Title
Applications of LDPC Codes to RF-Optical Hybrid Systems and the Line Product Code

Author
Nguyen, Jonathan Vu

Publication Date
2022

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Applications of LDPC Codes to RF-Optical Hybrid Systems and the Line Product Code

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Electrical and Computer Engineering

by

Jonathan Vu Nguyen

2022

ABSTRACT OF THE THESIS

Applications of LDPC Codes to RF-Optical Hybrid Systems
and the Line Product Code

by

Jonathan Vu Nguyen
Master of Science in Electrical and Computer Engineering
University of California, Los Angeles, 2022
Professor Richard D. Wesel, Chair

This thesis consists of an introduction, four main sections, and conclusion. The introduction gives a quick overview of communication channels and the basics of message passing algorithms. The first section focuses on the challenges of communicating over a fading channel using optical transmitters and receivers. Recognizing the limitations of a purely optical transmission system during environmental fades, the section proposes a hybrid RF-Optical system. The hybrid system consists of a high-throughput optical link closed by a Low Density Parity Check (LDPC) code alongside a separate RF link. The section proposes two different architectures for a hybrid system with varying degrees of mixing between the two links. For each architecture, their performance was evaluated during on a simulated fading channel and Additive White Gaussian Noise (AWGN) channel.

The second section of this thesis capitalizes on the recent advancements in deep learning and applies it to the most popular LDPC decoding process known as message passing. The section first deconstructs the steps of message passing into nodes that can be interpreted as a type of Neural Network. Then, utilizing gradient descent methods including Adap-

tive Movement Estimation (ADAM), multiplicative weights for message passing are found and optimized. The key contribution of the section is the connection between these multiplicative weights and properties intrinsic to the LDPC code structure. By exploiting this relationship, a weight-sharing paradigm based on node degree is proposed, resulting in a Neural-Normalized MinSum (N-NMS) decoder which dramatically reduces the complexity of both training and implementation compared to typical neural network based decoders.

The third section discusses a unique case study involving the Consultative Committee for Space Data Systems (CCSDS) 141.11-O-1 Line Product Code (LPC). Being such a short block-length code, two Maximum Likelihood (ML) decoders were evaluated against message passing decoders including MinSum (MS), Belief Propagation (BP), and Neural-Normalized MinSum (N-NMS). Analysis was carried out considering both the decoding performance and resulting hardware complexity of the decoders.

Continuing on this, the fourth and final section aims to give additional insight as to why the N-NMS decoder performed so well on the LPC. This section proposes that the unique graph structure of the LPC allowed for key optimizations such as breaking length-4 cycles that the N-NMS training process capitalized on during its training.

The thesis of Jonathan Vu Nguyen is approved.

Danijela Cabric

Achuta Kadambi

Lara Dolecek

Richard D. Wesel, Committee Chair

University of California, Los Angeles

2022

To my parents and Jodie

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis represents the culmination of my learned experiences as both an undergraduate and graduate researcher at the University of California, Los Angeles (UCLA). This thesis would not have been possible without the support and guidance of my family, friends, mentors, and colleagues and I would like to thank them properly here.

First and foremost, I would like to express my deepest gratitude and appreciation for my research advisor, Professor Richard D. Wesel. I first met Professor Wesel during my sophomore year of my undergraduate degree at somewhat of a turning point in my career. Up until that point, I had only taken circuits related courses and, not being very good at them, I (dramatically) began to doubt my future as an electrical engineer. However, upon joining the lab, I grew to find a deep appreciation and passion for both information and coding theory. While the learning curve was fairly steep at first, Professor Wesel gently guided me along, providing additional context during presentations I would have otherwise been lost during. Despite his busy schedule, he also set aside additional time and resources to bring me up to speed. Once I found my footing, his unwavering enthusiasm for coding theory coupled with his experiences enabled me to pursue my own research projects. The trajectory of my career changed when I joined the laboratory and I am grateful to Professor Wesel for providing me the opportunity to be a part of it.

I would like to thank Professor Dariush Divsalar. His long history with LDPC codes alongside his kindness and passion impacted every project he touched and I am honored to have worked with him.

Through my time in the Communication Systems Laboratory, I had the honor of working alongside truly talented and passionate individuals, all of whom I can call friends: Linfang Wang, Hengjie Yang, Ethan Liang, Chester Hulse, Amaael Antonini, and so many others. Ethan provided me with endless information and papers during my first three months in the lab to help me get started. Also being an undergraduate researcher, he inspired me with the

amazing work he did during his time in the lab. I especially want to thank Linfang, who served as my mentor throughout my time in the lab. I deeply cherish his enthusiasm and aptitude for research and for spending so much of his time to personally help in my own work.

I am also immensely grateful for the patience, dedication, and passion displayed by the teaching staff at UCLA. I would like to thank Professor Flavio Lorenzelli for teaching his Communications Systems Engineering course, and for always entertaining my ideas during office hours. My time in his course ultimately inspired me to pursue a career in digital communications systems. I also give special thanks to Professors Lieven Vandenberghe, Lara Dolecek, Greg Pottie, Danijela Cabric, Achuta Kadambi, Mike Briggs, and Hooman Darabi for teaching the courses that have formed the foundations of my engineering knowledge.

Outside of UCLA, I would like to thank my colleagues at Astranis Space Technologies, each of whom guided me through my internship and now full-time career. I thank Steve Joeseph, Ryan Stark, Stephen Spears, and Andrew Ramsey for their patience and camaraderie while working together on our own Software Defined Radios.

Most importantly, I dedicate this thesis to my parents. Without their support, love, and sacrifice, I would not be where I am today and I owe so much of who I am today them.

Finally, I would like to thank SA Photonics and Physical Optics Corporation for their financial support on the research projects I took part in. The knowledge I gained from working alongside the talented engineers at both companies has carried on well past the projects' completion. I would specifically like to thank Anthony Lai and Nathan Wong from Physical Optics Corporation for supporting my first research project and for taking me on as an intern. Additionally, I would like to thank Todd Drullinger and Todd Chauvin from SA Photonics for their guidance.

VITA

2021	B.S. (Electrical Engineering), UCLA, Los Angeles, California
2022	M.S. (Electrical Engineering), UCLA, Los Angeles, California
2019	Lab Intern, Physical Optics Corporation
2020	Digital Communications Intern, The Aerospace Corporation
2021	DSP/Communications Intern, Astranis Space Technologies
2022	Software Defined Radio Systems Engineer, Astranis Space Technologies

PUBLICATIONS

J. Nguyen, E. Liang, L. Wang, T. Drullinger, T. Chauvin, R. D. Wesel, "Comparison of Integrated and Independent RF/FSO Transceivers on a Fading Optical Channel," in *Asilomar Conference on Signals, Systems, and Computers*, 2020.

L. Wang, S. Chen, **J. Nguyen**, D. Dariush, R. Wesel, "Neural-Network-Optimized Degree-Specific Weights for LDPC MinSum Decoding", in *International Symposium on Topics in Coding (ISTC)*, 2021

J. Nguyen, L. Wang, C. Hulse, S. Dani, A. Antonini, T. Chauvin, D. Dariush, R. Wesel, "Neural Normalized Min-Sum Message-Passing vs. Viterbi Decoding for the CCSDS Line Product Code", in *IEEE International Conference on Communications (ICC)*, 2022

CHAPTER 1

Introduction

1.1 Overview of Communication Channels

Additive White Gaussian Noise (AWGN) is one of the most fundamental types of noise that appears in communication channels via background, thermal, or radiation noise. As the name implies, AWGN channels are applied additively to the signal of interest and follow a Gaussian probability distribution with mean 0 and variance σ^2.

$$y = x + n, \quad \text{where } n \sim \mathcal{N}(0, \sigma^2) \tag{1.1}$$

From Equation 1.1, x is the signal of interest, n is the noise, and y is the resulting signal. As the variance σ^2 increases, the range of values that n is likely to take on also increases. Larger realizations of the noise n cause the signal to become drowned out and for digital communication schemes, can result in bit errors. A common metric for describing the relative strength of a signal compared to the noise is known as the Signal to Noise Ratio (SNR). It is commonly expressed in decibels and is calculated via Equation 1.2. The higher the SNR, the more resistant the signal of interest is against the present noise. Other common metrics for describing the quality of a channel are E_b/N_0 and E_s/N_0 where E_b and E_s represent the energy per bit and symbol respectively and N_0 represents the noise power. For AWGN channels $N_0/2 = \sigma^2$.

$$SNR_{dB} = 10 \log_{10} \left(\frac{P_x}{P_n} \right) = 10 \log_{10} \left(\frac{P_x}{\sigma^2} \right) \tag{1.2}$$

In Binary Phase Shift Keying (BPSK), one of the most basic forms of digital communication, the signal of interest x takes on either $+1$ or -1 depending on whether the original bit value b was 0 or 1 respectively. Because x is transmitted over a noisy analog channel, by the time it is received, its value will not remain ± 1. Since the noise is additive and x takes on discrete values, we can succinctly express the probability density function of y conditioned on the original bit value b over a AWGN channel as:

$$p(y|b=0) = \mathcal{N}(1, \sigma^2) \tag{1.3}$$

$$p(y|b=1) = \mathcal{N}(-1, \sigma^2) \tag{1.4}$$

Graphically, these conditional probabilities are expressed as:

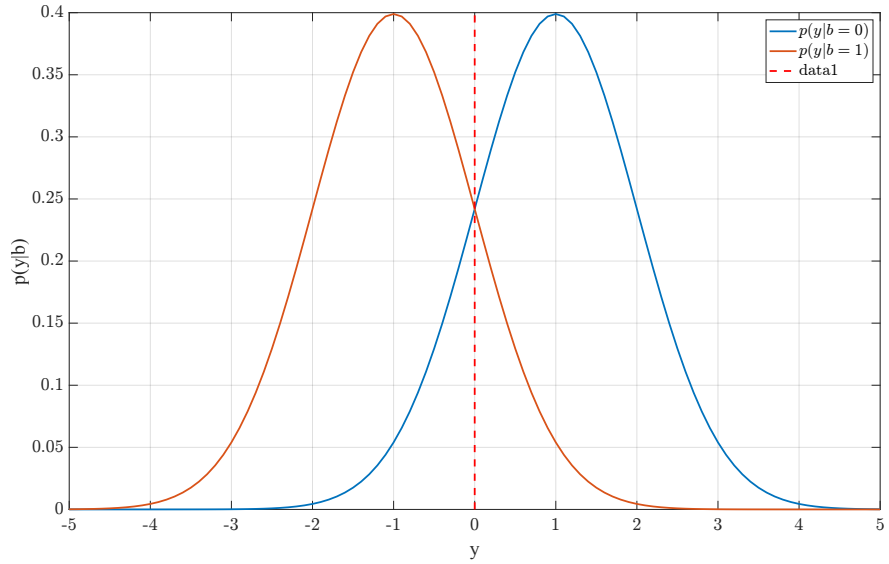


Figure 1.1: Conditional Probability Density Function for BPSK signal under AWGN ($\sigma^2 = 1$)

Because the decoded bits must take on a discrete value, some hard decision must be made to determine the bit value of the received signal y. Taking the midpoint of the two possible values of x as a decision boundary, we arrive at the following rule for making a hard decision. This is also shown in Figure 1.1 as the dotted red line.

$$\hat{b} = \begin{cases} 0 & \text{if } y > 0 \\ 1 & \text{if } y \leq 0 \end{cases} \tag{1.5}$$

Every signal starts at either ± 1 and through AWGN, ends up somewhere along each p.d.f. If the signal crosses over the decision boundary at $y = 0$, then it is decoded incorrectly and a bit error is made. Lower SNR values cause decision errors to occur more often, effectively increasing our Bit Error Rate (BER). Figure 1.2 showcases the relationship between SNR and BER.

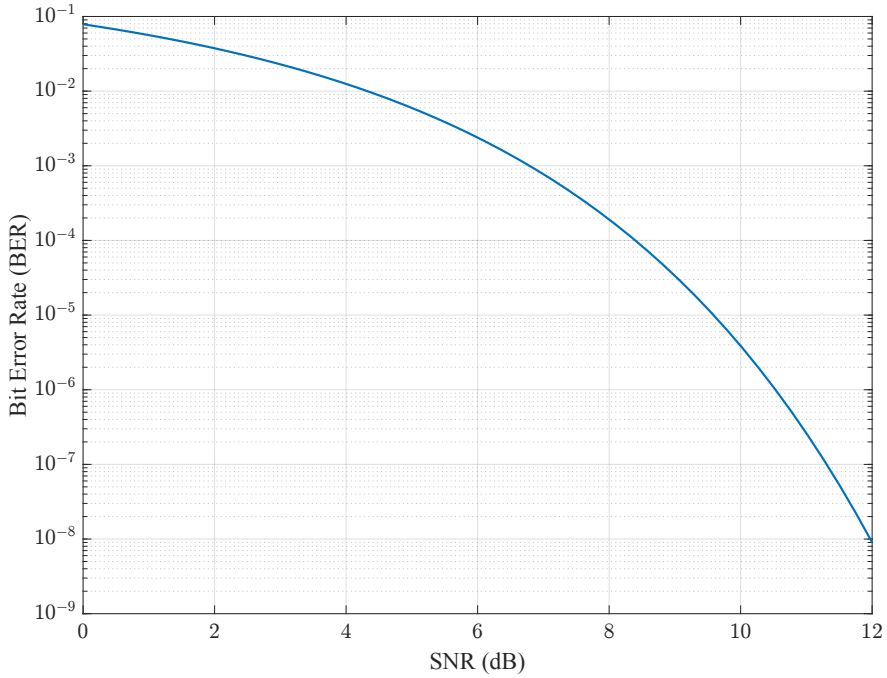


Figure 1.2: Bit Error Rate of BPSK modulated signal over AWGN

1.2 Overview of LDPC codes

Low Density Parity Check (LDPC) codes are a class of linear block codes proposed by Robert Gallager in his 1962 doctoral dissertation [Gal62]. LDPC codes, like all linear block codes, provide error detection and correction capabilities by encoding a block of bits such that the

result contains additional redundant parity bits. These parity bits allow for bit errors to be detected and corrected for at the expense of some throughput.

LDPC codes are fully described via the null space of a $(n-k) \times n$ parity check matrix $\boldsymbol{H}$. To give some context, n is known as the block-length and describes the length of the encoded block. On the other hand, k is the number of information bits, describing how much useful information is present in each transmitted block. Therefore, $n-k$ is the number of bits added on via the encoding process. Lastly, k/n, the proportion of information to transmitted bits, is known as the code rate. Higher code rates transmit a higher proportion of useful information but are more prone to errors because their are fewer redundant bits.

The entries of the parity check matrix are elements of $GF(2)$ meaning that they can only take on values of 0 or 1. The "low density" part of LDPC codes describes the fact that the proportion of 1s to 0s in the parity check matrix is small. Below, the parity check matrix of the 7-4 Hamming Code is shown. Here, $n = 7$ and $k = 4$.

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.6}$$

Because we encode k bits to n and $k \leq n$, there exist 2^n possible codewords and only 2^k codewords that correspond to a valid original message. For LDPC codes, $\boldsymbol{H}$ is constructed such that, for all valid codewords x, we must have $\boldsymbol{H}x = s = \mathbf{0}$ where s is the syndrome of x. In other words, the syndrome of a valid codeword must be $\mathbf{0}$.

1.3 Tanner Graphs Construction

Continuing from Gallager's work, Tanner proposed a graphical representation of LDPC codes as a bipartite graph, also known as a Tanner Graph [Tan81]. This bipartite graph can be drawn fairly simply given a code's parity check matrix.

To construct a Tanner Graph from a parity check matrix, we first define two types of nodes: variable nodes v_j and check nodes c_i. Each variable node represents a bit in the received codeword and is represented by each column in the parity check matrix. On the other hand, each check node represents the rules that define each parity bit, represented in each row of the parity check matrix. An edge is drawn between variable node v_j and check node c_i if $h_{ij} = 1$. Figure 1.3 is the Tanner Graph for the 7-4 Hamming Code shown earlier. Squares represent a check node and circles represent variable nodes. To satisfy the syndrome requirement $\boldsymbol{H}x = 0$, all variable nodes connected to a check node must XOR to 0 for all check nodes.

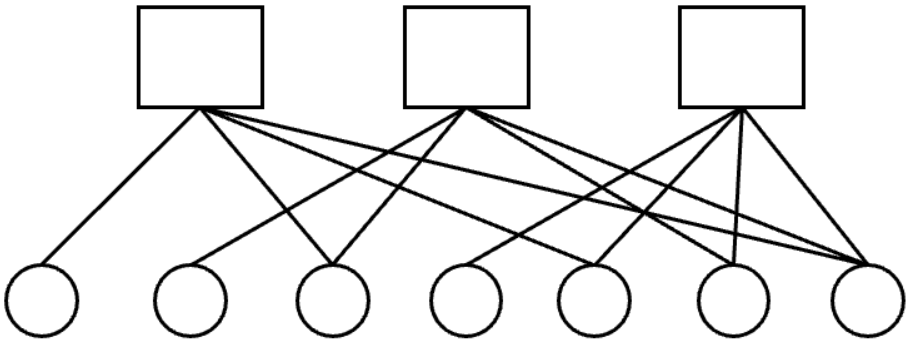


Figure 1.3: Bipartite Graph for the 7-4 Hamming Code

1.4 Message Passing Algorithms

Given the Tanner Graph representation of an LDPC code, the structure of the graph can be used to perform iterative decoding algorithms known broadly as message passing [KFL01] [FMI99]. The core principal of message passing is that variable nodes with low confidence can be corrected by other variable nodes of higher confidence. Through messages passed between variable and check nodes (and vice versa), this transfer of information and updating of beliefs can occur. The individual steps of message passing are summarized in each following

subsection. This section uses the notation as shown in [LC04].

1.4.1 Calculating Log-Likehood Ratios

Firstly, as bits enter the receiver, they are received as analog values. If the channel distribution is known, the conditional probabilities of the received signal given the received signal can be calculated. Recall from Equation 1.4 the conditional probability density functions of the received signal given the transmitted bit over a AWGN channel. Given the conditional probabilities, we can calculate the signal's Log-Likelihood Ratio (LLR). The LLR gives a metric of how likely the transmitted bit was 0 vs 1. The more positive the LLR is, the more confident we are that it was a 0. Negative LLRs represent higher confidence of a transmitted 1. These LLRs are used as the initial variable to check node messages in message passing decoders.

$$LLR = \log\left(\frac{p(y|b=0)}{p(y|b=1)}\right) \tag{1.7}$$

$$L_{j\to i} = \log\left(\frac{p(y_j|b=0)}{p(y_j|b=1)}\right) \tag{1.8}$$

1.4.2 Check Node Update

Following this initial broadcast, the check nodes collect their received messages and respond back to their connected variable nodes. There are two main methods which the outgoing check to variable message is calculated: Belief Propagation and MinSum.

For Belief Propagation (BP), Equation 1.9 is used to calculate the message $L_{i\to j}$ from check node c_i to variable node v_j. MinSum is shown in Equation 1.10. Note that $N(i)$ represents the set of all variable nodes connected to check node c_i. Therefore $N(i)\setminus\{j\}$ represents all variable nodes connected to c_i except for v_j. The exclusion of v_j in the calculation of the message back to it ensures that the update is made using extrinsic information. To see

why extrinsic information is important for decoding, imagine a scenario where one variable node has an extremely strong belief in its value that is incorrect. If all updates to that variable node included its own information, the variable node would continuously reinforce its incorrect belief.

$$L_{i\to j} = 2\tanh^{-1}\left(\prod_{j^*\in N(i)-\{j\}} \tanh\left(\frac{1}{2}L_{j^*\to i}\right)\right) \tag{1.9}$$

$$L_{i\to j} = \prod_{j^*\in N(i)-\{j\}} \operatorname{sgn}(L_{j^*\to i}) \cdot \min_{j^*\in N(i)-\{j\}} |L_{j^*\to i}| \tag{1.10}$$

Although the Belief Propagation operation is the most theoretically accurate, it is computationally expensive making MinSum the more commonly used operation in practice.

1.4.3 Syndrome Check

In order to determine when to stop the decoding process, each variable node constructs an estimate of its bit value based on the incoming check node messages. L_j is the initial LLR estimate of variable node v_j.

$$L_j^{total} = L_j + \sum_{i^*\in N(j)-\{i\}} L_{i^*\to j} \tag{1.11}$$

Given this total estimate, a hard decision is now made according to:

$$\hat{b}_j = \begin{cases} 0 & \text{if } L_j^{total} \geq 0 \\ 1 & \text{if } L_j^{total} < 0 \end{cases} \tag{1.12}$$

If $\boldsymbol{H}\hat{b} = \mathbf{0}$, then the estimated codeword is valid and the decoding process stops. Otherwise, repeat all steps until a valid codeword is found or the maximum number of iterations is reached. By the end of the decoding process, if the received codeword failed to pass the

syndrome check, the frame is declared to be a detected error and tossed. If the decoding process terminates but results in the wrong codeword, a undetected error has occurred.

1.4.4 Variable Node Update

Next, each variable node collects all incoming messages from its connected check nodes. The variable to check node update is calculated using Equation 1.13

$$L_{j\to i} = L_j + \sum_{i^*\in N(j)-\{i\}} L_{i^*\to j} \tag{1.13}$$

1.5 Pitfalls of Message Passing Algorithms

Recall from earlier how, when calculating a outgoing message to a specific variable or check node, that node's incoming message is excluded from the calculation. Additionally, recall that this is done such that only extrinsic information is passed around to prevent a single node from reaffirming its own beliefs.

One way that the assumption of extrinsic information can be broken is via cycles in the Tanner Graph. While excluding the node from message calculations back to itself prevents direct cycles, a variable node's incorrect belief can propagate through other check and variable nodes back to itself. The shorter the cycle, the easier it is for a node's belief to travel back to itself. The shortest cycle length in a code is called its girth and generally, higher girths lead to better decoding performance. Generally, high block-length codes will have more sparse parity check matrices and can avoid short cycles easier, thus explaining why longer codes perform better than shorter ones. To demonstrate what a cycle looks like, Figure 1.4 shows one length-4 cycle in red. Note that a length-4 cycle is the shortest possible cycle and therefore the most detrimental to decoding performance.

As explored in [Ric03], deficiencies in the structure of a code's Tanner Graph can lead

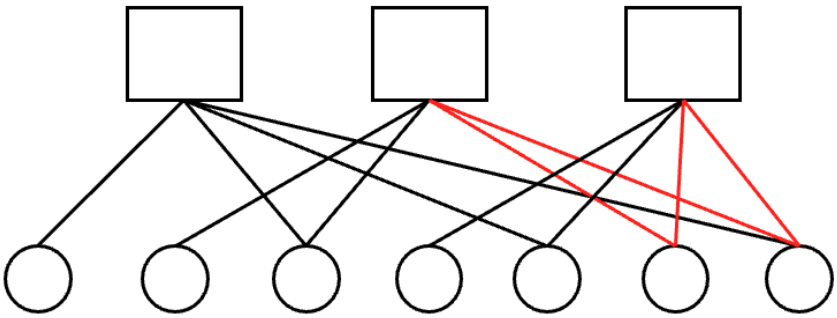


Figure 1.4: Length-4 Cycle of the 7-4 Hamming Code

to early error floors. In such a case, the error rate of a code ceases to improve, or improves much more slowly, after a certain SNR is reached. The authors of [LM07], [LHM09], and [MSW06] provide an analysis on the relationship between a code's girth and its trapping sets. Trapping sets refer to certain input patterns that cause message passing algorithms to fail. These trapping sets fail due to deficiencies in the code structure and commonly present themselves as "near-codewords" or continuous oscillations in variable node estimates.

CHAPTER 2

Hybrid Radio and Free Space Optical Modem Design on a Fading Channel

Several prior works [KGJ18] have shown that hybrid systems joining both Radio Frequency (RF) and Free Space Optical (FSO) transmitters have the ability to harvest the high data rates possible with optical transmitters while still maintaining a reliable RF link when clouds, fog, or dust interfere with the FSO channel. This chapter introduces two proposed architectures of such a hybrid system, incorporating Protograph-based Raptor-like (PBRL) LDPC codes developed at UCLA [CVD15]:

1. Integrated RF/FSO modem with joint physical-layer processing
2. Independent FSO and RF system with separate physical-layer processing assuming an intelligent IP router to divide data between the two physical layer modems.

2.1 Hybrid Modem Architectures

Both architectures listed above were tested assuming a fixed-rate PBRL Low-Density Parity Check (LDPC) code with either rate 1/2 or 8/9 on the FSO channel. PBRL codes utilize a capacity approaching Highest-Rate Code (HRC) whose rate can be adaptively lowered by the transmission of additional symbols produced by the Incremental Redundancy Code (IRC). The resulting LDPC code can support multiple code rates that each approach their respective Shannon Limit. For this project, the protograph formed from the HRC and IRC was lifted to

obtain much larger parity check matrices for the rate 1/2 and 8/9 codes. The lifting process, described in [TJV04] aims to maximize the length of short cycles that contribute to trapping sets. The paper introduces the Extrinsic Message Degree (EMD) metric which describes a cycle's propensity to induce decoding errors. By minimizing the EMD in the lifting process, PBRL codes of blocklength 16,384 and rates 1/2 and 8/9 were formed for this project.

2.1.1 Integrated RF/FSO Modem Design

In the case of the integrated architecture, as shown in Figure 2.1, bits sent over the RF channel were jointly encoded using the same LDPC code. The RF QPSK modulator is assumed to be 156.25 megasymbols per second, thus modulating 312.5 megabits per second of LDPC bits. Two baud rates were considered for the FSO on-off keying (OOK) modulator, 156.25 megasymbols per second or 2.5 gigasymbols per second. Thus, the FSO OOK modulator modulates either 156.25 megabits per second or 2.5 gigabits per second of LDPC bits, depending on the baud rate.

The choice of FSO baud rate affects both the overall data rate of the system and the mix of FSO and RF bits seen by the LDPC decoder. Higher baud rates correspond more bits being transmitted in a given period. Therefore, the modem, RF or FSO, with the higher baud rate will naturally transmit more LDPC encoded bits.

Table 2.1 shows the supported source data rates for various choices of FSO baud rate and LDPC code rate on the integrated RF/FSO modem. The ratio between FSO and RF bits has an important effect on system performance. For the system using the 156.25 Megabaud per second FSO OOK modulator, there are two RF QPSK bits for every one FSO OOK bit (recall that each QPSK symbol represents 2 bits). This means that the RF channel quality plays a dominant role in the performance of the hybrid modem in this case. In contrast, for the system using the 2.5 gigasymbols per second FSO OOK modulator, there are eight FSO OOK bits for every RF QPSK bit. Thus, the optical channel plays a dominant role in the performance of the hybrid modem in this case.

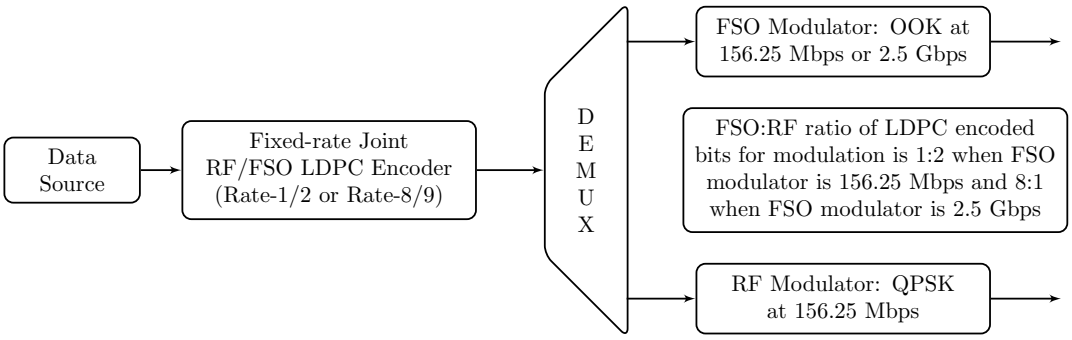


Figure 2.1: Transmitter architecture for an integrated RF/FSO design where a single LDPC encoder provides coded bits to modulators for both the RF and FSO channels.

Channels + Baud Rates	**Source data rate for LDPC rate-1/2**	**Source data rate for LDPC rate-8/9**
FSO OOK (156.25 Megabaud) + RF QPSK (156.25 Megabaud)	234.375 Mbps	416.67 Mbps
FSO OOK (2.5 Gigabaud) + RF QPSK (156.25 Megabaud)	1.40625 Gbps	2.5 Gbps

Table 2.1: Maximum data rates supported by the four configurations of the proposed **integrated** RF/FSO modem, assuming both links are closed.

2.1.2 Independent RF/FSO Modem Design

For the independent architecture, shown in Figure 2.2, a LDPC code is again used to encode bits destined for the FSO channel. However, instead of the LDPC code encoding both RF and FSO bits, RF bits will be exclusively and independently transmitted using a legacy 64-state rate-1/2 convolutional code operating on a closed QPSK link. In this sense, the RF link is meant to serve as a stable and constant link independent of fading.

Table 2.2 shows the supported source data rates for the proposed integrated RF/FSO modem. Unlike the integrated architecture, the RF link is fixed to a rate 1/2 convolutional

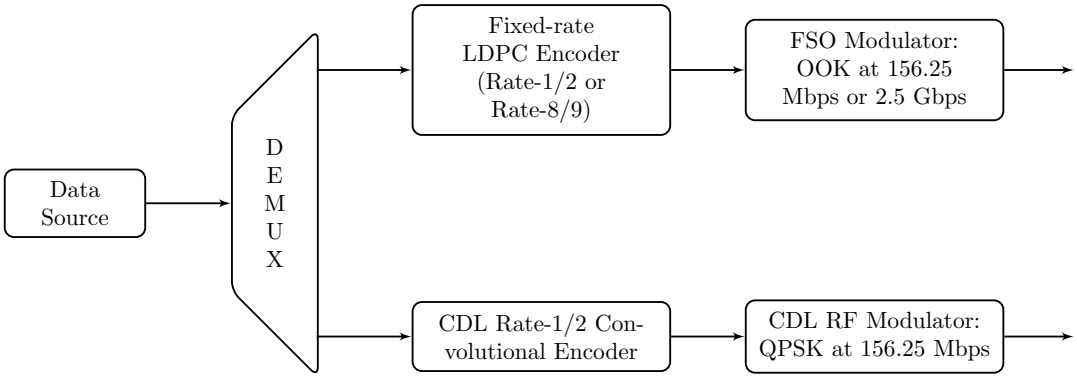


Figure 2.2: Transmitter architecture for an independent RF/FSO design where independent encoders provide coded bits separately to modulators for the RF and FSO channels.

Channels + Baud Rates	**Source data rate for LDPC rate-1/2**	**Source data rate for LDPC rate-8/9**
FSO OOK (156.25 Megabaud) + RF CDL QPSK (156.25 Megabaud)	234.375 Mbps	295.14 Mbps
FSO OOK (2.5 Gigabaud) + RF CDL QPSK (156.25 Megabaud)	1.40625 Gbps	2.376 Gbps

Table 2.2: Maximum data rates supported by the four configurations of the proposed **independent** RF/FSO modem, assuming both links are closed.

code in line with the Common Data Link (CDL) radio standard. As a result of this, the overall throughput of the independent modem is lower than that of the integrated one when the LDPC code rate is set to 8/9. The trade-off between the integrated and independent modems now becomes whether the added stability of the independent RF CDL link can make up for the lost throughput caused by the inflexibility of the CDL radio.

2.2 Channel Models

2.2.1 RF Channel

The RF channel model is QPSK modulation under additive white Gaussian noise (AWGN). A QPSK symbol can be modeled as $x = (x_I, x_Q) = (\pm\sqrt{E}, \pm\sqrt{E})$ where x_I and x_Q are the in-phase and quadrature components respectively. Depending on the channel conditions, the transmitted symbol can be scaled by fading parameter ρ and/or distorted by Additive White Gaussian Noise $z = (z_I, z_Q)$. Both components of the noise are modeled by i.i.d. Gaussian distributions with zero-mean and variance σ_z^2. The corresponding probability density function (p.d.f) is given by equations 2.1 and 2.2. The p.d.f is visualized in Figure 2.3.

$$f(z_I, z_Q) = \frac{1}{\sqrt{2\pi \det(\Sigma)}} \exp\left(-\frac{1}{2}\mathbf{z}^T \Sigma^{-1} \mathbf{z}\right) \tag{2.1}$$

where:

$$\Sigma = \begin{bmatrix} \sigma_z^2 & 0 \\ 0 & \sigma_z^2 \end{bmatrix}, \quad \Sigma^{-1} = \begin{bmatrix} \frac{1}{\sigma_z^2} & 0 \\ 0 & \frac{1}{\sigma_z^2} \end{bmatrix} \tag{2.2}$$

Therefore, the received bits y are modeled as:

$$y = \rho x + z \tag{2.3}$$

Following reception, bit-wise Log Likelihood Ratios (LLRs) of the QPSK symbols are calculated and fed as inputs into the LDPC decoder. The LLRs are calculated by Equation 2.4.

$$\lambda_I = \frac{2y_I \rho \sqrt{E}}{\sigma_z^2}, \quad \lambda_Q = \frac{2y_Q \rho \sqrt{E}}{\sigma_z^2} \tag{2.4}$$

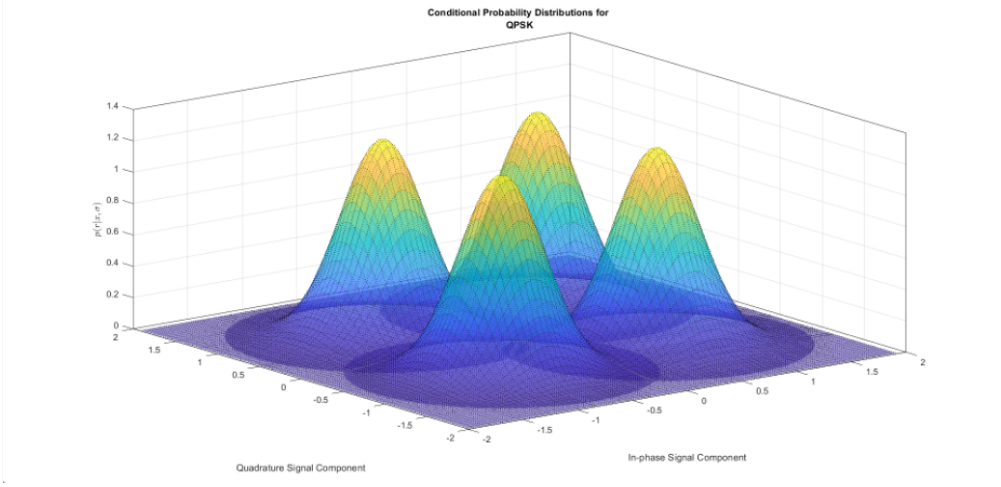


Figure 2.3: Probability Density Function of QPSK Symbol

In the independent architecture, the legacy CDL RF link is assumed to be closed providing a constant data rate of 156.25 megabits per second. For the integrated architecture, the RF symbols are processed by the PBRL LDPC code and transmitted assuming a E_s/N_0 of 4 dB and 7 dB at rate 1/2. If the LDPC code solely consisted of bits to be transmitted over the RF channel, a E_s/N_0 of 4 dB is sufficient to close the link of a rate-1/2 code. At 7 dB, a rate-8/9 link can be closed. However, because the LDPC code consists of bits from both the RF and FSO channels, selecting two different E_s/N_0 values gives insight on how much the reliability of the RF bits contributes to correcting errors on the FSO bits. Figure 2.4 shows the Frame Error Rate (FER) of the LDPC code operating solely on the RF channel for various E_s/N_0.

2.2.2 FSO Channel

Following [WSW05], simulations of the Free Space Optical channel are based on a Gaussian model for the avalanche photodiode detector using On-Off Keying (OOK) modulation. According to this model, each OOK slot contains either the signal or background noise, where both the signal and the background noise are modeled with Gaussian distributions. When

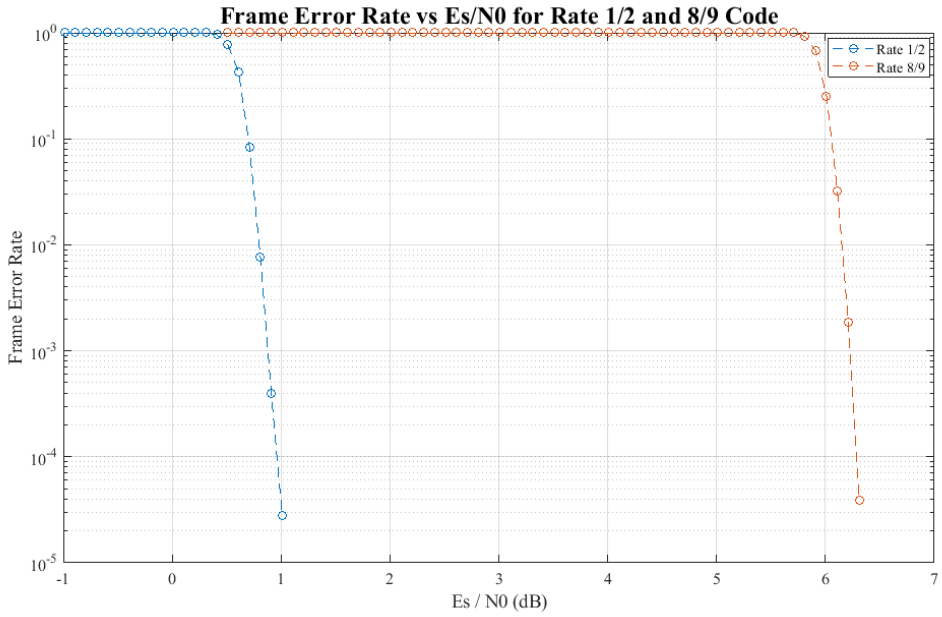


Figure 2.4: Frame Error Rate on the RF channel with QPSK modulation

the signal is present, the mean is μ_s and the variance is σ_s^2. When the signal is not present, the background noise has mean μ_b and the variance is σ_b^2.

The bit log likelihood ratios from OOK used by the LDPC decoder are computed as

$$\lambda = \frac{1}{2} \ln \frac{\sigma_b^2}{\sigma_s^2} + \frac{(y - \mu_b)^2}{2\sigma_b^2} - \frac{(y - \mu_s)^2}{2\sigma_s^2} \,. \tag{2.5}$$

The fading behavior is slow with respect to a codeword length allowing a block fading model for simulation following [Kol13]. The model generates a sequence of independent numbers according to a Gaussian distribution with zero mean and variance of ${\sigma_L}^2 = \log(PSI + 1)$. PSI is the Power Scintillation Index and describes how deep fade scintillations are. Next, these samples are passed through a low pass filter with frequency response given below in order to correlate them in time as shown in Equation 2.6.

$$H(f) = \sqrt{{\sigma_L}^2 \tau_0 \sqrt{\pi}} \exp\left[-\frac{1}{2}(\pi \tau_0 f)^2\right] \tag{2.6}$$

The variable τ_0 is called the turbulence coherence time and describes the time interval during which the fade characteristics change very little. Finally, the output of the low pass filter is passed through a non-linear transformation given below:

$$a_{T_i} = e^{\left(x_i - \sigma_L^2/2\right)} \tag{2.7}$$

Fades are generated at a rate according to the turbulence coherence time and interpolated to provide additional samples. The entire process of generating samples of the fading environment is summarized in Figure 2.5. An example of one realization of the fading channel is shown in Figure 2.6

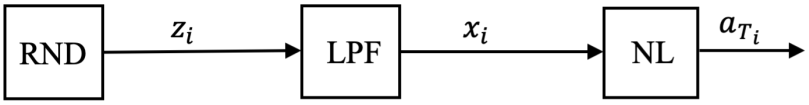


Figure 2.5: Generation of samples from fading environment

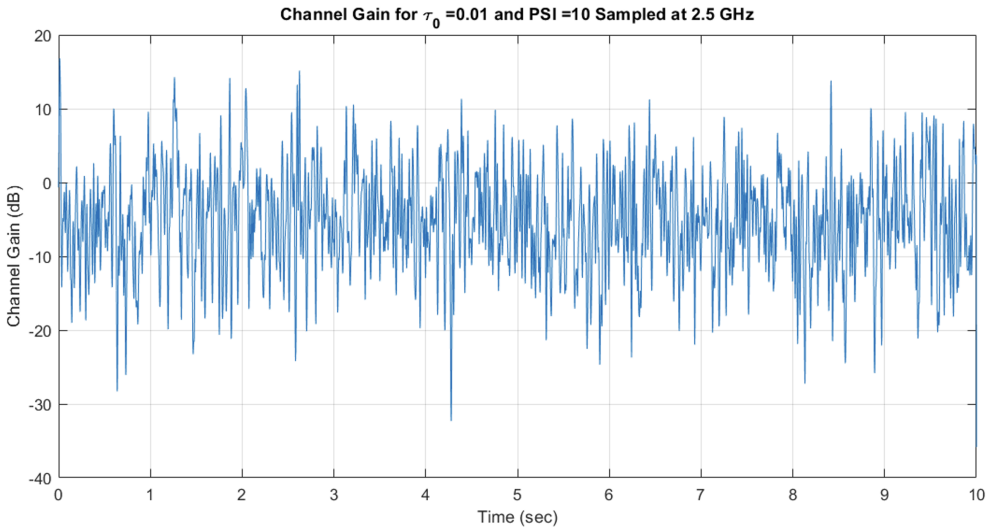


Figure 2.6: FSO Fading Channel Realization

2.3 Selecting LDPC rate to Maximize Throughput

Let g be the fading channel gain in dB and define $g_{1/2}$ and $g_{8/9}$ as the thresholds in channel gain at which the respective codes close the link. Let B be the rate at which coded bits are transmitted through the channel. For FSO OOK, this is the same as the baud rate and independent of the LDPC rate. The approximate throughput T for each of the two LDPC rates can be expressed as follows:

$$T_{\frac{1}{2}} \approx \frac{1}{2} \times B \times P\left(g > g_{1/2}\right) \tag{2.8}$$

$$T_{\frac{8}{9}} \approx \frac{8}{9} \times B \times P\left(g > g_{8/9}\right) \tag{2.9}$$

These equations reveal that $T_{\frac{8}{9}} > T_{\frac{1}{2}}$ whenever $\frac{P\left(g_{1/2} \leq g \leq g_{8/9}\right)}{P\left(g > g_{8/9}\right)} < \frac{7}{9}$. Figure 2.7 shows the empirical probability mass function for each rate. Since each LDPC rate utilizes a different codeword length, they have different fading processes, but very similar histograms. For purposes of analysis, let us assume that $g_{8/9} = 0$ dB and $g_{1/2} = -3$ dB, which are consistent with our choice of baseline power on detector of -53.9 dB. For both histograms we can see that the empirical probability that a fade is in the left red box region $P\left(g_{1/2} \leq g \leq g_{8/9}\right)$ is about 0.1524 and the region in the right red box $P\left(g > g_{8/9}\right)$ is about 0.2227. Since $\frac{0.1524}{0.2227} = 0.6843 < 0.7778 = \frac{7}{9}$, the long-term average throughput would be optimized by the rate-8/9 LDPC code rather than the rate-1/2 LDPC code. Specifically, the rate-8/9 code would provide average throughput of 0.198 bits per FSO symbol while the rate-1/2 code would provide 0.1875 bits per symbol. The actual choice of LDPC coding rate would depend on the specific empirical PMF of fading gains, but the rule that $T_{\frac{8}{9}} > T_{\frac{1}{2}}$ whenever $\frac{P\left(g_{1/2} \leq g \leq g_{8/9}\right)}{P\left(g > g_{8/9}\right)} < \frac{7}{9}$ should guide the choice.

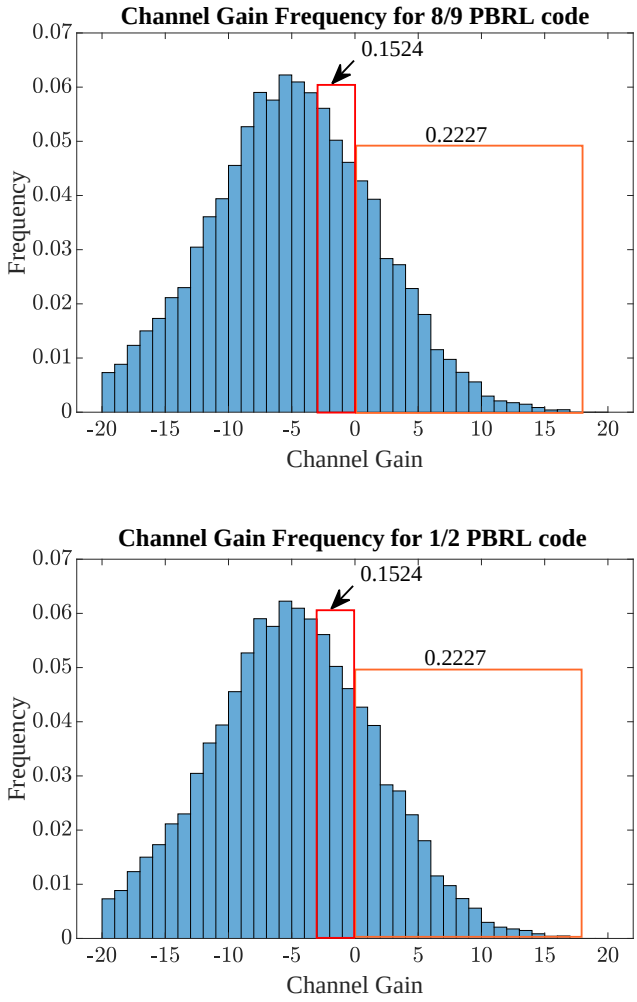


Figure 2.7: Histogram of FSO channel fades taken over 10 seconds.

2.4 Results and Conclusion

Figures 2.8 and 2.9 compares the performance of the independent and integrated RF/FSO modems on an optical fading channel. The top plot shows the instantaneous throughput of the standalone RF modem for both rate-1/2 and rate-8/9. As expected, the rate-1/2 code closes the link more often. However, when the rate-8/9 link is closed, it provides more instantaneous throughput compared to the rate-1/2 code.

The second plot shows the instantaneous throughput of the legacy CDL RF modem on

the RF channel. Recall that this modem is to be used in the independent architecture and is assumed to always close the link. The third plot shows the throughput of the independent architecture. Because the FSO and RF modems operate separately, the throughput of the independent modem is simply the summation of the first two plots.

The fourth and fifth plots from the top show the instantaneous throughput of the proposed integrated modem where RF and FSO bits are jointly encoded using a LDPC code. As noted in Figure 2.4, for a E_s/N_0 of 4 dB, only the rate-1/2 code can close the link on the RF channel. Here, the RF bits cannot provide enough information to support an LDPC code at rate 8/9. Even for the largest FSO gains in this simulation, the hybrid modem cannot close the link, indicating that the RF bits are the decoding bottleneck. For E_s/N_0 of 7 dB, both rates can consistently maintain the RF link. The decoding benefit of the RF link in the integrated architecture is most evident in Figure 2.8 around 30 ms for $E_s/N_0 = 4$ dB and 60 ms for $E_s/N_0 = 7$ dB. Here, the standalone FSO link could not be closed as indicated by the top plot. However, the introduction of reliable RF bits into the decoder allowed for the decoder to, albeit sporadically, recover codewords that were not previously decodeable.

Looking at $E_s/N_0 = 7$ dB, when the optical channel is favorable, both the independent and integrated systems achieve the same data rate of 234.375 megabits per second for the rate-1/2 code. However, when the optical channel is unfavorable, the data on both channels is lost by the hybrid modem while the architecture with two independent modems can rely on the stable RF link to provide 156.25 megabits per second. However, when the optical channel is favorable, the hybrid system has a higher rate of 416.67 megabits per second as compared to 295.14 megabits per second for the independent RF and FSO modems because the RF channel can utilize the efficiency of the rate 8/9 code.

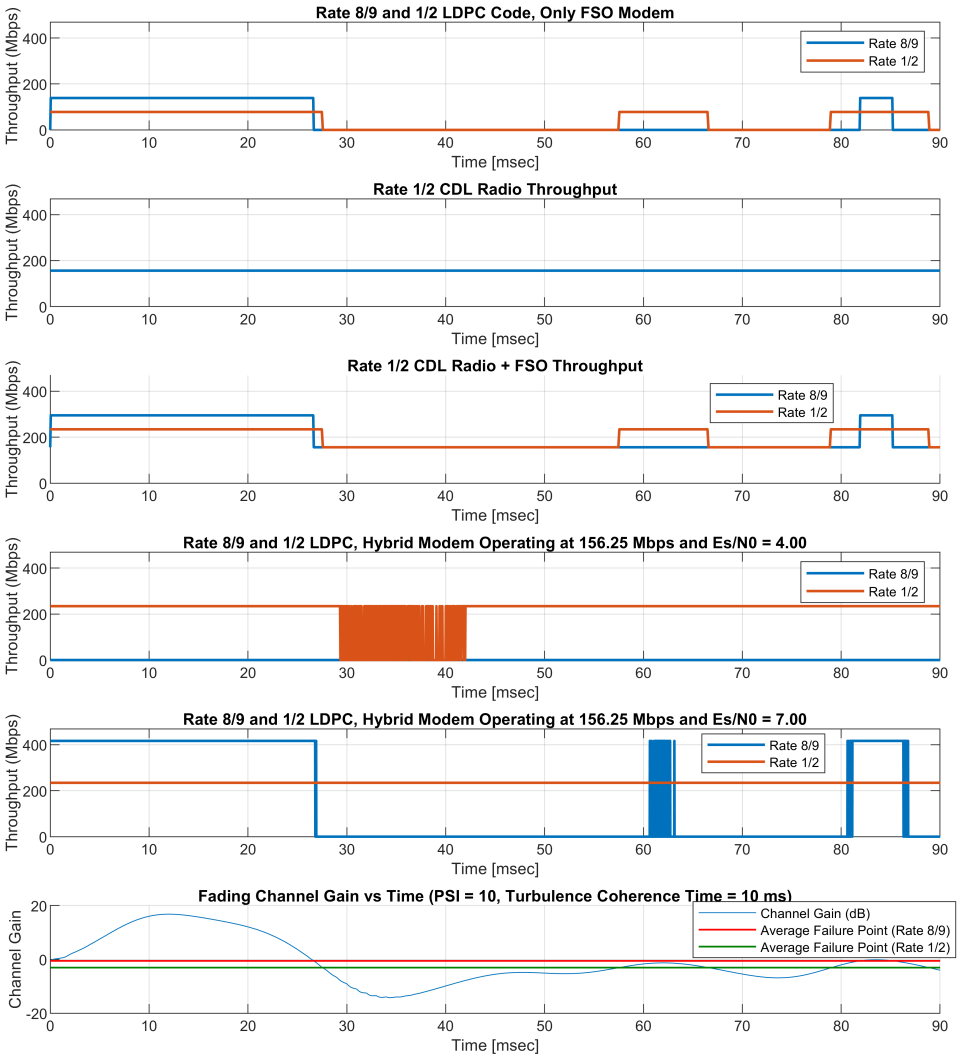


Figure 2.8: Simulation results for FSO Symbol Rate of 156.25 Megabauds per second

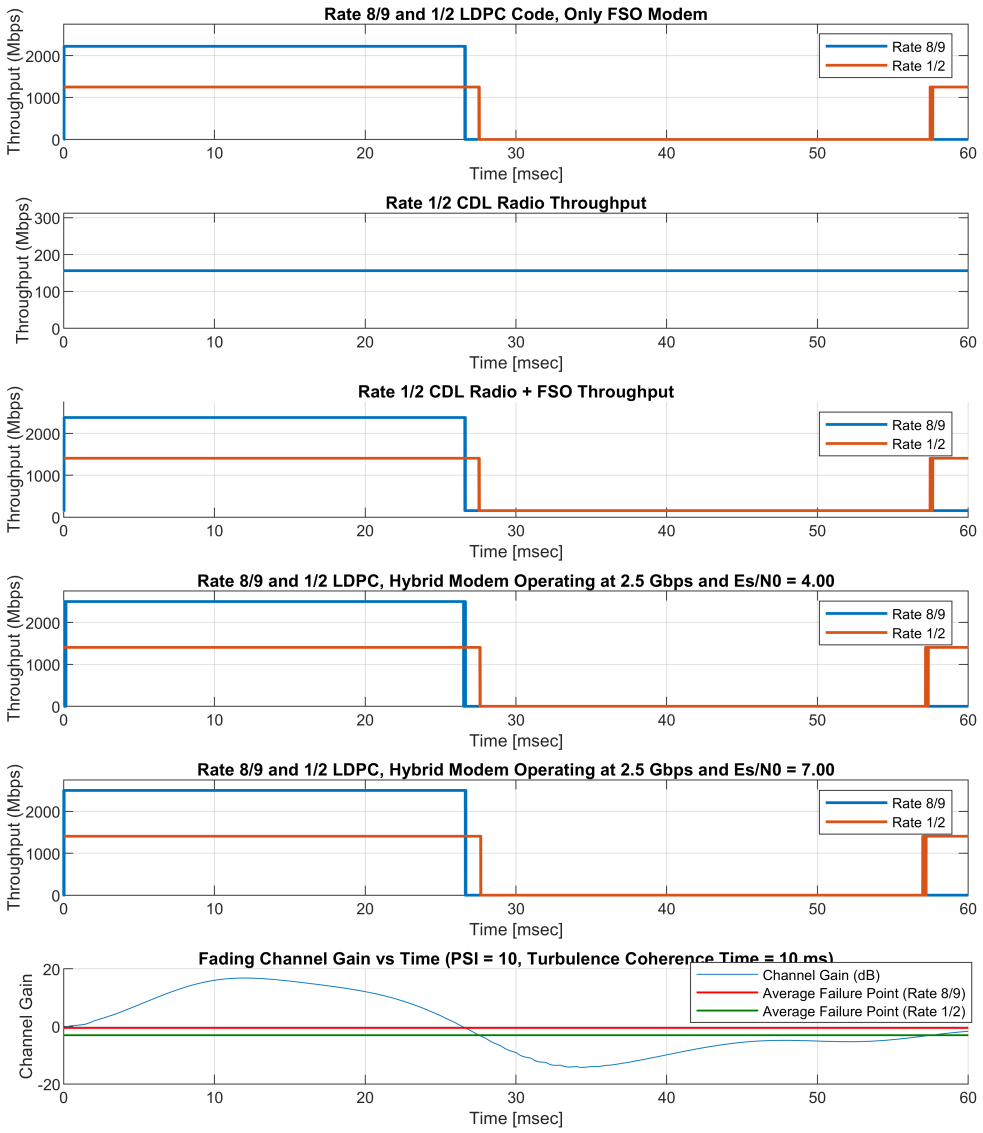


Figure 2.9: Simulation results for FSO Symbol Rate of 2.5 Gigabauds per second

Figures 2.10 and 2.11 show the average throughput of the various proposed designs for FSO symbol rates set to 156.25 megasymbols per second and 2.5 gigasymbols per second respectively. The two architectures achieve similar throughputs in most scenarios. However, for the low baud rate (156.25 Mbps) FSO system, the 4 dB RF channel forced throughput to zero for the rate-8/9 integrated RF/FSO architecture. Independent RF and FSO modems provide good throughput in every scenario while also providing the advantage of maintaining a reliable RF link rather than losing all communication during the optical fades that cause link failure in the integrated architecture. The essential throughput gain from utilizing both an RF and FSO channel can be obtained by deploying independent modems.

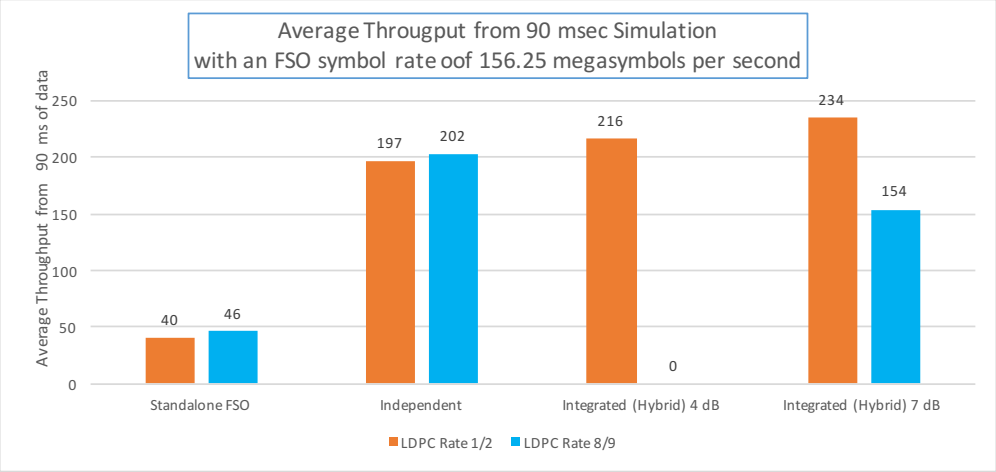


Figure 2.10: Average throughput from 90 msec simulation for the various systems with the FSO symbol rate set to 156.25 megasymbols per second.

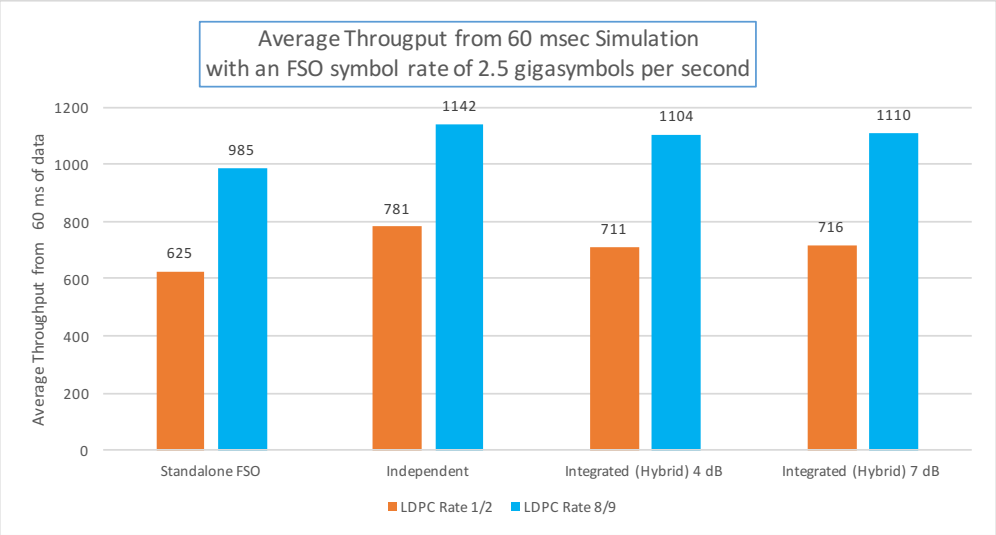


Figure 2.11: Average throughput from 60 msec simulation for the various systems with the FSO symbol rate set to 2.5 gigasymbols per second.

2.5 Acknowledgement

The majority of this chapter has been published or has been submitted for publication in [NLW20]. This research was conducted in collaboration with Linfang Wang, Ethan Liang, Todd Chauvin, Todd Drullinger, Dariush Divsilar, and Richard D. Wesel. I constructed the hybrid LDPC decoder which interleaved bits between the two channels and ran both fading and Frame Error Rate simulations. I would like to thank Linfang Wang and Ethan Liang for providing the avalanche photodiode detector model and base LDPC decoder. Lastly, I thank Todd Chauvin and Todd Drullinger at SA Photonics for providing the fading model and decoder specifications.

CHAPTER 3

Finding Optimal LDPC Message Passing Weights via Neural-Networks

3.1 Introduction

Message passing decoders including belief propagation (BP and Min-Sum alongside its variations are often used to decode Low Density Parity Check (LDPC) codes. These message passing algorithms are attractive because their complexity does not scale exponentially with block-length. This is due in part to the iterative nature of message passing along with the sparsity of LDPC parity check matrices. In practice, message passing decoders are sub-optimal because of the existence of cycles in the corresponding Tanner graph. These cycles break the assumption of extrinsic information when transmitting messages and lead to early error floors.

Recently, numerous works have proposed improving message passing decoders with neural networks [NBB16, LG17, NMB17, NML18, LSW18, WJZ18, LG18, LZJ18, XVT19, DB19, ABS19, BHP20, WWF20, LCH19]. The neural network is created by unfolding the message passing operations of each decoding iteration [NBB16].

Nachmani *et al.* in [NBB16] proposed improving BP decoding by assigning unique multiplicative weights to check-to-variable messages and the channel log-likelihood (LLR) of variables in each iteration. This so-called "Neural BP (NBP)" showed better performance than BP. Nachmani *et al.* and Lugosch *et al.* in [NML18, LG17, NBB16] proposed a Neural Normalized MinSum (N-NMS) decoder and Neural Offset MinSum (N-OMS) decoder to

improve the performance of the NMS and OMS decoder.

For longer block lengths, these edge-specific neural decoders become impractical because the number of edges scales rapidly. A possible solution is to share one parameter across all edges that have some similar property such as across the same iteration, or connecting to the same check/variable node. For an example, Wang *et al.* proposed to assign the same parameters for each check-to-variable layer and variable-to-check layer [WWF20], respectively. M. Lian *et. al.* in [LCH19] considered assigning same weight to all messages in one iteration.

With these previous papers, the focus on short block length codes ($N < 1000$) may have resulted from the fact that popular deep learning research platforms, such as Pytorch and Tensorflow, require impractical amounts of memory to calculate the gradient when the block length is long. However, as demonstrated in [ABS19], it is possible to train parameters for longer block lengths if resources are handled more efficiently. Abotabl *et al.* provided an efficient computation framework for optimizing the offset values in the N-OMS algorithm[ABS19], and trained an OMS neural network with edge-specific weights, iteration-specific weights, and a single weight.

This chapter discusses a family of neural 2-dimensional normalized MinSum (N-2D-NMS) decoders whose weights are optimized by a neural network based on node degree. This simplification over previous approaches that optimize the weights based on node degree leads to a much simpler optimization that provides excellent FER performance while still accommodating large block lengths of practical importance. The main contributions in this paper are:

- An efficient implementation of the N-NMS architecture. This part is related to the framework in [ABS19]. We showed that the memory issued for training long LDPC code faced by Tensorflow [Lug18] can be mitigated by efficiently storing the check-to-variable node and variable-to-check node messages. Separately, back propagation memory is also reduced by storing the gradient with respect to check-to-variable node

and variable-to-check node messages only for the previous iteration rather than all iterations.

- Empirical N-NMS results that show the dynamic weights exhibit a strong correlation with check node degree, variable node degree, and iteration.

- A family of N-2D-NMS decoders with various reduced parameter sets showing how performance varies with the parameter set selected. The N-2D-NMS decoding structure is a generalization of [JFD05] to allow variation with iteration. Simulation results on a (3096,1032) PBRL code show that N-2D-NMS decoder can achieve the same FER as N-NMS with significantly fewer parameters. A N-2D-NMS decoder trained on the (16200,7200) DVBS-2 LDPC code achieves a lower error floor than belief propagation.

- A hybrid decoding structure that combines a feedforward and recurrent structure that shows similar decoding performance as a full feedforward structure, but requires significantly fewer parameters.

3.2 Efficient implementation of N-NMS

3.2.1 Forward Propagation

Let H be the parity check matrix of an (n, k) LDPC code, where n and k represent the codeword length and dataword length, respectively. We use v_i and c_j to denote the i^{th} variable node and j^{th} check node, respectively. In each iteration, an NMS decoder uses the same constant value to scale all check-to-variable node messages, whereas an N-NMS decoder assigns distinct multiplicative parameters for each check-to-variable message in each iteration. In the t^{th} *decoding iteration*, N-NMS updates the check-to-variable node message, $u^{(t)}_{c_j \to v_i}$, the variable-to-check node message, $l^{(t)}_{v_i \to c_j}$, and posterior of each variable node, $l^{(t)}_{v_i}$,

by:

$$u_{c_i\to v_j}^{(t)} = \beta_{(c_i,v_j)}^{(t)} \times \prod_{v_{j'}\in N(c_i)/\{v_j\}} \operatorname{sgn}(l_{v_{j'}\to c_i}^{(t-1)}) \times \min_{v_{j'}\in N(c_i)/\{v_j\}} \left|(l_{v_{j'}\to c_i}^{(t-1)})\right| , \tag{3.1}$$

$$l_{v_j\to c_i}^{(t)} = l_{v_i}^{ch} + \sum_{c_{i'}\in N(v_j)/\{c_i\}} u_{c_{i'}\to v_j}^{(t)}, \tag{3.2}$$

$$l_{v_j}^{(t)} = l_{v_i}^{ch} + \sum_{c_{i'}\in N(v_j)} u_{c_{i'}\to v_j}^{(t)}. \tag{3.3}$$

$N(c_i)$ represents the set of the variable nodes that are connected to c_i and $N(v_j)$ represents the set of the check nodes that are connected to v_j. $l_{v_j}^{ch}$ is the LLR of channel observation of v_j. $\beta_{(c_i,v_j)}^{(t)}$ are multiplicative weights to be trained. The decoding process stops when all parity checks are satisfied or maximum iteration I_T is reached.

3.2.2 Backward Propagation

In this subsection, we derive the gradient of J with respect to the trainable weights, $\frac{\partial J}{\partial \beta_{(v_i,c_j)}^{(t)}}$, the check-to-variable message, $\frac{\partial J}{\partial u_{c_i\to v_j}^{(t)}}$, and variable-to-check message, $\frac{\partial J}{\partial l_{v_j\to u_i}^{(t)}}$. We show that in order to calculate the desired gradients, it is sufficient only to store, $l_{v_i}^{(t)}$,$\operatorname{sgn}(l_{v_j\to c_i}^{(t)})$, $\operatorname{sgn}(u_{c_i\to v_j}^{(t)})$, $\texttt{min1}_{c_i}^t$, $\texttt{min2}_{c_i}^t$, $\texttt{pos1}_{c_i}^t$ and $\texttt{pos1}_{c_i}^t$ when performing forward propagation, where

$$\texttt{min1}_{c_i}^t = \min_{v_{j'}\in N(c_i)} |l_{v_{j'}\to c_i}^{(t)}|, \tag{3.4}$$

$$\texttt{pos1}_{c_i}^t = \operatorname*{argmin}_{v_{j'}\in N(c_i)} |l_{v_{j'}\to c_i}^{(t)}|, \tag{3.5}$$

$$\texttt{min2}_{c_i}^t = \min_{v_{j'}\in N(c_i)/\{\texttt{pos1}_{c_i}^t\}} |l_{v_{j'}\to c_i}^{(t)}|, \tag{3.6}$$

$$\texttt{pos2}_{c_i}^t = \operatorname*{argmin}_{v_{j'}\in N(c_i)/\{\texttt{pos1}_{c_i}^t\}} |l_{v_{j'}\to c_i}^{(t)}|. \tag{3.7}$$

In this paper, multi-loss cross entropy [NBB16] is used as loss function. In iteration t,

$\frac{\partial J}{\partial l_{v_j \to u_i}^{(t)}}$ is updated as follows:

$$\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}} = \frac{\partial J}{\partial l_{v_j}^{(t)}} + \sum_{c_{i'} \in N(v_j)/\{c_i\}} \frac{\partial J}{\partial l_{v_j \to c_{i'}}^{(t)}}. \tag{3.8}$$

$\frac{\partial L}{\partial \beta_{c_i \to v_j}^{(t)}}$ is calculated by:

$$\frac{\partial J}{\partial \beta_{c_i \to v_j}^{(t)}} = u_{c_i \to v_j}^{(t)*} \frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}, \tag{3.9}$$

where

$$u_{c_i \to v_j}^{(t)*} = \text{sgn}(u_{c_i \to v_j}^{(t)*}) \times |u_{c_i \to v_j}^{(t)*}|, \tag{3.10}$$

$$\text{sgn}(u_{c_i \to v_j}^{(t)*}) = \prod_{v_{j'} \in N(c_i)/\{v_j\}} \text{sgn}(l_{v_{j'} \to c_i}^{(t-1)}), \tag{3.11}$$

$$|u_{c_i \to v_j}^{(t)*}| = \begin{cases} \texttt{min2}_{c_i}^t, & \text{if } v_j = \texttt{pos1}_{c_i}^t \\ \texttt{min1}_{c_i}^t, & \text{otherwise} \end{cases}. \tag{3.12}$$

With chain rule, we obtain the following for $\frac{\partial J}{\partial |u_{c_i \to v_j}^{(t)*}|}$:

$$\frac{\partial J}{\partial |u_{c_i \to v_j}^{(t)*}|} = \text{sgn}(u_{c_i \to v_j}^{(t)*}) \frac{\partial J}{\partial u_{c_i \to v_j}^{(t)*}}, \tag{3.13}$$

$$= \text{sgn}(u_{c_i \to v_j}^{(t)*}) \beta_{(c_i, v_j)}^{(t)} \frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}. \tag{3.14}$$

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

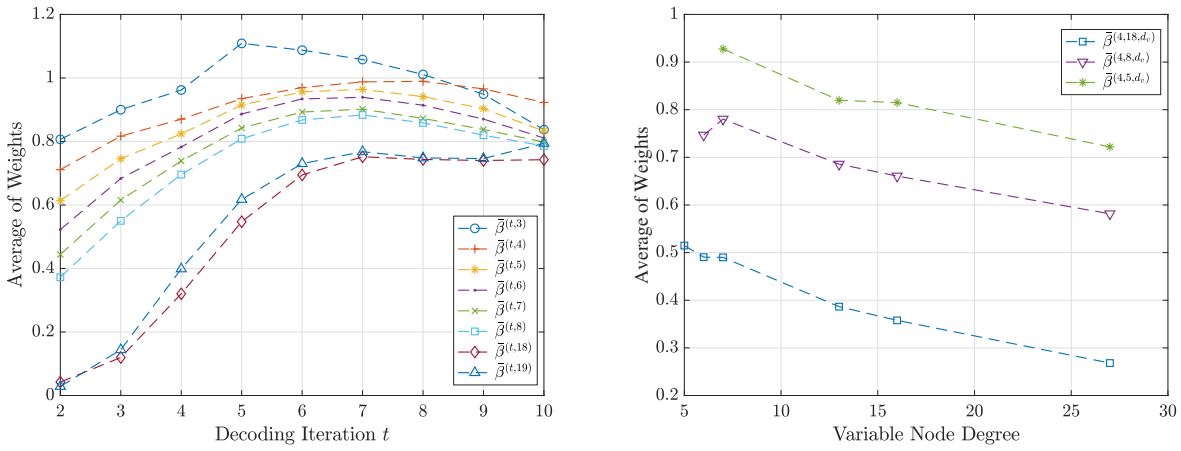
$$\begin{cases} \text{sgn}(l_{v_j \to c_i}^{(t-1)}) \sum_{v_{j'} \in N(c_i)/v_j} \frac{\partial J}{\partial |u_{c_i \to v_{j'}}^{(t)*}|} & , v_j = \texttt{pos1}_{c_i}^{(t)} \\ \text{sgn}(l_{v_j \to c_i}^{(t-1)}) \frac{\partial J}{\partial \left| u_{c_i \to \texttt{pos1}_{c_i}^{(t)}}^{(t)*} \right|} & , v_j = \texttt{pos2}_{c_i}^{(t)} \\ 0 & , \text{otherwise.} \end{cases} \tag{3.15}$$

Eqs. (3.8-3.15) indicate that $\frac{\partial J}{\partial u_{c_i \to v_j}^{(t)}}$ and $\frac{\partial J}{\partial l_{v_j \to c_i}^{(t)}}$ can be calculated iteratively. Therefore, back propagation does not need to store the gradients with respect to $u_{c_i \to v_j}$ and $l_{v_j \to c_i}$ for all

iterations. Both Pytorch and Tensorflow store all iterations of $u^{(t)}_{c_i \to v_j}$, $l^{(t)}_{v_j \to c_i}$ and $l^{(t)}_{v_j}$, making them inefficient for this purpose. However, we showed that the neuron values in each hidden layer can be stored compactly using the parameters $l^{(t)}_{v_i}$, $\mathrm{sgn}\left(l^{(t)}_{v_j \to c_i}\right)$, $\mathrm{sgn}\left(u^{(t)}_{c_i \to v_j}\right)$, $\texttt{min1}^t_{c_i}$, $\texttt{min2}^t_{c_i}$, $\texttt{pos1}^t_{c_i}$ and $\texttt{pos1}^t_{c_i}$, which results in a significant reduction in storage requirements. Using these two strategies, we resolve the Tensorflow storage obstacle identified by [Lug18].

3.2.3 Simulation Results

In this subsection, we use the efficient implementation described above to train the weights of N-NMS for a (3096,1032) protograph-based raptor-like (PBRL) LDPC code. The code we use is taken from [cls] (in [CVD15]). Encoded bits x are modulated by binary phase shift keying (BPSK) and transmitted through Additive White Gaussian Noise (AWGN) Channel. The N-NMS decoder is updated on a flooding schedule and the maximum number decoding iterations is 10. Define $\beta^{(t,d_c)} = \{\beta^{(t)}_{(c_i,v_j)} | \deg(c_i) = d_c\}$ and $\bar{\beta}^{(t,d_c)}$ as the mean value of $\beta^{(t,d_c)}$.



(a) (b)

Figure 3.1: Mean values of messages of FNNMS for a (3096,1032) PBRL code in each iteration show strong correlations to check and variable node degree.

Fig.3.1a shows $\bar{\beta}^{(t,d_c)}$ versus decoding iteration t with all possible check node degrees. Note

that the iteration number starts at 2 because most of edges have 0 valued messages in the first iteration as result of puncturing. The simulation shows a clear relationship between check node degree and $\bar{\beta}$, i.e. larger check node degrees correspond to smaller $\bar{\beta}$. This difference is significant in the first few iterations. Additionally, $\bar{\beta}^{(t,d_c)}$ changes significantly in first few iterations for all check node degrees d_c.

In order to investigate the relationship between weights and variable node degree given a check node degree and iteration number, we further define $\beta^{(t,d_c,d_v)} = \{\beta^{(t)}_{(c_i,v_j)} | \deg(c_i) = d_c, \deg(v_i) = d_v\}$. We denote $\bar{\beta}^{(t,d_c,d_v)}$ to be the average value of $\beta^{(t,d_c,d_v)}$. Fig.3.1b gives the average value of weights corresponding to various check and variable node degrees at iteration 4. Simulation results show that, given a specific iteration t' and check node degree d'_c, larger d'_v correspond to smaller $\bar{\beta}^{(t',d'_c,d'_v)}$.

In conclusion, the weights of N-NMS are correlated with check node degree, variable node degree, and iteration. Thus, node degrees should affect the weighting of messages on their incident edges when decoding irregular LDPC codes. Inspired by recent neural network decoders, we propose a family of N-2D-NMS decoders in this paper.

3.3 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 section, we propose neural 2-dimensional normalized MimSum (N-2D-NMS) decoders which has the following form:

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

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

Type	$\beta_*^{(t)}$	$\alpha_*^{(t)}$	The number of Required Parameters per Iteration	
			(16200,7200) DVBS-2 code	(3096,1032) PBRL code
No Weight Sharing				
0 [1]	$\beta_{(c_i,v_j)}^{(t)}$	1	$4.8 * 10^5$	$1.60 * 10^4$
Weight Sharing Based on Node Degree				
1	$\beta_{(d.(c_i),d.(v_j))}^{(t)}$	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 [19]	$\beta_{(\lfloor \frac{i}{f} \rfloor, \lfloor \frac{j}{f} \rfloor)}^{(t)}$	1	–	101
6	$\beta_{(\lfloor \frac{i}{f} \rfloor)}^{(t)}$	1	–	17
7	1	$\alpha_{(\lfloor \frac{j}{f} \rfloor)}^{(t)}$	–	25
Weight sharing based on Iteration Lian2019-jh,Abotabl2019-wt				
8	$\beta^{(t)}$	1	1	1

Table 3.1: Various N-2D-NMS Decoders and Required Number of Parameters per Iteration

$\beta_*^{(t)}$ and $\alpha_*^{(t)}$ are the multiplicative weights assigned to check and variable node messages, respectively. Table 3.1 lists different types of N-2D-NMS decoders, each identified in the first column by a type number. As a special case, we denote N-NMS as type 0. Columns 2 and 3 describe how each type assigns $\beta_*^{(t)}$ and $\alpha_*^{(t)}$, respectively. The subscript * is replaced in Table 3.1 with the information needed to identify the specific weight depending on the weight sharing methodology.

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 variable node degree and check node degree, respectively.

Dai. *et. al* in [DTS21] studied weight sharing based on the edge type of 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 edges with the same edge type, i.e. 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 parameter for each iteration, as in [LCH19, ABS19].

To further reduce the number of parameters, we consider a hybrid training structure that utilizes a neural network combining a feedforward module with a recurrent module . The corresponding decoder uses distinct trained parameters for each of the first I' decoder iterations and reuses the same parameters for the remaining $I_T - I'$ iterations. The motivation for the hybrid decoder is that the values of the trainable parameters change negligibly during the last few iterations, as illustrated in Sec. 3.4. Therefore, using the same parameters for the last few iterations doesn't cause a large performance degradation.

A (3096,1032) PBRL code and the (16200,7200) DVBS-2[ETS19] standard LDPC code are considered in this paper, and the number of parameters per iteration required for various N-2D-NMS decoders of these two codes are listed in column 4 and 5 in Table. 3.1, respec-

tively. It is shown that the number of parameters required by node-degree based weight sharing is less than that required by protomatrix based weight sharing.

3.4 Simulation Results

In this section, we investigate the decoding performance of various N-2D-NMS decoders for LDPC codes with different block lengths. All encoded bits are modulated by BPSK and transmitted through the AWGN channel.

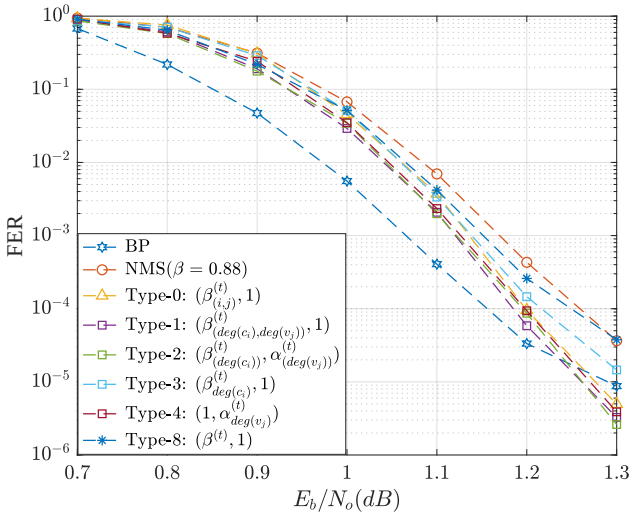


Figure 3.2: FER performances of various LDPC decoder for (16200,7200) DVBS-2 LDPC code.

3.4.1 (16200,7200) DVBS-2 LDPC code

Fig. 3.2 gives the FER performances of various LDPC decoder for (16200,7200) DVBS-2 LDPC code. All of the decoders are *flooding* scheduled and maximum decoding iteration is 50. It is shown that N-NMS decoder outperforms BP at $1.3dB$, with a lower error floor. The N-2D-NMS decoders of types 1 and 2 have a slightly better performance than N-NMS. Type 4 outperforms type 3, because the variable node weights of investigated code have a larger dynamic range than check node weights, as shown in Fig. 3.2.

Fig. 3.3 and 3.4 show 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, the weights change negligibly after iteration 20. Thus, the hybrid N-2D-NMS decoder of type 2 with $I' = 20$ delivers comparable performance to the full feedforward decoding structure, as shown in Fig. 3.4a. Fig. 3.4b shows that the parameters of type 8 converge to 0.885, which is close to the single weight of NMS decoder. As shown in Fig. 3.2, by only assigning iteration-specific parameters, N-2D-NMS decoder of type 8 appears an early error floor at 1.20 dB.

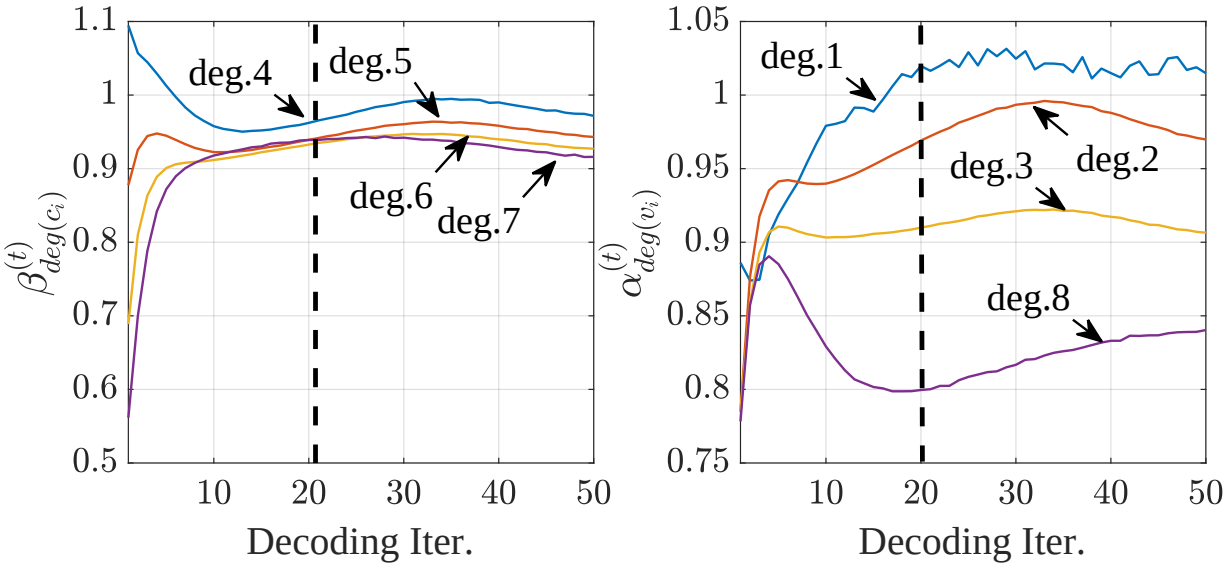
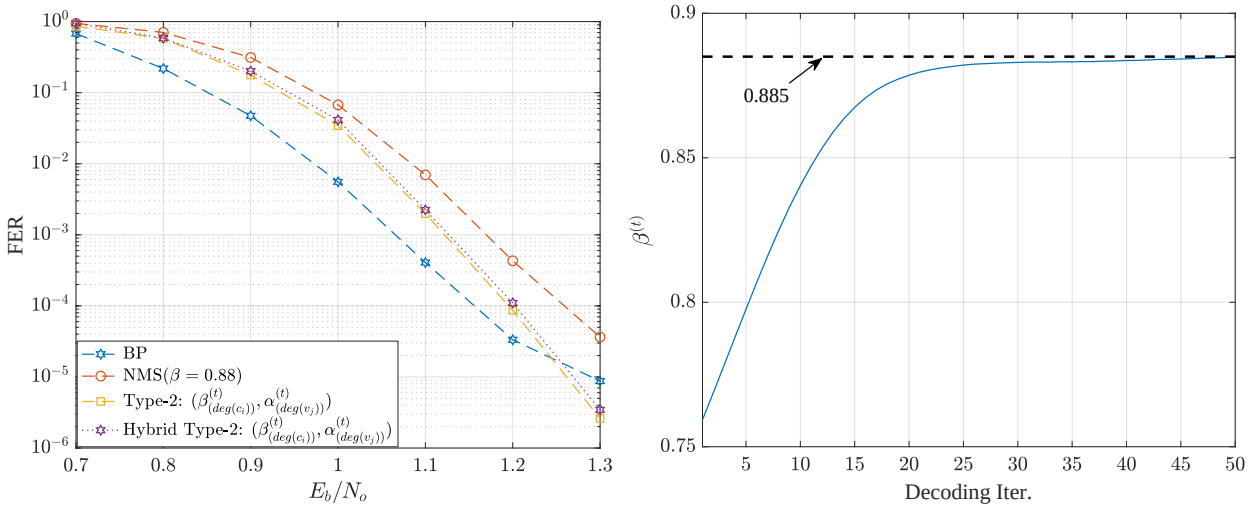


Figure 3.3: The weights of the Type-2 N-2D-NMS decoder for the (16200,7200) DVBS-2 LDPC code only change significantly in the first 20 iterations.

3.4.2 (3096,1032) PBRL LDPC Code

Fig. 3.5 compares the FER performance of various N-2D-NMS decoders with the N-NMS (type 0) and NMS. All of the decoders are implemented using a layered schedule with a maximum of 10 decoding iterations. The simulation results show that N-NMS has more than 0.5 dB improvement over the NMS decoder. N-2D-NMS decoders of types 1-7 are also simulated. Note that types 1, 2 and 5 have the same decoding performance as the N-NMS decoder, but the number of parameters is reduced by 99.7%, 99.9% and 99.3%, respectively. Thus, weight-sharing metrics based on check and variable node degree, or based on horizontal and vertical layer deliver lossless performance with respect to N-NMS. N-2D-NMS decoders of types 4 and 6 have a degradation of around 0.05 dB compared to N-NMS. N-2D-NMS decoders of types 5 and 7 have a degradation of around 0.2 dB compared with N-NMS. Thus, for the (3096,1032) PBRL code of Fig. 3.5, assigning weights based only on check nodes can gain more benefit than assigning weights only based on variable node.



(a) (b)

Figure 3.4: Fig. (a) illustrates that the Hybrid Type-2 N-2D-NMS decoder with $I' = 20$ shows comparable decoding performance to the full feedforward decoder. Fig.(b) shows that the weights of the Type-8 N-2D-NMS decoder for the (16200,7200) DVBS-2 LDPC code converge to 0.885.

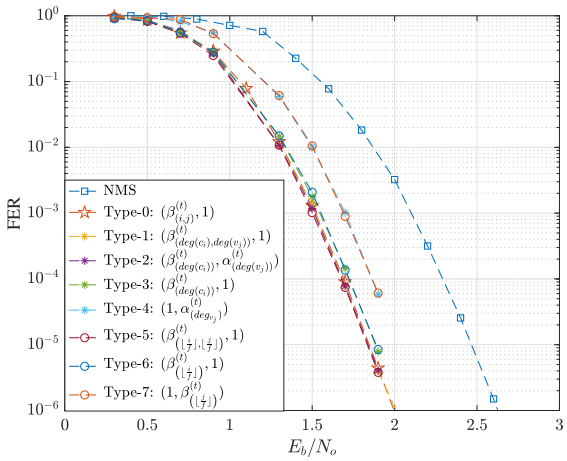


Figure 3.5: FER performance for various N-2D-NMS decoders for a (3096,1032) PBRL LDPC code compared with N-NMS (type 0) and NMS.

3.5 Conclusion

This chapter investigates MinSum LDPC decoders for which the normalization weights are optimized by a neural network. An initial neural network assigns a different weight to every edge. The statistics of the trained parameters show that the trained parameters depend on node degree. In particular, the trained weights have a smaller value for the neurons corresponding to a larger check/variable node degree. Neural 2D normalized MinSum (N-2D-NMS) decoders are introduced in this chapter with various weight-sharing techniques to reduce the number of parameters that must be trained. Simulation results on the (16200,7200) DVBS-2 standard LDPC code and a (3096,1032) PBRL code show that the N-2D-NMS decoder achieves comparable decoding performance to a N-NMS decoder but with dramatically fewer trained parameters. Furthermore, N-2D-NMS decoders can achieve a lower error floor than BP for some LDPC codes. Finally, this chapter proposes a hybrid neural network with both feedforward and recurrent modules for further parameter reduction.

3.6 Acknowledgement

The majority of this chapter has been published or has been submitted for publication in [WCN21]. This research was conducted in collaboration with Linfang Wang, Sean Chen, Dariush Divsilar, and Richard D. Wesel. I would like to thank Linfang Wang for leading the project, creating the N-NMS decoders, and for his derivations of the training process. Sean Chen trained several of the Neural Networks. I also trained several Neural Networks and produced simulations showing that the optimal training SNR is slightly lower than the testing SNR.

CHAPTER 4

Comparing Viterbi Decoding and Neural-Network Optimized Message Passing on the CCSDS Line Product Code

Line codes describe a set of encoding maps used to transmit digital data. The primary purpose of a line code is to manage the disparity of a transmission, which is defined as the difference between the number of transmitted 1s and 0s. Managing bit disparity has the benefit of minimizing DC components in transmissions which cannot be reliably transmitted over most long-distance communication channels. This chapter references the Consultative Committee for Space Data Systems' (CCSDS) 141.11-O-1 proposed line code, known as the Line Product Code (LPC) [boo08].

4.1 Introduction

While often impractical since their complexity scales at a rate of $O(2^n)$, maximum likelihood (ML) decoders represent the best possible decoding performance. Previous work including [Wol78] and [PHB98] have proposed methods of reducing the complexity of ML decoding for linear block codes by representing them as a trellis and performing Viterbi decoding. While still on the order of $O(2^n)$, these methods drastically reduce number of required operations, enough so that for a short blocklength code such as the LPC, ML decoding is considered.

Message passing algorithms, such as belief propagation or MinSum, are low-complexity

iterative decoders for linear block codes. However, message passing algorithms are sub-optimal because they assume that the Tanner graph defined by the parity check matrix has no cycles. As a result, for short block length codes with short cycles, message passing decoders do not provide satisfying performance.

Recently, numerous works have focused on improving the performance of message passing decoders with the aid of neural networks [NBB16, LG17, NMB17, NML18, LSW18, WJZ18, LG18, LZJ18, WCN21]. Nachmani *et al.* and Lugosch *et al.* in [NML18, LG17, NBB16] proposed Neural Normalized MinSum (N-NMS) and Neural Offset MinSum (N-OMS) decoders to improve the performance of the NMS and OMS decoders. Unlike NMS and OMS, which use a constant multiplicative or offset weight, N-NMS and N-OMS assign distinct trainable weights to each edge in each iteration. Simulations in [NML18, LG17, NBB16] show that N-NMS and N-OMS have the capability to drastically improve the decoding performance of NMS and OMS for short-blocklength codes. Therefore, this paper aims to reformulate the LPC as a linear block code to leverage the recent advancements in neural network-based decoders.

4.2 Line Product Code Encoding

The LPC encoder operates on blocks of 25 bits denoted by $LPCEncIn$[24:0]. The most significant bit $LPCEncIn$[24] is channel system data denoted as S. The LPC encoder discards $LPCEncIn$[23:16] (legacy implementation of the laser communication terminal encoding process), and maps $LPCEncIn$[15:0] to the following 4×4 matrix:

$u(0,0)$	$u(0,1)$	$u(0,2)$	$u(0,3)$
u(1,0)	u(1,1)	u(1,2)	u(1,3)
u(2,0)	u(2,1)	u(2,2)	u(2,3)
u(3,0)	u(3,1)	u(3,2)	u(3,3)

The LPC encoder generates the codewords of 24 bits. The codewords along with S forms

a 5×5 matrix:

$e^*(0,0)$	$e^*(0,1)$	$e^*(0,2)$	$e^*(0,3)$	$ph(0)$
$e^*(1,0)$	$e^*(1,1)$	$e^*(1,2)$	$e^*(1,3)$	$ph(1)$
$e^*(2,0)$	$e^*(2,1)$	$e^*(2,2)$	$e^*(2,3)$	$ph(2)$
$e^*(3,0)$	$e^*(3,1)$	$e^*(3,2)$	$e^*(3,3)$	$ph(3)$
$pv(0)$	$pv(1)$	$pv(2)$	$pv(3)$	S

The encoding of LPC consists of the following steps:

1. Calculate $e(i,j)$ $(i,j = 0,\ldots,3)$ using $LPCEncIn$[15:0] via differential encoding. In particular, we refer to $\{e(i,j)|i = 0,1,j = 0,\ldots,3\}$ as sub-block 1 and $\{e(i,j)|i = 2,3,j = 0,\ldots,3\}$ as sub-block 2.

2. Calculate the horizontal parity bits $ph(i)$ $(i = 0,\ldots,3)$ and vertical parity bits $pv(i)$ $(i = 0,\ldots,3)$.

3. Apply bit-wise inversion of sub-block 1 and/or 2 in order to minimize difference between the number of transmitted ones and zeros, which is also referred as disparity. We denote $e^*(i,j)$ $(i,j = 0,\ldots,3)$ as the sub-block bits after inversion process.

The following section describes these three steps in detail.

4.2.1 Differential Encoding

Differential encoding is performed on the two sub-blocks separately. For sub-block 1, initialize $e(0,0) = u(0,0)$ and $e(1,0) = u(1,0) \oplus e(0,3)$:

$$e(i,j) = u(i,j) \oplus e(i,j-1), \quad j > 0 \tag{4.1}$$

For sub-block 2, initialize $e(2,0) = u(2,0)$ $e(2,0) = u(2,0) \oplus e(0,3)$ as shown in Fig. 4.1. The remaining bits can be derived using equation (4.1). Fig. 4.1 shows the differential encoding on sub-block 1.

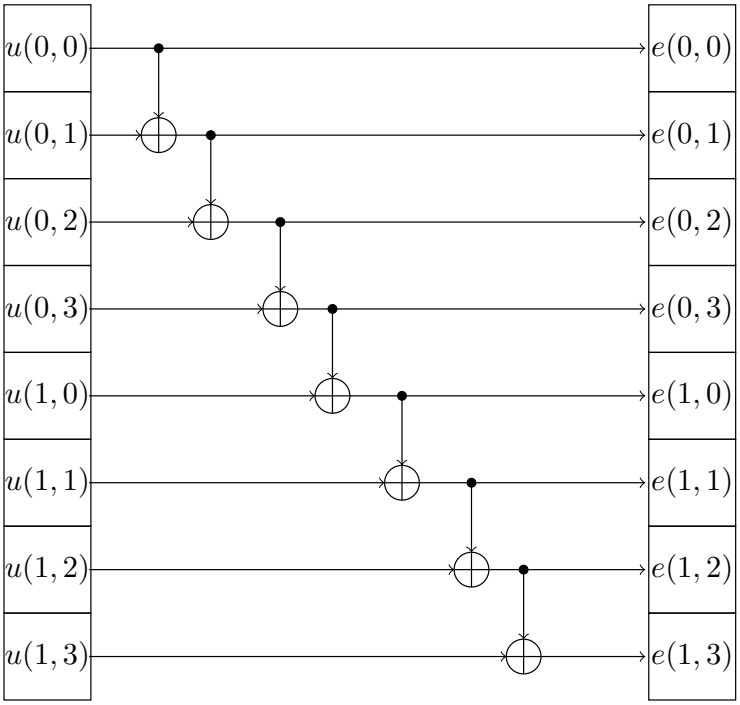


Figure 4.1: Differential encoding to calculate sub-block 1.

where $\oplus$ = logical XOR

4.2.2 Horizontal Parity bits

After calculating the $\{e(0,0) \ldots e(3,3)\}$ bits, the horizontal and vertical parity bits must be determined. The horizontal parity bit $ph(0)$ is always calculated for odd parity of the first row, meaning

$$\left[ph(0) + \sum_{k=0}^{3} e(0,k)\right] \text{mod} 2 = 1 \tag{4.2}$$

The other three parity bits $ph(1)$, $ph(2)$, and $ph(3)$ are determined not only by their

corresponding rows, but also by $e(0,0)$. Specifically, if $e(0,0) = 0$, then odd parity is used for $ph(1)$, $ph(2)$, and $ph(3)$ and its corresponding rows. Otherwise, the even parity must be satisfied. Therefore,

$$\left[ph(i) + \sum_{k=0}^{3} e(i,k)\right] \text{mod} 2 = 1 \oplus e(0,0),\ i = 1,2,3. \tag{4.3}$$

4.2.3 Vertical Parity Bits

The vertical parity bit $pv(0)$ is always calculated for even parity of the first row, meaning that

$$\left[pv(0) + \sum_{k=0}^{3} e(k,0)\right] \text{mod} 2 = 0 \tag{4.4}$$

The other vertical parity bits, $pv(1)$, $pv(2)$, and $pv(3)$ are calculated using their corresponding rows and $e(2,0)$. If $e(2,0) = 0$, then even parity is used for $pv(1)$, $pv(2)$, and $pv(3)$ and its corresponding rows. Otherwise, the odd parity must be satisfied. Therefore:

$$\left[pv(i) + \sum_{k=0}^{3} e(k,i)\right] \text{mod} 2 = e(2,0),\ i = 1,2,3 \tag{4.5}$$

4.2.4 Minimization of Disparity

The disparity of each sub-block is defined as the difference between the number of transmitted ones and zeros. The goal of the LPC encoder is to minimize the disparity of each sub-block so that a relatively equal number of 0s and 1s are transmitted. Consequently, sub-block 1, sub-block 2, both, or neither are inverted at the encoder's end depending on the value of the disparity bits of each sub-block. Define $DispSum[i](i = 0, \ldots, 3)$ as the disparity in the 5×5 matrix, after the inversion of none, one or both sub-blocks. The following table lists

inversion rules corresponding to each $DispSum[i]$ bit where $i = 0, \ldots, 3$:

	Inversion of Sub-block 1	Inversion of Sub-block 2
$DispSum[0]$	No	No
$DispSum[1]$	Yes	No
$DispSum[2]$	No	Yes
$DispSum[3]$	Yes	Yes

The LPC encoder performs sub-block inversion based on the rules shown in the previous table that provide minimum $DispSum[i]$. Based on these inversion rules, the $e^*(i,j)$'s are calculated as follows:

$$e^*(i,j) = \begin{cases} 1 - e(i,j) & \text{Inversion} \\ e(i,j) & \text{No Inversion} \end{cases}, \tag{4.6}$$

where $i = 0, 1$ for sub-block 1, $i = 2, 3$ for sub-block 2, and $j = 0, ..., 3$ for both sub-blocks.

4.3 Line Product Code Decoding

As a linear code, LPC can be represented by a parity check matrix H and corresponding bipartite graph $\mathcal{G}$. Let $\mathbf{v}$ be a codeword of LPC, and define $\mathbf{s}(\mathbf{v})$ by:

$$\mathbf{s}(\mathbf{v}) = H\mathbf{v}^T. \tag{4.7}$$

Note that for a conventional linear block code, $\mathbf{s}(\mathbf{v})$ is a vector that is independent with $\mathbf{v}$. However, in this case, there are four distinct $\mathbf{s}$ with each one corresponding to a single inversion rule. One possible solution is to perform the decoding process using four different $\mathbf{s}$ separately. This, however, will inevitably increase the hardware usage and decoding latency.

The following section shows that the four matrices can be combined into one by introducing two punctured variable nodes which indicate the inversion rule. As a result, decoding can be performed using only one matrix. The next sections describes the application of decoding methods including maximum likelihood and message passing to the LPC.

4.3.1 Parity Check Matrix Representation

Equations (4.2) through (4.5) put eight parity check constraints on $e(i, j)$, horizontal parity bits and vertical parity bits. The black, solid line portions in Fig. 4.2 represent the bipartite graph defined by these eight parity checks. The "box-plus" symbols and circles represent check nodes and variable nodes, respectively. Circles with a 1 represent a special variable node whose value is a constant 1. The eight check nodes are denoted as $c_0, ..., c_7$. The bipartite graph is drawn such that the modulo-2 sum of all variable nodes connected to each check node must equal zero. These are known as the parity checks.

Given a valid codeword, incorrectly inverting one sub-block will cause the new codeword to fail some parity checks. More specifically, the check nodes that connect to an odd number of variable nodes in one sub-block will no longer satisfy all parity checks if that sub-block gets inverted.

Therefore $\{c_1, c_2, c_3\}$ do not satisfy the parity check condition when sub-block 1 gets inverted and $\{c_5, c_6, c_7\}$ do not satisfy the parity check condition when sub-block 2 gets inverted. Two extra variable nodes $punc(1)$ and $punc(2)$ are introduced in order to make sure that the check nodes still satisfy the parity check condition after the sub-block inversion. $punc(1)$ connects the check nodes that have an odd number of variable node neighbors belonging to sub-block 1. When sub-block 1 gets inverted, $punc(1)$ equals 1 such that for each check node connected to $punc(1)$, all variable nodes connected to that check node sum to zero. Similarly, $punc(2)$ connects the check nodes that have an odd number of variable node neighbors belonging to sub-block 2. Fig. 4.2 shows the complete bipartite graph of LPC.

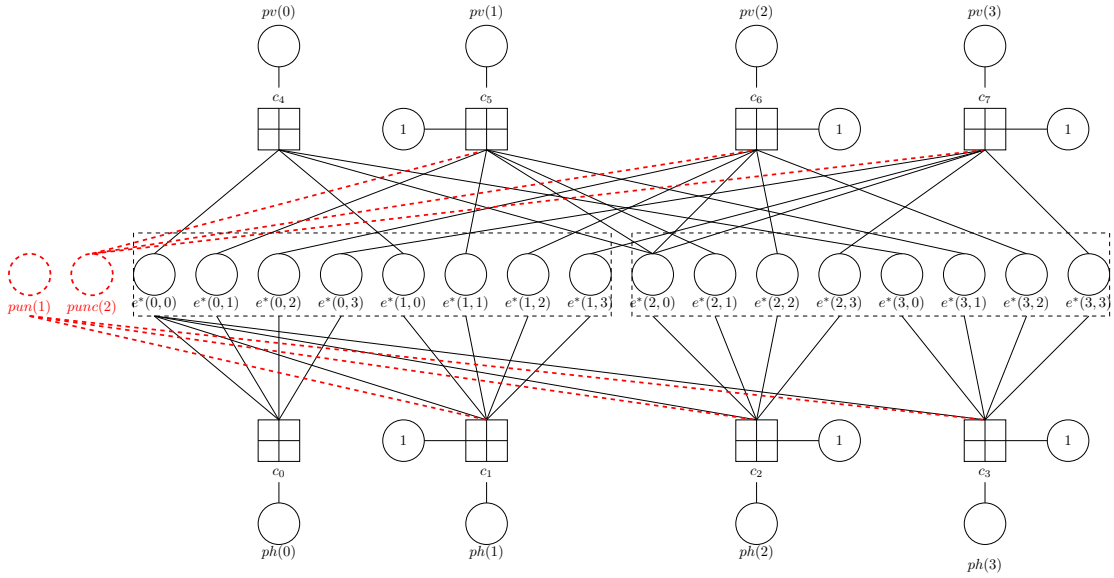


Figure 4.2: Bipartite graph of LPC. The red dashed circles are punctured variable node which indicate the sub-block inversion.

4.3.2 Maximum Likelihood Decoding via the Parity Check Matrix

In this section, we describe how to utilize Wolf's work in [Wol78] to represent the Line Product Code (LPC) as a trellis for performing maximum likelihood (ML) decoding. Since ML decoding represents the theoretical limit of decoding performance, practical decoders which achieve frame error rates closer to it are more desirable. Furthermore, because the LPC is a short blocklength code, reduction in complexity via a trellis representation such as [Wol78] may be feasible to implement in hardware.

Naive ML decoders simply compare the received codeword against all valid codewords. As a result, its complexity scales on the order of 2^k where k is the number of information bits in the codeword. According to the bipartite graph for the LPC shown in Figure 4.2, there are $262,144$ unique codewords making it infeasible in hardware.

As such, we leverage Wolf's framework in [Wol78] to create a trellis representation of the LPC. While sacrificing some parallelism, the final trellis contains only $2,764$ edges which

represents a 94-fold reduction in complexity compared to brute force ML decoding. To construct the trellis, we follow the procedure in [Wol78], also summarized below.

In the trellis, each row represents a unique syndrome with the all-zeros syndrome being at the bottom. Beginning at the all-zeros codeword, the trellis attempts to construct the most likely valid received message one bit at a time. The i^{th} stage of the trellis contains a blue and red branch for each node, representing whether the i^{th} bit is assumed to be 1 or 0 respectively. This continues until all n bits of the codeword are constructed. While the number of branches does indeed grow with $O(2^n)$, most branches are pruned away. The pruning process operates on the principal that only valid codewords have an associated syndrome of $\mathbf{0}$. Therefore, any codewords which do not satisfy this condition can be ignored in the deocding process. With the LPC, we terminate the trellis at the syndrome of $(1, 1, 1, 1, 0, 0, 0, 0)$ instead of the all-zeros syndrome. This is because the Line Product Code is NOT strictly linear due to the use of odd parity (XOR with a constant 1). This means that the trellis must terminate at a syndrome with 1s in the indices whose corresponding check node contains this constant 1, and 0 elsewhere.

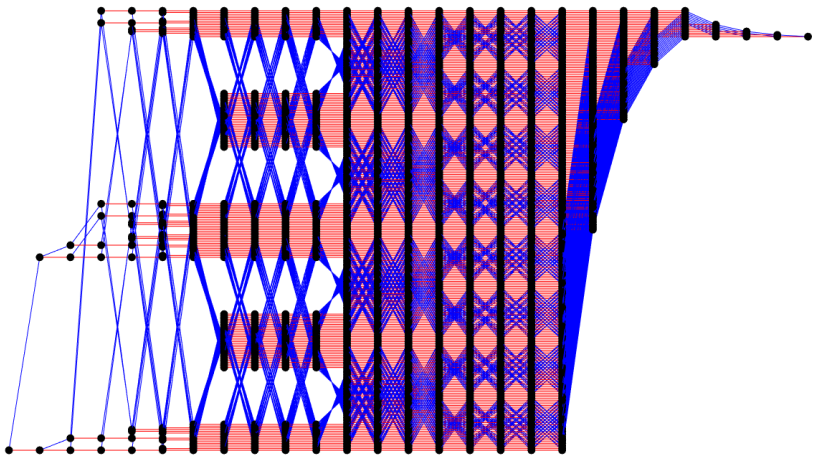


Figure 4.3: LPC Trellis Derived from Wolf's Method Following Pruning, with 2942 states and 5372 branches

4.3.3 Maximum Likelihood Decoding via the Generator Matrix

The trellis representation of a code can also be constructed using a trellis oriented generator matrix [PHB98]. A code's generator matrix G can be transformed to become "trellis oriented" via row operations. A permutation of the columns of G yields a code G' equivalent to G on memoryless channels [PHB98]. This permutation may yield a simpler trellis. However, finding this permutation is known to be an NP-hard problem. Using heuristics, a permutation that simplifies the trellis may be found. For example, a graph of a trellis obtained with one such permutation of the LPC yielded a trellis with 1098 states and 1908 branches, down from 2942 states and 5372 branches of the original code.

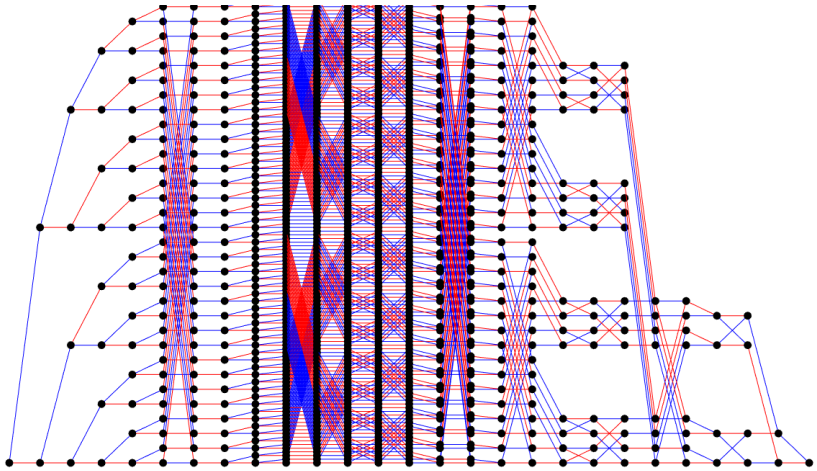


Figure 4.4: LPC Trellis Derived from a Generator Matrix Allowing Permutations, with 1098 states and 1908 branches

4.3.4 Message Passing Decoding

Message passing decoding algorithms, such as belief propagation and MinSum, provide an excellent decoding performance for linear block codes with large girths, defined as length of the shortest cycle in its Tanner graph. For the LPC, message passing decoders do not perform well, because its girth is only 4.

Recently, the neural-network-aided message passing decoders [NML18, LG17, NBB16] have shown substantial improvements compared to conventional message passing decoders. Neural-network-aided message passing decoders assign distinct weights to each message in each iteration, such that the decoder can overcome trapping sets with short cycle lengths. This paper considers a neural normalized MinSum (N-NMS) decoder with a flooding schedule. In the t^{th} decoding iteration, N-NMS updates the check-to-variable node message, $u^{(t)}_{c_j \to v_i}$, the variable-to-check node message, $l^{(t)}_{v_i \to c_j}$, and posterior of each variable node, $l^{(t)}_{v_i}$, by:

$$\begin{aligned} u^{(t)}_{c_i \to v_j} = \beta^{(t)}_{(c_i,v_j)} \times & \prod_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \operatorname{sgn}(l^{(t-1)}_{v_{j'} \to c_i}) \\ & \times \min_{v_{j'} \in \mathcal{N}(c_i)/\{v_j\}} \left| (l^{(t-1)}_{v_{j'} \to c_i}) \right| \end{aligned}, \tag{4.8}$$

$$l^{(t)}_{v_j \to c_i} = l^{ch}_{v_i} + \sum_{c_{i'} \in \mathcal{N}(v_j)/\{c_i\}} u^{(t)}_{c_{i'} \to v_j}, \tag{4.9}$$

$$l^{(t)}_{v_j} = l^{ch}_{v_i} + \sum_{c_{i'} \in \mathcal{N}(v_j)} u^{(t)}_{c_{i'} \to v_j}. \tag{4.10}$$

$\mathcal{N}(c_i)$ represents the set of the variable nodes connected to c_i and $\mathcal{N}(v_j)$ represents the set of the check nodes that are connected to v_j. $l^{ch}_{v_j}$ is the LLR of the channel observation of v_j. $\beta^{(t)}_{(c_i,v_j)}$ are multiplicative weights to be trained. The decoding process terminates when all parity checks are satisfied or when the maximum iteration count, I_T, is reached. In this paper, we follow the steps of [WCN21] to train the neural network.

4.4 Hardware Implementation of Message Passing Decoders

Despite ML decoding being the most optimal, its computational complexity for both the parity check and generator matrix derived trellises is too high to meet timing constraints for practical hardware implementation. Table 4.1 shows the worst case number of operations for each decoding method considered here. An operation is defined as an addition or multiplication, in the case of belief propagation (BP), we consider arctan, exp, and log as a single operation since they are typically implemented via a Lookup Table (LUT). Table

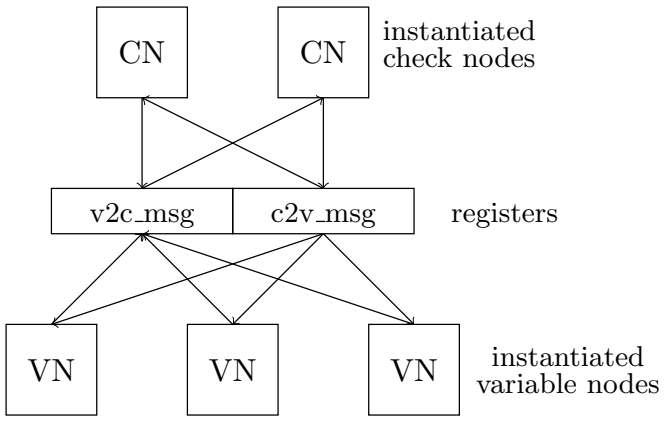


Figure 4.5: Block Diagram of FPGA architecture.

4.1 indicates that message passing algorithms, except belief propagation, utilize significantly fewer operations than ML decoders, indicating that they are the most feasible to implement on hardware. It should also be noted that the Table 4.1 assumes that the message passing decoders always run for 8 iterations. In reality, for higher E_b/N_0 (around 7.5 dB for the LPC), the number of required iterations approaches 1, making message passing even more attractive. As such, our focus for this section will be on the N-NMS decoder, with MS and NMS decoders included for the purpose of comparison. The field-programmable gate array (FPGA) device used for hardware implementation was the Zynq ZCU106 MPSoC.

The overall architecture consists of a bank of registers storing messages between check and variable nodes, and small modules to perform check node (CN) and variable node (VN) operations, as seen in Figure 4.5. The overall decoder controls the timing and coordinates the messages passed between the check and variable node modules for each decoding iteration. It also controls the terminating point by checking if the codeword estimate is valid or if the maximum number of iterations has been reached.

The initial FPGA implementation is a simple MinSum decoder, where check nodes search for the two minimums among its messages, and variable nodes compute simple summations. MinSum will be used as baseline to compare against the Normalized MinSum (NMS) and

Neural-Normalized MinSum (N-NMS) implementations.

However, in order to implement the N-NMS decoder, we must dynamically assign edge weights depending on the iteration of the decoder. This task is divided between 2 modules: the main decoder and the check node module. The multiplicative edge weights are first quantized to a 6-2 scheme, meaning the first 6 bits represent the integer part of the number and the last 2 bits represent the fractional part. Testing and simulations on the LPC showed that the 6-2 quantization achieved a satisfactory middle ground between accuracy and bit width.

Once the edge weights are quantized, they are stored in Block RAM (BRAM). The structure of the BRAM can represented as a 2-D matrix where each element represents a register that stores an 8-bit quantized edge weight. The index of a certain edge weight in the matrix also contains information regarding the iteration count and edge number in the bipartite graph. The main decoder module uses this information to assign weights to the proper edge depending on the iteration count. After the check node module calculates the check-to-variable node message, it then multiplies that by the incoming edge weight. The fixed point FER curves in the following section are generated using C++ simulations. To match the FPGA implementation, all edge weights and calculated messages are quantized according to the 6-2 scheme discussed earlier.

Table 4.1: Comparison of Decoding Complexity via Number of Operations.

Decoder	Worst case number of operations
BP*	10,192
standard MS*	3,200
NMS* / N-NMS*	3,616
ML (Brute force)	524,288
ML (Generator matrix trellis)	6,130

* Message passing algorithms are assumed to always run for 8 iterations

4.5 Simulation Results

4.5.1 Frame Error Rates for Various Decoders

In this section, we showcase our floating point simulation results for the Frame Error Rate (FER) of various decoding methods. The maximum likelihood FER was simulated using the trellis method described in sections 4.3.2 and 4.3.3. Additionally, in line with practical limitations on actual decoding hardware, we limit the number of decoding iterations to two and eight. At high E_b/N_0, most received codewords are low-noise, making the average number of iterations approach 1. However, errors involving trapping sets require more iterations to correct which explains the gap between the 2 and 8 iteration decoders.

The N-NMS decoder performed the closest to Maximum Likelihood out of the three decoders considered, even beating out Belief Propagation. These results line up with the findings of [WCN21]. To summarize, via its training process, the N-NMS decoder was able to adapt its weights to the particular structure of the LPC unlike belief propagation or normalized MinSum. In particular, the N-NMS weights are specifically trained to mitigate the decoding loss caused by trapping sets. With the LPC being such a short block-length

code, its cycles have particularly small girths making N-NMS the ideal decoding method for it.

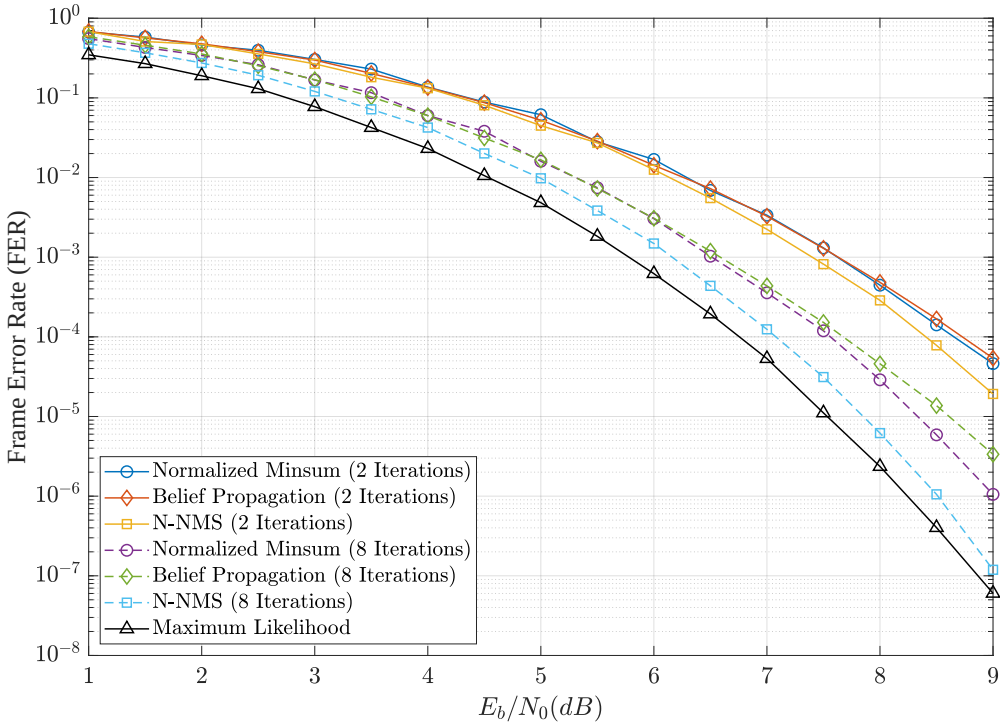


Figure 4.6: FER for various floating point decoding methods capped at 2 and 8 decoding iterations

4.5.2 Quantization Loss for Fixed Point Decoders

In Table 4.2, we observe a noticeable increase in the Look Up Tables (LUTs) used by the N-NMS implementations as compared to the baseline MS and NMS ones. However, it is important to note that the N-NMS decoders also perform better than their counterparts. So, essentially, we are trading extra hardware utilization for better performance.

Table 4.2: Decoder FER Performance and Resource Usage

decoder	**Eb/No (dB)**[a]	**LUT**	**Reg.**
baseline MS (8)	9.93	4953 (100%)	2201 (100%)
NMS	9.63	5205 (105%)	2201 (100%)
N-NMS(1[b])	13.36	5793 (117%)	2201 (100%)
N-NMS(2)	10.54	5784 (117%)	2206 (100%)
N-NMS(4)	9.28	5795 (117%)	2201 (100%)
N-NMS(8)	9.17	5796 (117%)	2206 (100%)

[a]Estimated $\left(\frac{E_b}{N_o}\right)$ to achieve FER of 10^{-7}.

[b](n) number of iterations spent decoding.

The 6-2 quantization used on our FPGA is inherently different than typical software simulations which utilize 64-bit floating point numbers. Since we utilize fewer bits in our fixed point implementation, its precision is comparatively diminished to floating point and we expect some deterioration in FER. The purpose of the simulations shown in Figure 4.7 is to demonstrate that with the N-NMS decoder, utilizing a fixed point quantization as opposed to floating point presents an almost negligible loss in frame error rate.

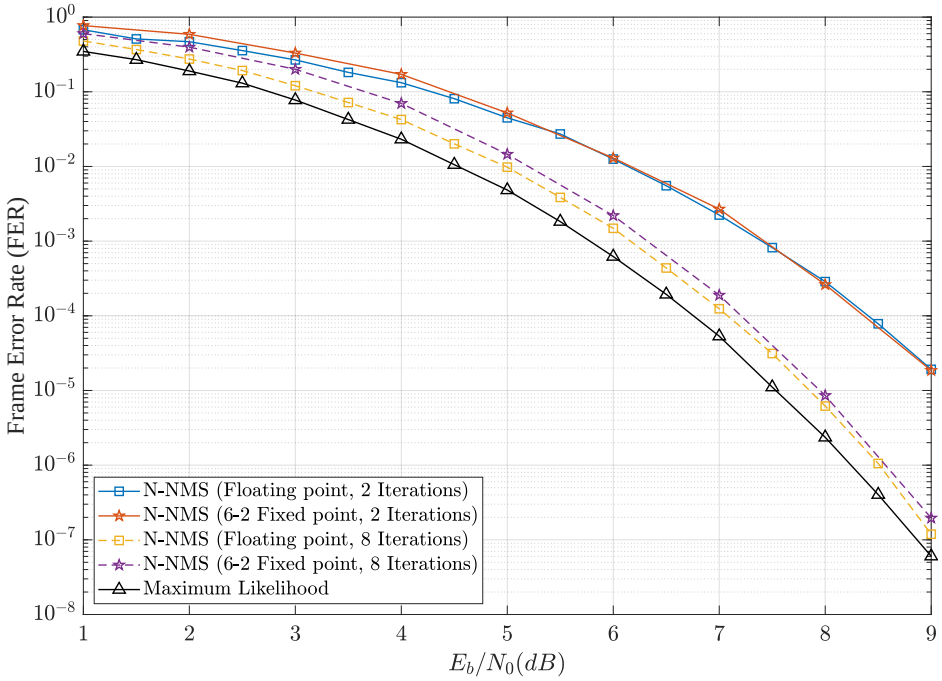


Figure 4.7: Floating vs fixed point FER for the N-NMS decoder capped at 2 and 8 decoding iterations

4.5.3 Reed Solomon Frame Error Rate

As noted in the CCSDS specification, the LPC serves as the inner code in conjunction to a (255,239) Reed-Solomon (RS) operating on GF(256). Since the RS code has 16 parity bytes, it can correct for up to 8 byte errors. Considering that the LPC encodes 2 bytes of data at a time, the RS code can correct for up to 4 LPC frame errors. Given the FER of the LPC, the corresponding FER of the RS code can be modeled via a binomial expression: $P_{RS}(e) = 1 - P(X < 5)$, where $X \sim B(128, P_{LPC}(e))$. Figure 4.8 shows the Reed Solomon FER for the N-NMS fixed and floating point implementations at various E_b/N_0 with the LPC as the inner code.

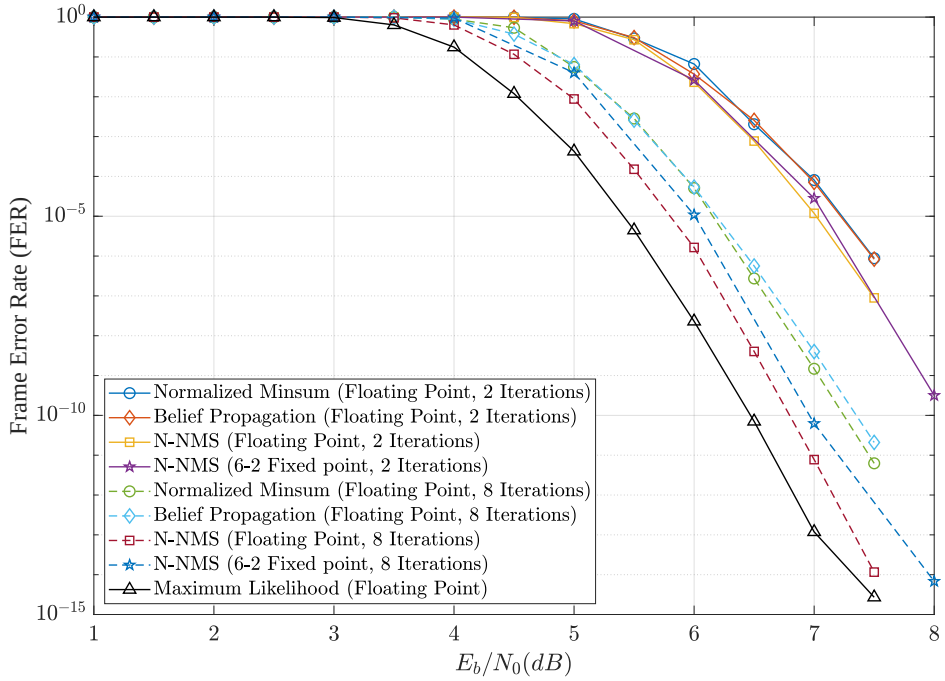


Figure 4.8: Reed Solomon Code FER for Floating vs Fixed point N-NMS and Maximum Likelihood Decoding on the LPC

4.6 Conclusion

This paper compares both the feasibility and decoding performance of maximum likelihood (ML), belief propagation, MinSum, and Neural Normalized MinSum (NNMS) decoders on the Line Product Code (LPC). An initial exploration of ML decoding was considered due to the LPC's short blocklength. However, simulation on an FPGA showed that, even with complexity reduction via a trellis, ML decoding failed to meet timing requirements. In lieu of this, we considered message passing algorithms. While less optimal than ML decoding, they can be performed iteratively and with fewer operations. Simulation results on the LPC show that, with sufficient iterations, these message passing algorithms approach the frame error rate achieved by ML decoding. In particular, the NNMS decoder, with only 8 iterations, show only a 0.5 dB loss compared to ML making it the most promising decoder considered here.

4.7 Acknowledgement

The majority of this chapter has been published or has been submitted for publication in [NWH22]. This research was conducted in collaboration with Linfang Wang, Chester Hulse, Sahil Dani, Amaael Antonini, Todd Chauvin, Dariush Divsilar, and Richard D. Wesel. I derived the Tanner Graph for the LPC, created the trellis decoder, and performed the Frame Error Rate (FER) simulations.I would like to thank Linfang Wang for providing the Neural-Normalized MinSum decoder and for advising on its use. Chester Hulse and Sahil Dani provided the fixed-point simulations and provided message passing simulation results on an FPGA. Amaael Antonini derived the generator matrix trellis structure. Todd Chauvin, Dariush Divsilar, and Richard D. Wesel advised and led the direction of the project.

CHAPTER 5

On the Effect of Puncturing for Neural-Network Optimized Weights

This final chapter is written as a continuation of the findings of [NWH22]. There, we found that a Neural-Normalized MinSum (N-NMS) decoder achieved frame error rates approaching that of Maximum Likelihood decoders while using drastically fewer computing resources. The results of the paper were surprising to say the least and further work was done to understand how the N-NMS training process achieved such a drastic improvement over traditional message passing algorithms. In this chapter, we leverage the training algorithm and decoder used in [WCN21] to analyze the weights assigned to the N-NMS decoder in the Line Product Code (LPC).

5.1 The Structure of the CCSDS Line Product Code

The CCSDS Line Product Code offers a unique opportunity to analyze the training process of the N-NMS decoder. Its short block-length ($n = 26$) makes it possible to analyze and observe its graph structure easily. Additionally, as shown in Figure 4.6, it has already been shown that N-NMS decoding provides **substantial** coding gain compared to Belief Propagation (BP) and traditional Normalized Min-Sum (NMS) on the LPC.

Figure 5.1 shows the associated Tanner Graph of the LPC. Note that the LPC was not designed with message passing based decoding in mind, as evidenced by structural deficiencies in its graph structure, the most notable of which is the abundance of length-

4 cycles. As discussed in the introduction chapter and [Ric03], short graph cycles allow for nodes to influence the messages coming back to it, violating the assumption of only extrinsic information being transmitted. In particular, length-4 cycles are especially detrimental to decoding performance because they are the shortest possible length a cycle can have.

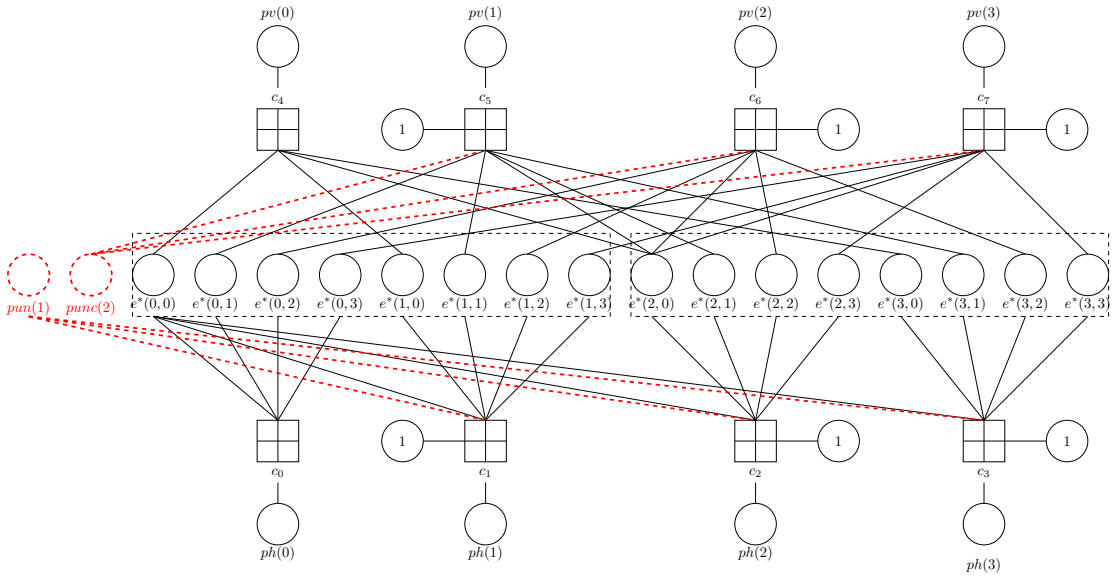


Figure 5.1: Bipartite Representation of the Line Product Code

5.2 Puncturing in the Line Product Code

Looking to literature, [TJV04] shows that both the length and connectivity of cycles play significant roles in decoding performance. The authors conclude that because not all short cycles are equally harmful, design of LDPC codes should focus on the selective treatment of more harmful cycles. In the case of the Line Product Code, the definition of a "harmful" cycle was shown to be these length-4 cycles which include a punctured variable node. Because punctured variable nodes are not transmitted to increase throughput, the decoder assumes no information about them. In terms of message passing, punctured variable nodes are assigned an initial Log-Likelihood Ratio of 0 indicating no confidence in the transmitted bit

being 0 or 1. This means that, in order to decode punctured nodes, non-punctured nodes in the graph must have sufficiently strong confidence in their estimates.

Figure 5.2 shows an induced subgraph of the Line Product Code. The subgraph was formed by analyzing each step of the message passing process for a specific received codeword which ultimately resulted in a decoding error. The variable nodes included in the subgraph are those whose estimate was incorrect at any point in the decoding process. Meanwhile, the check nodes shown are all nodes directly connect to the selected variable nodes.

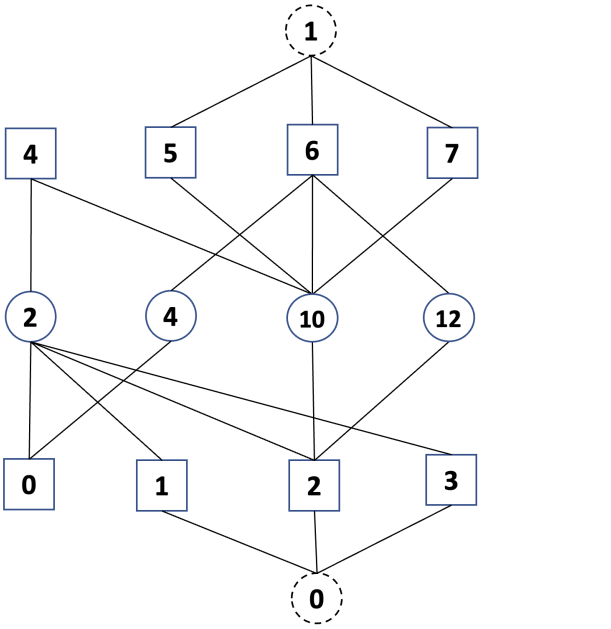
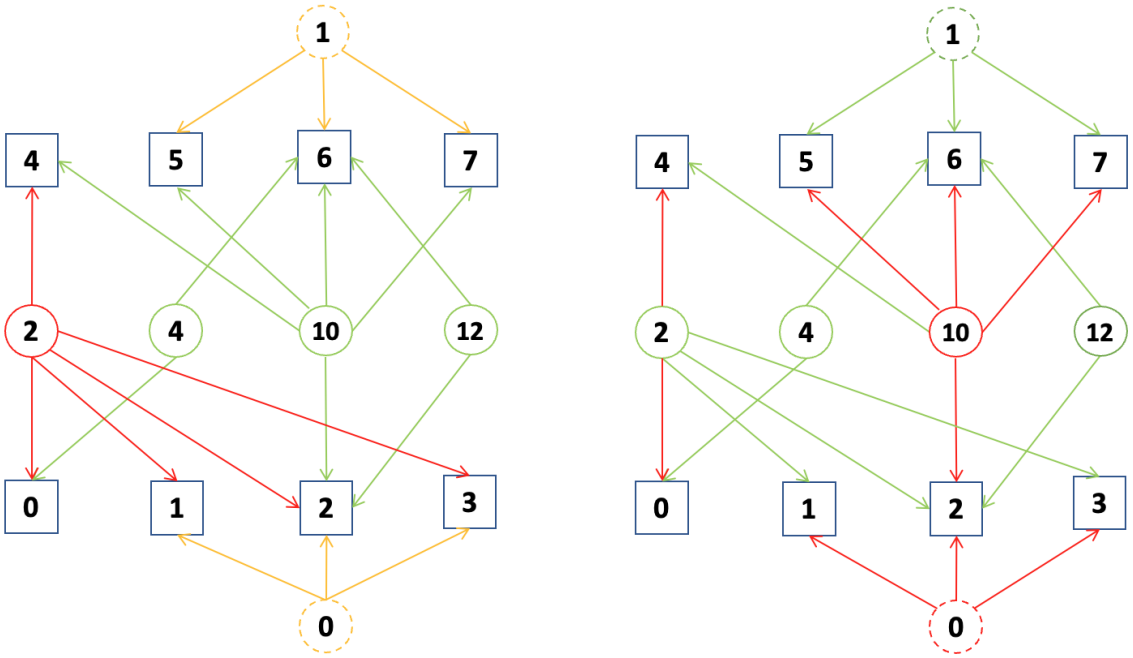


Figure 5.2: Induced Subgraph of Trapping Set on the LPC (dotted circles represent punctured nodes)

In Figure 5.2, each punctured variable node forms **3** length-4 cycles with a single variable node. In the case of punctured v_0, it forms these cycles with v_2. For punctured node v_1, these cycles are formed with v_{10}. In terms of message passing, this means that v_2 and v_{10} exert high levels of influence on the punctured nodes. For example, if v_2, is in error, the incorrect messages $v_2 \rightarrow (c_1, c_2, c_3) \rightarrow v_0$ may cause v_0 to take on the error. Even if v_2 corrects its estimate, v_0 can propagate its original error back using the same edges.

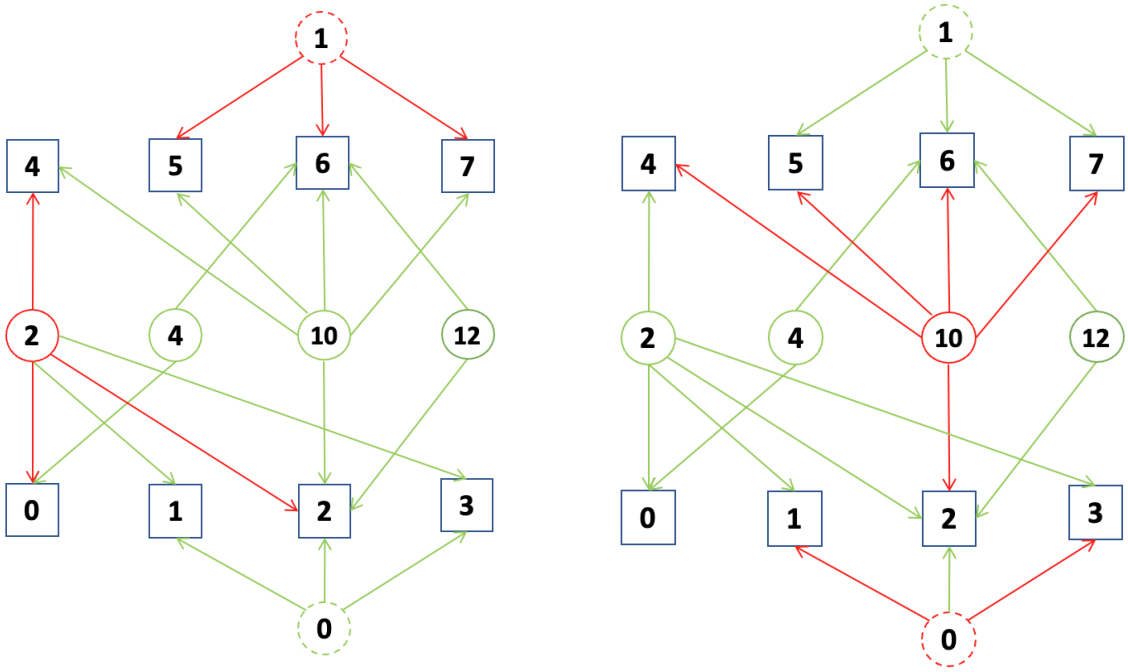
Figures 5.3 and 5.4 show an example of how errors can propagate back and forth using short cycles. The color of each node and edge indicates the correctness of it. For example, a green node indicates that the variable node's internal estimate corresponds to the correct initial bit value. A green edge indicates that the message being sent between two nodes indicates the correct bit value. Red means the node or message represents the wrong bit value. Finally, yellow represents a message or estimate indicating no confidence towards either bit value. Yellow nodes and edges are only present in the first step of decoding before punctured nodes receive any external information.



(a) Decoding Iteration 1

(b) Decoding Iteration 2

Figure 5.3: Example Error Propagation Iterations 1 and 2



(a) Decoding Iteration 3

(b) Decoding Iteration 4

Figure 5.4: Example Error Propagation Iterations 3 and 4

5.3 Neural-Network Optimized Weights Analysis

With the running hypothesis that short cycles involving punctured nodes v_0 and v_1 are the main drivers of decoding errors in the Line Product Code, we can now analyze how the Neural-Normalized MinSum decoder dynamic weights addresses them. Table 5.1 shows the weights trained by the N-NMS decoder. We see that 6 weights whose magnitudes are close to 0 and that that they only involve variable nodes v_2 and v_{10}. The effect of zeroing out these check-to-variable node message has the effect of breaking most of the length-4 cycles involving the punctured nodes. Figure 5.5 highlights where these near-zero weights appear on the Tanner Graph and how the length-4 cycles are broken. Recall that the N-NMS decoder used for this project only weights the **check-to-variable** node messages. Here, zeroing out the check-to-variable node messages prevents v_2 and v_{10} from receiving information from check nodes $c_1, c_2, c_3, c_5, c_6, c_7$. Also recall that Figure 5.5 only shows the induced subgraph

of the LPC. There are more variable nodes that connect to these check nodes and that will contribute to the outgoing messages.

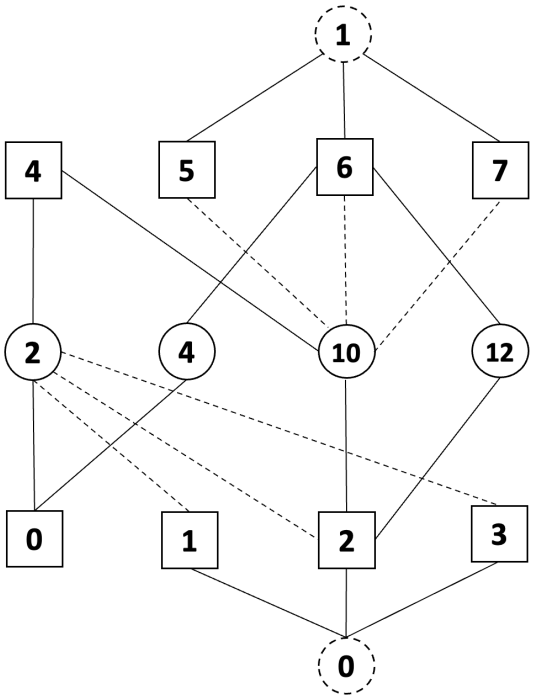


Figure 5.5: Induced Subgraph of the LPC with near-0 check-to-variable edges highlighted

To further convince ourselves that puncturing is the main driving factor in determining whether a check-to-variable edge weight is zeroed out, let us artificially "un-puncture" v_0 and v_1. If this theory holds, then we should expect fewer near-zero weights for fewer punctured nodes. Figure 5.6 shows the distribution of N-NMS trained weights assuming both v_0 and v_1 are punctured, only v_1 is punctured, and no nodes are punctured. As expected, if v_0 and v_1 are artificially un-punctured, then the near-zero edges associated with them return to normal.

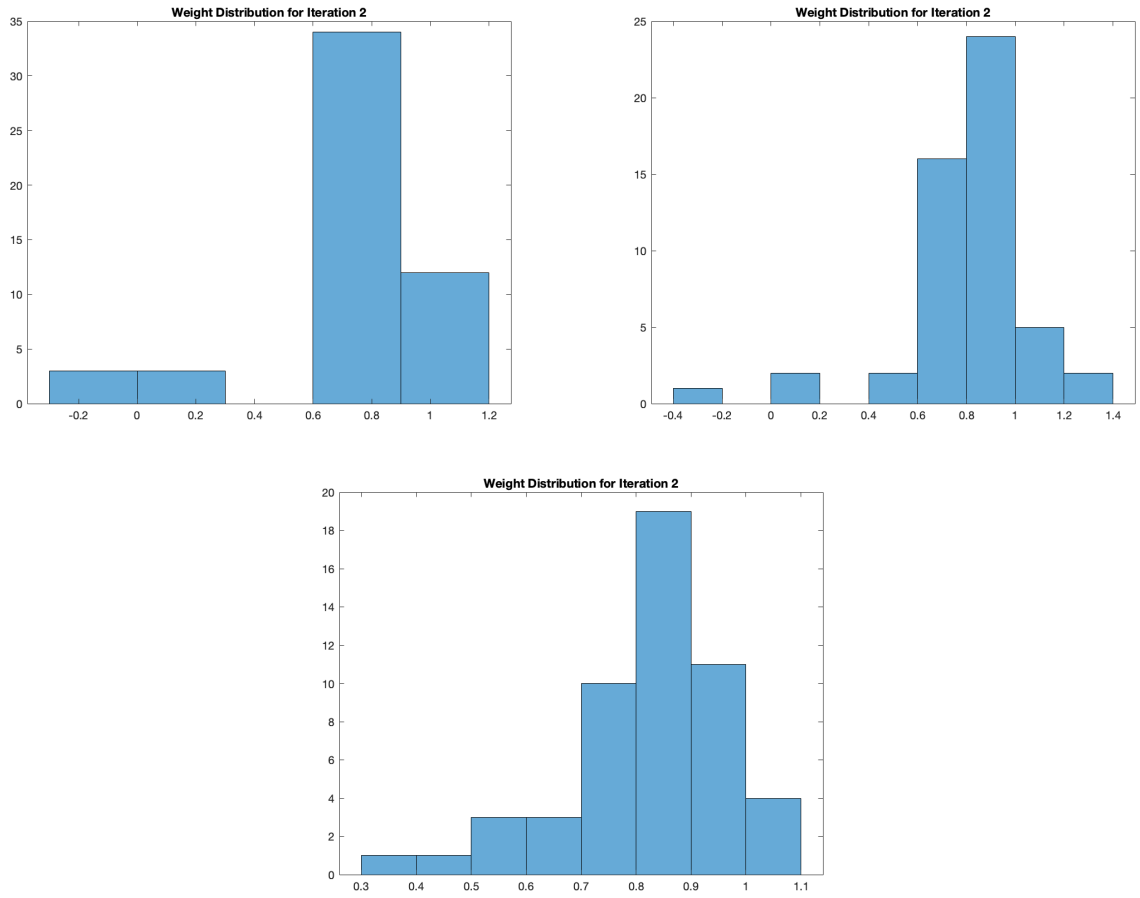


Figure 5.6: From top left, top right, and bottom: LPC N-NMS weight distribution when both v_0 and v_1 are punctured, only v_1 is punctured, and neither are punctured

5.4 Results

Knowing that the N-NMS decoder trained its weights to break length-4 cycles, the question now becomes how much of the decoding gain of Neural-Normalized Min-Sum is due to zeroing out edges vs dynamic weights that can change each decoding iteration. To check this, let us manually set the 6 near-zero edges to 0. Additionally, let us also fix the non-zero weights of the N-NMS weights to be some constant factor for all iterations. In terms of computing complexity, this modified N-NMS uses the same memory and computing resources as a

traditional MinSum decoder. Through trial and error, an optimal normalization factor was found. In theory, if the decoding gain on the LPC was due mainly to zeroing out the low-value edges, the frame error rate of this decoder should be close to that of a standard N-NMS one.

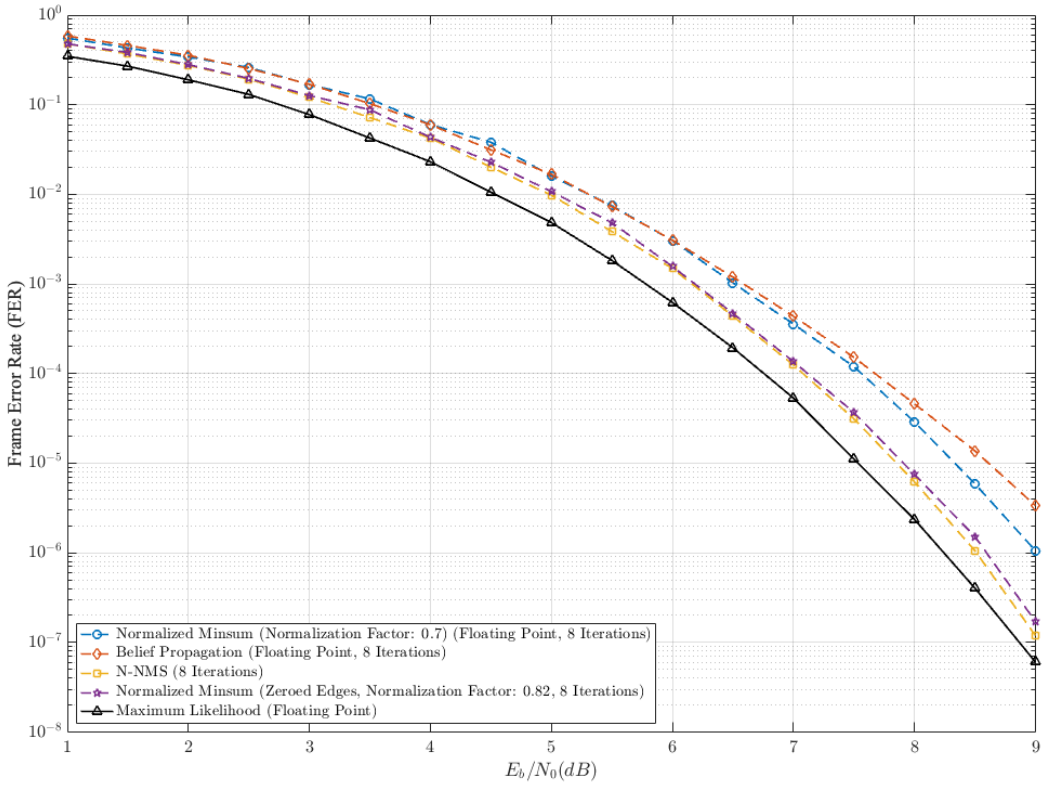


Figure 5.7: Frame Error Rate of Various Decoders

From Figure 5.7, the decoder formed by simply zeroing out the low-weight edges has almost identical frame error rate to the standard N-NMS decoder. This implies that, at least for the LPC, the primary benefit of the N-NMS decoder is this zeroing instead of dynamic weights. The loss in decoding performance compared to N-NMS is due to the constant weight not being exactly optimal (N-NMS weights go up to 6 decimal points while the fixed weight only had 3 decimal places).

5.5 Future Work

While the results of this chapter showed that the same FER as the N-NMS was achievable with lower computational complexity and a much simpler training process, the results are limited to the LPC. Unlike the LPC, more traditional LDPC codes are designed to mitigate structural graph deficiencies including short cycles and overly connected nodes. When examining the weights of more actual LDPC codes, the distinction between "low" and "high" edge weight values is not as clear as in the LPC. Figure 5.8 shows the histogram of N-NMS trained edge weights for a (3225,2193) PBRL code constructed using the lifting process described in [TJV04]. While this code also features 129 punctured variable nodes, none of edge weights approach 0 like the LPC. This indicates that the overall benefit of N-NMS trained edge weights is not solely in removing short cycles involving punctured nodes. Future contributions to this project would include formulating some lemma or set of rules that define when to weight graph edges with 0 and the appropriate time to reintroduce the edges into the decoding process.

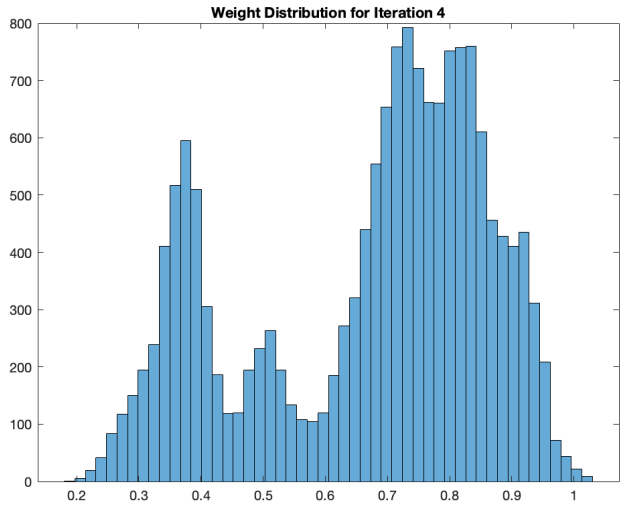


Figure 5.8: N-NMS trained weights for a (3225, 2193) PBRL code

Variable Node Index	Check Node Index	N-NMS Edge Weight
0	1	0.876696
0	2	0.793429
0	3	0.956476
1	5	0.770802
2	0	0.769365
2	1	-0.10364
2	2	0.04538
2	3	0.029803
2	4	0.98143
10	2	0.610683
10	4	0.610683
10	5	-0.02019
10	6	-0.0495
10	7	0.061089
12	2	0.851966
12	6	0.800699

Table 5.1: N-NMS weights assigned for check-to-variable messages on the LPC (most edges omitted for brevity)

CHAPTER 6

Conclusion

This thesis represents the culmination of three years of research, beginning in 2019 as an undergraduate student and ending in 2022 as a master's student. While the four projects discussed in this document cover a broad range of topics, they are all connected by their use and exploration of Low-Density Parity Check (LDPC) codes for unique applications.

Chapter 2 applied the principals of LDPC decoding to Free Space Optical (FSO) and RF communication channels. It asked whether the highly unreliable nature of FSO channels can be mitigated by combining it with a RF channel. In doing so, we hoped to combine the high throughput of FSO transceivers with the reliability of a RF link. To form this hybrid modem, bits from both channels were interleaved into a unified LDPC code. Although the interleaving of bits from different channels resulted in an overall loss of throughput over keeping them independent, it opens the door for further research. For example, if the interleaving was performed with more thought as opposed to randomly, perhaps taking advantage of the LDPC code's specific structure, could the hybrid modem have performed better? Is there a ratio of RF to FSO bits that would favor the hybrid modem over independently using the channels?

Chapter 3 takes a more fundamental approach and examines the way in which LDPC codes are decoded. By training Neural Networks to optimize the edge weights used in message passing algorithms, performance gains up to 0.5 dB were obtained for several codes. Given the weights trained by the Neural Network, the chapter then analyzes their structure. By realizing that nodes of the same degree were assigned extremely similar weights, a weight

sharing paradigm was proposed to drastically reduce decoding and training complexity. Finally, analyzing how these weights varied with decoding iteration opens the door for further optimizations and complexity reductions.

Chapters 4 and 5 introduce the unique case study of the Line Product Code (LPC). Originally only intended to serve as inner code to a Reed Solomon code in the CCSDS standard, the LPC was not designed with soft decision decoding in mind. However, recognizing the untapped potential of the parity check already present in the LPC, Chapter 4 reformulated the LPC as a modified linear block code. Given the derived parity check matrix of the LPC, it was possible to apply traditional soft decoding techniques to it. These included reformulation of the LPC into a trellis for Maximum Likelihood decoding as well as message passing algorithms. By the end, it was shown that Neural-Normalized MinSum (N-NMS) based message passing achieved the closest Frame Error Rates (FER) to the ML decoder while taking a fraction of the hardware resources.

Continuing from chapter 4, chapter 5 examined how the N-NMS decoder beat traditional message passing algorithms by over 0.5 dB while using similar resources. It was found that edges included in cycles with punctured variable nodes were essentially zeroed out. More interestingly, by manually zeroing out these edges and applying traditional MinSum decoding, almost all of the decoding gain of the N-NMS decoder was obtained. This revealed that the N-NMS training process was able to identify deficiencies in the graph structure and adjusted it weights accordingly. If these patterns could be understood, the main benefit of N-NMS decoding could be obtained while bypassing the prohibitively expensive and long training process.

REFERENCES

[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.

[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.

[boo08] "OPTICAL HIGH DATA RATE (HDR) COMMUNICATION — 1064 NM." Consultative Committee for Space Data Systems (CCSDS), Washington, DC, USA, 2008.

[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, May 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 *ICASSP 2019 - 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." *arXiv:2102.03828*, 2021.

[ETS19] 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)." 2019.

[FMI99] M.P.C. Fossorier, M. Mihaljevic, and H. Imai. "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation." *IEEE Transactions on Communications*, **47**(5):673–680, 1999.

[Gal62] R. Gallager. "Low-density parity-check codes." *IRE Transactions on Information Theory*, **8**(1):21–28, 1962.

[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.

[KFL01] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm." *IEEE Transactions on Information Theory*, **47**(2):498–519, 2001.

[KGJ18] Muhammad Nasir Khan, Syed Omer Gilani, Mohsin Jamil, Abdul Rafay, Qasim Awais, Bilal A. Khawaja, Muhammad Uzair, and Abdul Waheed Malik. "Maximizing Throughput of Hybrid FSO-RF Communication System: An Algorithm." *IEEE Access*, **6**:30039–30048, 2018.

[Kol13] Z. Kolka. "Channel Model for Monte-Carlo Simulation of Data Transmission on Terrestrial FSO Paths." In *International Conference on Emerging Trends in Engineering and Technology*, 2013.

[LC04] Shu Lin and Daniel J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., USA, 2004.

[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.

[LHM09] Stefan Laendner, Thorsten Hehn, Olgica Milenkovic, and Johannes B. Huber. "The Trapping Redundancy of Linear Block Codes." *IEEE Transactions on Information Theory*, **55**(1):53–63, 2009.

[LM07] Stefan Laendner and Olgica Milenkovic. "Codes Based on Latin Squares: Cycle Structure, Stopping Set, and Trapping Set Analysis." *IEEE Transactions on Communications*, **55**(2):303–312, 2007.

[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.

[Lug18] Loren Peter Lugosch. "Learning algorithms for error correctio.", 2018. master thesis, McGill University.

[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.

[MSW06] Olgica Milenkovic, Emina Soljanin, and Philip Whiting. "Trapping Sets in Irregular LDPC Code Ensembles." In *2006 IEEE International Conference on Communications*, volume 3, pp. 1101–1106, 2006.

[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.

[NLW20] Jonathan Nguyen, Ethan M. Liang, Linfang Wang, Richard D. Wesel, Todd Drullinger, and Todd Chauvin. "Comparison of Integrated and Independent RF/FSO Transceivers on a Fading Optical Channel." In *2020 54th Asilomar Conference on Signals, Systems, and Computers*, pp. 699–701, 2020.

[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.

[NWH22] 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." In *IEEE International Conference on Communications*, 2022.

[PHB98] Vera Pless, W. C. Huffman, and Richard A Brualdi. *Handbook of coding theory*. Elsevier, Amsterdam, 1998.

[Ric03] Thomas J. Richardson. "Error Floors of LDPC Codes." 2003.

[Tan81] R. Tanner. "A recursive approach to low complexity codes." *IEEE Transactions on Information Theory*, **27**(5):533–547, 1981.

[TJV04] Tao Tian, C.R. Jones, J.D. Villasenor, and R.D. Wesel. "Selective avoidance of cycles in irregular LDPC code construction." *IEEE Transactions on Communications*, **52**(8):1242–1247, 2004.

[WCN21] Linfang Wang, Sean Chen, Jonathan Nguyen, Divsalar Dariush, and Richard Wesel. "Neural-Network-Optimized Degree-Specific Weights for LDPC MinSum Decoding." *International Symposium on Topics in Coding*, 2021.

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

[Wol78] J. Wolf. "Efficient maximum likelihood decoding of linear block codes using a trellis." *IEEE Transactions on Information Theory*, **24**(1):76–80, 1978.

[WSW05] M. W. Wright, M. Srinivasan, and K. Wilson. "Improved Optical Communications Performance Using Adaptive Optics with an Avalanche Photodiode Detector." IPN Progress Report 42-161, Jet Propulsion Labroatory, California Institute of Technology, 2005.

[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.

[XVT19] 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.