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 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 Engineering
University of California, Los Angeles, 2023
Professor Richard D. Wesel, Chair

(Abstract omitted for brevity)

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) ...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

(Acknowledgments omitted for brevity.)

VITA

1974–1975	Campus computer center "User Services" programmer and consultant, Stanford Center for Information Processing, Stanford University, Stanford, California.
1974–1975	Programmer, Housing Office, Stanford University. Designed a major software system for assigning students to on-campus housing. With some later improvements, it is still in use.
1975	B.S. (Mathematics) and A.B. (Music), Stanford University.
1977	M.A. (Music), UCLA, Los Angeles, California.
1977–1979	Teaching Assistant, Computer Science Department, UCLA. Taught sections of Engineering 10 (beginning computer programming course) under direction of Professor Leon Levine. During summer 1979, taught a beginning programming course as part of the Freshman Summer Program.
1979	M.S. (Computer Science), UCLA.
1979–1980	Teaching Assistant, Computer Science Department, UCLA.
1980–1981	Research Assistant, Computer Science Department, UCLA.
1981–present	Programmer/Analyst, Computer Science Department, UCLA.

PUBLICATIONS

MADHOUS Reference Manual. Stanford University, Dean of Student Affairs (Residential Education Division), 1978. Technical documentation for the MADHOUS software system used to assign students to on-campus housing.

CHAPTER 1

Introduction

For text, let's use the first words out of the ispell dictionary.

CHAPTER 2

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

2.1 Introduction

Low-Density Parity-Check (LDPC) codes [Gal62] 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.

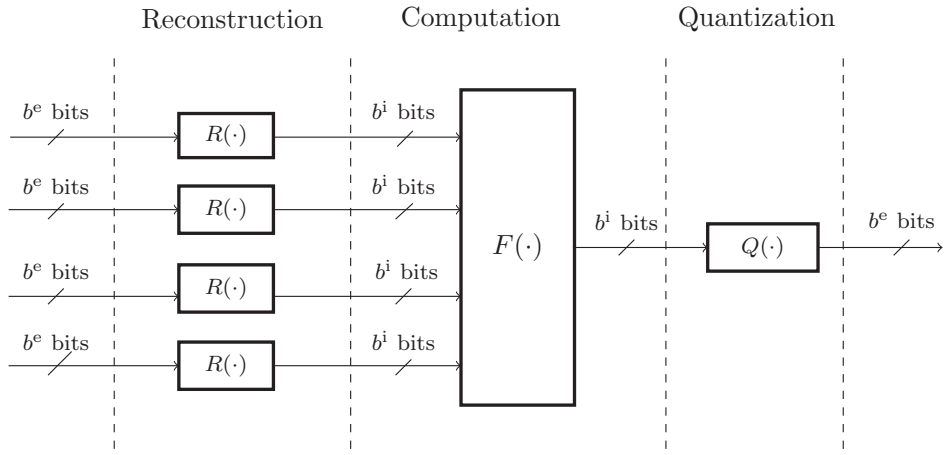


Figure 2.1: Illustration of a generalized RCQ unit which consists of three modules: *Reconstruction* that maps a b^{e}-bit value to a b^{i}-bit value, *Computation* that performs arithmetic operations, and *Quantization* that quantizes a b^{i}-bit value to a b^{e}-bit value.

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

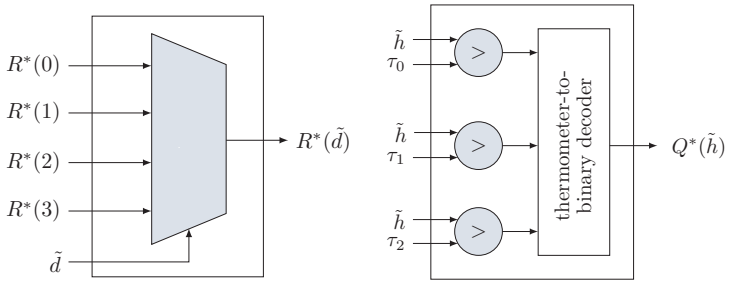
denote $b^{\mathrm{i,v}}$ by b^{v}. The CN min operation computes the XOR of the sign bits and finds the minimum of the extrinsic magnitudes. For a symmetric channel, the min operation can be computed by manipulating the external messages, because the external message delivers the *relative LLR meaning* 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, LB18a, 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 [LB18a], 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

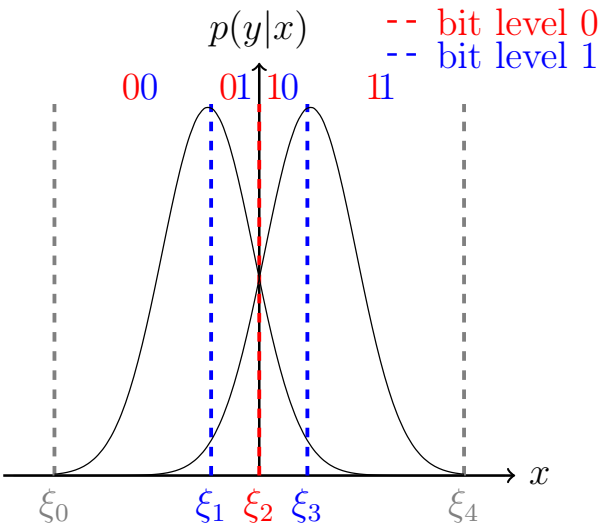


Figure 2.3: Given the conditional probability $p(y|x)$ of symmetric BI-AWGN channel, HDQ sequentially quantizing A/D output w into a 2-bit message by first finding the index ξ_2, then the indices ξ_1 and ξ_3.

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

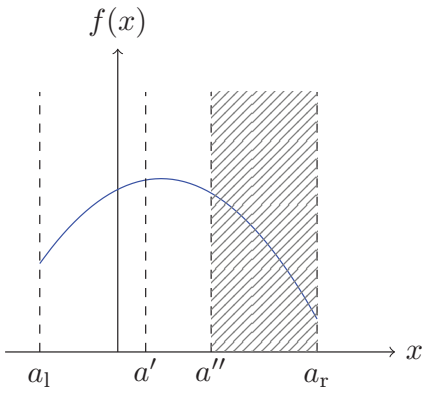


Figure 2.4: Illustration of one iteration of golden-section search for finding maximum point of $f(x)$ in the interval $[a_\mathrm{l}, a_\mathrm{r}]$. $a' = a_\mathrm{r} - \frac{a_\mathrm{r}-a_\mathrm{l}}{\gamma}$ and $a'' = a_\mathrm{l} + \frac{a_\mathrm{r}-a_\mathrm{l}}{\gamma}$. Because $f(a'') <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

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

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

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

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 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

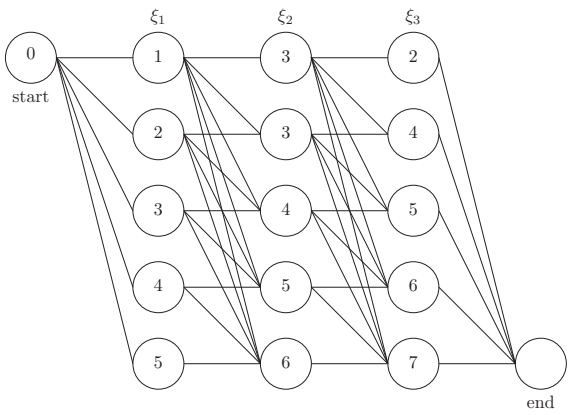


Figure 2.5: A trellis whose paths represent all 2-bit quantizers for a BI-DMC with 8 outputs. The vertices in column i are possible values for i^{th} threshold ξ_i. Each branch in the trellis identifies a quantization region.

that is proportional to:

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

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

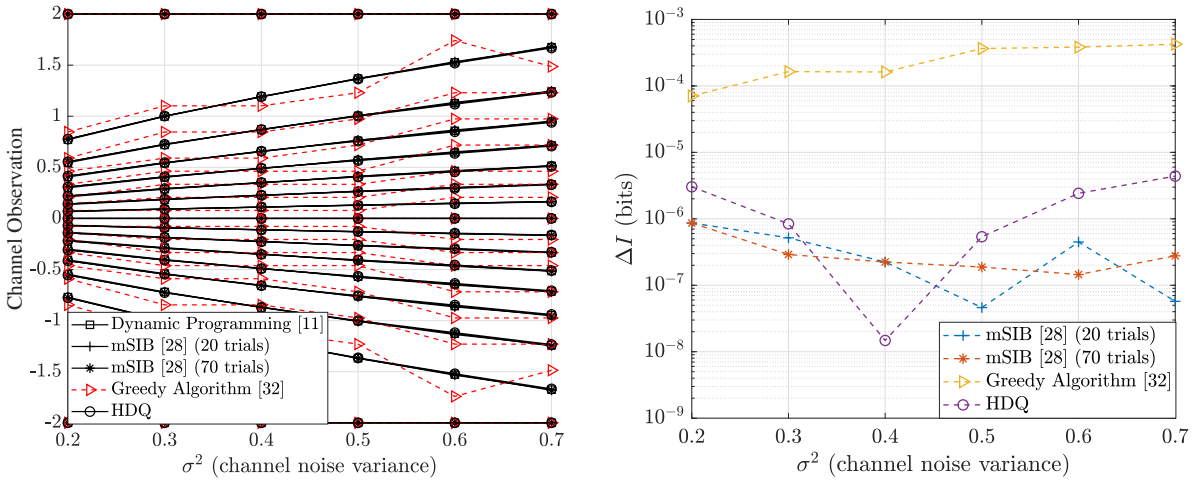
$$\leq \log_\gamma(B) + 2\log_\gamma\left(\frac{B}{2}\right) + \ldots + 2^{b-1}\log_\gamma\left(\frac{B}{2^{b-1}}\right) \tag{2.24}$$

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

$B_{j,i}$ 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



(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 $B = 2000$.

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 [LB18a]. 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 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:

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

where $I^{\mathrm{dp}}(X; D)$ and $I^{\mathrm{sub}}(X; D)$ 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 [LB18a], $|\mathbf{T}|=2^{m(d_c-1)}$ 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

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

$P^c_{out}(X,\mathbf{T})$ 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 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.:

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

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.

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 $\lambda(x) = \sum_{i=2}^{d_{v,max}} \lambda_i x^{i-1}$, $P^v_{out}(X,\mathbf{T})$ is given by:

$$P^v_{out}(X,\mathbf{T}) = P^{ch}(X,T) \boxdot \sum_{i=2}^{d_{v,max}} \lambda_i P^c(X,T)^{\boxdot(d_v-1)}. \tag{2.36}$$

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

$$\frac{E_b}{N_o}^* = \inf\left\{\frac{E_b}{N_o} : I^{(I_T)}\left(\frac{E_b}{N_o}\right) > 1 - \epsilon\right\}, \tag{2.37}$$

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

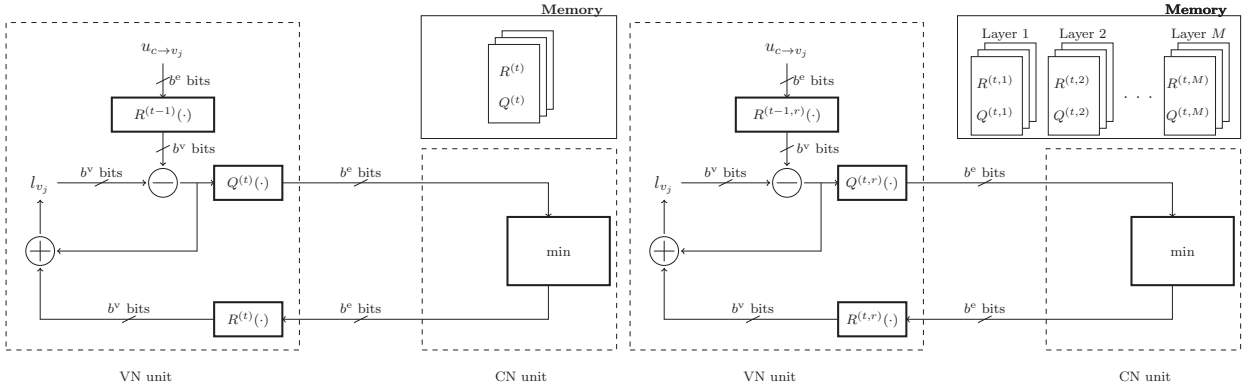
QC-LDPC codes are structured LDPC codes characterized by a parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ which consists of square sub-matrices with size S, which are either the all-zeros matrix or a cyclic permutation of the identity matrix. These cyclic permutations are also called circulants 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:

$$l_{v_{j'}} \leftarrow l_{v_{j'}} - u^{(t-1)}_{c_i \to v_{j'}} \quad \forall j' \in \mathcal{N}(c_i), \tag{2.38}$$

$$u^{(t)}_{c_i \to v_{j'}} = \left(\prod_{\tilde{j} \in \mathcal{N}(c_i)/\{j'\}} \operatorname{sign}(l_{v_{\tilde{j}}}) \right) \times \min_{\tilde{j} \in \mathcal{N}(c_i)/\{j'\}} |l_{v_{\tilde{j}}}|, \quad \forall j' \in \mathcal{N}(c_i), \tag{2.39}$$

$$l_{v_{j'}} \leftarrow l_{v_{j'}} + u^{(t)}_{c_i \to v_{j'}} \quad \forall j' \in \mathcal{N}(c_i). \tag{2.40}$$



(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.

$\mathcal{N}(c_i)$ 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 (2.38), (2.39), and (2.40) 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 $H_{i,j} = H_{i',j'} = 1$:

$$P^{(t)}_{(c_i,v_j)}(X, D) = P^{(t)}_{(c_{i'},v_{j'})}(X, D). \tag{2.41}$$

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

$$\log \frac{P^{(t)}_{(c_i,v_j)}(0, d)}{P^{(t)}_{(c_i,v_j)}(1, d)} = \log \frac{P^{(t)}_{(c_{i'},v_{j'})}(0, d)}{P^{(t)}_{(c_{i'},v_{j'})}(1, d)},\ d \in \{0, ..., 2^{b^e} - 1\}. \tag{2.42}$$

The msRCQ decoder with a flooding schedule obeys (2.41) and (2.42) 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:

$$l^{(t)}_{v_j \to c_i} = l^{(ch)}_{v_j} + \sum_{\{i' | i' \in \mathcal{N}(v_j), i' < i\}} u^{(t)}_{c_{i'} \to v_j} + \sum_{\{i' | i' \in \mathcal{N}(v_j), i' > i\}} u^{(t-1)}_{c_{i'} \to v_j}, \tag{2.43}$$

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 (2.41) and (2.42) no longer 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:

$$P_{XD^{\mathrm{ch}}}(x, d) = P_D(d)\frac{e^{(1-x)l_d}}{e^{l_d}+1},\ X \in \{0, 1\},\ l_d \in \mathcal{D}^{\mathrm{ch}}. \tag{2.44}$$

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:

$$\mathcal{R}(c) = \{r | H_{\mathrm{p}}(r, c) \neq 0\}, \quad \mathcal{C}(r) = \{c | H_{\mathrm{p}}(r, c) \neq 0\}. \tag{2.45}$$

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:

$$\begin{aligned} P_{\mathrm{v}}^{(t,r,c)}(X, \mathbf{D}) = P(X, D^{\mathrm{ch}}) \boxdot \left(\boxdot_{\substack{k \in \mathcal{R}(c) \\ k<r}} P^{(t,k)}(X, D^{\mathrm{c}}) \right) \\ \boxdot \left(\boxdot_{\substack{k \in \mathcal{R}(c) \\ k>r}} P^{(t-1,k)}(X, D^{\mathrm{c}}) \right), \end{aligned} \tag{2.46}$$

$\boxdot$ is defined as follows:

$$P(x,[d_1,d_2]) = P(X_1,D_1) \boxdot P(X_2,D_2) \tag{2.47}$$

$$\triangleq \frac{1}{P_X(x)} P_{X_1D_1}(x,d_1) P_{X_2D_2}(x,d_2), \tag{2.48}$$

$x \in \{0,1\},\ d_1,d_2 \in \{0,\ldots,2^{b^{\mathrm{e}}}-1\}$. 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 (2.46).

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:

$$\tilde{P}_{\mathrm{v}}^{(t,r)} = \left\{ \frac{1}{|\mathcal{C}(r)|} P_{\mathrm{v}}^{(t,r,c)}(X,\mathbf{D}) \mid c \in \mathcal{C}(r) \right\} \tag{2.49}$$

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:

$$\left[P^{(t,r)}(X,D^{\mathrm{v}}), Q^{(t,r)}(\cdot)\right] = \texttt{HDQ}\left(\tilde{P}_{\mathrm{v}}^{(t,r)}, 2^{b^{\mathrm{e}}}\right), \tag{2.50}$$

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:

$$l_{\mathrm{v}}^{(t,r)}(d) = \log \frac{P_{XD^{\mathrm{v}}}^{(t,r)}(0,d)}{P_{XD^{\mathrm{v}}}^{(t,r)}(1,d)}, \quad d = 0,...,2^{b^e}-1. \tag{2.51}$$

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

$$\begin{aligned} l_{\mathrm{MS}}^{\mathrm{out}} = &\min\left(|l_{\mathrm{v}}^{(t,r)}(d_1)|, |l_{\mathrm{v}}^{(t,r)}(d_2))|\right) \\ &\times \mathrm{sgn}(l_{\mathrm{v}}^{(t,r)}(d_1)) \times \mathrm{sgn}(l_{\mathrm{v}}^{(t,r)}(d_2)). \end{aligned} \tag{2.52}$$

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

$$l_{\mathrm{MS}}^{\mathrm{out}} = \log \frac{P_{XD^{\mathrm{v}}}^{(t,r)}(0, d^{\mathrm{out}})}{P_{XD^{\mathrm{v}}}^{(t,r)}(1, d^{\mathrm{out}})}. \tag{2.53}$$

At the check node output, l_{MS}^{out} will be assigned the label $d^{out} \in \mathcal{D}^{v}$ that satisfies (2.53). However, the LLR meaning associated with that d^{out} must be adjusted.

Define the follow function:

$$d^{\mathrm{out}} = \mathtt{MS}(d_1, d_2), \tag{2.54}$$

where $d^{\mathrm{out}}, d_1, d_2 \in \mathcal{D}^{\mathrm{v}}$. (2.54) holds if and only if (2.52) and (2.53) and are both satisfied.

Define the binary operation $\circledast$ by:

$$\tilde{P}_{XD}(x, d) = P(X_1, D_1) \circledast P(X_2, D_2) \tag{2.55}$$

$$\triangleq \sum_{\substack{d_1, d_2 : \mathtt{MS}(d_1, d_2) = d \\ x_1, x_2 : x_1 \bigoplus x_2 = x}} P_{X_1 D_1}(x_1, d_1) P_{X_2 D_2}(x_2, d_2). \tag{2.56}$$

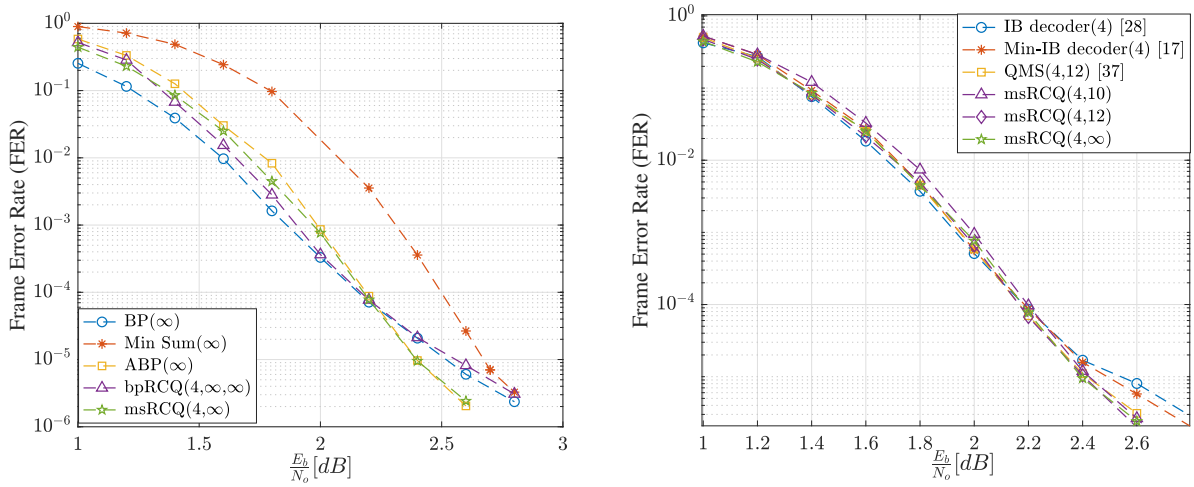
The joint PMF between code bit and external CN message in layer r and iteration t can be updated by:

$$P^{(t,r)}(X, D^{\mathrm{c}}) = P^{(t,r)}(X, D^{\mathrm{v}}) \circledast ... \circledast P^{(t,r)}(X, D^{\mathrm{v}}) \tag{2.57}$$

$$\triangleq P^{(t,r)}(X, D^{\mathrm{v}})^{\circledast(|\mathcal{C}(r)|-1)}. \tag{2.58}$$

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

$$R^{(t,r)}(d) = \log \frac{P_{XD^{\mathrm{c}}}^{(t,r)}(0, d)}{p_{XD^{\mathrm{c}}}^{(t,r)}(1, d)}, \ d \in \{0, ..., 2^{b^{\mathrm{e}}} - 1\}. \tag{2.59}$$



(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.

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:

$$\frac{E_b}{N_o}^* = \inf\left\{\frac{E_b}{N_o} : I^{(I_T,r)}\left(\frac{E_b}{N_o}\right) > 1-\epsilon, \forall r \in [1, M]\right\}. \tag{2.60}$$

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:

$$\lambda(x) = 0.2588x + 0.3140x^2 + 0.0465x^3 + 0.3837x^{10}, \tag{2.61}$$

$$\rho(x) = 0.8140x^6 + 0.1860x^7. \tag{2.62}$$

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 ∞ 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.

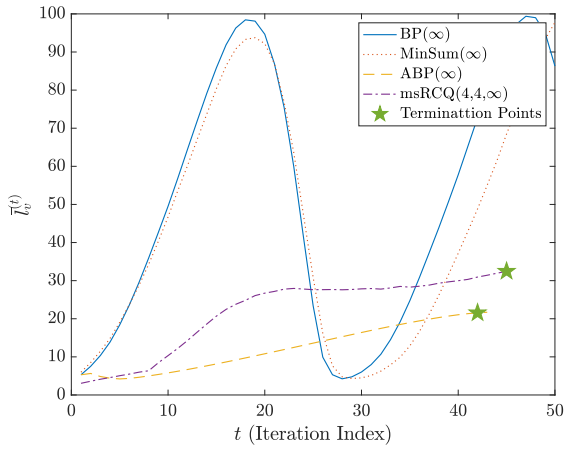


Figure 2.10: Average magnitudes of $l_v^{(t)}$ vs. iteration for BP, ABP, Min Sumand msRCQ for Fig. 6a simulation at $\frac{E_b}{N_o} = 2.6$ dB.

For the example of a degree-3 CN, (2.63) 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:

$$l_t = \log \frac{\sum_{\{(i,j)|t=MS(i,j)\}} P(0,i)P(0,j) + P(1,i)P(1,j)}{\sum_{\{(i,j)|t=MS(i,j)\}} P(1,i)P(0,j) + P(0,i)P(1,j)}. \tag{2.63}$$

Note that the boxplus operation is computed as follows :

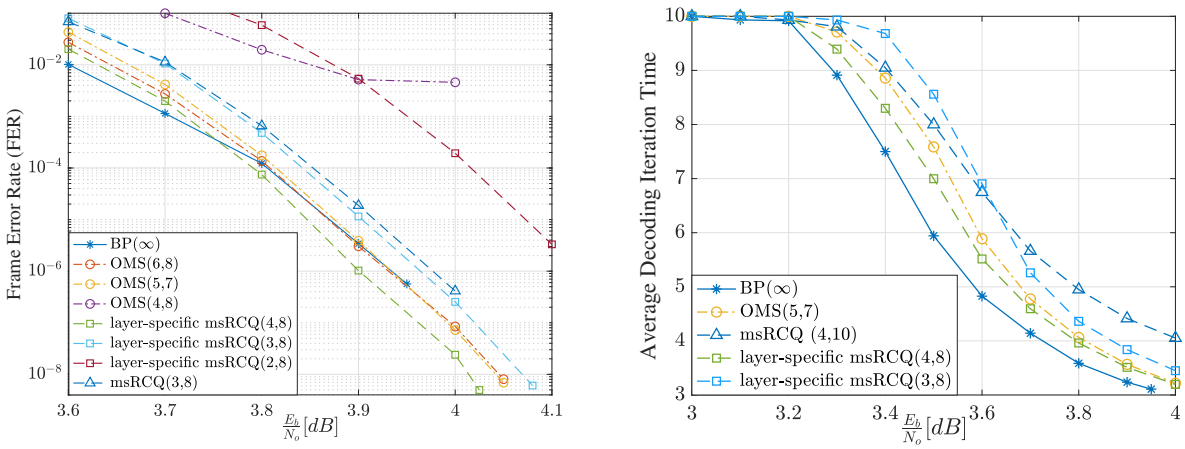
$$l_i \boxplus l_j = \log \frac{P(0,i)P(0,j) + P(1,i)P(1,j)}{P(0,i)P(1,j) + P(1,i)P(0,j)}. \tag{2.64}$$

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 with (2.64), it can be seen that (2.63) 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., 2.63) 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., (2.63)) 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 [LB18a] 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



(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.

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. 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

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%)

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 3.4 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 [Gal62] have 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, LB18b, 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) [NBB16, LG17, NMB17, NML18, LSW18, WJZ18, LG18, LZJ18, XVT19b, DB19, ABS19, BHP20, WWF20, LCH19, NWH21]. Nachmani *et al.* in [NBB16] 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, NBB16] 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/quantizer 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 a 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 schemes 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 6-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 [NBB16]. 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:

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

$\mathcal{N}(c_i)$ 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:

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

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

$\mathcal{N}(v_j)$ 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 [NBB16]. 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,

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

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

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

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

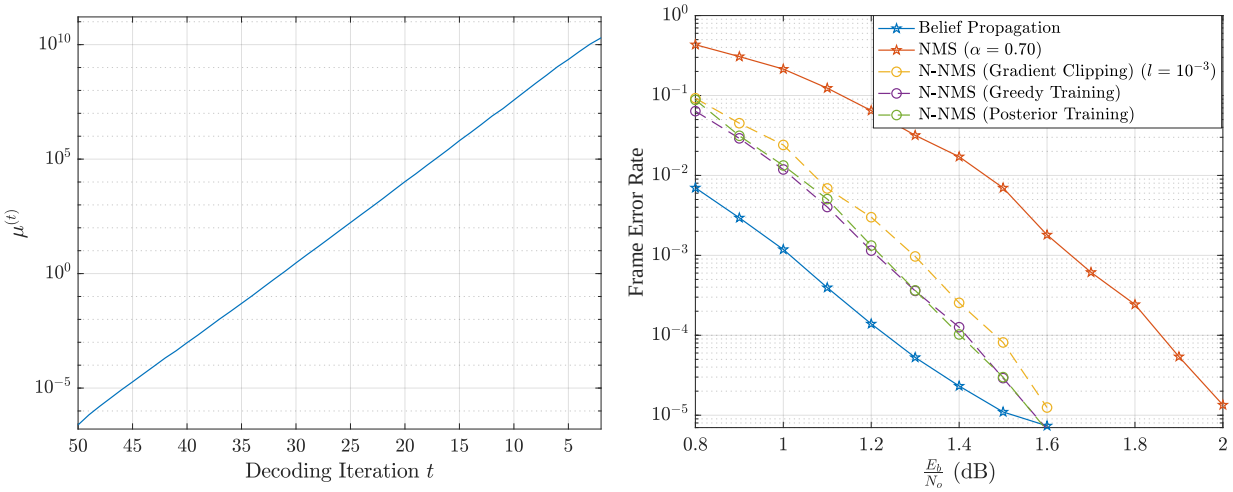
The term $\frac{\partial J}{\partial|u^{(t)*}_{c_i\to v_j}|}$ is calculated by:

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

(3.6)-(3.11) indicate that the neuron values in each hidden layer can be stored compactly with $\text{sgn}\left(l^{(t)}_{v_j\to c_i}\right)$, $\texttt{min1}^t_{c_i}$, $\texttt{min2}^t_{c_i}$, $\texttt{pos1}^t_{c_i}$ and $\texttt{pos1}^t_{c_i}$. 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 $\frac{\partial J}{\partial u^{(t)}_{c_i\to v_j}}$ and $\frac{\partial J}{\partial l^{(t)}_{v_j\to c_i}}$ 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 $\frac{\partial J}{\partial\beta^{(t)}_{(c_i,v_j)}}$. 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 $\texttt{pos1}^t_c$ and $\texttt{pos2}^t_c$ receive gradients from c. Besides, $|\mathcal{N}(c)|-1$ gradient terms flow to $\texttt{pos1}^1_c$. Hence, if check node c has a large degree, the gradient of J w.r.t. $\texttt{pos1}^t_c$ 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 $\frac{\partial J}{\partial u^{(t)}_{(c,v)}}$ leads to large magnitude of $\frac{\partial J}{\partial \beta^{(t)}_{(c,v)}}$ 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:

$$\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}} = \frac{\partial J}{\partial l_{v_j}^{(t)}}. \tag{3.12}$$

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

$$\frac{\partial J}{\partial \beta_{c_i \to v_j}^{(t)}} = \frac{u_{c_i \to v_j}^{(t)}}{\beta_{c_i \to v_j}^{(t)}} \frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}} = \frac{u_{c_i \to v_j}^{(t)}}{\beta_{c_i \to v_j}^{(t)}} \frac{\partial J}{\partial l_{v_j}^{(t)}}. \tag{3.13}$$

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 $\frac{\partial J}{\partial l_{v_j \to c_{i'}}^{(t)}}$ 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. $u_{c_i \to v_j}^{(t)}$ by:

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

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

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

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

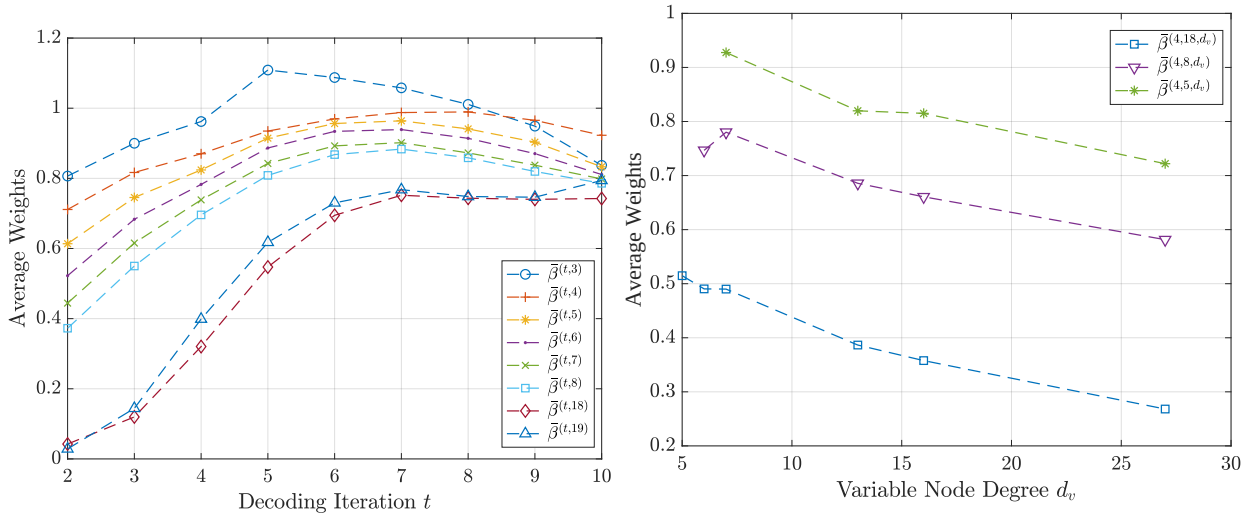
N-NMS and N-OMS decoder for the long-blocklength LDPC codes are impractical, because the number of parameters of these decoders is proportional to the number of edges in the corresponding Tanner graph. Weight sharing [XCB21] solves this problem by assigning one weight to different neurons in the NN. 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.

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



(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.

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

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 [NBB16]				
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

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.

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.

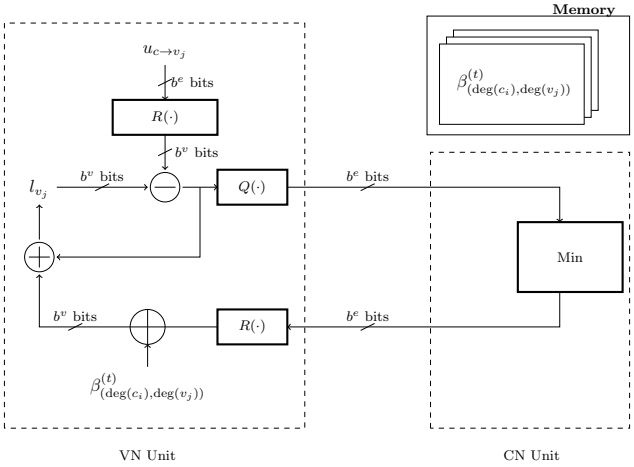


Figure 3.3: Layer-scheduled Neural Offset RCQ Decoder Structure

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

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

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

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

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

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

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 iteration in the local variable node units (VNUs) will cost large amount 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

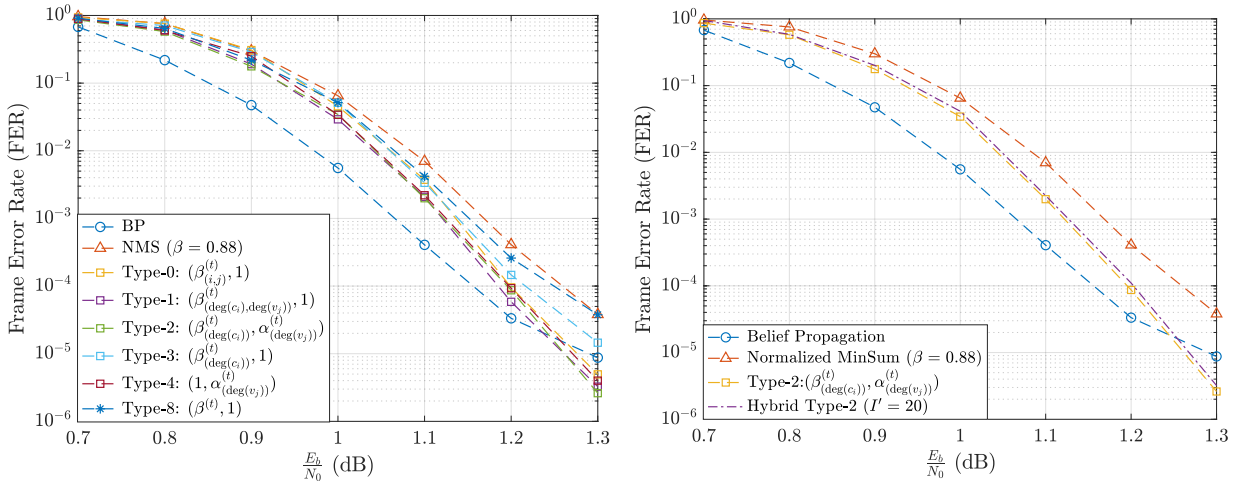
Code	Rate	Edge distribution
(16200,7200) DVBS-2 LDPC code [ETS]	$\frac{4}{9}$	$\lambda(x) = 2.06 * 10^{-5} + 0.3703x +$ $0.3333x^2 + 0.2963x^7$ $\rho(x) = 0.1186x^3 + 0.3332x^4 +$ $0.4445x^5 + 0.1037x^6$
(9472,8192) QC-LDPC code [WTS22]	$\frac{8}{9}$	$\lambda(x) = x^3$ $\rho(x) = 0.3919x^{28} + 0.6081x^{29}$
$k = 1032$ 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

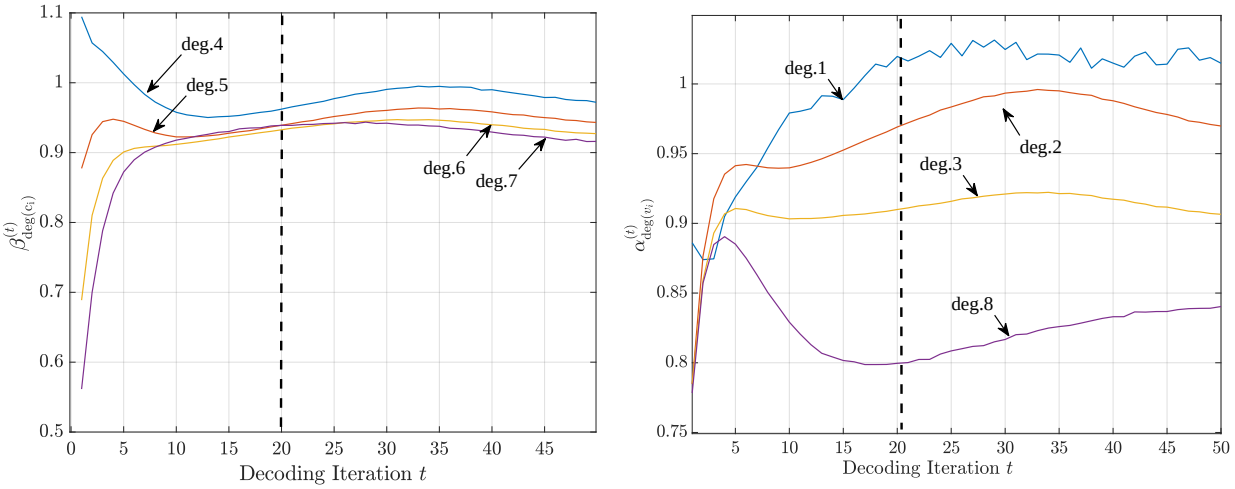


(a) (b)

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.

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



(a) (b)

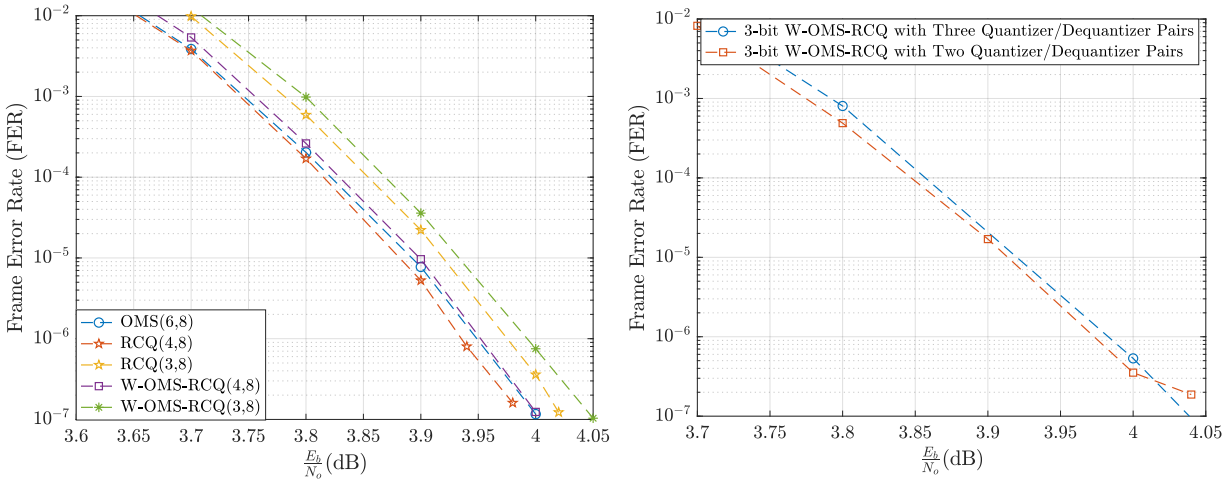
Figure 3.5: The change of weights of the type-2 N-2D-NMS decoder for (16200, 7200) DVBS-2 LDPC code w.r.t. check node degree, variable node degree and iteration index. Specifically, Fig. (a) gives $\beta^{(t)}_{(\deg(c_i))}$ for all possible check node degrees in each decoding iteration t, Fig. (b) gives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

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. 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



(a) (b)

Figure 3.6: Fig. (a): FER performance of W-OMS-RCQ decoders, RCQ decoders, and 6-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}.

and a 6-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].

Simulation result shows the 6-bit OMS decoder has the best FER performance. The 4-bit W-OMS-RCQ decoder and 4-bit RCQ decoder have similar FER performance, which are inferior to the 6-bit OMS decoder by 0.015 dB. However, as shown in Table 3.4, the 4-bit W-OMS-RCQ decoder requires much less resources compared with the 4-bit RCQ decoder and the 6-bit OMS decoder. Compared to the 6-bit OMS decoder, the 3-bit W-OMS-RCQ and 3-bit RCQ decoder have a gap of 0.04 and 0.06 dB, respectively. More specifically, the 3-bit RCQ decoder has a similar LUT, BRAM and routed net usage to the 4-bit W-OMS-

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	$C_1 = 3$, $\gamma_1 = 1.3$	$1 \sim 6$
	$C_2 = 5$, $\gamma_2 = 1.3$	$7 \sim 8$
	$C_3 = 7$, $\gamma_3 = 1.3$	$9 \sim 10$

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

Decoding Structure	LUTs	Registers	BRAMS	Routed Nets
OMS(6,8) (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%)

RCQ decoder. On the other hand, the 3-bit W-OMS-RCQ uses much fewer resource than the 4-bit W-OMS-RCQ decoder.

Fig. 3.6a compares the FER performances of W-OMS-RCQ decoders with RCQ decoders and a 6-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].

Simulation result shows the 6-bit OMS decoder has the best FER performance. The 4-bit W-OMS-RCQ decoder and 4-bit RCQ decoder have similar FER performance, which are inferior to the 6-bit OMS decoder by 0.015 dB. However, as shown in Table 3.4, the 4-bit W-OMS-RCQ decoder requires much less resources compared with the 4-bit RCQ decoder

and the 6-bit OMS decoder. Compared to the 6-bit OMS decoder, the 3-bit W-OMS-RCQ and 3-bit RCQ decoder have a gap of 0.04 and 0.06 dB, respectively. More 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 resource 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 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}.

subsection$k = 1032$ 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

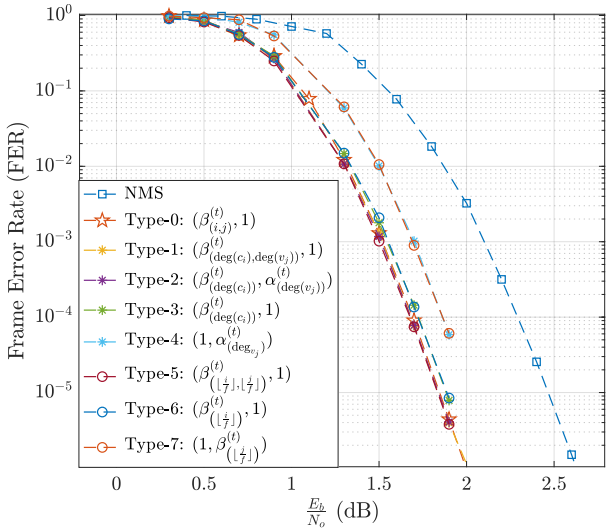
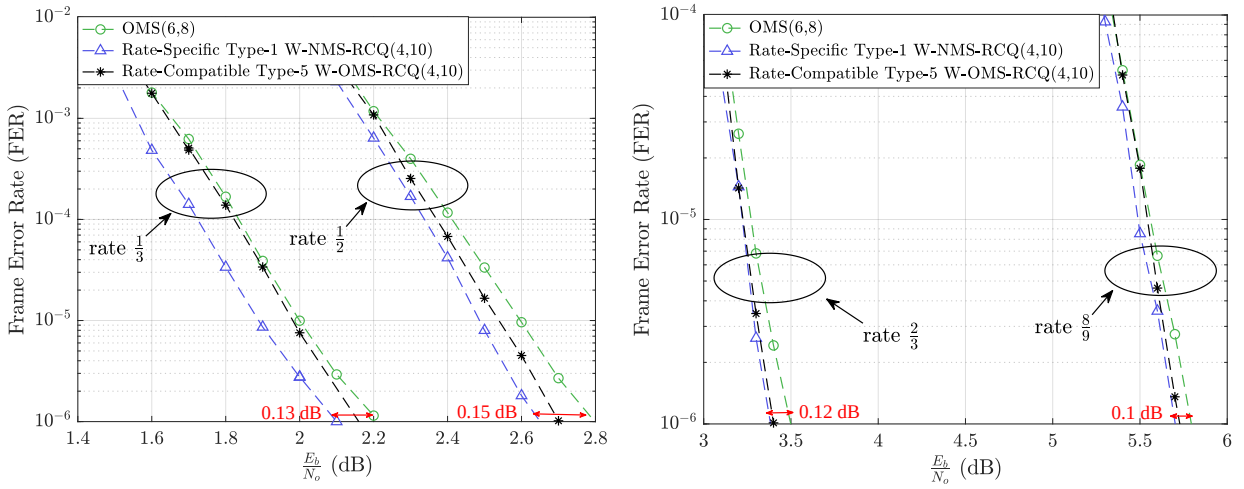


Figure 3.7: FER performance of N-2D-NMS decoders with various weight sharing types for a (3096,1032) PBRL LDPC code compared with N-NMS (type 0) and NMS.

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 $C_1 = 7$, $\gamma_2 = 1.7$



(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.

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.

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.

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.

[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.

[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.

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

[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).*

[Gal62] Robert G. Gallager. "Low-Density Parity-Check Codes." *IRE Trans. on Info. Theo.*, **8**(1):21–28, January 1962.

[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.

[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.

[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.

[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.

[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.

[NBB16] 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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.