be the random vector representing the binary sequence. Assume that the DM generates i.i.d. symbols, the G_i's that correspond to the same symbol bit position have the same distribution, i.e.:

$$P(G_i) = P(B_{i \pmod{k_0}}),\ i = 0, \ldots, k_0 l - 1. \tag{4.10}$$

Let $\mathbf{g} \in \mathbb{F}_2^{k_0 l}$ be a realization of $G^{k_0 l}$, and denote the polynomial form of $\mathbf{g}$ by $g(x) = \sum_{i=0}^{k_0 l-1} g_i x^i$.

An m-bit CRC is specified by a degree-m binary polynomial $p(x) = \sum_{i=0}^{m} p_i x^i$. Let the polynomial form of the output of the CRC encoder be $h(x) = \sum_{i=0}^{k_0 l+m-1} h_i x^i$. $h(x)$ is calculated by

$$h(x) = x^m g(x) + x^m g(x) \pmod{p(x)}. \tag{4.11}$$

The following theorem proves that the CRC check bits, h_i, $i = 0, \ldots, m-1$, can be arbitrarily close to be equally likely, with a proper choice of l.

Theorem 1. *For a length-l random vector A^l whose elements A_i, $i = 0, \ldots, l-1$, are i.i.d. random variables with alphabet $|\mathcal{A}| = \{0, 1, 2, \ldots, 2^{k_0} - 1\}$ and distribution $P(A)$. Let $G^{k_0 l}$*

be the binary representation of A^l and H^{k_0l+m} be the CRC output sequence by encoding G^{k_0l} with some degree-m CRC polynomial $p(x)$. For any $0 < \epsilon < 0.5$, there exists an l such that

$$|P_{H_i}(0) - 0.5| < \epsilon,\ i = 0, 1, \ldots, m-1.$$

Proof. Define the set of random variables in G^{k_0l} that belong to the j^{th} symbol bit position by $\mathcal{G}_j$, i.e., $\mathcal{G}_j = \{G_{k_0i+j}, i = 0, \ldots, l-1\}$, $j = 0, \ldots, k_0 - 1$. Based on (4.10), the random variables in the same set have same distribution. Let $P_{G_i}(0) = p_j$, if $G_i \in \mathcal{G}_j$.

A CRC code is a linear block code. Let $\mathcal{I}_i$, $i = 0, \ldots, m-1$, be the set of information bits that are constrained by i^{th} parity check. Let $J_{i,j}$ be the number of the elements belonged to both $\mathcal{I}_i$ and $\mathcal{G}_j$, i.e., $J_{i,j} = |\mathcal{I}_i \bigcap \mathcal{G}_j|$, where $i = 0, \ldots m-1$ and $j = 0, \ldots, k_0 - 1$. The PMF of i^{th} parity check bit, $P(H_i)$, can be calculated by: $P(H_i) = \circledast_{G_j \in \mathcal{G}_i} P(G_j)$, where $\circledast$ denotes circular convolution. Using the discrete Fourier transform, $P_{H_i}(0) = \frac{1}{2} + \frac{1}{2}\prod_{j=0}^{k_0-1}(1-2p_j)^{J_{i,j}}$. $P_{H_i}(0)$ is calculated by:

$$P_{H_i}(0) = \frac{1}{2} + \prod_{j=0}^{k_0-1}(1-2p_j)^{J_{i,j}}. \tag{4.12}$$

Because $|1 - 2p_j| < 1$, and $J_{i,j}$ gets larger with the increase of l, there exists an l such that for $i = 0, ..., m-1$, $\left|\prod_{j=0}^{\alpha-1}(1-2p_j)^{J_{i,j}}\right| < \epsilon$. □

Note that Theorem 1 can be generalized to any systematic linear block code, and it validates the uniform check bit assumption in [BSS15].

Example 1. *Let P(A)=(0.072,0.165,0.321,0.442). For a degree-5 CRC with $p(x) = x^5 + x^4 + x^2 + 1$, the minimum l that gives $|P_{H_i}(0) - 0.5| < 10^{-4}$, $i = 0, \ldots, 4$, is 20.*

Remark. *Theorem 1 can be generalized to any systematic linear block code, and it validates the uniform check bit assumption in [BSS15].*

4.3.2 Symmetry of Channel Input Distribution

Consider a length-n, rate-$\frac{k_0}{k_0+1}$, systematic, and recursive TBCC with ν memory elements. Denote the input symbol in stage t by $u_t \in \mathcal{U}$, $t = 0, ..., n-1$, and denote the state at time instant t by $v_t \in \mathcal{V}$, $t = 0, \ldots, n$. Let $\mathbf{u}_t \in \mathbb{F}_2^{k_0 \times 1}$ and $\mathbf{v}_t \in \mathbb{F}_2^{\nu \times 1}$ be the binary representation of u_t and v_t, respectively. Based on the state-space representation of convolutional code [WBR01, FW99], $\mathbf{v}_{t+1}$ is a function of $\mathbf{v_t}$ and $\mathbf{u_t}$, i.e., $\mathbf{v}_{t+1} = \mathbf{A}\mathbf{v}_t + \mathbf{B}\mathbf{u}_t$, where $\mathbf{A} \in \mathbb{F}_2^{\nu \times \nu}$ and $\mathbf{B} \in \mathbb{F}_2^{\nu \times k_0}$. The initial state $\mathbf{v}_0$ of a recursive TBCC codeword can be determined by the following equation:

$$\mathbf{v}_0 = (\mathbf{A}^n + \mathbf{I}_\nu)^{-1}\mathbf{v}_n^{[zs]}, \tag{4.13}$$

where $\mathbf{I}_\nu$ is a size ν identity matrix and $\mathbf{A}^n + \mathbf{I}_\nu$ is an invertible matrix [WBR01]. The term $\mathbf{v}_N^{[zs]}$ is referred to as zero-state solution and is the final state by encoding the dataword with initial state 0. The encoding of tail-biting convolutional code has two steps:

1. Run encoding process first time by setting $v_0 = 0$ and record $v_n^{[zs]}$.
2. Run encoding process second time by setting v_0 using (4.13) and generate output symbols.

Therefore, in order to study the distribution of the output symbols of a recursive TBCC, we need to know the distribution of $v_n^{[zs]}$ by analyzing the first encoding process.

For the CRC-TCM-PAS system, the data frames, i.e., input symbols, of TBCC are the outputs of CRC encoder. Because the CRC encoder is systematic, the first $n - \frac{m}{k_0}$ input symbols of TBCC have DM output symbol distribution $P(\bar{A})$. Based on Theorem 1, the last $\frac{m}{k_0}$ input symbols have uniform distributions. This subsection uses state-space representation of convolution code in [WBR01, FW99] to analyze the PMF of the state in time instant t, V_t. The PMF of V_t, is calculated by:

$$P_{V_t}(v_t) = \sum_{v_{t-1} \in \mathcal{V}} P(v_{t-1}) \sum_{(v_{t-1}, o_t} , v_t) \in \mathcal{E} P(o_t, v_t | v_{t-1}). \tag{4.14}$$

Let $u_t = g^{-1}(v_{t-1}, o_t, v_t) \in \mathcal{U}$ be the input symbol that associates to the edge (v_{t-1}, o_t, v_t). Hence, $P(o_t, v_t|v_{t-1}) = P_{U_t}(g^{-1}(v_{t-1}, o_t, v_t))$. If the convolution code is systematic, the input corresponded to (v_{t-1}, o_t, v_t) can be solely determined by *ccoutputrealize*$_t$, we use $g^{-1}(o_t) = g^{-1}(v_{t-1}, o_t, v_t)$ as a simplification. Define the matrix $\mathbf{C}_{t-1} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ as follows:

$$\mathbf{C}_{t-1}(v_t, v_{t-1}) = P(v_t|v_{t-1}) = \sum_{(v_{t-1},o_t,v_t)\in\mathcal{E}} P(o_t, v_t|v_{t-1}). \tag{4.15}$$

Let $\mathbf{p_t} = [P_{V_t}(0) \dots P_{V_t}(2^\nu - 1)]^T$, (4.14) can be rewritten as:

$$\mathbf{p}_t = \mathbf{C}_{t-1}\mathbf{p}_{t-1} = \left(\prod_{i=0}^{t-1} \mathbf{C}_i\right) \mathbf{p}_0, \ t = 1, 2, ..., n. \tag{4.16}$$

(4.15) implies that $\mathbf{C}_{t-1}$ is a left stochastic matrix, i.e., each column in $\mathbf{C}_{t-1}$ is a probability vector. Moreover, $\mathbf{C}_{t-1}$ is also right stochastic, meaning that each row has a sum of 1. To see this, for the trellis of a convolutional code, for each $v_t \in \mathcal{V}$, there are 2^{k_0} edges that connect v_t and each edge associates a distinct input in $\mathcal{U}$. As a result, $\mathbf{C}_{t-1}$ is a doubly stochastic matrix.

Theorem 2. *For an (γ_0, k_0, ν) convolutional code with any initial state distribution $P(V_0)$, if the data frames are i.i.d. random variables with PMF $P(U)$ and $P_U(u) > 0$ for any $u \in \mathcal{U}$. Let V_t be the state at time instant t, then the random sequence $V_0, V_1, \dots$ converges in distribution to a uniform random variable V_{uni}, i.e., $V_t \xrightarrow{d} V_{\text{uni}}$.*

Proof. Because all the data frames have the same distribution, it has $\mathbf{C}_t = \mathbf{C}$. Hence, (4.16) can be rewritten as $\mathbf{p}_t = \mathbf{C}^t\mathbf{p}_0$. $\mathbf{C}$ is not only a doubly stochastic matrix but also a regular matrix. For a convolutional code, any state $v_i \in \mathcal{V}$ can always reach any state $v_j \in \mathcal{V}$ with a finite-length path. $\mathbf{C}$ retains this property, because $P_U(u) > 0$, for any $u \in \mathcal{U}$. As a result, $\mathbf{C}$ is regular. Based on Perron-Frobenius theorem [Gan00], the non-negative and regular matrix $\mathbf{C}$ has the following properties:

1. $\mathbf{C}$ has $\lambda_1 = 1$ as an eigenvalue of multiplicity 1, and the normalized right eigenvector corresponded to eigenvalue 1 is $\mathbf{q}^* = \left[\frac{1}{\sqrt{2^\nu}} \ \frac{1}{\sqrt{2^\nu}} \dots \frac{1}{\sqrt{2^\nu}}\right]^T$.

2. For all other eigenvalues λ_j, $j = 2, ..., q$, it has $|\lambda_j|$ is *strictly* smaller than 1, i.e., $|\lambda_j| < 1$.

Let $\mathbf{J} = \mathbf{Q}^{-1}\mathbf{C}\mathbf{Q}$ be the Jordan canonical form of $\mathbf{C}$. Based on Perron-Frobenius theorem, $\mathbf{J} = \text{diag}(1, \mathbf{J}_2, \dots, \mathbf{J}_q)$, where $\mathbf{J}_2, \dots, \mathbf{J}_q$ are Jordan block matrices that correspond to eigenvalues $\lambda_2, \dots, \lambda_q$, respectively. Let $\mathbf{Q} = [\mathbf{q}_1 \dots \mathbf{q}_{2^\nu}]$ and, $\mathbf{q}_1$ is the eigenvector associated to eigenvalue 1, $\mathbf{q}_1 = \alpha\mathbf{q}_1^*$, $\alpha \in \mathbb{R}$. Let $\mathbf{p}_0 = \sum_{i=1}^{2^\nu} c_i\mathbf{q}_i = \mathbf{Q}\mathbf{c}$, it has $\mathbf{p}_t = \mathbf{C}^t\mathbf{p}_0 = \mathbf{Q}\mathbf{J}^t\mathbf{c}$. Because $\lim_{t\to\infty} \mathbf{J}_i = \mathbf{0}$ for $j = 2, \dots, 2^\nu$, it has $\lim_{t\to\infty} \mathbf{p}_t = c_1\alpha\mathbf{q}_1^* = \left[\frac{1}{2^\nu} \dots \frac{1}{2^\nu}\right]^T$. □

Example 2. *Consider the (3,2,3) convolution code shown in Fig. 4.1. Let the initial state be 0 and* $P(U) = (0.5742, 0.3188, 0.01642, 0.09048)$. *When* $t = 12$, $|P_{V_{12}}(v) - \frac{1}{8}| < 10^{-4}$, $v = 0, \dots, 7$.

Besides, if the state distribution at time t is uniform, the state distribution at time $t+1$ is also uniform, no matter what $P(U_t)$ is. Hence, the zero-state solution, as well as the initial state of TBCC, have a uniform distribution. As indicated in (4.13), the TBCC initial state is a one-to-one mapping of zero-state solution, thus the initial state of TBCC has uniform distribution. As a result, the states at all $n+1$ time instants in second encoding process have uniform distribution.

Now, we show that if the state at time instant t is uniform, then the (k_0+1, k_0, ν) systematic recursive TBCC generates an equally likely parity check bit in stage t. First of all, the following theorem gives that distribution of output symbol in stage t.

Theorem 3. *Consider a* (k_0+1, k_0, ν) *systematic recursive convolutional code that is defined by state set* $\mathcal{V}$, *edge set* $\mathcal{E}$, *input set* $\mathcal{U}$, *and output set* $\mathcal{O}$. *If the state distribution at time instant* t *is uniform, i.e.,* $\mathbf{p}_t = \left[\frac{1}{2^\nu}\frac{1}{2^\nu} \dots \frac{1}{2^\nu}\right]^T$, *then the output symbol distribution in stage* t, $P_{O_t}(o_t) = \frac{1}{2}P_{U_t}(g^{-1}(o_t))$, $\forall o_t \in \mathcal{O}$.

Proof. Define matrix $\mathbf{D}_t \in \mathbb{R}^{|\mathcal{O}|\times|\mathcal{V}|}$ with $\mathbf{D}_t(o_t, v_{t-1}) = P(o_t|v_{t-1})$, where $o_t \in \mathcal{O}$ and $v_{t-1} \in \mathcal{V}$. Define $\mathbf{q}_t = [P_{O_t}(0) \dots P_{O_t}(|\mathcal{O}|-1)]^T$. $\mathbf{q}_t$ can be calculated by $\mathbf{q}_t = \mathbf{D}_{t-1}\mathbf{p}_t$.

Because the TBCC is systematic, $\mathbf{D}_t(o_t, v_{t-1}) = P_{U_t}(g^{-1}(o_t))$. Hence, one property of $\mathbf{D}_t$ is that the non-zero elements in each row have the same value.

The other property is that $\mathbf{D}_t$ contains $2^{\nu-1}$ non-zero elements for each row, i.e., given any output $o_t \in \mathcal{O}$, there are only $2^{\nu-1}$ possible states from which o_t can be generated. This is because for a rate-$\frac{k_0}{k_0+1}$, systematic, recursive convolution code, the register adjacent to the output is determined by o_t, hence the freedom of v_{t-1} is reduced by 1. Based on the two properties of $\mathbf{D}$, for any $o_t \in \mathcal{O}$, it has: $P_{O_t}(o_t) = \sum_{i=1}^{2^\nu} \mathbf{D}(l, i)P_{V_t}(i) = \frac{1}{2}P_{U_t}(g^{-1}(o_t))$. □

Theorem 3 implies that, if the state distribution at time t is uniform, then the parity bit generated by the convolutional code at stage t is uniform. Because the sign value of the channel input symbol at stage t, X_t, is determined by parity bit, it has $P_{X_t}(x) = P_{X_t}(-x)$, for $x \in \mathcal{X}$. Note that one rule of channel-signal mapping function in CRC-TCM-PAS is that information bits indicate magnitude and parity check bit indicates sign.

Because the states of each time instant of TBCC have uniform distribution, the channel inputs in each stage have symmetric distributions. Besides, the magnitude distributions of first $n - \frac{m}{k_0}$ and last $\frac{m}{k_0}$ channel inputs follow $P(\bar{A})$ and uniform distribution, respectively.

4.4 FER Upper Bound for CRC-TCM-PAS System

In this section, we derive the FER upper bound for the CRC-TCM-PAS system with the specified CC, CRC, and an ideal distribution matcher that generates length-l symbol sequences with the desired distribution $P(\hat{A}^l)$. The upper bound is computed using the generating function of an equivalent convolutional code whose error events correspond exactly to the undetectable error events of the concatenation of the original CRC and CC.

4.4.1 Equivalent Code for CRC-Aided Convolutional Code

As shown in Fig. 4.1, the binary representation of the symbol sequence generated by a distribution matcher is encoded by a CRC and a TBCC serially. We begin our analysis by replacing the CRC and convolutional encoder with a single convolutional encoder whose input is the quotient of dividing the CRC codeword by the CRC polynomial.

Let $\mathbf{h}$ be a length-$(\tilde{l}+m)$ CRC codeword with polynomial form $h(x) = \sum_{t=0}^{\tilde{l}+m+1} h_t x^t$. Based on the notation in Fig. 4.1, $\tilde{l} = k_0 l$. For a rate-$\frac{k_0}{k_0+1}$ convolutional code, there are k_0 input branches. Let the input of the i^{th} branch be $\mathbf{h}^{(i)}$, and let the corresponding polynomial be $h^{(i)}(x)$. $\mathbf{h}^{(i)} = [h_i \; h_{k_0 i} \dots h_{\tilde{l}+m-k_0+i}]$ is obtained by sampling $\mathbf{h}$ every k_0 positions starting from i^{th} position, and $h^{(i)}(x) = \sum_{t=0}^{(\tilde{l}+m)/k_0-1} h_{k_0 t+i} x^t$, $i = 0, \dots, k_0 - 1$.

Let $\mathbf{q}$ be the quotient of dividing the CRC output by the CRC polynomial. The polynomial form of $\mathbf{q}$, $q(x)$, is calculated by

$$q(x) := h(x)/p(x). \tag{4.17}$$

Theorem 4. *Consider an m-bit CRC encoder which is specified by an m-degree polynomial $p(x)$. Let the number of input bits be $\tilde{l}$. Let k_0 be an integer that divides $m + \tilde{l}$. Then for any codeword polynomial $h(x)$, its k_0-split polynomial vector, $\mathbf{h}_{k_0}(x)$ can be calculated by $\mathbf{h}_{k_0}(x) = \mathbf{q}_{k_0}(x)\mathbf{P}_{\text{eq}}(x)$.*

$$\mathbf{h}_{k_0}(x) = \mathbf{q}_{k_0}(x)\mathbf{P}_{\text{eq}}(x), \tag{4.18}$$

where $\mathbf{q}_{k_0}(x)$ is the k_0-split polynomial vector of $q(x) = h(x)/p(x)$ and $\mathbf{P}_{\text{eq}}(x) \in \mathbb{F}_2[x]^{k_0 \times k_0}$ is a $k_0 \times k_0$ square binary polynomial matrix.

Proof. Based on the relationship $h(x) = p(x)q(x)$, the t^{th} bit of $\mathbf{h}^j$, $h_t^{(j)}$ is calculated by:

$$h_t^{(j)} = h_{k_0t+j} = \sum_{s=0}^{m} q_{k_0t+j-s}p_s \tag{4.19}$$

$$= q_{k_0t+j}p_0 + \sum_{\ell=0}^{m/k_0-1}\sum_{s'=1}^{k_0} q_{k_0t+j-k_0\ell-s'}p_{k_0\ell+s'} \tag{4.20}$$

$$= \sum_{\ell=0}^{m/k_0-1}\sum_{i=0}^{j} q_{k_0(t-\ell)+i}p_{k_0\ell+j-i} + \sum_{\ell=1}^{m/k_0}\sum_{i=j+1}^{k_0-1} q_{k_0(t-\ell)+i}p_{k_0\ell+j-i} + q_{k_0t+j-m}p_m \tag{4.21}$$

Let $p_t^{(i)} = p_{kt+i}$, $h_t^{(j)}$ can be rewritten as:

$$h_t^{(j)} = \sum_{i=0}^{j}\sum_{\ell=0}^{m/k_0-\mathbb{1}(i\neq j)} q_{t-\ell}^{(i)}p_\ell^{(j-i)} + \sum_{i=j+1}^{k_0-1}\sum_{\ell=0}^{m/k_0-1} q_{t-\ell-1}^{(i)}p_{\ell+1}^{(j-i+k_0)}. \tag{4.22}$$

Define $p^{(i)}(x) = \sum_{t=0}^{m/k_0-\mathbb{1}(i=0)} p_{k_0t+i}x^t$. The $h^{(j)}(x)$ can be calculated by:

$$h^{(j)}(x) = \sum_{i=0}^{j} q^{(i)}(x)p^{(j-i)}(x) + \sum_{i=j+1}^{k_0-1} xq^{(i)}(x)p^{(j-i+k_0)}(x)\,. \tag{4.23}$$

(4.23) implies that, by choosing the polynomial of i^{th} row and j^{th} column of $\mathbf{P}_{\text{eq}}(x)$ as:

$$\mathbf{P}_{\text{eq}}(x)_{i,j} = p^{(j-i)}(x)\mathbb{1}(i \leq j) + xp^{(j-i+k_0)}(x)\mathbb{1}(i > j), \tag{4.24}$$

it has

$$\mathbf{h}_{\text{split}}(x) = \mathbf{q}_{\text{split}}(x)\mathbf{P}_{\text{eq}}(x). \tag{4.25}$$

□

As a result, the concatenation of a CRC with generator polynomial $p(x)$ and a rate-$\frac{k_0}{k_0+1}$ convolutional code with generator matrix $\mathbf{G}(x)$ is equivalent to a convolutional code with generator matrix $\mathbf{G}_{\text{eq}}(x)$, which is defined as follows:

$$\mathbf{G}_{\text{eq}}(x) = \mathbf{P}_{\text{eq}}(x)\mathbf{G}(x). \tag{4.26}$$

The error events of the equivalent convolutional code correspond exactly to the error events of the original concatenation of CRC and convolutional code. Because the concatenation of a CRC expurgates the original TBCC by removing the codewords whose corresponding messages do not pass the CRC, the remaining codewords all meet the tail-biting condition so that the equivalent convolutional code is still tail-biting.

4.4.2 FER Upper Bound

This subsection bounds the FER for the CRC-TCM-PAS system. Based on the analysis in the previous subsection, the CRC-aided TBCC can be replaced by an equivalent TBCC with the generator matrix given in (4.26). The final computation of FER requires the output symbol distributions. For the purposes of this analysis, we assume a distribution matcher that generates l i.i.d. symbols with the target symbol distribution $P(A)$. After the distribution matcher, $n-l$ CRC symbols are appended to the sequence. Based on Theorem 1, these CRC symbols should be approximated as having a uniform distribution rather than $P(A)$. The output symbol distributions for the analyzed system of the equivalent TBCC with the generator matrix given in (4.26) with our idealized distribution matcher are thus l output symbols distributed according to $P(A)$ and $n-l$ output symbols distributed according to a uniform distribution.

Let $\mathcal{C}_{\mathrm{T}} \subset \mathcal{X}^n$ be the codebook of TCM. Let $\mathbf{x}_c \in \mathcal{C}_{\mathrm{T}}$ be the transmitted codeword, and let $\mathbf{y}$ be the channel observation over AWGN channel. Let $\varepsilon_{\mathbf{x}_c}$ denote the event that, given observation $\mathbf{y}$, an ML decoder selects $\hat{\mathbf{x}} \neq \mathbf{x}_c$. Let $e_{\mathbf{x}_c,\mathbf{x}_e}$ denote the event that, given $\mathbf{y}$, codeword $\mathbf{x}_e$ is more likely than codeword $\mathbf{x}_c$. The FER of CRC-TCM-PAS transmission

system P_e is upper bounded by the union bound:

$$P_e = \sum_{\mathbf{x}_c \in \mathcal{C}_{\mathrm{CT}}} P(X^n = \mathbf{x}) P\left(\varepsilon_{\mathbf{x}_c}\right) \tag{4.27}$$

$$= \sum_{\mathbf{x}_c \in \mathcal{C}_{\mathrm{CT}}} P(X^n = \mathbf{x}) P\left(\bigcup_{\substack{\mathbf{x}_e \in \mathcal{C}_{\mathrm{CT}} \\ \mathbf{x}_c \neq \mathbf{x}_e}} e_{\mathbf{x}_c, \mathbf{x}_e}\right) \tag{4.28}$$

$$\leq \sum_{\mathbf{x}_c \in \mathcal{C}_{\mathrm{T}}} P(X^n = \mathbf{x}_c) \sum_{\substack{\mathbf{x}_e \in \mathcal{C}_{\mathrm{T}} \\ \mathbf{x}_e \neq \mathbf{x}_c}} P\left(e_{\mathbf{x}_c, \mathbf{x}_e}\right). \tag{4.29}$$

The probability $P\left(e_{\mathbf{x}_c, \mathbf{x}_e}\right)$ is referred as the pairwise error probability (PEP). With the assumption of i.i.d. symbols of distribution matcher, and based on the analysis on CRC bits and channel inputs, Because the input symbols are independent, it has $P(X^n = \mathbf{x}) = \prod_{i=0}^{n} P(X_i = x_i)$. Not that the distributions of $P(X_i)$, $i = 0, ..., l-1$ can be derived from $P(\bar{A})$ and last $\frac{m}{k_0}$ symbols in $\mathbf{x}$ have uniform distribution.

Because $P(X^n)$ is non-uniform[1], choosing the codeword that has the smallest Euclidean distance with the channel observation is no longer optimal. Let $\mathbf{u}_c, \mathbf{u}_e$ denote the convolutional inputs corresponding to outputs $\mathbf{x}_c, \mathbf{x}_e$, $e_{\mathbf{x}_c, \mathbf{x}_e}$ happens if $P_{X^n|Y^n}\left(\mathbf{x}_e|\mathbf{y}\right) > P_{X^n|Y^n}\left(\mathbf{x}_c|\mathbf{y}\right)$, this condition is equivalent to:

$$\log\left(f(\mathbf{y}|\mathbf{x}_e) P_{X^n}(\mathbf{x}_e)\right) > \log\left(f(\mathbf{y}|\mathbf{x}_c) P_{X^n}(\mathbf{x}_c)\right) \tag{4.30}$$

$$\iff ||\mathbf{x}_c - \mathbf{y}||_2^2 - ||\mathbf{x}_e - \mathbf{y}||_2^2 > 2\sigma^2 \log\left(\frac{P_{X^n}(\mathbf{x}_c)}{P_{X^n}(\mathbf{x}_e)}\right) \tag{4.31}$$

$$\iff 2\left\langle \mathbf{y} - \mathbf{x}_c, \mathbf{x}_e - \mathbf{x}_c \right\rangle - ||\mathbf{x}_c - \mathbf{x}_e||_2^2 > 2\sigma^2 \log\left(\frac{P_{X^n}(\mathbf{x}_c)}{P_{X^n}(\mathbf{x}_e)}\right). \tag{4.32}$$

$\langle \cdot, \cdot \rangle$ represents the inner product and $|| \cdot ||_2$ represents l^2-norm.

Define z' as follows:

$$z' = \frac{\left\langle \mathbf{y} - \mathbf{x}_c, \mathbf{x}_e - \mathbf{x}_c \right\rangle}{||\mathbf{x}_c - \mathbf{x}_e||_2}, \tag{4.33}$$

[1]In a practical CRC-TCM-PAS system, the codewords are uniform, specifically, $P_{X^n}(\mathbf{x}) = \frac{1}{2^k} \mathbb{1}(\mathbf{x} \in \mathcal{X}_{\mathrm{CTP}})$.

it can be proved that $z' \sim \mathcal{N}(0, \sigma^2)$. Manipulating (4.32) reveals that $e_{\mathbf{x}_c,\mathbf{x}_e}$ occurs if the following inequality is satisfied:

$$z' > \frac{1}{2}||\mathbf{x}_c - \mathbf{x}_e||_2 + \frac{\sigma^2}{||\mathbf{x}_c - \mathbf{x}_e||_2} \log\left(\frac{P_{X^n}(\mathbf{x}_c)}{P_{X^n}(\mathbf{x}_e)}\right) \tag{4.34}$$

$$\triangleq \frac{1}{2} d(\mathbf{x}_c, \mathbf{x}_e). \tag{4.35}$$

Note that d is not a metric as $d(\mathbf{x}_c, \mathbf{x}_e) \neq d(\mathbf{x}_e, \mathbf{x}_c)$.

Applying (4.34) yields

$$P\left(e_{\mathbf{x}_c,\mathbf{x}_e}\right) = Q\left(\frac{\sqrt{d^2\left(\mathbf{x}_c, \mathbf{x}_e\right)}}{2\sigma}\right), \tag{4.36}$$

where

$$d^2(\mathbf{x}_c, \mathbf{x}_e) = ||\mathbf{x}_c - \mathbf{x}_e||_2^2 + 4\sigma^2 \log\left(\frac{P_{X^n}(\mathbf{x}_c)}{P_{X^n}(\mathbf{x}_e)}\right) + \left(\frac{2\sigma^2}{||\mathbf{x}_c - \mathbf{x}_e||_2} \log\left(\frac{P_{X^n}(\mathbf{x}_c)}{P_{X^n}(\mathbf{x}_e)}\right)\right)^2. \tag{4.37}$$

Define $d^2_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e)$ by neglecting the last squared term in (4.37), i.e.:

$$d^2_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e) = ||\mathbf{x}_c - \mathbf{x}_e||_2^2 + 4\sigma^2 \log\left(\frac{P_{X^n}(\mathbf{x}_c)}{P_{X^n}(\mathbf{x}_e)}\right). \tag{4.38}$$

Because $d^2_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e) \leq d^2(\mathbf{x}_c, \mathbf{x}_e)$, the PEP $P(e_{\mathbf{x}_c,\mathbf{x}_e})$ is upper bounded by:

$$P(e_{\mathbf{x}_c,\mathbf{x}_e}) = Q\left(\frac{\sqrt{d^2(\mathbf{x}_c, \mathbf{x}_e)}}{2\sigma}\right) \leq Q\left(\frac{\sqrt{d^2_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e)}}{2\sigma}\right). \tag{4.39}$$

Hence, P_e is further bounded by:

$$P_e \leq \sum_{\mathbf{x}_c \in \mathcal{C}_{\text{T}}} P(X^n = \mathbf{x}_c) \sum_{\substack{\mathbf{x}_e \in \mathcal{C}_{\text{T}} \\ \mathbf{x}_e \neq \mathbf{x}_c}} Q\left(\frac{\sqrt{d^2_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e)}}{2\sigma}\right). \tag{4.40}$$

Based on the ideal DM assumption and our analysis of CRC and TBCC encoding, the output symbols of the CRC-TCM-PAS system are independent of each other. Hence,

$$d^2_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e) = \sum_{i=1}^{n} d^2_{\text{prox}}\left(x_{c,i}, x_{e,i}\right), \tag{4.41}$$

where $x_{c,i}$ and $x_{e,i}$ are the i^{th} element in $\mathbf{x}_c$ and $\mathbf{x}_e$, respectively, and

$$d^2_{\text{prox}}\left(x_{c,i}, x_{e,i}\right) = (x_{c,i} - x_{e,i})^2 + 4\sigma^2 \log \frac{P_{X_i}(x_{c,i})}{P_{X_i}(x_{e,i})}. \tag{4.42}$$

Besides, the omitted term

$$\left(\frac{2\sigma^2}{||\mathbf{x}_c - \mathbf{x}_e||_2} \log\left(\frac{p(\mathbf{u}_c)}{p(\mathbf{u}_e)}\right)\right)^2 \sim \left(\sigma^2\right)^2. \tag{4.43}$$

As a result, d^2_{prox} approaches d^2 quadratically with SNR.

4.4.3 Generating Function with State-Reduction Method

This subsection derives the generating function of non-uniform-input TCM using Biglieri's product state method [Big84], with state-reduction method as described in [Wes04]. The product state diagram [Big84] is built by replacing each state in the error state diagram with a complete encoder state diagram. Hence, for a convolutional code that has ν memory elements, there are totally $2^{2\nu}$ states in the product state diagram. Wesel in [Wes04] reduces the total number of states by proposing an "equivalence-class encoder" with ν_x memory elements. Because $\nu_x < \nu$, the state-reduction method requires fewer states than the product state diagram.

For an equivalence-class encoder, denote the set of output by $\mathcal{O}_{\text{eq}}$. Let $q \in \mathcal{O}_{\text{eq}}$ be an output of the equivalent-class encoder. Let $e_o \in \mathcal{O}$ be a symbol error. As a reminder, $\mathcal{O}$ is the set of TBCC output symbols. Let x_q, x_{qe_o} be any constellation point that belongs to equivalent class q and the constellation point that x_q moves to because of e_o. We define $d^2_{\text{prox}}(q, e_o)$ as follows:

$$d^2_{\text{prox}}\left(q, e_o\right) = (x_q - x_{qe_o})^2 + 4\sigma^2 \log \frac{P_X(x_q)}{P_X(x_{qe_o})}. \tag{4.44}$$

We follow the notations in [Wes04] to describe the state-reduced product state diagram. Denote the set of equivalence-class encoder states and the set of error states by $\mathcal{S}_q$ and $\mathcal{S}_e$, respectively. The pair $(s_q, s_e) \in \mathcal{S}^* = \mathcal{S}_q \times \mathcal{S}_e$ describes where the states "should be" if

there is no error occurs, and where the state is "drifted to" because of some error event. The notation "$\times$" means Cartesian product. Let $(s_q, s_e), (s'_q, s'_e) \in \mathcal{S}^*$, we label the state transition $(s_q, s_e) \to (s'_q, s'_e)$ with

$$P\left(s_q \to s'_q\right) \sum_{e_o} \sum_{\tilde{q}} P\left(\tilde{q} | s_q \to s'_q\right) W^{d^2_{\text{prox}}(\tilde{q}, e_o)}, \tag{4.45}$$

where $s_q \to s'_q$ is the event that the state of the equivalent class encoder transits from s_q to s'_q. The first summation is over all possible symbol error e_0 due to error state diagram transition $s_e \to s'_e$, and the second summation is over all possible equivalent class q' due to equivalent-class encoder state diagram transition $s_q \to s'_q$.

Based on the channel-signal mapping rule, the constellation of TCM output is symmetric with respect to 0 and the equivalence class is determined by the systematic bits. Thus, one generator polynomial matrix of the minimal equivalent-class encoder for the rate-$\frac{k_0}{k_0+1}$, systematic TBCC in TCM is simply a size-k_0 identity matrix. Thus, by Theorem 1 in [Wes04], it is sufficient to use the error state diagram to compute the transfer function, and the label of transition $s_e \to s'_e$ is $\sum_{e_o} \sum_{q \in \mathcal{O}_{\text{eq}}} P(q)\, W^{d^2_{\text{prox}}(q, e_o)}$. The equivalent class q of the constellation of TCM output is associated with the magnitude of the constellation point, which has either capacity-approaching distribution $P(A)$ for the first $n - l$ output symbols or uniform distribution for the last l output symbols. In the proposed transmission system, the magnitudes of the channel inputs are provided by either distribution matcher or CRC bits. Hence, $P(q)$ is either the empirical distribution of distribution matcher $P(\bar{A})$ or the uniform distribution P_{uni}, depending on whether q is correspond to the distribution matcher symbols or the CRC symbols.

Define $|\mathcal{S}_e| \times |\mathcal{S}_e|$ matrices $\mathbf{G}_A(W)$ and $\mathbf{G}_{\text{uni}}(W)$ that enumerate all possible state transitions with equivalent-class PMFs of $P(A)$ and uniform distribution as follows:

$$\mathbf{G}_A(W)_{s_e, s'_e} = \sum_{e_o} \sum_{q} P_A(q) W^{d^2_{\text{prox}}(q, e_o)}, \tag{4.46}$$

$$\mathbf{G}_{\text{uni}}(W)_{s_e, s'_e} = \sum_{e_o} \sum_{q} \frac{1}{|\mathcal{A}|} W^{d^2_{\text{prox}}(q, e_o)}. \tag{4.47}$$

We define the generating function as

$$T_{\text{TBCC}}(W) = -1 + \sum_{i=0}^{\mathcal{S}_e} \mathbf{e}_i \mathbf{G}_A^l(W) \mathbf{G}_{\text{uni}}^{n-l}(W) \mathbf{e}_i^{\text{T}}, \tag{4.48}$$

where $\mathbf{e}_i$ is a length $|\mathcal{S}_e|$ indicator vector where $e_{i,j} = \mathbb{1}(j = i)$. For the TBCC, the error events must be tail-biting paths, $\mathbf{v}_i$ selects the starting/ending state of the error events.

Define the free distance, $d_{\text{free}} = \min_{\mathbf{x}_c, \mathbf{x}_e \in \mathcal{C}_{\text{T}}} d_{\text{prox}}(\mathbf{x}_c, \mathbf{x}_e)$. With the inequality:

$$Q\left(\frac{\sqrt{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)}}{2\sigma}\right) \leq Q\left(\frac{\sqrt{d_{\text{free}}^2}}{2\sigma}\right) \exp\left(\frac{d_{\text{free}}^2 - d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)}{8\sigma^2}\right), \tag{4.49}$$

P_e in (4.40) is further bounded by:

$$\begin{aligned} P_e \leq\ & Q\left(\frac{\sqrt{d_{\text{free}}^2}}{2\sigma}\right) \exp\left(\frac{d_{\text{free}}^2}{8\sigma^2}\right) \times \\ & \sum_{\mathbf{x}_c \in \mathcal{C}_{\text{T}}} \sum_{\substack{\mathbf{x}_e \in \mathcal{C}_{\text{T}} \\ \mathbf{x}_e \neq \mathbf{x}_c}} \prod_{i=1}^{n} \left[\exp\left(-\frac{d_{\text{prox}}^2(x_{c,i}, x_{e,i})}{8\sigma^2}\right) P_{X_i}(x_{c,i})\right]. \end{aligned} \tag{4.50}$$

Note the (4.49) can be proved by $Q(\sqrt{x+y}) \leq Q(\sqrt{x})e^{-\frac{y}{2}}$, for $x, y \geq 0$. The double summation term in (4.50) can be rewritten as follows:

$$\sum_{\mathbf{x}_c \in \mathcal{C}_{\text{T}}} \sum_{\substack{\mathbf{x}_e \in \mathcal{C}_{\text{T}} \\ \mathbf{x}_e \neq \mathbf{x}_c}} \left[\exp\left(-\frac{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)}{8\sigma^2}\right) P_{X^n}(\mathbf{x}_c)\right], \tag{4.51}$$

$$= \sum_{\mathbf{x}_c \in \mathcal{C}_{\text{T}}} \sum_{\substack{\mathbf{x}_e \in \mathcal{C}_{\text{T}} \\ \mathbf{x}_e \neq \mathbf{x}_c}} \left[W^{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)} P_{X^n}(\mathbf{x}_c)\right]\Big|_{W = e^{-\frac{1}{8\sigma^2}}}, \tag{4.52}$$

$$= \sum_{\mathbf{x}_c, \mathbf{x}_e \in \mathcal{C}_{CT}} \left[W^{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)} P_{X^n}(\mathbf{x}_c)\right]\Big|_{W = e^{-\frac{1}{8\sigma^2}}} - \sum_{\mathbf{x}_c \in \mathcal{C}_{CT}} \left[W^{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_c)} P_{X^n}(\mathbf{x}_c)\right]\Big|_{W = e^{-\frac{1}{8\sigma^2}}} \tag{4.53}$$

$$= \sum_{\mathbf{x}_c, \mathbf{x}_e \in \mathcal{C}_{CT}} \left[W^{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)} P_{X^n}(\mathbf{x}_c)\right]\Big|_{W = e^{-\frac{1}{8\sigma^2}}} - \sum_{\mathbf{x}_c \in \mathcal{C}_{\text{CT}}} \left[W^0 P_{X_i}(x_{c,i})\right] \tag{4.54}$$

$$= \sum_{\mathbf{x}_c, \mathbf{x}_e \in \mathcal{C}_T} \left[W^{d_{\text{prox}}^2(\mathbf{x}_c, \mathbf{x}_e)} P_{X^n}(\mathbf{x}_c)\right]\Big|_{W = e^{-\frac{1}{8\sigma^2}}} - 1, \tag{4.55}$$

Table 4.2: Optimized Convolutional Code and CRC Pairs. All the parameters are optimized while SNR equals 11 dB.

		$H^0(D)$	$H^1(D)$	$H^2(D)$	$p(x)$	FER bound
$\nu = 3$	Ung.	13	04	00	7	6.65e-4
$m = 2$	Opt.	13	06	00	5	5.80e-4
$\nu = 5$	Ung.	45	10	00	5	8.20e-5
$m = 2$	Opt.	43	26	00	5	6.58e-5
$\nu = 7$	Ung.	235	126	000	5	1.15e-5
$m = 2$	Opt.	211	142	000	5	8.96e-6

$$= \sum_{\substack{\mathbf{q}\in\mathcal{O}_{\text{eq}}^n \\ \mathbf{e}\in\mathcal{C}_{\text{T}}^n}} \prod_{i=1}^{n} \left[W^{d^2_{\text{prox}}(q_i,e_i)} P(q_i) \right] \Big|_{W=e^{-\frac{1}{8\sigma^2}}} - 1, \tag{4.56}$$

$$= \sum_{i=0}^{|\mathcal{S}_e|} \mathbf{e}_i \mathbf{G}_A^l(W) \mathbf{G}_{\text{uni}}^{n-l}(W) \mathbf{e}_i^{\text{T}} \Big|_{W=e^{-\frac{1}{8\sigma^2}}} - 1. \tag{4.57}$$

As a result, the FER upper bound can be calculated using the generating function by

$$P_e \leq Q\left(\frac{\sqrt{d^2_{\text{free}}}}{2\sigma}\right) \exp\left(\frac{d^2_{\text{free}}}{8\sigma^2}\right) T_{\text{TBCC}}\left(W = e^{-\frac{1}{8\sigma^2}}\right). \tag{4.58}$$

4.5 Simulation results

This section evaluates the performance of the CRC-TCM-PAS system over AWGN channel with different DMs and decoding methods. The CRC-TCM-PAS systems use degree-2 CRCs and rate-2/3 TBCCs. The channel inputs are equidistant 8-PAM symbols. We use the magnitudes (0.449, 1.348, 2.247, 3.146) with the PMF (0.5877, 0.3120, 0.0144, 0.0859) that is optimized for an SNR of 8 dB using a version of DAB that constrains the points to be equally spaced [XWS21]. The capacity-approaching amplitude distribution $P(\hat{A})$ that DMs target is optimized using the DAB algorithm [XWS21] at an SNR of 8 dB.

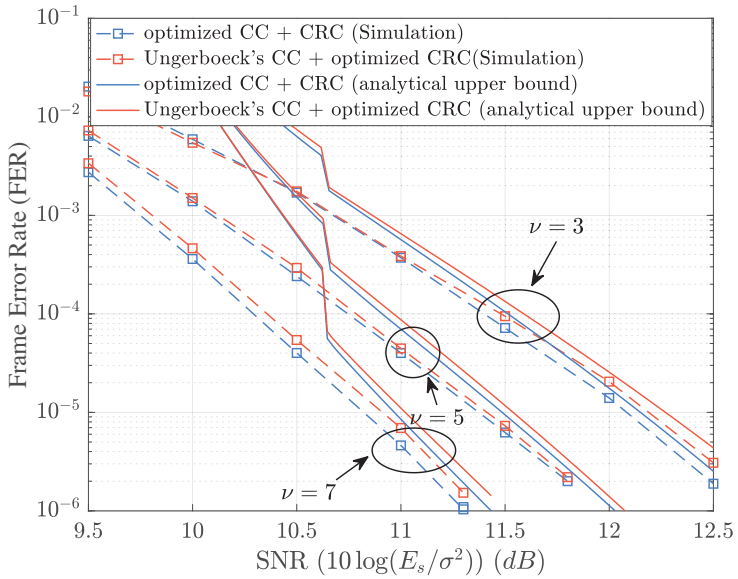


Figure 4.4: The upper bounds and FER simulations of the simplified CRC-TCM-PAS system with a degree-2 CRC. The simplified system takes length-64 i.i.d. 4-ary amplitude symbol sequences and generates length-65 8-AM symbol sequences.

Fig. **??** considers a CRC-TCM-PAS system with $k = 87$ input bits and $n = 65$ output symbols. We use the FER upper bound derived in Section 4.4 as an objective function to jointly optimize the CRC and TBCC. As a baseline, we adopt the convolutional codes optimized in Ungerboeck's paper [Ung82], and the CRC is optimized by fixing the convolutional code. We consider the number of memory elements of the convolutional code $\nu = 3$, 5, and 7. Table 4.2 lists the optimized TBCCs and CRCs in octal form. All the parameters are optimized for an SNR of 11 dB. Table 4.2 also provides the FER upper bounds at 11 dB. For the joint optimization, the optimized CRC polynomial is $p(x)$ and the optimized TBCC generator matrix is

$$\left[\begin{array}{cc:c} 1 & 0 & H^2(D)/H^0(D) \\ 0 & 1 & H^1(D)/H^0(D) \end{array}\right]. \tag{4.59}$$

Fig. 4.4 presents analytical upper bounds and simuulation results that compare FERs for

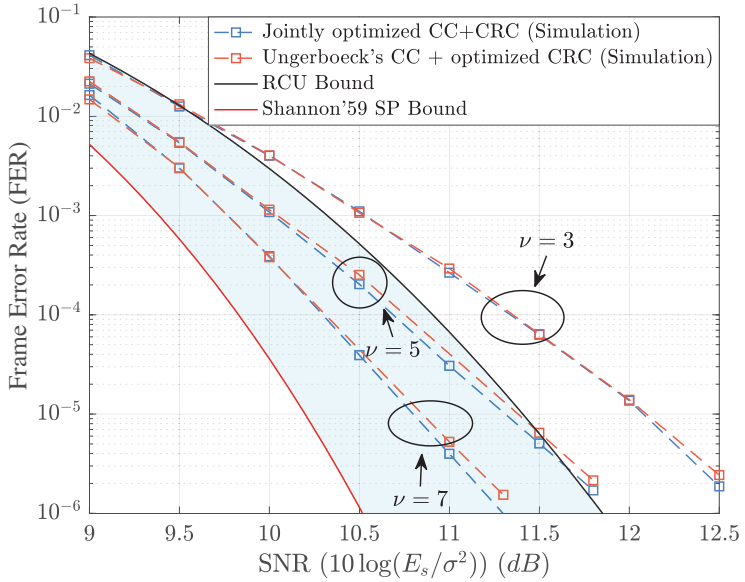


Figure 4.5: The FER curves of the practical CRC-TCM-PAS transmission system that uses MCDM with $\mathcal{C}_{\text{HP}}$. This system takes 87 input bits and generates 65 8-AM symbols.

the optimized convolutional codes to Ungerboeck's convolutional codes for a CRC-TCM-PAS system that assumes an ideal DM. Hence, the system input "messages" are length-64 i.i.d. magnitude symbol sequences according to the PMF $P(\hat{A})$. The magnitude sequences are encoded and modulated by CRC-aided TCM to length-65 8-AM symbol sequences. Simulation results show that maximizing the FER upper bound finds slightly better convolutional codes than those in Ungerboeck's paper. Note that in both cases the FER upper bound was used to optimize the CRC polynomial.

The system uses TBCCs and CRCs from Table 4.2. The receiver uses an ML decoder. Shannon's 1959 sphere packing (SP) bound [Sha59] and Polyanskiy's random coding union (RCU) bound [PPV10] are also shown. Note that the last channel input of the CRC-TCM-PAS system is uniform [WSA22]. When calculating the RCU bound, we assume all channel inputs have the DM output distribution. Fig. 4.5 shows that, when a practical DM is considered, the optimized convolutional codes deliver a slightly better performance than Ungerboeck's convolutional codes. When $\nu = 7$, the FER performance of the CRC-TCM-

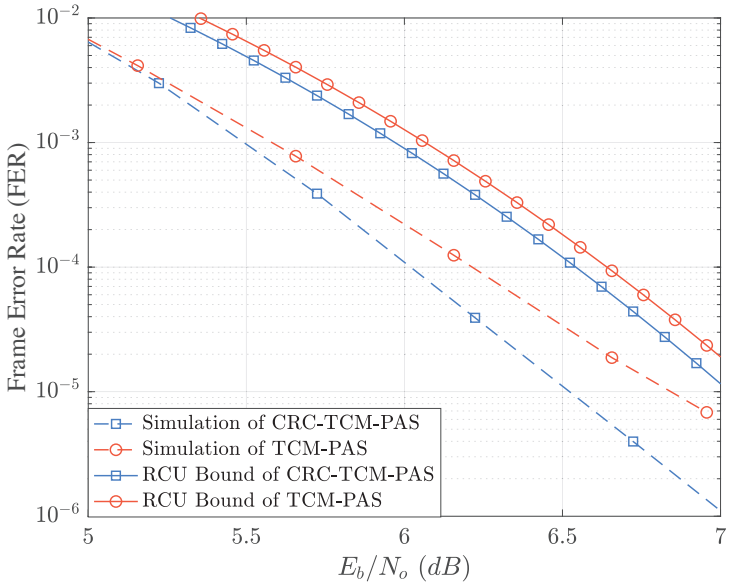
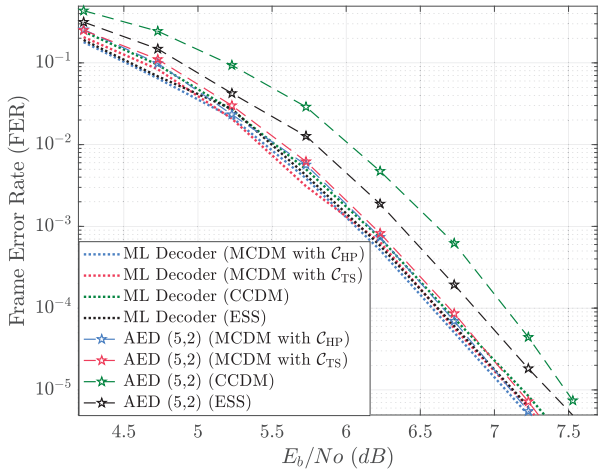


Figure 4.6: The FER curves and RCU bounds of the CRC-TCM-PAS system and TCM-PAS system. The gap between the two curves indicates the contribution of the 2-bit CRC. This system takes 87 input bits and generates 65 8-AM symbols.

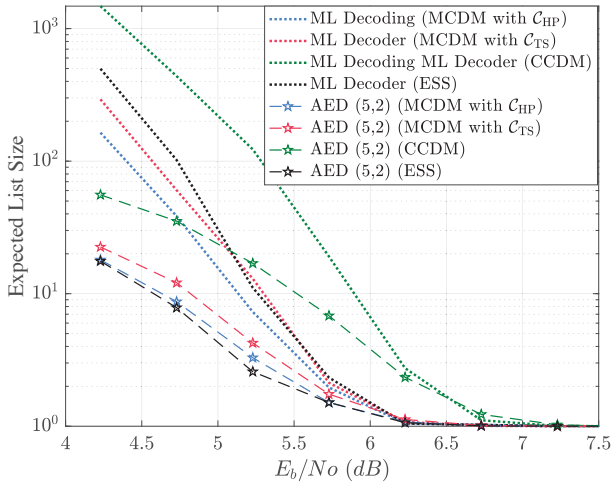
PAS system with optimized CRC and TBCC is better than RCU bound by 0.55 dB at the FER of 10^{-6}. Note that the FER curves from the simulation with the ideal DM in Fig. 4.4 are similar to those with the real DM in Fig. 4.5.

Fig. 4.6 evaluates the contribution of the 2-bit CRC of the CRC-TCM-PAS system with $\nu = 7$ TBCC in Fig. 4.5. We refer to the system without CRC as the TCM-PAS system. Hence, the TCM-PAS system takes 87 input bits and generates 64 8-AM symbols. The FER curve and the RCU bound for the two systems are given in Fig. 4.6. It can be seen that the CRC-TCM-PAS system outperforms the TCM-PAS system by about 0.3 dB at the FER of 10^{-5}, which implies the importance of the 2-bit CRC.

Fig. 4.7 investigates the CRC-TCM-PAS system that uses various DMs and two decoders, ML decoding and a sub-optimal but less complex AED(5,2) decoder. The system in Fig. 4.7 has $k = 96$ input bits and $n = 64$ output symbols, and the transmission rate is 1.5 bits/real channel use. The CRC-aided TCM uses the jointly optimized $\nu = 7$, rate-2/3 TBCC, and

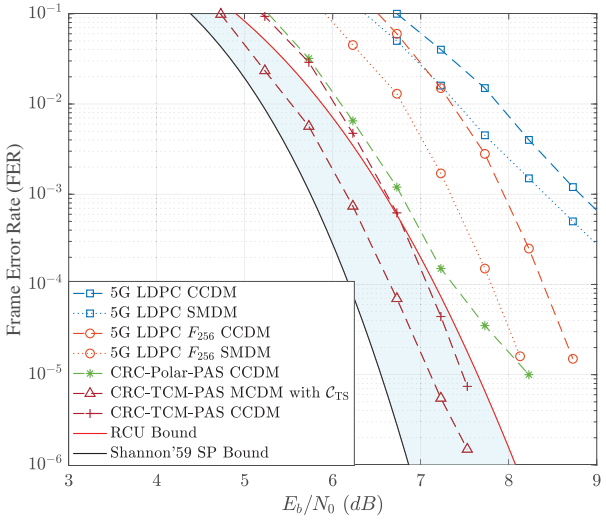


(a)

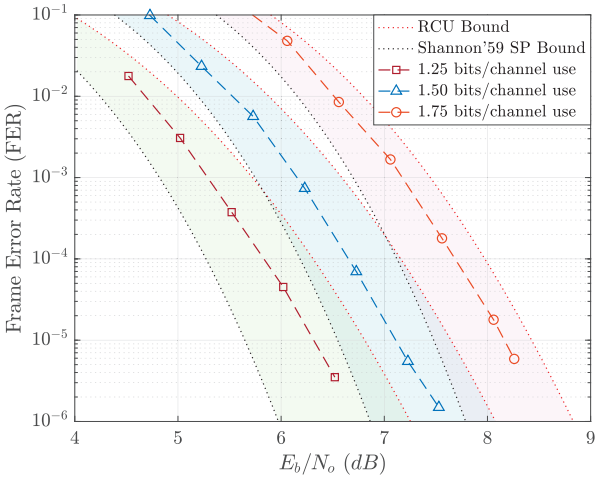


(b)

Figure 4.7: The performance of a CRC-TCM-PAS transmission system with various DMs and decoders. The system takes 96 input bits and generates 64 output symbols. Fig. (a) and (b) give the FER and expected list size, respectively.



(a)



(b)

Figure 4.8: (a): The FER curves of PAS systems with different FECs. All the PAS systems generate 64 8-AM symbols with a transmission rate of 1.5 bit/real channel use. The CRC-TCM-PAS system utilizes CCDM and MCDM with $\mathcal{C}_{\mathrm{TS}}$ as the DM. The decoder of the CRC-TCM-PAS system is AED(5,2) with a maximum list size of 100. (b): The FER curves of CRC-TCM-PAS systems with various rates. The CRC-TCM-PAS systems generate 64 8-AM symbols, with transmission rates of 1.25, 1.5, and 1.75 bit/real channel use, respectively.

the 2-bit CRC in Table 4.2. Fig. 4.7a and 4.7b give the FER performances and expected list sizes, respectively.

We first investigate the performances of the CRC-TCM-PAS systems with various DMs and the ML decoder. The simulation results show that the four considered distribution matchers, i.e., ESS, CCDM, MCDM with $\mathcal{C}_{\text{HP}}$ and $\mathcal{C}_{\text{TS}}$ deliver similar FER performances under ML decoding. However, the CCDM requires more list size than the other three DMs. Fig. 4.7 also presents the FER performance when the AED(5,2) is used. The maximum list size of all 2-States decoders in AED(5,2) is 100. As shown in Fig. 4.7, when AED(5,2) is used as the decoder, the CRC-TCM-PAS system with CCDM delivers the worst FER and largest expected list size. On the other hand, the CRC-TCM-PAS systems that use the MCDM with $\mathcal{C}_{\text{HP}}$ and $\mathcal{C}_{\text{TS}}$ deliver the near-optimal FER performance and outperform the system that uses ESS.

Fig. 4.8a compares the decoding performance of CRC-TCM-PAS system with other PAS systems that use various FEC codes in [CDJ19, Fig.14]. All systems have 96 input bits, and the transmission rate is 1.5 bits/real channel use. For the CRC-TCM-PAS, two distribution matchers are considered, i.e., MCDM with $\mathcal{C}_{\text{TS}}$ and CCDM. The decoder uses AED(5,2) with a maximum list size of 100. The details of other PAS systems are described in [CDJ19]. The simulation results show that the CRC-TCM-PAS system with MCDM delivers the best performance and outperforms the CRC-Polar-PAS system by nearly 1dB. Since the CRC-Polar-PAS system uses CCDM as the distribution matcher, the gain of CRC-TCM-PAS over CRC-Polar-PAS can come from two factors: the choice of DM or the coded modulation scheme. As shown in Fig. 4.8a, with CCDM as the distribution matcher, the CRC-TCM-PAS system still outperforms the CRC-Polar-PAS system but does not perform as well as CRC-TCM-PAS with MCDM. Notably, the CRC-TCM-PAS system doesn't display the error floor of the CRC-Polar-PAS system, which shows an error floor at FER of 10^{-5}. Hence, the gap between the FER curves of the CRC-TCM-PAS with CCDM and the CRC-Polar-PAS with CCDM can be treated as the gain of CRC-TCM code over CRC-Polar code, and the

gap between the FER curves of the CRC-TCM-PAS with CCDM and the MCDM can be treated as the gain of MCDM over CCDM.

The error floor seen in the CRC-Polar-PAS with CCDM could be due to a variety of factors. One factor is the sub-optimality in the decoder. Serial list Viterbi decoding of CRC-TBCC either chooses the ML codeword or reports an erasure with each growing list size. In contrast, successive cancellation list (SCL) decoding of CRC-polar codes sometimes selects non-ML codewords with a fixed list size of 32. Thus, CRC-polar codes decoded using adaptive list sizes will display an error floor if the initial list size is small. If the list size is fixed at a value such as 32, then performance will be limited by that small list size. For polar codes, list sizes larger than 32 are typically not considered because of complexity limitations. The error floor could also be due to a CRC that is too short, not optimized for high SNR, or otherwise sub-optimal.

Fig. 4.8b evaluates the CRC-TCM-PAS system with various transmission rates. We design three CRC-TCM-PAS systems that take 80, 96, and 112 information bits, respectively, and generate 64 8-AM symbols. The resultant transmission rates are 1.25, 1.50, and 1.75 bits/real channel use, respectively. We design the MCDM with $\mathcal{C}_{\text{HP}}$ for all three transmission rates as distribution matcher. All three transmission rates employ the $\nu = 7$ CC and the 2-bit CRC in Table 4.2. AED(5,2) with a maximum list size of 100 is used as the decoder. Fig. 4.8b gives the FER curves, as well as the RCU bound and Shannon's 59 SP bound, of all three transmission rates. The simulation result shows that the FER curves for all three rates lie between the RCU and the SP bound, which indicates excellent decoding performance.

4.6 Conclusion

Shannon's proof of the channel coding theorem [Sha48] generates a random codebook that has an optimal distribution and then performs an expurgation to improve the codebook. The CRC-TCM-PAS system described in this paper follows that paradigm. The DM plays the

role of random codebook generation and the selection of that TCM and CRC polynomials expurgates that code to make it stronger. While there are many recent PAS systems, CRC-TCM-PAS allows the use of the tight FER upper bound derived in this paper for a precise expurgation of the codebook produced by the DM. The TCM and CRC can be jointly selected to optimize FER performance. This also paper proposes a new multi-composition DM (MCDM), which allows codewords with different compositions. The new MCDM provides a significant benefit when decoding complexity is limited. Simulation results show that the optimized CRC-TCM-PAS system with MCDM exceeds the RCU bound for various rates and outperforms the PAS systems with various FEC codes studied in [CDJ19].

CHAPTER 5

Conclusion

This dissertation investigates two topics in channel coding theory: 1) low-complexity decoder design for low-density parity-check (LDPC) codes (Chapters 2 and 3); 2)reliable communication in the short-blocklength regime (Chapter 4). The investigated topics cover the practical and theoretical aspects of channel coding theory. Below we discuss the open problems and direction in each topic.

Chapter 3 describes the optimization of degree-specific weights by training the associated neural network. The open problems in this topic are: 1) why the neural network can be used to train the weights? 2) Why the weights associated with larger node degrees are smaller? For the second problem, we can intuitively answer that the larger a node's degree, the more uncertainty. Thus, the decoder assigns smaller weights to the nodes with a larger degree. It would be great if this observation could be mathematically proved. For the W-RCQ decoder, we first fix the quantizer/dequantizer parameters and then train the neural weight. One interesting direction is jointly optimizing the neural weights and the quantizer/dequantizer parameters.

Chapter 4 presents the CRC-TCM-PAS transmission system. One limitation of this system is that the system only has an excellent performance in the short-blocklength regime. The performance of the CRC-TCM-PAS system degrades for the moderate block length, for example, 200. One conjecture is that longer CRCs could help improve decoding performance. However, the search space of CRC is exponential to the number of CRC bits, and besides, the list decoder's expected list size also grows when the CRC is increased. Hence,

an efficient CRC searching algorithm and low-complexity decoders are desired to improve the performance of the CRC-TCM-PAS system for moderate (or even longer) block length.

REFERENCES

[80212] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, 2012.

[AB13] Rana Ali Amjad and Ing Georg Böcherer. *"Algorithms for simulation of discrete memoryless sources."*. Master's thesis, Technische Universität München, 2013.

[ABS19] A Abotabl, J H Bae, and K Song. "Offset min-sum Optimization for General Decoding Scheduling: A Deep Learning Approach." In *2019 IEEE 90th Vehicular Tech. Conf. (VTC2019-Fall)*, pp. 1–5, September 2019.

[AGG19] Abdelkerim Amari, Sebastiaan Goossens, Yunus Can Gültekin, Olga Vassilieva, Inwoong Kim, Tadashi Ikeuchi, Chigo M Okonkwo, Frans MJ Willems, and Alex Alvarado. "Introducing enumerative sphere shaping for optical communication systems with short blocklengths." *Journal of Lightwave Technology*, **37**(23):5926–5936, 2019.

[AKK19] Rajagopal Anantharaman, Karibasappa Kwadiki, and Vasundara Patel Kerehalli Shankar Rao. "Hardware Implementation Analysis of Min-Sum Decoders." *Advances in Electrical and Electronic Engineering*, **17**(2):179–186, June 2019.

[AV18] M P Ajanya and George Tom Varghese. "Thermometer code to Binary code Converter for Flash ADC - A Review." In *2018 Inter. Conf. on Control, Power, Comm. and Comp. Tech. (ICCPCCT)*, pp. 502–505, March 2018.

[BHP20] Andreas Buchberger, Christian Häger, Henry D Pfister, Laurent Schmalen, and Alexandre Graell i Amat. "Pruning and Quantizing Neural Belief Propagation Decoders." *IEEE Journal on Selected Areas in Communications*, 2020.

[Big84] E. Biglieri. "High-Level Modulation and Coding for Nonlinear Satellite Channels." *IEEE Trans. on Comm.*, **32**(5):616–626, 1984.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." *arXiv preprint arXiv:1308.3432*, 2013.

[Boc17] Georg Böcherer. "Achievable rates for probabilistic shaping." *arXiv preprint arXiv:1707.01134*, 2017.

[BSS15] Georg Böcherer, Fabian Steiner, and Patrick Schulte. "Bandwidth efficient and rate-matched low-density parity-check coded modulation." *IEEE Trans. on comm.*, **63**(12):4651–4665, 2015.

[BSS19] Georg Böcherer, Patrick Schulte, and Fabian Steiner. "Probabilistic shaping and forward error correction for fiber-optic communication systems." *Journal of Lightwave Technology*, **37**(2):230–244, 2019.

[CDJ19] Mustafa Cemil Coşkun, Giuseppe Durisi, Thomas Jerkovits, Gianluigi Liva, William Ryan, Brian Stein, and Fabian Steiner. "Efficient error-correcting codes in the short blocklength regime." *Physical Communication*, **34**:66–79, 2019.

[cls] "UCLA Communications Systems Laboratory." `http://www.seas.ucla.edu/csl/#/publications/published-codes-and-design-tools`.

[CVD15] Tsung-Yi Chen, Kasra Vakilinia, Dariush Divsalar, and Richard D. Wesel. "Protograph-Based Raptor-Like LDPC Codes." *IEEE Transactions on Communications*, **63**(5):1522–1532, 2015.

[DB19] C Deng and S L Bo Yuan. "Reduced-complexity Deep Neural Network-aided Channel Code Decoder: A Case Study for BCH Decoder." In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1468–1472, May 2019.

[DTS21] Jincheng Dai, Kailin Tan, Zhongwei Si, Kai Niu, Mingzhe Chen, H Vincent Poor, and Shuguang Cui. "Learning to decode protograph LDPC codes." *IEEE Journal on Selected Areas in Communications*, 2021.

[DVP13] D Declercq, B Vasic, S K Planjery, and E Li. "Finite Alphabet Iterative Decoders—Part II: Towards Guaranteed Error Correction of LDPC Codes via Iterative Decoder Diversity." *IEEE Trans. Comm.*, **61**(10):4046–4057, October 2013.

[ETS] ETSI EN 302 307. *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2).*

[FGL84] G. Forney, R. Gallager, G. Lang, F. Longstaff, and S. Qureshi. "Efficient Modulation for Band-Limited Channels." *IEEE Journal on Selected Areas in Communications*, **2**(5):632–647, 1984.

[FMK18] Tobias Fehenberger, David S Millar, Toshiaki Koike-Akino, Keisuke Kojima, and Kieran Parsons. "Multiset-partition distribution matching." *IEEE Trans. on Comm.*, **67**(3):1885–1893, 2018.

[For92] G David Forney. "Trellis shaping." *IEEE Trans. on Info. Theory*, **38**(2):281–300, 1992.

[FW99] Christine Fragouli and Richard D Wesel. "Convolutional codes and matrix control theory." In *Proceedings of the 7th Inte. Conf. on Advances in Comm. and Cont., Athens, Greece*. Citeseer, 1999.

[FWS01] C. Fragouli, R.D. Wesel, D. Sommer, and G.P. Fettweis. "Turbo codes with non-uniform constellations." In *ICC 2001. IEEE Inter. Conf. on Comm.. Conference Record (Cat. No.01CH37240)*, volume 1, pp. 70–73 vol.1, 2001.

[Gal62a] Robert Gallager. "Low-density parity-check codes." *IRE Transactions on information theory*, **8**(1):21–28, 1962.

[Gal62b] Robert G. Gallager. "Low-Density Parity-Check Codes." *IRE Trans. on Info. Theo.*, **8**(1):21–28, January 1962.

[Gal68] Robert G Gallager. *Information theory and reliable communication*, volume 588. Springer, 1968.

[Gan00] Feliks Ruvimovich Gantmakher. *The Theory of Matrices, Volume 2*, volume 133. American Mathematical Soc., 2000.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[GBM18] R Ghanaatian, A Balatsoukas-Stimming, T C Müller, M Meidlinger, G Matz, A Teman, and A Burg. "A 588-Gb/s LDPC Decoder Based on Finite-Alphabet Message Passing." *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, **26**(2):329–340, February 2018.

[GEE22] Marvin Geiselhart, Moustafa Ebada, Ahmed Elkelesh, Jannis Clausius, and Stephan ten Brink. "Automorphism Ensemble Decoding of Quasi-Cyclic LDPC Codes by Breaking Graph Symmetries." *arXiv preprint arXiv:2202.00287*, 2022.

[HCM19a] X He, K Cai, and Z Mei. "On Mutual Information-Maximizing Quantized Belief Propagation Decoding of LDPC Codes." In *2019 IEEE Global Comm. Conf. (GLOBECOM)*, pp. 1–6, December 2019.

[HCM19b] Xuan He, Kui Cai, and Zhen Mei. "On Mutual Information-Maximizing Quantized Belief Propagation Decoding of LDPC Codes." In *2019 IEEE Global Comm. Conf. (GLOBECOM)*, pp. 1–6, 2019.

[JF05] J. Zhang and M. P. C. Fossorier. "Shuffled iterative decoding." *IEEE Trans. on Comm.*, **53**(2):209–213, 2005.

[JFD05] Juntan Zhang, M. Fossorier, Daqing Gu, and Jinyun Zhang. "Improved min-sum decoding of LDPC codes using 2-dimensional normalization." In *IEEE Global Comm. Conf., 2005.*, volume 3, pp. 1187–1192, 2005.

[KCH20] Peng Kang, Kui Cai, Xuan He, Shuangyang Li, and Jinhong Yuan. "Generalized Mutual Information-Maximizing Quantized Decoding of LDPC Codes with Layered Scheduling." *arXiv preprint arXiv:2011.13147*, 2020.

[Kie53] J Kiefer. "Sequential Minimax Search for a Maximum." *Proc. Am. Math. Soc.*, **4**(3):502–506, 1953.

[KKY22] Jacob King, Alexandra Kwon, Hengjie Yang, William Ryan, and Richard D Wesel. "CRC-Aided List Decoding of Convolutional and Polar Codes for Short Messages in 5G." *arXiv preprint arXiv:2201.07843*, 2022.

[KP93] F.R. Kschischang and S. Pasupathy. "Optimal nonuniform signaling for Gaussian channels." *IEEE Transactions on Information Theory*, **39**(3):913–929, 1993.

[KY14] B M Kurkoski and H Yagi. "Quantization of Binary-Input Discrete Memoryless Channels." *IEEE Trans. Inf. Theory*, **60**(8):4544–4552, August 2014.

[LB18a] J Lewandowsky and G Bauch. "Information-Optimum LDPC Decoders Based on the Information Bottleneck Method." *IEEE Access*, **6**:4054–4071, 2018.

[LB18b] J Lewandowsky and G Bauch. "Information-Optimum LDPC Decoders Based on the Information Bottleneck Method." *IEEE Access*, **6**:4054–4071, 2018.

[LCH19] Mengke Lian, Fabrizio Carpi, Christian Häger, and Henry D Pfister. "Learned Belief-Propagation Decoding with Simple Scaling and SNR Adaptation." In *2019 IEEE Inter. Symp. on Information Theory (ISIT)*, pp. 161–165, July 2019.

[LFT94] Rajiv Laroia, Nariman Farvardin, and Steven A Tretter. "On optimal shaping of multidimensional constellations." *IEEE Trans. on Info. Theory*, **40**(4):1044–1056, 1994.

[LG17] L Lugosch and W J Gross. "Neural offset min-sum decoding." In *2017 IEEE Inter. Symp. on Info. Theory (ISIT)*, pp. 1361–1365, June 2017.

[LG18] L Lugosch and W J Gross. "Learning from the Syndrome." In *2018 52nd Asilomar Conf. on Signals, Systems, and Computers*, pp. 594–598, October 2018.

[LM05] S. Landner and O. Milenkovic. "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes." In *2005 International Conference on Wireless Networks, Communications and Mobile Computing*, volume 1, pp. 630–635, 2005.

[LSW18] F Liang, C Shen, and F Wu. "An Iterative BP-CNN Architecture for Channel Decoding." *IEEE J. Sel. Top. Signal Process.*, **12**(1):144–159, February 2018.

[LT05a] J K Lee and J Thorpe. "Memory-efficient decoding of LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, pp. 459–463, September 2005.

[LT05b] J K Lee and J Thorpe. "Memory-efficient decoding of LDPC codes." In *Proc. Int. Symp. on Info. Theory, ISIT 2005.*, pp. 459–463, September 2005.

[LYD19] Ethan Liang, Hengjie Yang, Dariush Divsalar, and Richard D. Wesel. "List-Decoded Tail-Biting Convolutional Codes with Distance-Spectrum Optimal CRCS for 5G." In *2019 IEEE Glob. Comm. Conf. (GLOBECOM)*, pp. 1–6, 2019.

[LZJ18] W Lyu, Z Zhang, C Jiao, K Qin, and H Zhang. "Performance Evaluation of Channel Decoding with Deep Neural Networks." In *2018 IEEE Inter. Conf. on Comm. (ICC)*, pp. 1–6, May 2018.

[LZS17] Yanhuan Liu, Chun Zhang, Pengcheng Song, and Hanjun Jiang. "A high-performance FPGA-based LDPC decoder for solid-state drives." In *2017 IEEE 60th Inter. Midwest Symp. on Circuits and Systems (MWSCAS)*, pp. 1232–1235, August 2017.

[MBB15] M Meidlinger, A Balatsoukas-Stimming, A Burg, and G Matz. "Quantized message passing for LDPC codes." In *2015 49th Asilomar Conference on Signals, Systems and Computers*, pp. 1606–1610, November 2015.

[Mik12] Tomáš Mikolov. *Statistical language models based on neural networks.* PhD thesis, Brno University of Technology, 2012.

[MM17] M Meidlinger and G Matz. "On irregular LDPC codes with quantized message passing decoding." In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, July 2017.

[MMB20] M Meidlinger, G Matz, and A Burg. "Design and Decoding of Irregular LDPC Codes Based on Discrete Message Passing." *IEEE Trans. Commun.*, **68**(3):1329–1343, March 2020.

[NBB16a] E Nachmani, Y Be'ery, and D Burshtein. "Learning to decode linear codes using deep learning." In *2016 54th Annual Allerton Conference on Communication, Control, and Computing*, pp. 341–346, September 2016.

[NBB16b] Eliya Nachmani, Yair Be'ery, and David Burshtein. "Learning to decode linear codes using deep learning." In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346, 2016.

[NMB17] Eliya Nachmani, Elad Marciano, David Burshtein, and Yair Be'ery. "RNN Decoding of Linear Block Codes." *CoRR*, **abs/1702.07560**, 2017.

[NML18] E Nachmani, E Marciano, L Lugosch, W J Gross, D Burshtein, and Y Be'ery. "Deep Learning Methods for Improved Decoding of Linear Codes." *IEEE J. Sel. Top. Sig. Pro.*, **12**(1):119–131, February 2018.

[NWH21] Jonathan Nguyen, Linfang Wang, Chester Hulse, Sahil Dani, Amaael Antonini, Todd Chauvin, Divsalar Dariush, and Richard Wesel. "Neural Normalized Min-Sum Message-Passing vs. Viterbi Decoding for the CCSDS Line Product Code." *arXiv preprint arXiv:2111.07959*, 2021.

[PDD13a] S K Planjery, D Declercq, L Danjean, and B Vasic. "Finite Alphabet Iterative Decoders—Part I: Decoding Beyond Belief Propagation on the Binary Symmetric Channel." *IEEE Trans. Comm.*, **61**(10):4033–4045, October 2013.

[PDD13b] Shiva Kumar Planjery, David Declercq, Ludovic Danjean, and Bane Vasic. "Finite Alphabet Iterative Decoders—Part I: Decoding Beyond Belief Propagation on the Binary Symmetric Channel." *IEEE Trans. on Comm.*, **61**(10):4033–4045, 2013.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1310–III–1318, June 2013.

[PPV10] Yury Polyanskiy, H. Vincent Poor, and Sergio Verdu. "Channel Coding Rate in the Finite Blocklength Regime." *IEEE Trans. on Info. Theory*, **56**(5):2307–2359, 2010.

[PX19a] Marcin Pikus and Wen Xu. "Arithmetic coding based multi-composition codes for bit-level distribution matching." In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6. IEEE, 2019.

[PX19b] Marcin Pikus and Wen Xu. "Arithmetic Coding Based Multi-Composition Codes for Bit-Level Distribution Matching." In *2019 IEEE Wireless Comm. and Networking Conf. (WCNC)*, pp. 1–6, 2019.

[RK16] F J C Romero and B M Kurkoski. "LDPC Decoding Mappings That Maximize Mutual Information." *IEEE J. Sel. Areas Commun.*, **34**(9):2391–2401, September 2016.

[RU01] T J Richardson and R L Urbanke. "The capacity of low-density parity-check codes under message-passing decoding." *IEEE Trans. on information*, 2001.

[SB15] Patrick Schulte and Georg Böcherer. "Constant composition distribution matching." *IEEE Trans. on Info. Theory*, **62**(1):430–434, 2015.

[SBW20a] M Stark, G Bauch, L Wang, and R D Wesel. "Information Bottleneck Decoding of Rate-Compatible 5G-LDPC Codes." In *ICC 2020 - 2020 IEEE Inter. Conf. on Comm. (ICC)*, pp. 1–6, June 2020.

[SBW20b] Maximilian Stark, Gerhard Bauch, Linfang Wang, and Richard D Wesel. "Information bottleneck decoding of rate-compatible 5G-LDPC codes." In *ICC 2020-2020 IEEE Inte. Conf. on Comm. (ICC)*, pp. 1–6. IEEE, 2020.

[Sch20] Patrick Schulte. *Algorithms for Distribution Matching*. PhD thesis, Technische Universität München, May 2020.

[SFR01] S.-Y. Chung, G D Forney, T J Richardson, and R Urbanke. "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit." *IEEE Commun. Lett.*, **5**(2):58–60, February 2001.

[SH90] Frank K Soong and Eng-Fong Huang. "A fast tree-trellis search for finding the N-best sentence hypotheses in continuous speech recognition." *The Journal of the Acoustical Society of America*, **87**(S1):S105–S106, 1990.

[SH16] Ahmed M Sadek and Aziza I Hussein. "Flexible FPGA implementation of Min-Sum decoding algorithm for regular LDPC codes." In *2016 11th Inter. Conf. on Comp. Engi. Systems (ICCES)*, pp. 286–292, December 2016.

[Sha48] Claude Shannon. "A mathematical theory of communication." *The Bell system technical journal*, **27**(3):379–423, 1948.

[Sha59] Claude E Shannon. "Probability of error for optimal codes in a Gaussian channel." *Bell System Tech. Jour.*, **38**(3):611–656, 1959.

[SLB18] M Stark, J Lewandowsky, and G Bauch. "Information-Optimum LDPC Decoders with Message Alignment for Irregular Codes." In *2018 IEEE Glob. Comm. Conf. (GLOBECOM)*, pp. 1–6, December 2018.

[SLF03] R.Y. Shao, Shu Lin, and M.P.C. Fossorier. "Two decoding algorithms for tail-biting codes." *IEEE Trans. on Comm.*, **51**(10):1658–1665, 2003.

[SS94] Nambirajan Seshadri and Cs Sundberg. "List Viterbi decoding algorithms with applications." *IEEE trans. on comm.*, **42**(234):313–323, 1994.

[SS19] Patrick Schulte and Fabian Steiner. "Divergence-optimal fixed-to-fixed length distribution matching with shell mapping." *IEEE Wireless Communications Letters*, **8**(2):620–623, 2019.

[Sta21] Maximilian Stark. *Machine learning for reliable communication under coarse quantization*. PhD thesis, Technische Universität Hamburg, 2021.

[SWB20] M Stark, L Wang, G Bauch, and R D Wesel. "Decoding Rate-Compatible 5G-LDPC Codes With Coarse Quantization Using the Information Bottleneck Method." *IEEE Open Journal of the Communications Society*, **1**:646–660, 2020.

[TJ06] MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 2006.

[TV13] I. Tal and A. Vardy. "How to Construct Polar Codes." *IEEE Trans. on Info. Theo.*, **59**(10):6562–6582, 2013.

[TWC21a] Caleb Terrill, Linfang Wang, Sean Chen, Chester Hulse, Calvin Kuo, Richard Wesel, and Dariush Divsalar. "FPGA Implementations of Layered MinSum LDPC Decoders Using RCQ Message Passing." *arXiv preprint arXiv:2104.09480*, 2021.

[TWC21b] Caleb Terrill, Linfang Wang, Sean Chen, Chester Hulse, Calvin Kuo, Richard Wesel, and Dariush Divsalar. "FPGA Implementations of Layered MinSum LDPC Decoders Using RCQ Message Passing." In *2021 IEEE Global Comm. Conf. (GLOBECOM)*, pp. 1–6, 2021.

[Ung82] Gottfried Ungerboeck. "Channel coding with multilevel/phase signals." *IEEE Trans. on Info. Theory*, **28**(1):55–67, 1982.

[WBR01] Christian Weiß, Christian Bettstetter, and Sven Riedel. "Code construction and decoding of parallel concatenated tail-biting codes." *IEEE Trans. on Info. Theory*, **47**(1):366–386, 2001.

[WCN21] Linfang Wang, Sean Chen, Jonathan Nguyen, Divsalar Dariush, and Richard Wesel. "Neural-Network-Optimized Degree-Specific Weights for LDPC MinSum Decoding." In *2021 11th International Symposium on Topics in Coding (ISTC)*, pp. 1–5, 2021.

[WCS11] Jiadong Wang, Thomas Courtade, Hari Shankar, and Richard D. Wesel. "Soft Information for LDPC Decoding in Flash: Mutual-Information Optimized Quantization." In *2011 IEEE Glob. Tele. Conf. (GLOBECOM)*, pp. 1–6, 2011.

[Wes04] R.D. Wesel. "Reduced-state representations for trellis codes using constellation symmetry." *IEEE Trans. on Comm.*, **52**(8):1302–1310, 2004.

[WJZ18] X Wu, M Jiang, and C Zhao. "Decoding Optimization for 5G LDPC Codes by Machine Learning." *IEEE Access*, **6**:50179–50186, 2018.

[WLW19] Nathan Wong, Ethan Liang, Haobo Wang, Sudarsan V. S. Ranganathan, and Richard D. Wesel. "Decoding Flash Memory with Progressive Reads and Independent vs. Joint Encoding of Bits in a Cell." In *2019 IEEE Glob. Comm. Conf. (GLOBECOM)*, pp. 1–6, 2019.

[WSA22] Linfang Wang, Dan Song, Felipe Areces, and Richard D. Wesel. "Achieving Short-Blocklength RCU bound via CRC List Decoding of TCM with Probabilistic Shaping." In *2022 International Conference on Communications (ICC)*, 2022.

[WTS22] Linfang Wang, Caleb Terrill, Maximilian Stark, Zongwang Li, Sean Chen, Chester Hulse, Calvin Kuo, Richard D. Wesel, Gerhard Bauch, and Rekha Pitchumani. "Reconstruction-Computation-Quantization (RCQ): A Paradigm for Low Bit Width LDPC Decoding." *IEEE Transactions on Communications*, **70**(4):2213–2226, 2022.

[WVC14] Jiadong Wang, Kasra Vakilinia, Tsung Chen, Thomas Courtade, Guiqiang Dong, Tong Zhang, Hari Shankar, and Richard Wesel. "Enhanced Precision Through Multiple Reads for LDPC Decoding in Flash Memories." *IEEE J. Sel. Areas in Comm.*, **32**(5):880–891, 2014.

[WWF20] Q Wang, S Wang, H Fang, L Chen, L Chen, and Y Guo. "A Model-Driven Deep Learning Method for Normalized Min-Sum LDPC Decoding." In *2020 IEEE Inter. Conf. on Com. Workshops*, pp. 1–6, June 2020.

[WWS20a] L Wang, R D Wesel, M Stark, and G Bauch. "A Reconstruction-Computation-Quantization (RCQ) Approach to Node Operations in LDPC Decoding." In *GLOBECOM 2020 - 2020 IEEE Glob. Comm. Conf.*, pp. 1–6, December 2020.

[WWS20b] L Wang, R D Wesel, M Stark, and G Bauch. "A Reconstruction-Computation-Quantization (RCQ) Approach to Node Operations in LDPC Decoding." In *2020 IEEE Global Comm. Conf. (GLOBECOM)*, pp. 1–6, December 2020.

[XCB21] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Lanfei Wang, Zhengsu Chen, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. "Weight-sharing neural architecture search: A battle to shrink the optimization gap." *ACM Computing Surveys (CSUR)*, **54**(9):1–37, 2021.

[XVT19a] X Xiao, B Vasic, R Tandon, and S Lin. "Finite Alphabet Iterative Decoding of LDPC Codes with Coarsely Quantized Neural Networks." In *2019 IEEE Glob. Comm. Conf. (GLOBECOM)*, pp. 1–6, December 2019.

[XVT19b] X Xiao, B Vasic, R Tandon, and S Lin. "Finite Alphabet Iterative Decoding of LDPC Codes with Coarsely Quantized Neural Networks." In *2019 IEEE Global Comm. Conf. (GLOBECOM)*, pp. 1–6, December 2019.

[XVT20a] X Xiao, B Vasić, R Tandon, and S Lin. "Designing Finite Alphabet Iterative Decoders of LDPC Codes Via Recurrent Quantized Neural Networks." *IEEE Trans. Commun.*, **68**(7):3963–3974, July 2020.

[XVT20b] X Xiao, B Vasić, R Tandon, and S Lin. "Designing Finite Alphabet Iterative Decoders of LDPC Codes Via Recurrent Quantized Neural Networks." *IEEE Trans. Commun.*, **68**(7):3963–3974, July 2020.

[XWS21] Derek Xiao, Linfang Wang, Dan Song, and Richard D Wesel. "Finite-Support Capacity-Approaching Distributions for AWGN Channels." In *2020 IEEE Information Theory Workshop (ITW)*, pp. 1–5. IEEE, 2021.

[YLP22] Hengjie Yang, Ethan Liang, Minghao Pan, and Richard D. Wesel. "CRC-Aided List Decoding of Convolutional Codes in the Short Blocklength Regime." *IEEE Transactions on Information Theory*, **68**(6):3744–3766, 2022.

[YLY19] Hengjie Yang, Ethan Liang, Hanwen Yao, Alexander Vardy, Dariush Divsalar, and Richard D Wesel. "A List-Decoding Approach to Low-Complexity Soft Maximum-Likelihood Decoding of Cyclic Codes." In *2019 IEEE Glob. Comm. Conf. (GLOBECOM)*, pp. 1–6. IEEE, 2019.

[YRW18] Hengjie Yang, Sudarsan VS Ranganathan, and Richard D Wesel. "Serial list Viterbi decoding with CRC: Managing errors, erasures, and complexity." In *2018 IEEE Glob. Comm. Conf. (GLOBECOM)*, pp. 1–6. IEEE, 2018.

[ZDN06] Zhengya Zhang, Lara Dolecek, Borivoje Nikolic, Venkat Anantharam, and Martin Wainwright. "GEN03-6: Investigation of Error Floors of Structured Low-Density Parity-Check Codes by Hardware Emulation." In *2006 IEEE Glob. Comm. Conf. (Globecom)*, pp. 1–6, 2006.

[ZFG06] J. Zhang, M. Fossorier, D. Gu, and J. Zhang. "Two-dimensional correction for min-sum decoding of irregular LDPC codes." *IEEE Communications Letters*, **10**(3):180–182, 2006.

[ZS14] X Zhang and P H Siegel. "Quantized Iterative Message Passing Decoders with Low Error Floor for LDPC Codes." *IEEE Trans. Commun.*, **62**(1):1–14, January 2014.