Doctoral Dissertation

MAPPING FUNCTIONS THAT MAXIMIZE MUTUAL INFORMATION FOR DECODING LDPC CODES

FRANCISCO JAVIER CUADROS ROMERO

Supervisor: Professor Brian Michael Kurkoski

School of Information Science
Japan Advanced Institute of Science and Technology

March, 2017

Reviewed by
Professor Mineo Kaneko
Professor Hideki Yagi
Professor Dirk Wubben
Professor Tadashi Matsumoto

Abstract

Low-density parity-check (LDPC) codes have been reported to perform close to the channel capacity. LDPC decoders and channel quantization algorithms are usually implemented using floating point simulations in Matlab/C or another programming languages. Once these algorithms are carefully optimized, the next step is to carry out their corresponding hardware implementation in a very-large-scale integrated (VLSI) circuit. In such implementation, LDPC decoders and channel quantization algorithms are converted to a fixed-point representation. For example, the offset min-sum (OMS) algorithm for decoding LDPC codes uses real-valued operations: addition, min. But the channel and decoder messages are usually quantized to a bit width of 4 to 7 bits, depending on the performance/complexity tradeoff. In this research, floating-point algorithms are not used. Instead, the central method is "direct design" of VLSI circuits for LDPC decoders and channel quantizers.

The objective of this research is to design LDPC decoder schemes and channel quantizers that can be implemented in VLSI circuits. For LDPC decoders, the goal is designs that achieve high throughput (a few iterations) and low gate count (a few bits per message). For channel quantization, the goal is to find an optimal quantization scheme, for a fixed bit width, even when the error distribution model is based only on sample data.

In this dissertation, we have developed a technique where the LDPC decoders and channel quantization implementations, including quantization of messages, are designed using only the probability distribution from the channel. Given a probability distribution, our method designs a lookup table (LUT) that maximizes mutual information, and LUTs are implemented directly in VLSI circuits. This is the "max-LUT method".

The proposed lookup tables are sometimes referred as mapping functions. The mapping functions we propose are used for channel quantization and for message-passing decoding of LDPC codes. These mapping functions are not derived from belief-propagation decoding or one of its approximations, instead, the decoding mapping functions are based on a channel quantizer that maximizes mutual information. More precisely, the construction technique is a systematic method which uses an optimal

quantizer at each step of density evolution to generate message-passing decoding mappings.

In a simple manner, the design of LDPC decoders by maximization of mutual information is analogous to finding non-uniform quantization schemes where the quantization can vary with each iteration.

The proposed decoding mapping functions are particularly well suited for data storage applications, because they can be designed from non-parametric and irregular noise distributions. Though finite-length simulations show that the proposed decoding mappings functions present good performance for a variety of code rates.

Numerical results show that using 4 bits per message and a few iterations (10–20 iterations) are sufficient to approach the error-rate decoding performance of full (without quantization) sum-product algorithm (SPA), less than 5–7 bits per message typically needed to perform around 1 dB away from the error-rate decoding performance of full SPA.

Another result of this research is that the construction technique for the mapping functions is flexible since it can generate maps for arbitrary number of bits per message, and can be applied to arbitrary binary-input memoryless channels.

Keywords: LDPC decoding, mapping functions, lookup tables, quantization, sum-product algorithm.

Acknowledgments

First of all, I wish to thank to all committee members who kindly accepted to be part of it. Undoubtedly, Prof. Brian Kurkoski is the first person whom I really want to show my gratitude to. His guidance and support throughout the time of my dissertation have an incalculable value. This dissertation brings me back memories of the days when I was an exchange master student in the University of Electro-Communications (UEC) in Tokyo. I was really lucky to get in touch with Prof. Brian M. Kurkoski for the very first time. I remember he accepted me in his lab knowing that my background in information theory and coding theory was really low. I used to be an image processing guy, but not anymore. Other professor to whom I would like to thank is Prof. Hideki Yagi who took care of me when Prof. Brian left UEC to work in JAIST. I remember that he always had time to solve my questions in a simple and clear way. Later when I became Ph. D. student at JAIST, I was lucky again because I had the chance to meet Prof. Tad Matsumoto who always has the tricky questions that make me study harder.

I would like to thank Dr. Khoirul Anwar, assistant professor in our laboratory, for his selfless help. Also, I would like to say thank you to all my lab colleagues who already left as well as those who still stay there, Pen-Shun Lu, Hui Zhou, Ade Irawan, Xin He, Shen Qian, Kun Wu, Muhammad Reza Kahar Aziz, Ricardo Antonio Parrao Hernandez, Erick Garcia Alvarez, Ryouta Sekiya, Mohammad Nur Hasan and Fan Zhou for their kind help and friendship.

Before to finish, I want to thank to the Consejo Nacional de Ciencia y Tecnologia – CONACYT (the Mexican National Council for Science and Technology) because it supported part of my Ph. D. studies in JAIST.

Moreover, I want to thank all members of the university staff who managed my living in JAIST well so that I can concentrate to the research work. Finally, thanks to my parents who are thousands miles away in Mexico. Their spiritual support will be treasured forever in my deep heart.

Contents

List of symbols

K^{code} Amount of incoming data bits to the channel encoder.

N Length of the code. Size of an outgoing codeword from the channel encoder.

R Rate of the code.

$\mathbf{u}$ Binary codeword of length N.

$\hat{\mathbf{u}}$ Estimated binary codeword by the decoder of length N.

$\mathbf{H}$ Parity-check matrix.

M Number of rows in a parity-check matrix $\mathbf{H}$.

i Indicates a row in a parity-check matrix $\mathbf{H}$.

j Indicates a bit position inside of a codeword $\mathbf{u}$ or estimated codeword $\hat{\mathbf{u}}$. It is also used to indicate a column in a parity-check matrix $\mathbf{H}$.

$h_{i,j}$ Value of the element in the row i and column j of a parity-check matrix $\mathbf{H}$.

d_v Degree of the variable node. Number of ones in a column of the parity-check matrix.

d_c Degree of the check node. Number of ones in a row of the parity-check matrix.

d_{min} Minimum distance of a code.

K Number of quantization levels.

u_j Encoded bit belonging to the codeword $\mathbf{u}$.

$\hat{u}_j$ Estimated decoded bit belonging to an estimated codeword $\hat{\mathbf{u}}$. This variable is also used in Chapter 3 as a generic binary random variable of a generic function f.

y_j Indicates the jth output from a binary-input discrete memoryless channel.

$\hat{y}_j$ Indicates the binary jth output from a binary-input discrete memoryless channel.

$\hat{u}_j$ Decoded bit belonging to the estimated codeword $\hat{\mathbf{u}}$.

P_b Bit-error probability.

P_{cw} Codeword-error probability.

X A discrete random variable. Input to a binary-input discrete memoryless channel.

$\mathcal{X}$ Alphabet for the discrete random variable X.

Y A discrete random variable. Output of a binary-input discrete memoryless channel.

$\mathcal{Y}$ Alphabet for the discrete random variable Y.

$H(\cdot)$ Entropy of a discrete random variable.

$H(\cdot|\cdot)$ Conditional entropy between two discrete random variables.

$p(\cdot)$ Probability mass function.

$p(\cdot|\cdot)$ Conditional probability mass function.

$p(\cdot,\cdot)$ Joint probability of two variables.

$I(\cdot;\cdot)$ Mutual information between two discrete random variables.

C Channel capacity.

C_{BSC} Channel capacity for the binary symmetric channel (BSC).

$h(\cdot)$ Binary entropy function.

ε Cross-over probability in a binary symmetric channel (BSC).

$\mathbf{x}$ Vector of length N which is used as the input to a binary-input discrete memoryless channel.

$\mathbf{y}$ Binary decoded vector of length N.

x_j Indicates a mapping point for the BPSK modulation i.e. $x_j \in \{-1, 1\}$.

E_b Energy per message bit.

E_c Energy per transmitted coded bit.

a It is equal to the square root of the energy per transmitted coded bit.

E_c Energy per transmitted coded bit.

σ Standard deviation for a BI-AWGNC.

σ^2 Variance for a BI-AWGNC.

ϖ Gaussian noise vector of length N.

ϖ_j jth Gaussian noise value of the vector ϖ.

N_0 Noise espectral density.

E_b/N_0 Bit signal-to-noise ratio.

$\mathbb{R}$ Set of the real numbers.

$\mathbb{E}$ Expected value of a discrete random variable.

$\mathcal{N}(0, \sigma^2)$ Gaussian distribution with mean 0 and variance σ^2.

$C_{BI-AWGNC}$ Channel capacity for a binary-input additive white Gaussian noise channel (BI-AWGNC).

δ Arbitrary small value.

$Q^{func}(\cdot)$ Q-function.

$\mathcal{N}(i)$ Set of indices that are nonzero elements in the row i of a parity-check matrix $\mathbf{H}$.

$\mathcal{M}(j)$ Set of indices that are nonzero elements in the column j of a parity-check matrix $\mathbf{H}$.

$\mathcal{N}(i)\backslash j$ Set of indices that are nonzero elements in the row i of a parity-check matrix $\mathbf{H}$ without the index j.

$\mathcal{M}(j)\backslash i$ Set of indices that are nonzero elements in the column j of a parity-check matrix $\mathbf{H}$ without the index i.

$V_{j\to i}$ Decoder message from the variable node j to the check node i.

$L_{i\to j}$ Decoder message from the check node i to the variable node j.

L_j Log-likelihood ratio of the jth bit.

f Generic function used in Chapter 3.

$\sum_{\sim\{\cdot\}}$ Variables inside of the braces indicate the variables not being summed over.

g Generic global function used in Chapter 3.

r Generic binary ransom variable used in Chapter 3.

W Number of factors of a global function g or f.

w A specific factor of a global function g or f.

$g_w(r,\dots)$ Factor w of a global function g with root r.

C^{code} A linear block code.

(p_0, p_1) Vector of probabilities of a binary random variable which represent a decoder message.

(q_0, q_1) Vector of probabilities of a binary random variable which represent a decoder message.

Φ Mapping function or lookup table that performs the check node update.

ϕ Mapping function or lookup table that performs part of the check node update in a decomposed check node.

Ψ Mapping function or lookup table that performs the variable node update.

ψ Mapping function or lookup table that performs part of the variable node update in a decomposed check node.

Γ Mapping function or lookup table that performs the hard decision in a variable node.

γ Mapping function or lookup table that performs part of the hard decision in a decomposed variable node.

$\Lambda\{\cdot\}$ Parametrization used in a message-passing algorithm.

Ω Generic variable used in Chapter 3 to explain an approximation of the variable node update.

d Degree of a node.

ℓ Number of iteration.

Z Channel message.

$\mathcal{Z}$ Alphabet for the channel message.

L Check-to-variable node message.

$\mathcal{L}$ Alphabet for the check-to-variable node messages.

S Message between a pair of mapping functions ϕ in a decomposed check node.

$\mathcal{S}$ Alphabet for the interconnecting messages S in a decomposed check node.

V Variable-to-check node message.

$\mathcal{V}$ Alphabet for the variable-to-check node messages.

T Message between a pair of mapping functions ψ in a decomposed variable node.

$\mathcal{T}$ Alphabet for the interconnecting messages T in a decomposed variable node.

Q A quantizer.

$\mathcal{Q}$ Set of all possible quantizers.

Q^* Optimal quantizer that maximizes mutual information.

a Boundary in a finely quantized channel output.

a^* Optimal boundary that maximizes mutual information.

p_x Input distribution of the channel input X.

$p_{z|x}$ Transition probability between the input to the channel x and the quantizer output z.

$Q_{z|y}$ Transition probability between the quantizer output z and the output of the channel y, i.e. $Q_{z|y} \in \{0, 1\}$.

$\mathcal{A}$ Subset of channel outputs y.

ι Partial mutual information.

$\rho_z(y)$ State of the quantization algorithm that represents the maximum partial mutual information when 1 to y values of Y are quantized to 1 to z values of Z.

$h_z(a)$ Used to save a local decision during the quantization algorithm that finds the optimal quantizer Q^*.

α^* Decoding threshold computed by the density evolution algorithm.

L_0 Initial channel message.

m_0 Mean of the initial channel message L_0.

$m^{(\ell)}$ Mean of the variable-to-check message $V_{j\to i}$.

$n^{(\ell)}$ Mean of the check-to-variable message $L_{i\to j}$.

$p(\cdot)$ Probability density function.

$r^{(0)}(x_0, y_0)$ Initial channel transition probability at iteration ℓ.

t Generic probability distribution used during the DEA.

$\tilde{t}$ Generic joint probability distribution used during the DEA.

$r^{(\ell)}$ Probability distribution for V at iteration ℓ.

$\tilde{r}^{(\ell)}$ Joint distribution distribution of the incoming messages to the variable node at iteration ℓ.

$l^{(\ell)}$ Probability distribution for L at iteration ℓ.

$\tilde{l}^{(\ell)}$ Joint distribution distribution of the incoming messages to the check node at iteration ℓ.

f_c Function for the check node (modulo two addition).

$Q_c^{(\ell)}$ Optimal quantizer that maximizes mutual information at the iteration ℓ in the check node.

f_v Function for the variable node (equality).

$Q_v^{(\ell)}$ Optimal quantizer that maximizes mutual information at the iteration ℓ in the variable node.

$\otimes$ Kronecker product.

K Number of quantization levels.

$\mathbf{y}'$ Concatenation of the incoming messages to a node.

Abbreviations

ARQ Automatic request-for-repeat

BER Bit-error rate

BI-AWGNC Binary-input additive white Gaussian noise channel

BP *Belief propagation*

BPSK Binary phase-shift keying

Blue-ray Digital optical data storage format

BSC Binary symmetric channel

CD Compact disc

dB Decibel

DMC Discrete memoryless channels

DVD Digital versatile disc or digital video disc

FAID Finite alphabet iterative decoder

FEC Forward-error-correction

FER Frame-error rate

FPGA Field-programmable gate array

IC Integrated circuit

IEEE Institute of electrical and electronics engineers

LD Likelihood difference

LDPC Low-density parity-check

LLR Log-likelihood ratio

LUT Lookup table

max-LUT Lookup table that maximizes mutual information

MD Mapping decoder

MPF Marginalize-product-of-function

MS Min-sum

NQBPA Non-uniform quantized belief propagation algorithm

OMS Offset min-sum

pdf Probability density function

PLR Parity likelihood ratio

RS-LDPC Reed-Solomon based LDPC

SNR Signal-to-noise ratio

SPA *Sum-product* algorithm

SSD Solid-state drive

USB Universal serial bus

UTP Unshielded twisted-pair

VLSI Very-large-scale integration

WER Word-error rate

Wi-Fi It is a trademark of the Wi-Fi Alliance

WiMax Worldwide interoperability for microwave access

List of Figures

List of Tables

Chapter 1

Introduction

Source coding and channel coding were initially presented in *A Mathematical Theory of Communications* [1], the groundbreaking paper published by Shannon in 1948. In this paper, Shannon defined *channel capacity* and proved that it is the upper bound of the rate at which we can transmit information over a noisy channel with a probability of error negligibly small. In the following years a variety of codes were proposed. However, it was not until 1993 when *turbo codes* were published [2], the first class of codes reporting a performance close to the channel capacity. Later, around 1996 a rediscovery of low-density parity-check (LDPC) codes were also shown to have near-capacity performance, even though LDPC codes (sometimes called Gallager codes) were conceived by Gallager in 1961 [3], they were mainly forgotten due to the complexity involved in their implementation. Turbo codes lead themselves to a passionate study but they are outside the scope of this work, this work is completely related to the decoding of LDPC codes.

1.1 Transmission and storage of data

Digital communication and storage systems helping people to share and save their information can be seen everywhere at anytime. Some of the most common examples of digital communication systems include smart phones, tablets, smart digital television via satellite or cable, internet access either wired via cable modem and wirelessly via Wi-Fi and WiMax. On the side of digital storage systems, we can mention optical disk drives (e.g. CD, DVD, Blue-ray), solid-state drives (SSD), memory cards, USB flash drives, and magnetic disk drives, although the latter is increasingly disappearing from personal devices.

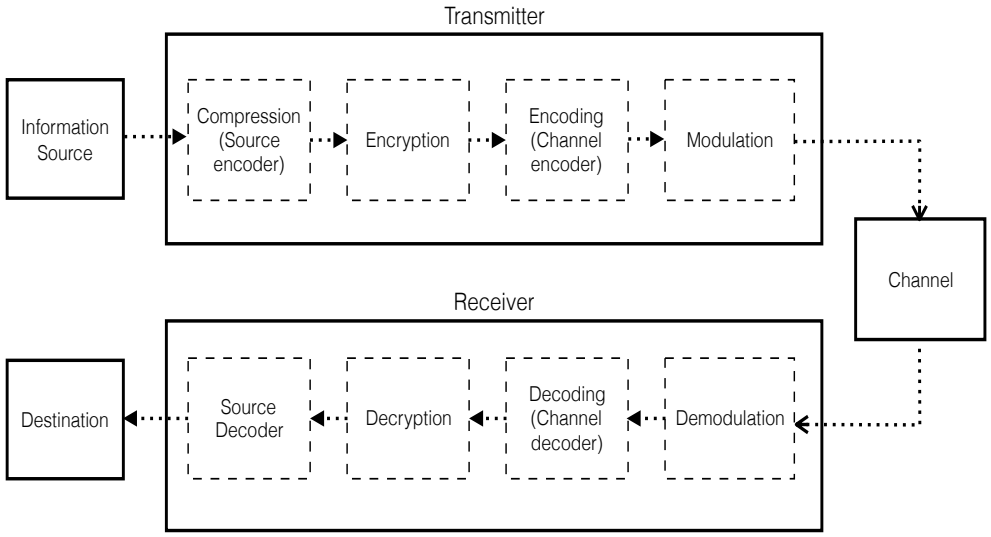


Figure 1.1: Block diagram of general digital communication system.

All the above examples of communication and storage systems can be more globally put into a simple and common framework. Such framework was first proposed by Shannon in [1].

Originally, the block diagram proposed by Shannon of a general communication system had five blocks; an information source, transmitter, channel, receiver and destination, see Fig. 1.1. Each of the blocks is described below. Given that some of the blocks in the diagram perform more than one operation, those are decomposed and described as a set of well defined operations.

1. *The information source* is considered a stream of random numbers (commonly binary) that follow a probability distribution and represent some type of data that a user (a person or a system) wants to communicate to other user. The incoming signal to the information source block may be digital (e.g. computer file) or analog (e.g., light being sensed by a digital camera, sound capture by a microphone, etc.), in such a case, an analog-to-digital conversion is applied to produce a digitized output signal.

2. *The transmitter* is a compound of the following four operations. Since Shannon refers to these operations as a whole, they are drawn inside of the transmitter block, see Fig. 1.1.

    - *Compression* (or source coding) can be seen as the operation in the communication process, where the existing redundancy

in the user data is eliminated, thus the output of this operation produces equiprobable outputs. Depend upon the application, the compression can be lossless (lower bounded by the entropy of the data source) or lossy (governed by the *rate-distortion theorem* [4, p. 301]).

- *Encryption* is sometimes considered in the communication system, and it can be described as a mapping from user data into a "secret" code, so that non authorized users cannot recognize relevant data.
- *Encoding* (or channel coding) is an addition method of structured redundancy to enable error detection/correction capability. Commonly, every incoming sequence of K^{code} symbols, called *message*, is mapped to another sequence of N symbols, called *codeword*, always having $N > K^{code}$. The ratio K^{code}/N is called *code rate* and is normally denoted by R such that $0 < R = K^{code}/N < 1$.
- *Modulation* takes the codewords which have some useful and efficient redundancy and generates the waveforms that meet the requirements of the specified noisy channel.

3. *The channel* is the physical medium whereby the modulated output is conveyed "through space when signaling from here to there (transmission), or through time when signaling from now to then (storage)" (Hamming [5, p. 20]).

4. *The receiver* is the counterpart of the transmitter block. Therefore, it is also decomposed into the corresponding "inverse" set of operations for those in the transmitter block. These operations are wrapped inside of the receiver block, see Fig. 1.1.

    - *The demodulation* is the part of the receiver in a communication system where the output from the channel is converted into noisy sequences (other important operations are performed in this sub block, but they fall beyond the scope of this dissertation).
    - *The channel decoder* attempts to recover the original data encoded by the channel encoder, starting from the demodulated noisy sequences (or corrupted codewords). It produces valid messages for the following processes wrapped in the receiver side. This is the main subject of this dissertation.

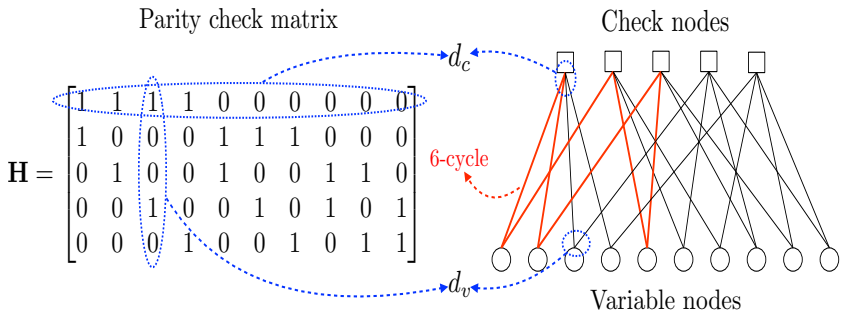


Figure 1.2: Graphical representation of a parity-check matrix $\mathbf{H}$ by a Tanner graph. Edges interconnecting nodes of different types are drawn wherever there is a one in the matrix $\mathbf{H}$.

- *Decryption.* Removes any encryption.
- *The source decoder* recovers the compressed data.

5. *The destination* represents the user for whom data is intended.

1.2 LDPC codes and their key properties

An LDPC code is a block code for channel coding that has a parity-check matrix $\mathbf{H}$ which is sparse. There are two types of LDPC codes: *regular* and *irregular*. Regular LDPC codes have a constant number of ones d_v in each column (column weight) and a constant number of ones d_c in each row (row weight), otherwise the code is defined as an irregular LDPC code. LDPC codes can be analysed using a Tanner graph, which is a bipartite graph that separates the nodes into *variable nodes* (graphically represented by circles corresponding to columns of $\mathbf{H}$) and *check nodes* (graphically represented by squares corresponding to rows of $\mathbf{H}$), see Fig 1.2 for an example of a parity-check matrix $\mathbf{H}$ with $d_v = 2, d_c = 4$ whose Tanner graph is also shown.

The following results are provided by Gallager [3] and Mackay [6]. LDPC codes have a quite simple construction (randomly generated parity check matrix), given an optimal decoder, LDPC codes are *good codes* (code families that achieve arbitrary small probability of error at non-zero communication rates up to some maximum rate that may be less than the capacity of a given channel), and they have *good distance* (the minimum distance d_{min} of the code divided by the length N of that such

code tends to a constant greater than zero). These results hold for any column weight $d_v \geq 3$. Furthermore, there are sequences of LDPC codes in which d_v increases gradually with the length N of the code, in such a way that the ratio d_v/N still goes to zero, this property gives a good distance [6, p. 557].

1.3 Decoding of LDPC codes

Using the Tanner graph of an LDPC code, a *cycle* is defined as a sequence of edges that form a closed path. For instance, in Fig 1.2 we can observe a cycle whose length is equal to the number of edges that form it, for this specific example such length is equal to 6 and as a result this cycle is denoted as *6-cycle*. If an LDPC code is drawn as a tree (which is possible only if there are no cycles in the Tanner graph or in the parity-check matrix), optimal message-passing decoding can be achieved, unfortunately at the same time, LDPC codes with good minimum distance properties cannot be found for this setting [7, p. 64]. On the other hand, the existence of cycles leads to a suboptimal iterative message-passing decoding which requires a large length (e.g. 10^7) to have a probability of error negligibly small [8].

The best iterative message passing decoding algorithm known for LDPC codes is the *sum-product* algorithm (SPA) (from now on, we will sometimes omit the word "iterative" when we refer to decoding algorithms for LDPC codes, since it is understood that the decoding process is iterative), also known as iterative probabilistic decoding or *belief propagation* (BP).

It is well known that the best decoding performance of LDPC codes can be achieved using the SPA with irregular LDPC codes [9]. In irregular LDPC codes the degree distributions of the nodes are optimized causing nodes with different degrees. However, that increases the complexity of the hardware implementation for LDPC decoders. Another problem of employing irregular LDPC codes is that the optimal degree distributions in the nodes generates 4-cyles which generates a decrement in the decoding performance by causing an abrupt change in the slope of the resulting error probability curve, this phenomenon is called the "error floor" [10, p. 399]. In contrast, despite regular LDPC codes having an error-rate performance penalty respect to that achieved by irregular LDPC codes, they provide an easy way to design an efficient hardware implmentation of LDPC decoders due to their structure (i.e. constant d_c and d_v in rows and columns respectively). In this work, we only use regular LDPC codes due to their friendly design (e.g. regular LDPC codes based on array codes, shortened

array codes, finite geometries, etc.) and hardware implementation (e.g. generic node operations due to constant degree of the nodes, fully parallel, serial or hybrid manageable architectures that reduce the integrated circuit (IC) resources and speed up the throughput of the LDPC decoder which allows a scalable design [11]).

1.4 Discrete LDPC decoding algorithms

Although it was mentioned above that the SPA provides the best decoding performance (i.e. error-rate probability close to the the channel capacity), when it comes to its hardware implementation, it becomes a problem due to the fact that this algorithm employs nonlinear functions that need a high resolution (i.e. 12 or more bits [12]) to represent each coded bit in a codeword. This issue also requires that the corresponding architecture work with high resolution variables that at the same time demand the necessary arithmetical and logical units to process them.

Depending of the number of bits (resolution) utilized by a defined variable to represent a coded bit at the decoder, we can define two types of decoding: hard-decision, when one bit per coded bit is used, and soft-decoding, when high number of bits e.g. 64 bits are used to represent each coded bit. Commonly such variables receive the name of *messages*. Therefore, 4-bit per message means that each variable that represents a coded bit has a resolution of 4 bits which gives 16 possible values to represent and processing a given noisy coded bit at the designed message-passing decoder, e.g. the SPA. As one might expect, the error-rate probability improves as the number of bits per message increases, for example, soft-decision (more than one bit) gives better decoding performance than hard-decision (one bit). As a result, the latency for reading and processing the messages in an LDPC decoder is proportional to number of bits per message used to represent such messages. Thus, the target of discrete LDPC decoding algorithms is to reduce the number of bits as much as possible to reduce the latency of the decoding process. The problem is that at the same time, for the quantization of LDPC decoding algorithms, a reduction of the number of bits per message can lead to a performance penalty [3], [13], [14]. Indeed, this topic has received substantial attention in both the research and engineering communities. Past work on discrete message-passing decoding algorithms is summarized below.

One of the first works about the implications related to quantization of the SPA was carried out by Li Ping *et al.* [12], in this work, it is shown that a quantized SPA using 12 bits per message achieves error-rate

performance close to that obtained by SPA without quantization, but still using 12 bits per message an error floor is observed. In [12], a binary-input additive white Gaussian noise channel (BI-AWGNC) is considered. To overcome the problem of quantization of the SPA, in [12] a parity likelihood ratio (PLR) technique is proposed. In [12], using 6 bits per message an error-rate performance close to non-quantized SPA is shown.

Due to the complex operations involved in the SPA (or BP) to generate the check-to-variable node messages, SPA is usually implemented using approximations. One of the most common is the so-called BP-based approximation [15] (commonly known as the "min-sum" (MS) approximation [16]). Thus, in [17] Chen *et al.* proposes two BP-based decoding algorithms to reduce the decoding complexity. Using 6 bits per message a gap of around 0.1 dB respect to full SPA (without quantization) on the BI-AWGNC is presented.

MS decoding reduces the implementation complexity of the iterative decoding process performing just a few tenths of a decibel inferior to BP performance. In [18] Zhao *et al.* study the effects of clipping and quantization on the performance of MS over a BI-AWGNC. The best error-rate performance is achieved using 6-bits per message with a gap of around 0.1 dB respect to that achieved by full SPA.

Chen *et al.* in [19] reported results using 5, 6 and 7 bits per message with an uniform quantization scheme. In this work, using 6 bits per message on a BI-AWGNC the proposed message-passing decoding algorithm shows error-rate performance identical to full SPA.

In [20] Lee *et al.* proposed the idea of designing message-passing decoding algorithms using maximization of mutual information. They used a nonuniform quantization scheme for a regular $(d_v = 3, d_c = 6)$-LDPC code. Comparing with floating point SPA on a BI-AWGNC, 0.2 dB and 0.1 dB gaps are observed using 3 and 4 bits per message respectively, albeit a significant amount of hand-optimization is mentioned and the optimization procedures were not explained in detail.

From an engineering perspective, error floors of the (2048, 1723) Reed-Solomon based LDPC (RS-LDPC) code and (2209,1978) array-based LDPC code on a BI-AWGNC are studied in [21] by Z. Zhang *et al.* In it, 6-bit uniform quantization is employed for a SPA decoder using a parallel-serial decoder architecture in a field programmable gate array (FPGA). Part of those 6 bits control the range of the quantization (lower negative real value and upper positive real value), while the remaining bits define the resolution (quantization step), a parallel-serial decoder architecture in a field programmable gate array (FPGA) was used.

Offset min-sum (OMS) over a binary symmetric channel (BSC) using 4 bits per message and OMS over a BI-AWGNC using 5 and 6 bits per message were proposed in [22] by X. Zhang *et al.*. In this work, using 4 bits per message on the BSC is enough to produce identical error-rate performance than that obtained by OMS without quantization. On the other hand, using 6 bits per message is sufficient to achieve the error-rate performance of the full OMS over the BI-AWGNC.

More recently, in [23] Planjery *et al.* propose a 3-bit finite alphabet iterative decoder (FAID). FAIDs are designed using the knowledge of potentially harmful subgraphs that could be present in a given code. Presented results focus on column-weight-three codes over the (BSC), and in all cases FAIDs decoding performance is better than that obtained by full SPA.

Lewandowsky *et al.* also applied the information bottleneck method to the implementation of quantization in LDPC decoders [24]. Using 4 bits per message over a BI-AWGNC, a gap around 0.2 dB respect to the error-rate performance of full SPA is shown.

In this work we employ binary phase-shift keying (BPSK) modulation since all the aforementioned work also implemented it in all the simulations results. Thus a fair comparison can be made with all the above proposed discrete LDPC decoding algorithms.

In Table 1.1, simulation parameters such as type of quantization, gap with respect to full SPA (if available), number of bits per message, maximum number of iterations, type of considered channel, as well as other details to identify and analyze each of the above discrete LDPC decoding algorithms are described. Also the details about the proposed decoding mapping functions for decoding LDPC codes denoted as "*This work*" are presented.

In the following section, the idea behind the proposed decoding mapping functions and their benefits compared with the above described discrete LDPC decoding algorithms are delineated.

1.5 Proposed technique for LDPC decoding

In this work, we propose a method to find message-passing decoding mapping functions for regular LDPC codes which can surpass the error-rate decoding performance of sum-product algorithm (or BP) using only 4 bits per message. These results are shown on the BSC and in the BI-AWGNC. From the algorithms listed in Table 1.1 only FAIDs using 3 bits per message have presented similar results, but only for the BSC.

Table 1.1: List of various proposed discrete message-passing decoding algorithms using a certain number of bits to represent each received coded bit beloging to a received noisy codeword.
PLR: Parity likelihood ratio, MS: min-sum, NQBPA: non-uniform quantized belief propagation algorithm, SPA: sum-product algorithm, OMS: offset min-sum, FAID: Finite alphabet iterative decoder, MD: mapping decoder, max-LUT: lookup table that maximizes mutual information, BSC: Binary symmetric channel, BI-AWGNC: Binary-input additive white Gaussian noise channel.

No.	Author	Algorithm	Quantization	Gap respect to SPA	No. of bits	Max. no. of iterations	Channel
I	Ping *et al.* (2000) [12]	PLR	Uniform	0.05	6	40	BI-AWGNC
II	Chen *et al.* (2002) [17]	BP-based	Uniform	0.1 dB	6	100	BI-AWGNC
III	Zhao *et al.* (2005) [18]	MS	Uniform	0.1 dB	5–6	200	BI-AWGNC
IV	Chen *et al.* (2005) [19]	MS	Uniform	Identical	6	30	BI-AWGNC
V	Lee *et al.* (2005) [20]	NQBPA	Non-uniform	0.1 dB	3–4	Not mentioned	BI-AWGNC
VI	Z. Zhang *et al.* (2009) [21]	SPA	Uniform	none	6	200	BI-AWGNC
VII	X. Zhang *et al.* (2012) [22]	OMS	Quasi-uniform	Identical (OMS)	4 & 5–6	200	BSC/BI-AWGNC
VIII	Planjery *et al.* (2013) [23]	FAID	Uniform	Better	3	100	BSC
IX	Lewandowsky *et al.* (2016) [24]	MD	Non-uniform	0.2 dB	4	50	BI-AWGNC
	This work	max-LUT	Non-uniform	Better	3–4	25 (average 10–15)	BSC/BI-AWGNC

More precisely, the proposed technique is a systematic method which uses an optimal quantizer at each step of density evolution to generate message-passing decoding mappings which maximize mutual information. Previously in [20] the maximization of mutual information was utilized to design decoding mapping functions too, but the technique was limited only to a specific code rate. On the other hand, in this work the proposed technique allows different LDPC codes and as a consequence different code rates which makes the proposed maps suitable for different applications.

FAIDs along with the mapping decoder proposed in [24] by Lewandowsky *et al.*, represent the most similar works on the design of decoding mapping functions. Compare with FAIDs, the proposed technique uses an optimal quantizer to construct the decoding mapping functions while FAIDs use the information of trapping sets existing in the codes. In second place, comparing with row IX in Table 1.1, they proposed to use the information bottleneck method to design the mapping functions, even though they use a discretized density evolution algorithm they have to carry out an extensive search for a good set of mapping functions to decode a specified code. In this work instead of using the information bottleneck method, we use systematically an optimal quantizer. In our case, we can determine a theoretical threshold for a specified LDPC code which is the designed parameter to construct the decoding mapping functions, in this way, we avoid an extensive search for a good set of decoding mapping functions.

The resulting message-passing decoding mappings are not quantized versions of the sum-product algorithm, or min-sum decoding algorithm nor modifications of these algorithms as in I, II, III, IV, VI and VII in Table 1.1, instead, the proposed maps are based on an optimal quantizer which maximizes mutual information [25].

Although the proposed mapping functions achieve near-SPA error-rate performance using 4 bits per message, it is possible to construct them for an arbitrary number of bits per message, as an example of this, in Chapter 5 also results using 3 bits per message are shown.

Our approach has both theoretical and practical aspects. The theoretical approach of this work is derived from a strong connection between the problem of classification in statistical learning theory, and the problem of optimal quantization of discrete memoryless channels (DMC) in information theory. On the practical side, finite-length results for various regular LDPC code rates show that using 4 bits per message is sufficient to perform close to theoretical limits, achieving or surpassing the error-rate performance of full SPA.

The proposed maps do not necessarily correspond to elementary math-

ematical operations, but may be implemented by a lookup table (LUT). This can lead to a hardware implementation of an LDPC decoder with high throughput (number of decoded bits per second). Another significant benefit of the proposed decoding mapping functions towards a high throughput LDPC decoder, is that the required number of maximum number of iterations is the lowest among all listed algorithms in Table 1.1, even better, in average the number of required iterations decreases to 10–15 depending of code rate. Added to this, benefits of using a few bits per message (3 or 4) include: reduction of the memory needed to store the messages generated along the message-passing decoding process, reduction in the number of interconnect wires utilized between variable and check nodes, reduced complexity of interconnect routing and reduced logic complexity [20].

1.6 Summary of contributions

Throughout this work, we aim to describe how to design decoding mapping functions to decode regular (d_v, d_c)-LDPC codes which can be implemented in integrated circuits using very-large-scale integration (VLSI). For LDPC decoding, the goal is to design decoding algorithms able to meet three features: 1) error-rate performance close to that of SPA (robust decoding algorithm able to work in different channels), 2) high throughput (a few bits per message and a few number of iterations) and 3) low gate count (a few resources for hardware implemntation). The problem is that normally if a decoding algorithm achieves 1), it cannot meet 2) and 3) due to the complexity associated to accomplish 1). On the other hand, if a decoding algorithm meets 2) and 3) it cannot fulfill 1) due to low resolution representation of the variables implicated to estimate valid codewords during the decoding process, or due to excessive assumptions that become the decoding algorithm efficient for a few particular scenarios.

In this dissertation, a decoding algorithm that meets 1), 2) and 3) is presented. More precisely, the proposed algorithm is an iterative message-passing decoding algorithm that only uses mapping functions (lookup tables) to perform the local decisions involved in a common LDPC decoding algorithm (e.g. SPA). The proposed algorithm only performs searches for lookup tables to produce channel messages, decoder messages and estimations of valid codewords, that is, the proposed algorithm does not require any arithmetical operation, instead all messages are positive integers that are used to search the corresponding value according to the type of node and the value of the incoming messages to the node (i.e. the combination

of incoming messages represents an address in a lookup table).

The proposed algorithm is the result of the combination of previous accomplishments that represent the contributions of this dissertation. Such contributions are described as follows:

- **Max-LUT method**. In this research, floating-point algorithms are not used. Instead, the central method is "direct design" of VLSI for decoders and channel quantizers. We have developed a technique where the decoder implementation, including quantization of messages, are designed using only the probability distribution from the channel. **Given a probability distribution, our method designs a lookup table (LUT) that maximizes mutual information, and LUTs are implemented directly in VLSI. This is the "max-LUT method".** It is well-known that maximization of mutual information is Shannon's channel capacity, and in numerical results so far, the proposed method has excellent quantization/performance tradeoff.

- **Quantized density evolution**. Since we were interested in predicting the decoding performance of the proposed message-passing decoding algorithm, we derive a density evolution algorithm that systematically at each step of the density evolution process performs an optimal quantization (optimal in terms of maximizing mutual information). The quantized density evolution algorithm that we proposed, allows us to compute the theoretical decoding threshold for a regular (d_v, d_c)-LDPC code ensemble and a specified number of quantization levels K under the proposed decoding algorithm based on mapping functions that maximize mutual information.

- **Efficient implementation of LDPC decoders**. For the design of LDPC decoders, the max-LUT method is analogous to finding non-uniform quantization schemes where the quantization can vary with each iteration. In our finite-length results: the proposed decoding mapping functions using 3 bits per message have a gap around 0.4 dB with respect to the error-rate performance achieved by full SPA. On the other hand, the proposed decoding mapping functions using 4 bits per message are usually sufficient to achieve the error-rate performance of full SPA. Under the proposed decoding technique, the usual complexity of non-uniform quantization is avoided by using lookup tables. Lastly, the proposed decoding mapping functions using 4 bits per message show lower error floor than full SPA.

1.7 Dissertation outline

This dissertation is organized as follows.

In Chapter 1, introduction to LDPC codes and their basic properties as well as their decoding is mentioned. In this chapter also motivation on discrete LDPC decoding algorithms is presented. Later, the properties of the proposed decoding mapping functions are delineated as well as their main results. At the end of the chapter, the summary of the contributions of this dissertation are described.

In Chapter 2, some fundamental concepts and useful facts about coding theory are formally described. The framework for the proposed research is established in this chapter. Fundamentals about LDPC codes and the sum-product algorithm are described in this chapter.

In Chapter 3, we aim to describe the origin of the so-called *sum-product* algorithm starting from its graphical representation in a tree until its graphical representation in the so-called Tanner graph. Later in this chapter, we present the max-LUT method and its application to channel quantization and its application to designing local decoding lookup tables.

In Chapter 4, we first introduce the idea behind the density evolution algorithm. Later, we describe a discretized density evolution algorithm which uses the max-LUT method to systematically perform quantization on the conditional probability distributions to generate the proposed decoding mapping functions. We also describe how to compute thresholds for a given number of quantization levels K and for a given regular (d_v, d_c)-LDPC code.

In Chapter 5, we analyze the error-rate performance of the proposed decoding mapping functions of a wide range of extensive simulation results for finite-length LDPC codes considering the BI-AWGNC and the BSC.

Finally, conclusions, as well as the future work, are presented in Chapter 6.

Chapter 2

Preliminaries

In this chapter, we firstly establish the conventional performance measures for message-passing decoders. Later, we recall the channel capacity and its application for discrete memoryless channels of interest. At the end of this chapter, we formally introduce the matrix and graphical representation of LDPC codes, to later describe the sum-product algorithm for different channel models.

2.1 Performance measures

Automatic request-for-repeat (ARQ) technique and forward error correction (FEC) technique can be seen as the two branches of *error-control coding*. Firstly, ARQ is a technique which aims to carry out the task of error detection using retransmission requests; in other words, its goal is to detect whether or not a received sequence of symbols (commonly bits) has errors, which are produced due to transmission through a noisy channel. In the case that an ARQ has detected errors in the received sequence, a request of retransmission of the last sequence is sent to the transmitter from the receiver. Secondly, FEC is the scheme whereby existing errors in the received sequence (codeword) are corrected applying an error-correction code. FEC systems normally target a low probability of decoding error. There are systems which mix both schemes to guarantee reliable transmissions, an example of this is the IEEE 802.16-2005 standard for mobile broadband wireless access, also known as "mobile WiMAX".

Although ARQ techniques are enormously useful, in this work we concentrate on LDPC decoding which is a FEC technique.

Consider the transmission of the binary codeword $\mathbf{u}$. The *bit-error probability* P_b or sometimes also referred as BER is the probability that

the jth estimated bit $\hat{u}_j$ at the channel decoder output is not equal to the encoded bit u_j at the channel encoder output, this is,

$$P_b = \Pr\{\hat{u}_j \neq u_j\}. \tag{2.1}$$

Another performance measure commonly found in coding theory literature is the codeword-error probability, P_{cw} also referred as *word-error rate* (WER) or *frame-error rare* (FER). P_{cw} is defined as the probability that the estimated channel decoder codeword $\hat{\mathbf{u}}$ is not equal to the channel encoder codeword $\mathbf{u}$, this is,

$$P_{cw} = \Pr\{\hat{\mathbf{u}} \neq \mathbf{u}\}. \tag{2.2}$$

When it comes to compare decoding algorithms, commonly one is able to see the decoding results as graphs where bit-error rate (BER)/frame-error rate (FER) curves evaluated in a chosen range of signal-to-noise ratio (SNR) are presented. In this work, we shall use the above performance measures to present the decoding performance for the proposed decoding algorithms.

2.2 Channel capacity

Information theory is incredibly relevant for coding theory, it establishes the "playground" of coding schemes by clearly defining the performance bounds. The most important equation in information theory is the equation to calculate the mutual information between two random variables X and Y. The mutual information is the average information that one random variable has about another random variable. Recalling the general communication system model shown in Fig. 1.1, X normally represents the channel input, while Y represents the channel output. When these two random variables X and Y take values from discrete alphabets $\mathcal{X}$ and

$\mathcal{Y}$ respectively, the mutual information can be found as

$$I(X;Y) = H(Y) - H(Y|X) \tag{2.3}$$

$$I(X;Y) = -\sum_{y\in\mathcal{Y}} p(y)\log_2 p(y) - \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \tag{2.4}$$

$$I(X;Y) = -\sum_{y\in\mathcal{Y}} p(y)\log_2 p(y) \tag{2.5}$$

$$-\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log_2 p(y|x) \tag{2.6}$$

$$I(X;Y) = -\sum_{y\in\mathcal{Y}} p(y)\log_2 p(y) \tag{2.7}$$

$$-\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(y|x), \tag{2.8}$$

where $H(Y)$ is the entropy of the channel output Y, and $H(Y|X)$ is conditional entropy of Y given X. Mutual information has various properties [4], one of its properties is that it is symmetric in X and Y, such that

$$I(X;Y) = I(Y;X) \tag{2.9}$$

$$= H(X) - H(X|Y). \tag{2.10}$$

The *channel capacity* C of a discrete memoryless channel (DMC) [1] with input X and output Y is the maximization of mutual information $I(X;Y)$, where the maximization is over the channel input probability distribution $\{p(x)\}$, resulting in

$$C = \max_{\{p(x)\}} I(X;Y). \tag{2.11}$$

The idea behind the channel capacity C is of wide interest for communication systems because it aims to find the maximum achievable rate R at which we can reconstruct the channel input sequences (codewords) at the channel output with a negligible probability of error P_b.

2.3 Channel capacity for useful DMCs

In all practical communication systems, the goal is to transmit data reliably through a noisy channel at the maximum possible rate. In order to be

[1] A channel is said to be *memoryless* if the probability distribution of the output depends only on the input at that time and is conditionally independent of previous channel inputs or outputs.

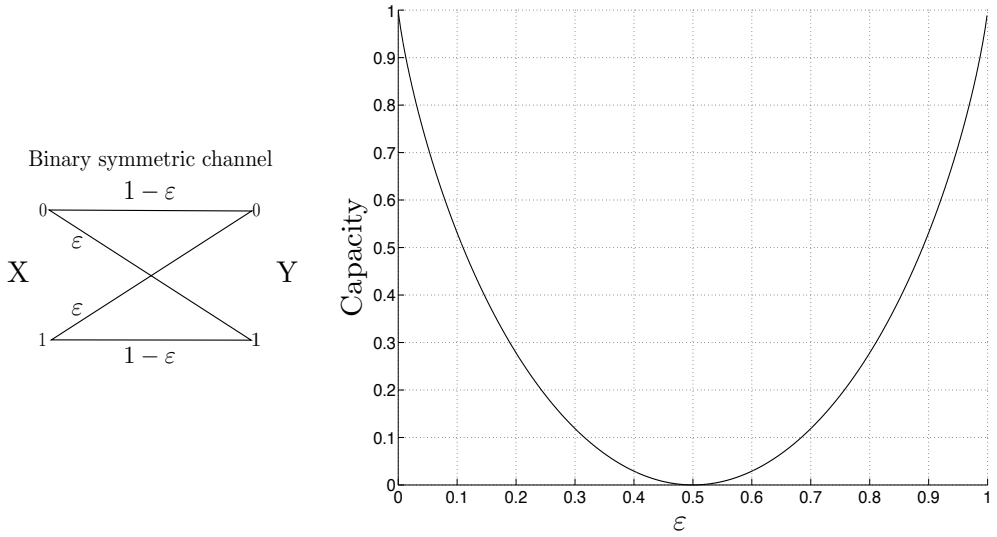


Figure 2.1: Diagram and channel capacity plot for the binary symmetric channel.

able to do this, we need a noisy channel model that emulates the random changes that a sequence conveyed through a physical channel suffers.

2.3.1 The binary symmetric channel

In this DMC, a binary input $X \in \{0, 1\}$ and a binary output Y are assumed. The probability of a bit error is defined by ε, see Fig. 2.1. The channel capacity for the binary symmetric channel C_{BSC}, plotted in Fig. 2.1, can be computed through (2.11), resulting in

$$C_{BSC} = 1 - h(\varepsilon), \tag{2.12}$$

where $h(\varepsilon)$ is the *binary entropy function*

$$h(\varepsilon) = -\varepsilon \log_2(\varepsilon) - (1 - \varepsilon) \log_2(1 - \varepsilon). \tag{2.13}$$

2.3.2 The binary-input AWGN channel

When a coded bit $u_j \in \{0, 1\}$ is transmitted through an additive white Gaussian noise channel (AWGNC), first, it is mapped to $x_j = a(-1)^{u_j}$, where $a = \sqrt{E_c}$ (employing a binary phase-shift keying (BPSK) signal), and E_c is the energy per transmitted coded bit, which is related to the energy per message bit E_b such that $E_c = RE_b$, being $R = K^{code}/N$ the rate of the code. Once u_j is mapped to x_j, x_j is transmitted through

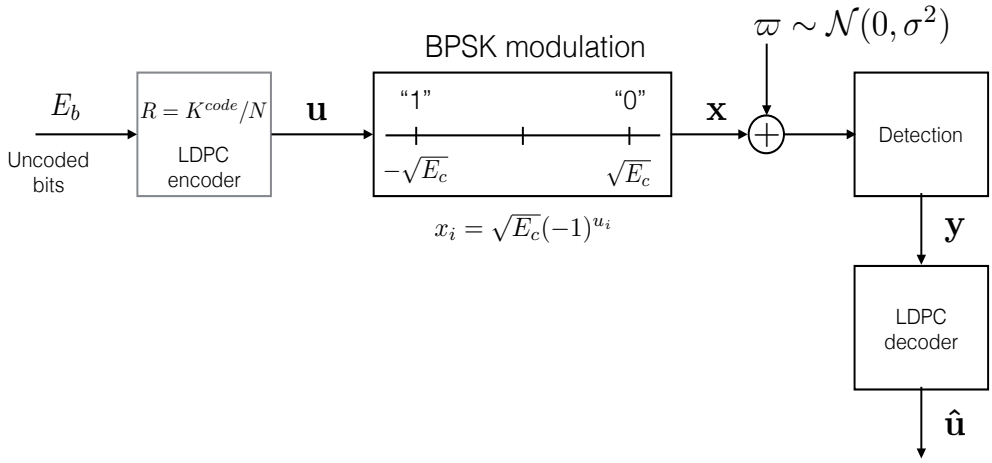


Figure 2.2: Block diagram that represents the transmission of a binary codeword $\mathbf{u}$ through the BI-AWGNC. Coding a decoding are assumed to be carried out by LDPC codes.

a channel which adds a Gaussian noise value ϖ_j with zero mean and variance $\sigma^2 = N_0/2$, i.e., $\varpi_j \sim \mathcal{N}(0, \sigma^2)$. At the output of the channel the real value $y_j \in \mathbb{R}$, being $y_j = x_j + \varpi_j$ is received. Therefore, this channel is known as the *binary-input* AWGN channel (BI-AWGNC). The transmission of the binary codeword $\mathbf{u}$ through the BI-AWGNC is shown in Fig 2.2, where coding and decoding are assumed to be performed using LDPC codes.

The capacity of the BI-AWGNC is

$$C_{BI-AWGNC} = 0.5 \sum_{x=\pm a} \int_{-\infty}^{\infty} p(y|x) \log_2 \left(\frac{p(y|x)}{p(y)} \right) dy, \tag{2.14}$$

where

$$p(y|x = \pm a) = \frac{1}{\sqrt{2\pi}\sigma} \exp[-(y \pm a)^2/2\sigma^2] \tag{2.15}$$

and

$$p(y) = \frac{1}{2}[p(y|x = +a) + p(y|x = -a)]. \tag{2.16}$$

Following (2.11), we can derive another formula to compute $C_{BI-AWGNC}$, this is, $C = H(Y) - H(Y|X) = H(Y) - H(Z)$, where $H(Z) = 0.5 \log_2(2\pi e \sigma^2)$, thus we have

$$C_{BI-AWGNC} = -\int_{-\infty}^{\infty} p(y) \log_2(p(y))\, dy - 0.5 \log_2(2\pi e \sigma^2). \tag{2.17}$$

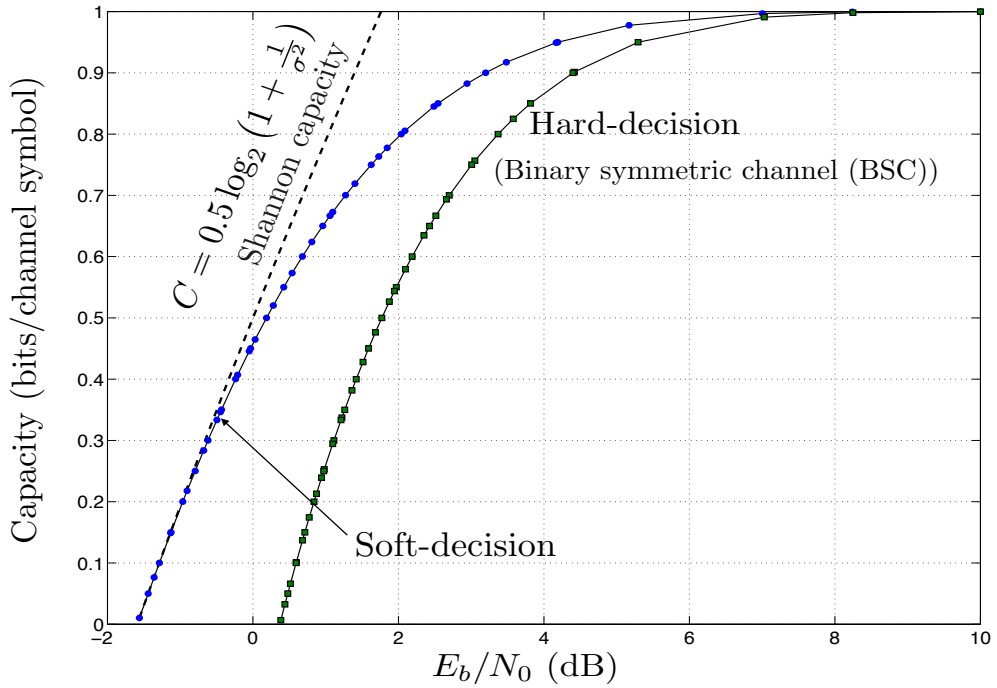


Figure 2.3: Plotting soft-decision and hard-decision capacity curves for the BI-AWGNC, along with the curve for the *Shannon capacity*.

Note that the integral in (2.17) may be estimated as

$$\mathbb{E}\{-\log_2(p(y))\} \simeq -\frac{1}{J}\sum_{j=1}^{J}\log_2(p(y_j)), \tag{2.18}$$

where $\{y_j : j = 1, \ldots, J\}$ is a large group of realizations of Y (say 10^6) and $\mathbb{E}\{\cdot\}$ indicates the expected value of the discrete random variable.

In Fig. 2.3, the capacity curve $C_{BI-AWGNC}$ (labeled "*soft-decision*") versus the (bit) signal-to-noise ratio E_b/N_0 in dB is plotted . Recall that $E_b/N_0 = \mathbb{E}[x_j^2]/2R\sigma^2$ (in this work we assume $\mathbb{E}[x_j^2] = 1$). The value R used in the E_b/N_0 is considered as $R = C_{BI-AWGNC}$, because R is assumed to be lower than $C_{BI-AWGNC}$ just by an arbitrary small value δ, this is

$$E_b/N_0(\text{dB})_{soft} = 10\log_{10}(1/(2C_{BI-AWGNC}\sigma^2)). \tag{2.19}$$

In order to compute the *hard-decision* BI-AWGNC capacity curve (labeled "*hard-decision*"), the hard-decisions $\hat{y_j}$ from y_j must to be obtained as follows

$$\hat{y_j} = \begin{cases} 1 & \text{if } y_j \leq 0 \\ 0 & \text{if } y_j > 0. \end{cases} \tag{2.20}$$

Note that these hard-decisions transform the BI-AWGNC into a binary symmetric channel with error probability ε such that, first, we can apply the Q-function[2] to estimate ε, resulting in

$$\varepsilon = Q(\sqrt{2RE_b/N_0}), \tag{2.21}$$

later recalling (2.12), we write $C_{BSC} = 1 - h(\varepsilon)$ to finally produce

$$E_b/N_0(\text{dB})_{hard} = 10\log_{10}(1/(2C_{BSC}\sigma^2)). \tag{2.22}$$

In Fig. 2.3, the *Shannon capacity* curve is also shown, without loss of generality, this is

$$C_{Shannon} = 0.5\log_2\left(1 + 2R\frac{E_b}{N_0}\right), \tag{2.23}$$

and then

$$E_b/N_0(\text{dB})_{Shannon} = 10\log_{10}(1/(2C_{Shannon}\sigma^2)). \tag{2.24}$$

2.4 Representation of LDPC codes

Even though we introduced LDPC codes in section 1.2 of Chapter 1, here we concentrate our attention in the description of the sum-product algorithm. Fig. 1.2 is used again but, modified to describe the flow of the messages in the context of message-passing decoding and thus be able to explain some useful equations of this section.

Due to complexity implementation requirements associated to the LDPC codes in 1960, these linear block codes were forgotten for a while, until in the mid 1990s with the work of MacKay, Luby, and others, they came back again, but this time they continue being an interesting research topic.

The main reason why these codes are frequently studied, is because they have shown to have decoding performance close to the Shannon capacity [8].

This section follows the presentation of LDPC codes in [10] and [26]. In this research, only binary LDPC codes are considered. The representation of LDPC codes can be carried out either in matrix form or in a graphical form.

[2]The Q-function is the probability that a unit Gaussian $N \sim \mathcal{N}(0,1)$ exceeds x [26]: $Q^{func}(x) = \Pr(N > x) = \frac{1}{\sqrt{2\pi}}\int_x^\infty \exp(-n^2/2)\,dn$.

2.4.1 Matrix representation

An LDPC code is a linear block code given by the null space of an $M \times N$ parity-check matrix $\mathbf{H}$, which has a low density of ones. A *regular* LDPC code has a constant number of ones d_v in each column and a constant number of ones d_c in each row, otherwise the code is called *irregular*. Thus, the code rate R of a regular LDPC is

$$R \geq 1 - \frac{M}{N} = 1 - \frac{d_v}{d_c} \tag{2.25}$$

with equality when $\mathbf{H}$ is full rank. An example of a parity-check matrix is as follows

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \tag{2.26}$$

where $M = 5, N = 10, d_v = 2$ and $d_c = 4$. Looking closely at the parity check matrix in (2.26), we can see that the first row is not independent, this is, it is the addition of the rest four rows. As a result $R = 1 - 4/10 = 3/5$.

2.4.2 Graphical representation

To graphically represent a parity-check matrix $\mathbf{H}$ of an LDPC code, we use a Tanner graph, which is a bipartite graph, that is, the nodes are separated into two types: check nodes, denoted as CN and variables nodes, denoted as VN, with edges connecting only nodes from different types. For each $\mathbf{H}$ there are M check nodes and N variable nodes. The Tanner graph construction is as follows: each check node CN i is connected to a VN j whenever element $h_{i,j}$ of $\mathbf{H}$ is equal to 1. Considering the parity-check matrix in (2.26), the corresponding Tanner graph is depicted in Fig. 2.4.

Before describing the iterative decoding process of the SPA, we first need to define some useful notation. We denote the set of VNs j that participate in the CN i as

$$\mathcal{N}(i) = \{j : h_{i,j} = 1\}, \tag{2.27}$$

in a similar manner, we denote the set of CNs i that participate in the VN j as

$$\mathcal{M}(j) = \{i : h_{i,j} = 1\}. \tag{2.28}$$

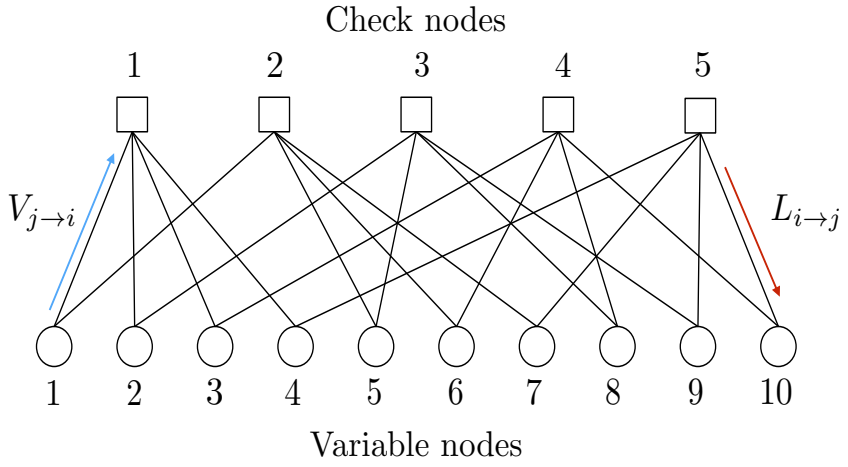


Figure 2.4: Tanner graph for the parity-check matrix $\mathbf{H}$ in (2.26)

Using (2.27) and our example of a parity check matrix in (2.26), we can write the set of VNs j that participate in each CN i as follows

$$\mathcal{N}(1) = \{1, 2, 3, 4\}, \mathcal{N}(2) = \{1, 5, 6, 7\}, \mathcal{N}(3) = \{2, 5, 8, 9\},$$
$$\mathcal{N}(4) = \{3, 6, 8, 10\}, \mathcal{N}(5) = \{4, 7, 9, 10\}.$$

In a similar way but using (2.28), we can write the set of CNs i that participate in each VN j as follows

$$\mathcal{M}(1) = \{1, 2\}, \mathcal{M}(2) = \{1, 3\}, \mathcal{M}(3) = \{1, 4\}, \mathcal{M}(4) = \{1, 5\},$$
$$\mathcal{M}(5) = \{2, 3\}, \mathcal{M}(6) = \{2, 4\}, \mathcal{M}(7) = \{2, 5\}, \mathcal{M}(8) = \{3, 4\},$$
$$\mathcal{M}(9) = \{3, 5\}, \mathcal{M}(10) = \{4, 5\}.$$

We use $\mathcal{N}(i)\backslash j$ to indicate the set $\mathcal{N}(i)$ without the element j, e.g., $\mathcal{N}(1)\backslash 3 = \{1, 2, 4\}$. Note that in the Tanner graph the messages sent from VN j to the CN i are denoted as $V_{j\to i}$, while the messages sent from the CN i to the VN j are denoted as $L_{i\to j}$, this can be observed in Fig. 2.4.

2.5 The Gallager sum-product algorithm

In this section we describe the sum-product algorithm (SPA). At the beginning of SPA, we initialize the variable-to-check node messages $V_{j\to i}$ with the log-likelihood ratio (LLR) as

$$L_j = L(u_j|y_j) = \log\left(\frac{\Pr(u_j = 0|y_j)}{\Pr(u_j = 1|y_j)}\right), \tag{2.29}$$

whenever $h_{i,j} = 1$. Below, we define the LLRs for the BSC and for the BI-AWGNC. For the BSC with $b \in \{0, 1\}$ and b^c as a complement (i.e. when $b = 0, b^c = 1$ and vice versa), we have

$$L_j = L(u_j|y_j) = (-1)^{y_j} \log\left(\frac{1-\varepsilon}{\varepsilon}\right), \tag{2.30}$$

where the channel output $y_j \in \{0, 1\}$ and $\varepsilon = \Pr(y_j = b^c|u_j = b)$. In the case of the BI-AWGNC, we have

$$L_j = L(u_j|y_j) = 2y_j/\sigma^2, \tag{2.31}$$

being $u_j \in \{0, 1\}$, $x_j = (-1)^{u_j}$ and the channel output $y_j = x_j + \varpi_j$, where the ϖ_j are independent and normally distributed as $\mathcal{N}(0, \sigma^2)$. Once the LLRs in (2.30) and (2.31) have been defined, the iterative SPA is as follows:

1. **Initialization**. For all j, initialize L_j according to (2.29) for the appropriate channel model used. Then, for all i, j that $h_{i,j} = 1$, set $V_{j\to i} = L_j$.

2. **Check node update.** Compute $L_{i\to j}$ for each check node as

$$L_{i\to j} = 2\tanh^{-1}\left(\prod_{j'\in\mathcal{N}(i)\setminus j} \tanh\left(\frac{1}{2}V_{j'\to i}\right)\right), \tag{2.32}$$

   and then transmit to the variable nodes.

3. **Variable node update.** Compute $V_{j\to i}$ for each variable node as

$$V_{j\to i} = L_j + \sum_{i'\in\mathcal{M}(j)\setminus i} L_{i'\to j} \tag{2.33}$$

   and then transmit to the check nodes.

4. **LLR total**. For $j = 1, 2, \ldots, N$ compute

$$L_j^{total} = L_j + \sum_{i\in\mathcal{M}(j)} L_{i\to j}. \tag{2.34}$$

5. **Stoping criteria**. For $j = 1, 2, \ldots, N$, set

$$\hat{u}_j = \begin{cases} 1 & \text{if } L_j^{total} < 0 \\ 0 & \text{else ,} \end{cases} \tag{2.35}$$

   to obtain $\hat{\mathbf{u}}$. if $\hat{\mathbf{u}}\mathbf{H}^T = \mathbf{0}$ or the number of iterations equals the *maximum number of iterations*, stop; else, go to step 2.

The equation (2.32) for the check node update, is the part of the SPA that increases the complexity of a hardware implementation and at the same time is quite sensible to quantization, this happends due to the product, tanh and $\tanh^{-1}$ operations involved. Mainly the complexity issues of the equation (2.32) are the motivation of all discrete LDPC decoding algorithms shown in Table 1.1 in page 9 on chapter 1, as well as the motivation of this work.

2.6 Summary

In this chapter, first we formally describe the bit-error rate and word/frame-error rate as performance measures for decoding algorithms. In chapter 5, we will use this measures to analyze the error-rate performance of the proposed decoding mapping functions for regular (d_v, d_c)-LDPC codes.

Later, we described shortly the channel capacity and we write its corresponding equation for some discrete memoryless channels that we use in this work.

At the end of the Chapter, we present the sum-product algorithm and we mentioned that the complexity of the check node update equation (2.32) for its hardware implementation is the motivation of this work and that of others proposed decoding algorithms.

Chapter 3

Max-LUT method: Maximizing mutual information

In this chapter, we describe a technique where the factor-graph-based decoders and channel quantizer implementations, including quantization of messages, are designed using only the probability distribution from the channel. Given a probability distribution, our method designs a lookup table (LUT) that maximizes mutual information. In addition, LUTs are desirable by engineers who design very-large-scale integration (VLSI) hardware implementations of the above operations. This method is called the "max-LUT method".

Before presenting the max-LUT method, we are interested in describing the origin of the sum-product algorithm (SPA) and its variations that lead to some of its approximations (e.g. min-sum). This is important in the first place to clarify the difference between the SPA and its approximations with respect to the proposed decoding algorithm. In the second place, the description of the origin of SPA will also give a landscape over possible range of applications that the proposed factor-graph-based decoding algorithm could have. Also the decomposition of the local decoding functions can be understood by describing the core of the SPA.

3.1 Factorization of a global function

Algorithms that have to deal with marginals of multivariate functions, normally exploit the factorization of the global function. This leads to the computation of a set of simpler *local functions* which only receive

as arguments a few random variables of the global function. An specific example of these type of algorithms are the decoding algorithms based on graphs, e.g sum-product algorithm to decode LDPC codes.

The essential idea of using the product of local functions to solve a marginalize-product-of-functions (MPF) problem was first explicitly presented by Aji and McEliece [27]. Aji and McEliece in [28] proposed a *generalized distributive law* which may solve some MPF problems using junction trees (i.e. a mapping of a graph into a tree), but more importantly, it can also be used in *factor graphs* to describe the functionality of a generic message-passing decoding algorithm commonly called the *sum-product* algorithm. This result is significant because algorithms developed in digital communications and other disciplines may be derived as a particular case of the sum-product algorithm attached to a suitable factor graph.

As an example of a global function f and its factorization, consider a set of six binary random variables $\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6 \in \{0, 1\}$ such that $f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6)$. Suppose, f and its factorization are as follows

$$f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6) = f_1(\hat{u}_1, \hat{u}_2, \hat{u}_3) f_2(\hat{u}_1, \hat{u}_4, \hat{u}_6) f_3(\hat{u}_4) f_4(\hat{u}_4, \hat{u}_5). \quad (3.1)$$

For this specific example, the global function f is factorized into four factors f_1, f_2, f_3 and f_4.

Taking f and its factorizarion, we are interested in a graphical representation. For this matter, we can draw a *factor graph* as follows: each variable $\hat{u}$; in the global function f is represented with a *variable node* (circle) and each factor of f in (3.1) is represented by a *factor node* (square). The corresponding factor graph of the example in (3.1) is shown in the left hand side of Fig. 3.1. Then, using edges connect a variable node to a factor node whenever such variable node is an argument of that factor node, e.g. in Fig. 3.1, we connect the variable nodes $\hat{u}_1, \hat{u}_2$ and $\hat{u}_3$ to the factor node f_1.

Note that the factor graph in Fig. 3.1 is in fact a tree with root in $\hat{u}_1$ for convenience. In this tree edges only connect nodes of different types; in other words, the factor graph of the example in (3.1) is a bipartite tree since it has two types of nodes; variable and factor nodes. As a result, there is only one path between two nodes, e.g. there is only one path between the variable nodes $\hat{u}_3$ and $\hat{u}_5$; this is indicated on Fig. 3.1 with a dashed line.

The property that a global function can be represented by a bipartite tree (no closed paths) is useful because it leads to the computation of a

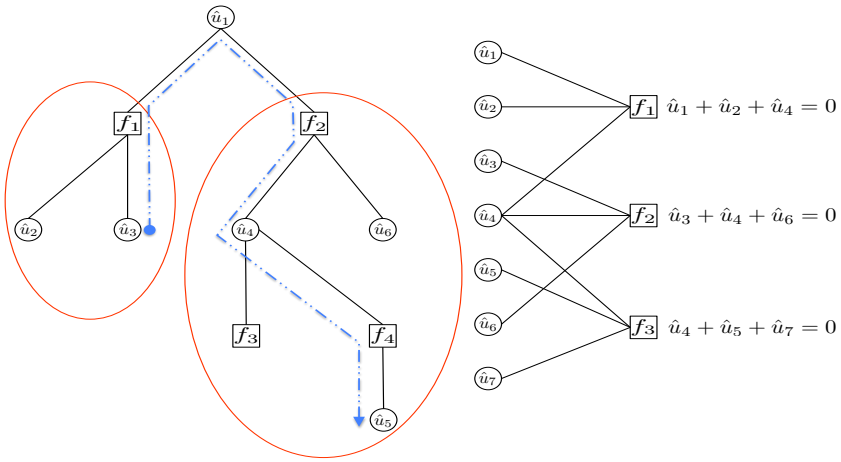


Figure 3.1: Representation of a bipartite tree with two factors (subtrees) closed by ellipses (left hand side). Tanner graph of a code C^{code} with its corresponding check node equations (right hand side).

generic factorization of the global function. This property later will allow the computation of exact marginals that become the main assumption behind the node equations for the sum-product algorithm.

At the beginning of this section, we mentioned that the factorization of a global function e.g. f, is useful for decoding algorithms, in order to show the connection, we will use the following example. Consider a binary linear code C^{code} whose parity check matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{3.2}$$

Therefore, C^{code} has a set of binary codewords $\hat{\mathbf{u}}$ of the form $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6, \hat{u}_7)$. In this example, the task of the global function f is to verify if each incoming binary sequence $\hat{\mathbf{u}}$ is a codeword of C^{code} (in this example f has seven arguments and different factorization than in (3.1)). Thus, the global function f is

$$f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6, \hat{u}_7) = \begin{cases} 1, & \text{if } \hat{\mathbf{u}}\mathbf{H}^T = \mathbf{0}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Using the nonzero elements in each row of the parity check matrix $\mathbf{H}$ in (3.2), the global function f can be factorized as

$$f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6, \hat{u}_7) = f_1(\hat{u}_1, \hat{u}_2, \hat{u}_4) f_2(\hat{u}_3, \hat{u}_4, \hat{u}_6) f_3(\hat{u}_4, \hat{u}_5, \hat{u}_7), \tag{3.4}$$

where each factor f_1, f_2 and f_3 of f is equal to 1 if the modulo two addition (i.e. checksum) of its arguments $\hat{u}$ is equal to 0, otherwise is 0. In other words, the global function f is satisfied when f_1, f_2 and f_3 are equal to 1, meaning that the incoming sequence $\hat{\mathbf{u}}$ satisfies $\hat{\mathbf{u}}\mathbf{H}^T = \mathbf{0}$. The graphical representation of the above example receives the name of *Tanner graph* and is shown in right hand side of Fig. 3.1. Since each factor of f in (3.4) performs checksums, in the literature the factor nodes of the Tanner graph are commonly called *check nodes*. In this way, we can say that a Tanner graph is a particular case of a factor graph applied to the description of codes.

3.2 Recursive determination of marginals

Employing the example in (3.1), suppose that we are interested in computing the marginal of the global function f respect to $\hat{u}_1$, this is,

$$\begin{aligned} f(\hat{u}_1) &= \sum_{\hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6} f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6) \\ &= \sum_{\sim\{\hat{u}_1\}} f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6). \end{aligned} \tag{3.5}$$

Note that we introduce the *not-sum* or *summary* denoted as $\sum_{\sim\{\cdot\}}$ which indicates the variables not being summed over, i.e. the marginal of f respect to $\hat{u}_1$ is the not-sum for $\hat{u}_1$ of f.

Now, consider a generic global function g whose graphical representation is a bipartite tree. Then, we are interested in computing the marginal

$$g(r) = \sum_{\sim\{r\}} g(r, \dots). \tag{3.6}$$

In order to reduce the complexity of the marginalization in (3.6), we will exploit the fact that g can be represented as a tree. Therefore, g has a generic factorization as follows

$$g(r,\dots) = \prod_{w=1}^{W} [g_w(r,\dots)], \tag{3.7}$$

where W is an integer. An important property of the factorization in (3.7) is that the variable r is an argument of each factor g_w, but all other variables only appears in one of the factors.

Thus, the factorization in (3.7) applied to the example in (3.1) gives

$$f(\hat{u}_1,\hat{u}_2,\hat{u}_3,\hat{u}_4,\hat{u}_5,\hat{u}_6) = \underbrace{[f_1(\hat{u}_1,\hat{u}_2,\hat{u}_3)]}_{\text{First factor}}\underbrace{[f_2(\hat{u}_1,\hat{u}_4,\hat{u}_6)f_3(\hat{u}_4)f_4(\hat{u}_4,\hat{u}_5)]}_{\text{Second factor}}, \tag{3.8}$$

where $W = 2$. In a graphical way, the above factorization of f is depicted in Fig 3.1. In this figure the factorization of f in (3.8) is graphically the partition of the tree into two subtrees with root $\hat{u}_1$, this is highlighted by ellipses closing the two subtrees or factors of f.

Using the generic factorization of the function g in (3.7) and the distributive law, we can rewrite the marginalization of g respect to r as follows

$$g(r) = \sum_{\sim\{r\}} g(r,\dots) = \sum_{\sim\{r\}} \prod_{w=1}^{W} [g_w(r,\dots)] = \prod_{w=1}^{W} \left[\sum_{\sim\{r\}} g_w(r,\dots) \right]. \tag{3.9}$$

Note that the marginal $\sum_{\sim\{r\}} g(r,\dots)$ is the product of individual marginals $\sum_{\sim\{r\}} g_w(r,\dots)$. Therefore, the application of (3.9) to the marginalization of f respect to $\hat{u}_1$ will produce

$$f(\hat{u}_1) = \left[\sum_{\sim\{\hat{u}_1\}} f_1(\hat{u}_1,\hat{u}_2,\hat{u}_3) \right] \left[\sum_{\sim\{\hat{u}_1\}} f_2(\hat{u}_1,\hat{u}_4,\hat{u}_6)f_3(\hat{u}_4)f_4(\hat{u}_4,\hat{u}_5) \right]. \tag{3.10}$$

Since we want to decompose g in small pieces to perform simper local operations, we define a generic factorization for each g_w which contains a *kernel* and the product of factors, this is,

$$g_w(r,\dots) = \underbrace{\eta(r, r_1, \dots, r_J)}_{\text{Kernel}} \prod_{j=1}^{J} \underbrace{\Big[\eta_j(r_j,\dots)\Big]}_{\text{Factors}}. \tag{3.11}$$

Thus, the kernel is the function contained in the root node of the current factor g_w. Note that the above generic factorization of g_w only has the variable r in the kernel while each of the other variables r_j appears at most twice; sometimes in the kernel and in one of the factors $\eta_j(r_j,\dots)$. Identifying the structure of (3.11) in f, we can write f as

$$\begin{aligned} f(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6) = & \\ & \underbrace{\Big[\underbrace{f_1(\hat{u}_1, \hat{u}_2, \hat{u}_3)}_{\text{Kernel}} \underbrace{[1]}_{\hat{u}_2} \underbrace{[1]}_{\hat{u}_3}\Big]}_{\text{First factor}} \cdot \\ & \underbrace{\Big[\underbrace{f_2(\hat{u}_1, \hat{u}_4, \hat{u}_6)}_{\text{Kernel}} \underbrace{[f_3(\hat{u}_4) f_4(\hat{u}_4, \hat{u}_5)]}_{\hat{u}_4} \underbrace{[1]}_{\hat{u}_6}\Big]}_{\text{Second factor}}. \end{aligned} \tag{3.12}$$

Taking advantage of (3.11), we can compute the marginal $\sum_{\sim\{r\}} g_w(r,\dots)$ by multiplying the kernel $\eta(r, r_1, \dots, r_J)$ with the individual marginals $\sum_{\sim\{r_j\}} \eta_j(r_j,\dots)$ and summing all the remaining variables different than r, more precisely

$$\sum_{\sim\{r\}} g_w(r,\dots) = \sum_{\sim\{r\}} \eta(r, r_1, \dots, r_J) \prod_{j=1}^{J} \Big[\sum_{\sim\{r_j\}} \eta_j(r_j,\dots)\Big] \tag{3.13}$$

At this point we can apply (3.9) and (3.13) to recurvely fragment the marginalization of g respect to r until we reach the leaves of the tree. Note that the marginalization process is governed by the structure of the tree.

In the following section, we describe the method to compute the marginals of the global function g whose graphical representation is a bipartite tree, such a method is called *message-passing algorithm*.

3.3 Message-passing algorithm in cycle-free factor graphs

The message-passing algorithm in cycle-free factor graphs (trees) is a method that sends messages along the edges of the tree, starting from the leaves and ending at the root. These messages are marginals of part of the global function g that at the end are combined to compute the marginal of the global function respect of a given variable r (root).

Assuming a tree with root r (or a rooted cycle-free factor graph in r), the message-passing algorithm is as follows:

1. the calculation begins in the leaf nodes of the tree;
    - if the leaf node is a variable node, it sends an *identity function* (i.e., 1) to its parent node,
    - if the leaf node is a factor node, it sends a description of f to its parent node.
2. Each node must to wait to receive all incoming messages from its children to compute its outgoing message that will be transmited to its parent;
    - if the the node is a variable node, it sends the product of all incoming messages from its children,
    - if the the node is a factor node, it performs the product of its function f with all incoming messages from its children, and then, it applies the summary $\sum_{\sim\{r'\}}$ or *not-sum* of r', where r' is its parent.

We have described in two general steps the computation of the marginalization of a global function g respect to r, but we are interested in computing the marginals for each variable in the global function g.

Computing all marginals of a global function g is possible by applying the previously described message-passing algorithm to different trees which have one of the variables of g as a root. On the other hand, a better approach is to compute the marginals of each variable of g in the same tree. In order to do that, we start at all leaf nodes and for each node we compute an outgoing message once we have all other incoming messages from the rest of adjacent edges to the given node. Each node performs these operations until all edges have transmitted messages in both directions. At the end, each variable node of the tree is able to calculate the corresponding marginal.

The initialization conditions as well as the node operations of the message-passing algorithm are shown in Fig 3.2. Looking at this figure, note that a considerable amount of operations performed are sums and products, carried out along the message-passing algorithm, for this reason, sometimes it is called *sum-product* algorithm.

Since we are interested in the decoding of binary LDPC codes, it is important to mention that the function f that appears in the factor node in Fig 3.2 is the the addition modulo 2 of all its arguments which define a binary sequence. The purpose of this function when decoding binary LDPC codes is to add all probabilistic products of sequences that have even number of ones and on the other side, add all the probabilistic products of sequences that have odd number of ones.

Later in section 3.5, we will explain some of the most common parametrization of the node function that lead to different approaches to perform the message-passing decoding algorithm to decode LDPC codes.

3.4 Message-passing algorithm in factor graphs with cycles

A linear block code C^{code}, defined by its parity-check matrix $\mathbf{H}$ with M rows and N columns, can be represented by a factor graph with N variables nodes and M factor nodes. As an example, consider a linear block code C^{code} whose parity-check matrix is

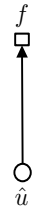
$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{3.14}$$

the corresponding factor graph for the above matrix can be drawn as shown on the left hand side of Fig. 3.3. On the other hand, a more useful and structured graph is the Tanner graph which is depicted on the right hand side of Fig. 3.3. Although both graphs are graphical representations of C^{code}, normally the Tanner graph is used to analyze the local decoding functions involved in the message-passing algorithm. In Fig. 3.3, a closed dashed path is shown, this is called a *cycle* whose *girth* is equal to four since the cycle is constructed by four edges; in the literature this is written as a 4-*cycle*.

The implication of an existing cycle in a parity check matrix implies that C^{code} cannot be represented by a tree. This further implies that

Initialization in the leaf nodes

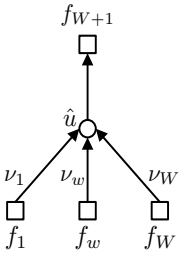
$\nu(\hat{u}) = f(\hat{u})$



$\nu(\hat{u}) = 1$

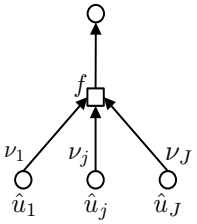
Variable node

$\nu(\hat{u}) = \prod_{w=1}^{W} \nu_w(\hat{u})$



Factor node

$\nu(\hat{u}) = \sum_{\sim\{\hat{u}\}} f(\hat{u}, \hat{u}_1, \ldots, \hat{u}_J) \prod_{j=1}^{J} \nu_j(\hat{u}_j)$



Marginalization

$\prod_{w=1}^{W+1} \nu_w(\hat{u})$

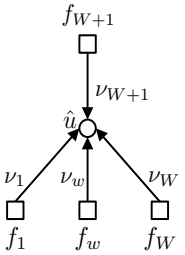


Figure 3.2: Initialization conditions and node operations of the message-passing algorithm on a bipartite tree.

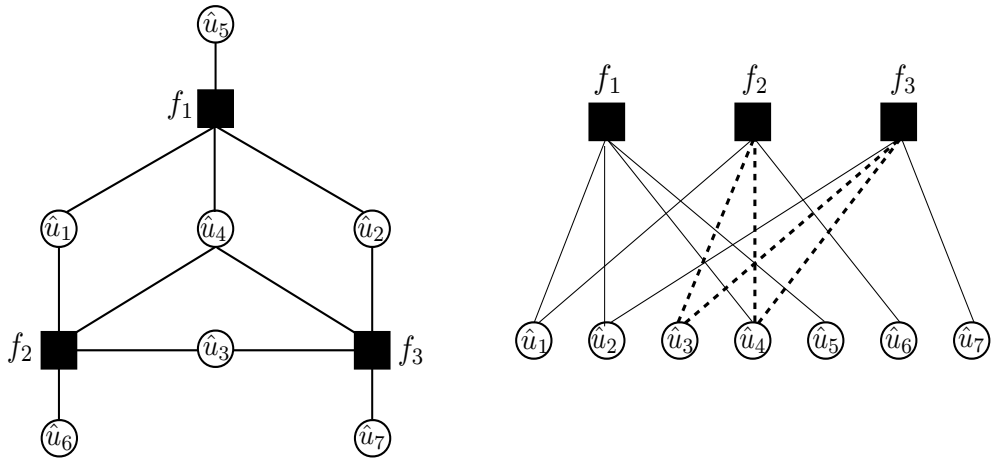


Figure 3.3: A factor graph representation of a linear block code (left). Tanner graph representation of a linear block code emphasizing an existing 4-cycle by dashed lines (right)

the variable nodes are not statistically independent and as a result the marginals of the message passing decoding are not exact, which leads to a suboptimal decoding.

On the other hand, if a linear block code C^{code} of length N has a Tanner graph without cycles, the *maximum-likelihood soft-decision* decoding of C^{code} can be achieved in time $O(n^2)$. However, in [29] it has been shown that cycle-free Tanner graphs cannot support good codes. Specifically, it is proved that if the rate R of a code C^{code} is greater than or equal to 0.5 the *minimum distance* d_{min} is lower than or equal to 2. On the other hand, if $R < 0.5$, C^{code} is obtained from a code of rate ≥ 0.5 and distance ≤ 2 by simply repeating some symbols.

Even though the propagation rules of the sum-product algorithm shown in Fig 3.2 are considered for a cycle-free graphs (trees), they may also be applied to factor graphs with cycles. Although the results of the message-passing algorithm operating in a factor graph with cycles cannot in general be interpreted as exact function summaries, positive results from various decoding codes such as turbo codes and *low-density parity-check* (LDPC) codes have been reported [2], [30], [31].

3.5 Message-passing decoding and its variations

In this section we are interested in the structure of the equations of the sum-product algorithm and their variations. The generic updating rules of the nodes of a factor graph were shown in Fig 3.2. In this section we are interested is defining the variable node update and the check node update as two mapping functions Φ and Ψ respectively. This is to make a distinction between the general equations for a factor graph and the equations used for decoding LDPC codes in a Tanner graph. Some of the following representations of the SPA date back to the work of Gallager [3].

Consider a variable node with degree three which receives two incoming messages $L_{i\to j}$ and $L_{i'\to j}$ (we preserve the notation of the messages of the SPA introduced in section 2.4 on page 20) whose probability mass function may be represented by the vectors (p_0, p_1) and (q_0, q_1) respectively.

Following the generic updating rule of the sum-product algorithm, when the two incoming messages $L_{i\to j}$ and $L_{i'\to j}$ arrive at the variable node, we can write the outgoing normalized message as

$$\Phi(p_0, p_1, q_0, q_1) = \left(\frac{p_0 q_0}{p_0 q_0 + p_1 q_1}, \frac{p_1 q_1}{p_0 q_0 + p_1 q_1} \right) \qquad \text{(Variable)}. \tag{3.15}$$

Note that the text *Variable* just indicates that this equation is performed in the variable node. Similarly, at a check node with degree 3 which represents the function $f(\hat{u}_1, \hat{u}_2, \hat{u}_3) = (\hat{u}_1 \oplus \hat{u}_2 \oplus \hat{u}_3)$ ($\oplus$ means the modulo-2 addition), we have

$$\Psi(p_0, p_1, q_0, q_1) = (p_0 q_0 + p_1 q_1, p_0 q_1 + p_1 q_0) \qquad \text{(Check)}. \tag{3.16}$$

In a similar manner to the variable node, the text *Check* just indicates that this equation is performed in the check node.

Binary probability mass functions can be represented using a single value. *Variable node function* Φ and *check node function* Ψ are described below for different parametrizations. To avoid an excessive notation we define the parametrization for each case using $\Lambda\{\cdot\}$.

- Likelihood ratio (LR):

$$\Lambda(p_0, p_1) = p_0/p_1 \qquad \text{(Definition)} \tag{3.17}$$

$$\Phi(\Lambda_1, \Lambda_2) = \Lambda_1 \Lambda_2 \qquad \text{(Variable)} \tag{3.18}$$

$$\Psi(\Lambda_1, \Lambda_2) = \frac{\Lambda_1 \Lambda_2}{\Lambda_1 + \Lambda_2} \qquad \text{(Check)}. \tag{3.19}$$

- Log-Likelihood ratio (LLR):

$$\Lambda(p_0, p_1) = \ln(p_0/p_1) \qquad \text{(Definition)} \tag{3.20}$$

$$\Phi(\Lambda_1, \Lambda_2) = \Lambda_1 + \Lambda_2 \qquad \text{(Variable)} \tag{3.21}$$

$$\begin{aligned} \Psi(\Lambda_1, \Lambda_2) &= \ln(\cosh((\Lambda_1 + \Lambda_2)/2)) \\ &\quad - \ln(\cosh((\Lambda_1 - \Lambda_2)/2)) \qquad \text{(Check)} \\ &= 2\tanh^{-1}(\tanh(\Lambda_1/2)(\tanh(\Lambda_2/2)) \end{aligned} \tag{3.22}$$

  Note that when $\Omega \gg 1$, $\ln(\cosh(\Omega) \approx |\Omega| - \ln(2))$ (in this specific case $|\cdot|$ indicates absolute value), thus the approximation is

$$\begin{aligned} \Psi^{min}(\Lambda_1, \Lambda_2) &\approx \left|\frac{(\Lambda_1 + \Lambda_2)}{2}\right| - \left|\frac{(\Lambda_1 - \Lambda_2)}{2}\right| \\ &= \operatorname{sgn}(\Lambda_1)\ \operatorname{sgn}(\Lambda_1)\ \min(|\Lambda_1|, |\Lambda_2|), \end{aligned} \tag{3.23}$$

  which is known as the *min-sum* update rule.

- Likelihood difference (LD):

$$\Lambda(p_0, p_1) = p_0 - p_1 \qquad \text{(Definition)} \tag{3.24}$$

$$\Phi(\Lambda_1, \Lambda_2) = \frac{\Lambda_1 + \Lambda_2}{1 + (\Lambda_1\Lambda_2)} \qquad \text{(Variable)} \tag{3.25}$$

$$\Psi(\Lambda_1, \Lambda_2) = \Lambda_1\Lambda_2 \qquad \text{(Check).} \tag{3.26}$$

All the above representations of Λ are used to generate different decoding algorithms. For example the most common hardware implementation of SPA is using the parametrization in (3.21) and (3.23).

In the case that the nodes have degree larger than three, the above functions may be applied in a "nested" fashion as

$$\Phi(\Lambda_1, \Lambda_2, \ldots, \Lambda_n) = \Phi(\Lambda_1, \Phi(\Lambda_2, \ldots, \Lambda_n)) \tag{3.27}$$

$$\Psi(\Lambda_1, \Lambda_2, \ldots, \Lambda_n) = \Psi(\Lambda_1, \Psi(\Lambda_2, \ldots, \Lambda_n)), \tag{3.28}$$

this approach is useful when the functions Φ and Ψ in the nodes are implemented as a *lookup table* (LUT). Another similar approach is to decompose nodes in pairs to perform the functions Φ and Ψ in a parallel manner, for instance if the degree of the nodes is 4, we have

$$\Phi(\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4) = \Phi(\Phi(\Lambda_1, \Lambda_2), \Phi(\Lambda_3, \Lambda_4)) \tag{3.29}$$

$$\Psi(\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4) = \Psi(\Psi(\Lambda_1, \Lambda_2), \Psi(\Lambda_3, \Lambda_4)). \tag{3.30}$$

Combining both approaches, lookup tables and parallel implementation, the speed of the decoding process is governed by the resolution of the variables Λ_i (i.e. number of bits to represent a variable Λ_i).

Note that the fragmentations in (3.27)–(3.30) of the local node functions Φ and Ψ, can lead to faster hardware implementation for LDPC decoders due to the capability of pipelining or semi-parallel implementation of the node operations. Since our approach is to design Φ and Ψ as lookup tables, we will take advantage of the decomposition of the nodes to reduce the size of Φ and Ψ which brings a significant decrement in the necessary resources for a hardware implementation. Detailed explanation and an example of the benefits are described in subsection 3.6.2 on page 40.

3.6 Discretized message-passing decoding

In this section, a message-passing decoding algorithm using lookup tables for the local functions Φ and Ψ is proposed. A decomposition in the nodes is necessary to reduce complexity in the implementation of the proposed decoding lookup tables. More precisely each variable and check node is decomposed into a set of degree-3 nodes i.e. two inputs and one output.

The decomposition presented in this section will be used in Chapter 4 to describe the construction of the proposed decoding lookup tables via quantized density evolution. First conventional message-passing decoding on a Tanner graph without decomposition is introduced, later decomposition is applied, graphical representations of both cases are shown in Fig. 3.4.

3.6.1 Discretized message-passing decoding on a Tanner graph

Here we consider a Tanner graph without decomposition for a regular (d_v, d_c)-LDPC code. Message-passing decoding for this case is as follows.

At iteration ℓ, the check node with degree d_c finds the check-to-variable node message $L_1, \ldots, L_{d_c}$ from alphabet $\mathcal{L}$ using $d_c - 1$ incoming messages $V_1, \ldots, V_{d_c-1}$ from alphabet $\mathcal{V}$. The decoding mapping function that performs these operations can be described as follows:

$$\Psi_c^{(\ell)} : \mathcal{V}^{d_c-1} \to \mathcal{L}. \tag{3.31}$$

This step is shown diagrammatically in Fig. 3.4-(a).

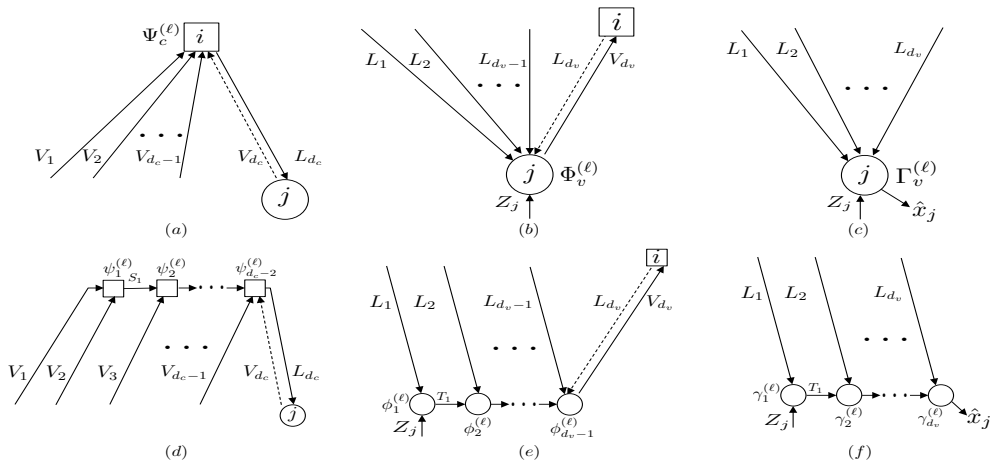


Figure 3.4: Decomposition of the variable node and check node into a set of two-input mapping functions (or two-input lookup tables). (a) Check node update operation. (b) Variable node update operation. (c) Hard decision operation on the variable node. (d) Decomposition of the check node update operation $\Psi_c^{(\ell)}$ into the set of two-input mapping functions $\psi_1^{(\ell)}, \ldots, \psi_{d_c-2}^{(\ell)}$. (e) Decomposition of the variable node update operation $\Phi_v^{(\ell)}$ into the set of two-input mapping functions $\phi_1^{(\ell)}, \ldots, \phi_{d_v-1}^{(\ell)}$. (f) Decomposition of the hard decision operation $\Gamma_v^{(\ell)}$ into the set of two-input mapping functions $\gamma_1^{(\ell)}, \ldots, \gamma_{d_v}^{(\ell)}$.

Similarly, at iteration ℓ, the variable node with degree d_v finds the variable-to-check messages $V_1, \ldots, V_{d_v}$ using the channel value Z from alphabet $\mathcal{Z}$ and $d_v - 1$ incoming messages $L_1, \ldots, L_{d_v-1}$. The decoding mapping function that performs this operation can be described as follows:

$$\Phi_v^{(\ell)} : \mathcal{Z} \times \mathcal{L}^{d_v-1} \to \mathcal{V}. \tag{3.32}$$

This step is shown diagrammatically in Fig. 3.4-(b).

At iteration ℓ, the variable node with degree d_v calculates the estimate $\hat{u} \in \{0, 1\}$ using the channel value Z and d_v incoming messages $L_1, \ldots, L_{d_v}$:

$$\Gamma_v^{(\ell)} : \mathcal{Z} \times \mathcal{L}^{d_v} \to \{0, 1\}. \tag{3.33}$$

This step is shown diagrammatically in Fig. 3.4-(c). This proceeds iteratively for $\ell = 1, 2, \ldots$ until convergence is detected, or the maximum number of iterations is reached.

3.6.2 Discretized message-passing decoding on a decomposed Tanner graph

Throughout this section, we use the terms mapping and LUTs interchangeably. In the context of LDPC decoding, a degree-d node could naively use a lookup table with $d-1$ inputs to produce an output.

A general lookup table of Δ address bits requires 2^{Δ} memory locations or indices. Assuming each input has the same resolution in bits denoted by Δ, the number of indices (memory locations) for a LUT with $d-1$ inputs is $2^{(d-1)\Delta}$. On the other hand, performing a decomposition of the same degree-d node into a series of degree-3 nodes, and considering a LUT implementation for each, the number of indices is $(d-2)2^{2\Delta}$; clearly $2^{(d-1)\Delta} > (d-2)2^{2\Delta}$ for cases of interest.

As an example to illustrate the above observation, in Fig. 3.5, we use a check node with degree equal to 6 ($d_c = 6$) and we assume that each incoming message V is represented using 3 bits, this is, $\Delta = 3$. The number of locations (addresses) needed to implement the check node decoding lookup table $\Psi_c^{(\ell)}$ without decomposition is equal to 32768 locations. On the other hand, if the check node decoding lookup table $\Psi_c^{(\ell)}$ is implemented using the set of two-input lookup tables $\psi_1^{(\ell)}, \ldots, \psi_4^{(\ell)}$, the number of locations in drastically decreased to 256 locations.

From the above example, we can see that using a decomposed node drastically reduces the hardware implementation resources, but in contrast, this introduces a delay in time processing which can be overcome or at least mitigated by applying pipeline or semi-parallel implementation previously described in (3.27)–(3.30).

When high rate LDPC codes are considered, the decomposition reduces dramatically the memory requirements for the implementation of a message-passing decoding using LUTs. This worst-case analysis assumes arbitrary lookup tables; symmetries in the lookup tables may reduce the memory requirements.

Now, we proceed to describe the decomposition of the variable and check nodes into a set of degree-3 nodes i.e. two inputs and one output. Consider that the message-passing decoding maps $\Psi_c^{(\ell)}$, $\Phi_v^{(\ell)}$ and $\Gamma_v^{(\ell)}$ depicted in Fig. 3.4-(a), Fig. 3.4-(b) and Fig. 3.4-(c) respectively can be implemented by lookup tables (LUTs). Then the mapping function $\Psi_c^{(\ell)}$ (check node c at iteration ℓ) is decomposed into a series of smaller mapping functions $\psi_1^{(\ell)}, \ldots, \psi_{d_c-2}^{(\ell)}$, each with two input variables. As a result of the decomposition, to communicate $\psi_1^{(\ell)}, \ldots, \psi_{d_c-2}^{(\ell)}$, we introduce $S_i \in \mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$, for $i = 1, \ldots, d_c - 3$, where $|\cdot|$ indicates the

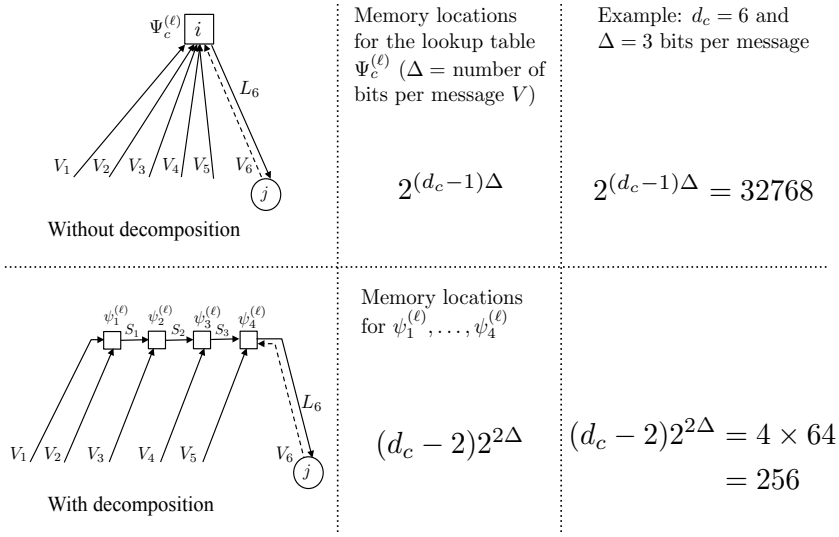


Figure 3.5: Required memory locations to implement both the decoding lookup table $\Psi_c^{(\ell)}$ and its decomposition $\psi_1^{(\ell)}, \ldots, \psi_4^{(\ell)}$. This example consider a check node with degree $d_c = 6$ and incoming messages V with a resolution of $\Delta = 3$ bits.

cardinality of the set. Thus, the decomposition is:

$$\psi_1^{(\ell)} : \mathcal{V}^2 \to \mathcal{S}, \tag{3.34}$$

and for $i = 2, 3, \ldots, d_c - 3$,

$$\psi_i^{(\ell)} : \mathcal{S} \times \mathcal{V} \to \mathcal{S} \tag{3.35}$$

and finally,

$$\psi_{d_c-2}^{(\ell)} : \mathcal{S} \times \mathcal{V} \to \mathcal{L}. \tag{3.36}$$

The decomposition of $\Psi_c^{(\ell)}$ is shown in Fig. 3.4-(d).

Continuing the decomposition process, the decoding mapping function $\Phi_v^{(\ell)}$ (variable node v at iteration ℓ) is decomposed into a series of smaller mapping functions $\phi_1^{(\ell)}, \ldots, \phi_{d_v-1}^{(\ell)}$ each with two input variables. To communicate $\phi_1^{(\ell)}, \ldots, \phi_{d_v-1}^{(\ell)}$ we introduce $T_i \in \mathcal{T} = \{1, \ldots, |\mathcal{T}|\}$, for $i = 1, \ldots, d_v - 2$. Thus, the decomposition is:

$$\phi_1^{(\ell)} : \mathcal{Z} \times \mathcal{L} \to \mathcal{T}, \tag{3.37}$$

and for $i = 2, 3, \ldots, d_v - 2$,

$$\phi_i^{(\ell)} : \mathcal{T} \times \mathcal{L} \to \mathcal{T} \tag{3.38}$$

and finally,

$$\phi_{d_v-1}^{(\ell)} : \mathcal{T} \times \mathcal{L} \to \mathcal{V}. \tag{3.39}$$

The decomposition of $\Phi_v^{(\ell)}$ is shown in Fig. 3.4-(e).

Lastly, the mapping function $\Gamma_v^{(\ell)}$ (hard decision on variable node v at iteration ℓ) is decomposed into a set of smaller mapping functions $\gamma_1^{(\ell)}, \ldots, \gamma_{d_v}^{(\ell)}$, each has two input variables. For interconnecting these subdivisions, we use T_i as in the case of $\Phi_v^{(\ell)}$ but, for $i = 1, \ldots, d_v - 1$.

Hence, the decomposition is as follows:

$$\gamma_1^{(\ell)} : \mathcal{Z} \times \mathcal{L} \to \mathcal{T}, \tag{3.40}$$

and for $i = 2, 3, \ldots, d_v - 1$

$$\gamma_i^{(\ell)} : \mathcal{T} \times \mathcal{L} \to \mathcal{T} \tag{3.41}$$

and finally,

$$\gamma_{d_v}^{(\ell)} : \mathcal{T} \times \mathcal{L} \to \{0, 1\}. \tag{3.42}$$

The decomposition of $\Gamma_v^{(\ell)}$ is shown in Fig. 3.4-(f). It is worth mentioning that the set of two-input mapping functions $\{\phi_1^{(\ell)}, \ldots, \phi_{d_v-1}^{(\ell)}\}$ used to generate the variable-to-check messages V are exactly the same as the set of two-input mapping functions $\{\gamma_1^{(\ell)}, \ldots, \gamma_{d_v-1}^{(\ell)}\}$ which generate the hard decision estimates $\hat{u}$.

So far we have only described the decomposition of the variable and check nodes into a set of degree-3 nodes. The reason of this decomposition is because we want to allocated two-input decoding lookup tables in each of this degree-3 nodes i.e. because this decomposition reduces the requirements of such decoding lookup tables.

Note that somehow at this moment we only have a decomposed Tanner graph and we know that the decoding will be discrete and will be performed by lookup tables, but what is the content of these lookup tables?. The answer requires that first we present the optimal quantization algorithm that we are going to use to generate discretized channel and decoder messages, section 3.7. Later, using this quantization algorithm we will be able to explain the max-LUT method which is the method to design a decoding lookup table that maximizes mutual information, section 3.8.

3.7 Optimal quantizer that maximizes mutual information

Recently, the problem of optimal quantization of communications channels has been shown to have a connection with the problem of classification from statistical learning theory [25]. A common classification problem deals with a Markov chain $X \to Y \xrightarrow{Q} Z$, where X is a variable of interest, Y is an observation, Q is a function called a *classifier* and Z is the classification, essentially an estimate of X [32]. A well-known metric for classification is to minimize conditional entropy [33]:

$$\min_{Q} H(X|Z), \tag{3.43}$$

which is equivalent to:

$$\max_{Q} I(X;Z) = H(X) - \min_{Q} H(X|Z). \tag{3.44}$$

Since channel capacity is the maximization of mutual information, a reasonable metric for designing channel quantizers is to similarly maximize the mutual information between the channel input and the quantizer output.

The optimal quantizer Q^* which maximizes the mutual information between X and Z:

$$Q^* = \operatorname*{argmax}_{Q \in \mathcal{Q}} I(X;Z), \tag{3.45}$$

can be found efficiently over the set of all possible quantizers $\mathcal{Q}$ using dynamic programming [25], when X is binary. The optimal quantizer Q^* that maximizes mutual information between X and Z is deterministic, meaning that Q^* is a $|\mathcal{Y}| \times |\mathcal{Z}|$ binary matrix. This optimal quantizer is found by an algorithm summarized below; in this work this algorithm is referred as the *Quantization Algorithm*.

For a given discrete memoryless channel (DMC) with input X and output Y using the alphabets $\mathcal{X}$ and $\mathcal{Y}$ respectively, finding the optimal quantizer Q^* which maps outputs Y of the DMC to a certain number of quantization levels Z using alphabet $\mathcal{Z}$, i.e. $X \to Y \xrightarrow{Q^*} Z$, reduces to finding the boundaries

$$\{a_1^*, a_2^*, \dots a_{|\mathcal{Z}|-1}^*\} \in \mathcal{Y} \tag{3.46}$$

which maximize mutual information between X and Z ($|\cdot|$ indicates cardinality of the set).

Let the channel input $X \sim p_x$ such that $p_x = \Pr(X = x)$. Let the transition probabilities between X and Z be $P_{z|x} = \Pr(Z = z|X = x)$.

Thus the highest mutual information between X and Z that we can achieve by using the optimal quantizer Q^* is

$$I(X;Z) = \sum_z \sum_x p_x P_{z|x} \log \frac{P_{z|x}}{\sum_{x'} p_{x'} P_{z|x'}}. \tag{3.47}$$

Clearly, if $|\mathcal{Z}| \geq |\mathcal{Y}|$, the trivial quantizer which maps each channel output y to a unique quantizer output z will result. On the other hand, when $|\mathcal{Z}| < |\mathcal{Y}|$ the optimal quantizer Q^* maps each channel output y to a single value z', for which $Q_{z'|y} = 1$, and for all other values of z, $Q_{z|y} = 0$.

Doing this, the optimal quantizer Q^* builds $|\mathcal{Z}|$ sets $(\mathcal{A}_1, \ldots, \mathcal{A}_{|\mathcal{Z}|})$ consisting of consecutive channel outputs y sorted according to

$$\log \frac{P_{1|0}}{P_{1|1}} < \log \frac{P_{2|0}}{P_{2|1}} < \cdots < \log \frac{P_{|\mathcal{Y}||0}}{P_{|\mathcal{Y}||1}} \tag{3.48}$$

[25, Lemma 3].

Thus, $\mathcal{A}_1$ is the set

$$\{1, 2, \ldots, a_1^*\}, \tag{3.49}$$

$\mathcal{A}_2$ is the set

$$\{a_1^* + 1, \ldots, a_2^*\}, \tag{3.50}$$

etc., and $\mathcal{A}_{|\mathcal{Z}|}$ is the set

$$\{a_{|\mathcal{Z}|-1}^* + 1, \ldots, |\mathcal{Y}|\}, \tag{3.51}$$

with

$$1 \leq a_1^* < a_2^* < \cdots < a_{|\mathcal{Z}|-1}^* < |\mathcal{Y}|. \tag{3.52}$$

The quantization algorithm that finds the optimal quantizer Q^* is implemented using dynamic programming [25], and its implementation in Matlab is available in [34].

3.7.1 Partial mutual information

For the purpose of describing the algorithm that finds the optimal quantizer Q^*, partial mutual information is presented. Partial mutual information ι is the contribution that one or more quantizer values makes to the total mutual information. For a deterministic quantizer, the total mutual information is

$$I(X;Z) = \sum_{z=1}^{|\mathcal{Z}|} \sum_{x=1}^{2} p_x \sum_{y \in \mathcal{A}_z} P_{y|x} \log \frac{\sum_{y' \in \mathcal{A}_z} P_{y'|x}}{\sum_{x'} p_{x'} \sum_{y' \in \mathcal{A}_z} P_{y'|x'}}, \tag{3.53}$$

since $Q_{z|y} = 1$ if and only if $y \in \mathcal{A}_z$.

Under the quantization mapping from Y to Z, the preimage of quantizer output z is $\mathcal{A}_z$. The partial mutual information ι_z for this z is:

$$\iota_z = \sum_{x=1}^{2} p_x \sum_{y \in \mathcal{A}_z} P_{y|x} \log \frac{\sum_{y' \in \mathcal{A}_z} P_{y'|x}}{\sum_{x'} p_{x'} \sum_{y' \in \mathcal{A}_z} P_{y'|x'}} \tag{3.54}$$

so total mutual information is the sum of all partial mutual information terms:

$$I(X;Z) = \sum_{z=1}^{|\mathcal{Z}|} \iota_z. \tag{3.55}$$

Further, let consecutive values $a' + 1$ to a, with $a' < a < |\mathcal{Y}|$, be assigned to a single quantizer output. Denote by $\iota(a' \to a)$, the partial mutual information:

$$\iota(a' \to a) = \sum_{x=1}^{2} p_x \sum_{y=a'+1}^{a} P_{y|x} \log \frac{\sum_{y'=a'+1}^{a} P_{y'|x}}{\sum_{x'} p_{x'} \sum_{y'=a'+1}^{a} P_{y'|x'}}. \tag{3.56}$$

So if $\mathcal{A}_z = \{a_{z-1} + 1, \ldots, a_z\}$, then $\iota_z = \iota(a_{z-1} + 1 \to a_z)$.

3.7.2 Quantization algorithm

The quantization algorithm uses dynamic programming principles applied to partial mutual information. The algorithm has state value $\rho_z(y)$, which is the maximum partial mutual information when 1 to y values of Y are quantized to 1 to z values of Z. This can be computed recursively by conditioning on the state value at time index $z - 1$:

$$\rho_z(a) = \max\big(\rho_{z-1}(a') + \iota(a' \to a)\big), \tag{3.57}$$

where the maximization is taken over $a' \in \{z-1, \ldots, a-1\}$.

For convenience the algorithm is denoted as $Q^* = \text{Quant}(P_{x,y}, |\mathcal{Z}|)$, where Q^* is a $|\mathcal{Z}| \times |\mathcal{Y}|$ matrix.

The steps of the quantization algorithm are as follows:

1. Inputs: joint distribution $P_{x,y}$ and fix $|\mathcal{Z}|$.
2. Initialize $\rho_0(0) = 0$
3. Precompute partial mutual information. For each $a' \in \{0, 1, \ldots, |\mathcal{Y}|-1\}$ and for each $a \in \{a'+1, \ldots, t\}$ (where) $t = \min(a'+1+|\mathcal{Y}|-|\mathcal{Z}|, |\mathcal{Y}|)$:
   - compute $\iota(a' \to a)$ according to (3.56)
4. Recursion. For each $z \in \{1, \ldots, |\mathcal{Z}|\}$ and for each $a \in \{z, \ldots, z+|\mathcal{Y}|-|\mathcal{Z}|\}$,
   - compute $\rho_z(a)$ according to (3.57),
   - store the local decision $h_z(a)$:
     $h_z(a) = \text{argmax}\big(\rho_{z-1}(a') + \iota(a' \to a)\big)$,
     where the maximization is taken over $a' \in \{z-1, \ldots, a-1\}$.
5. Find the optimal quantizer by traceback. Let $a^*_{|\mathcal{Z}|} = |\mathcal{Y}|$. For each $z \in \{|\mathcal{Z}|-1, |\mathcal{Z}|-2, \ldots, 1\}$:
   $a^*_z = h_{z+1}(a^*_{z+1})$.
6. Outputs:
   - The optimal boundaries $a^*_1, a^*_2, \ldots, a^*_{|\mathcal{Z}|-1}$. Equivalently, output the matrix Q^*, where row z of Q^* has ones in columns $a_{z-1}+1$ to a_z and zeros in all other columns.
   - The maximum mutual information, $\rho_{|\mathcal{Z}|}(|\mathcal{Y}|)$.

In following sections the quantization algorithm that finds the optimal quantizer Q^* is denoted as $Q^* = \text{Quant}(P_{x,y}, |\mathcal{Z}|)$. This algorithm receives two inputs: $P_{x,y}$ which is a $2 \times |\mathcal{Y}|$ joint probability matrix and an integer $|\mathcal{Z}|$, the output is Q^* which is a $|\mathcal{Y}| \times |\mathcal{Z}|$ binary matrix. Since we can compute $P_{y|x}$, the matrix multiplication of $P_{y|x}$ and Q^* will produce a reduced conditional probability distribution $P_{z|x}$ which is a $2 \times |\mathcal{Z}|$ matrix. Using $P_{z|x}$, we can compute the mutual information $I(X;Z)$ using (3.47) on page 44.

3.8 Max-LUT method

To describe the max-LUT method the following observations are useful.

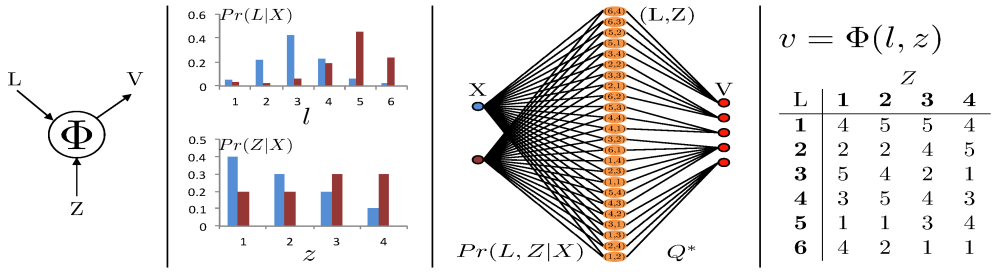
- **Observation one**. Quantization schemes using a few bits per message are inherent to modern communication systems, mainly because the implementation of the algorithms is made in fixed precision hardware (e.g. mobile devices, SSD, etc.).

- **Observation two**. Recalling the mathemathical definition of channel capacity in Chapter 2, we learned that the maximization of mutual information reduces the uncertainty of the channel output. In other words, maximization of mutual information is a natural metric for communications channels because it reduces the uncertainty of a random variable in such a way to maximize achievable communication rates. This simple insight on the channel output tell us that the decoder will be immediately affected by any type of quantization scheme concatenated to the output of the channel, this is, if the the channel quantizer increases the mutual information between its output and the input to the channel, the decoder will have a better chance to correct errors successfully and vice versa.

- **Observation three**. Recalling the message-passing decoding on factor graphs, we note that every node in the factor graph performs a local computation/marginalization using its incoming inputs as arguments. This means that the local functions seek to pass good beliefs about a random variable, in others words, local decoding functions in graphs as well as channel quantizers try to maximize mutual information.

- **Observation four**. Lookup tables are widely used in modern hardware implementations because they can replace real-valued or fixed-point operations (e.g. multiplications, divisions, etc.), while producing higher data rates and a potentially easier VLSI implementation because of their suitability for synthesization through Boolean algebra and advanced optimization techniques.

Combining the above observations, a good idea will be to design a lookup table that maximizes mutual information for both channel quantization and message-passing decoding algorithms based on graphs (local decoding). Precisely the above description is the result of the max-LUT method. Explicitly, the max-LUT method will produce a suitable lookup



(a) (b) (c) (d)

Figure 3.6: Overview of the designing process for the mapping function Φ. (a) degree-2 LDPC variable node with inputs L and Z and output V. (b) Input distributions $\Pr(L|X)$ and $\Pr(Z|X)$. (c) Joint distribution $\Pr(L, Z|X)$ quantized to five-valued variable V using the optimal quantizer Q^* which maximizes the mutual information between X and V. (d) The resulting lookup table corresponding to Q^*. This lookup table computes $V = \Phi(L, Z)$ to maximize mutual information.

table that implicitly through its "decisions" maximizes mutual information and, it can be applied to channel quantization and message-passing decoding on factor graphs.

3.9 Constructing a decoding mapping function via max-LUT method

In this subsection, we present the idea of mapping functions for LDPC decoding and its connection with the optimal quantizer Q^* mentioned in Subsection 3.7. Here a decoding mapping function is considered a lookup table (LUT) in which each input has a defined resolution in bits.

Given two messages L and Z, a lookup table Φ provides the value V:

$$V = \Phi(L, Z), \tag{3.58}$$

if the check-to-variable node message is L and the channel message is Z, and L and Z are two messages about an information bit $X \in \{0, 1\}$, then we want to find the optimal lookup table Φ^* which maximizes mutual information between X and V, that is:

$$\Phi^* = \arg\max_{\Phi} I(X; V), \tag{3.59}$$

This problem is identical to (3.45), if we consider the pair (L, Z) as a single variable.

The direct design of the mapping function Φ can be described using a simple degree-2 variable node, see Fig. 3.6-(a). The variable node represents an information bit (or codeword symbol) $X = 0$ or 1. The incoming check-to-variable node message is L, quantized to 6 values (not a power of 2, to make the example clear), and a channel message Z has been quantized to 2 bits. Assume the conditional distributions:

$$P_{L|X}(l|x) = \Pr(L = l|X = x) = \begin{bmatrix} 0.05 & 0.22 & 0.42 & 0.23 & 0.06 & 0.02 \\ 0.03 & 0.02 & 0.06 & 0.19 & 0.45 & 0.25 \end{bmatrix} \text{ and}$$

$$P_{Z|X}(z|x) = \Pr(Z = z|X = x) = \begin{bmatrix} 0.40 & 0.30 & 0.20 & 0.10 \\ 0.20 & 0.20 & 0.30 & 0.30 \end{bmatrix}$$

illustrated in Fig. 3.6-(b).

Then the joint distribution $P_{L,Z|X}(l, z|x)$ can be easily constructed by computing

$$P_{L,Z|X}(l, z|x) = \begin{bmatrix} P_{L|X}(l|x = 0) \otimes P_{Z|X}(z|x = 0) \\ P_{L|X}(l|x = 1) \otimes P_{Z|X}(z|x = 1) \end{bmatrix} =$$

$$\begin{bmatrix} 0.020 & 0.088 & 0.168 & \dots & 0.023 & 0.006 & 0.002 \\ 0.006 & 0.004 & 0.012 & \dots & 0.057 & 0.135 & 0.075 \end{bmatrix}, \tag{3.60}$$

where $\otimes$ is the Kronecker product. Each column in the resulting 2×24 matrix $P_{L,Z|X}(l, z|x)$ can be labeled as a pair (l, z), for instance the second column in (3.60) is $(l, z) = (1, 2)$ and the 23th column is $(l, z) = (6, 3)$.

In order to meet the sorting condition in (3.48) (i.e. the columns of the matrix $P_{L,Z|X}(l, z|x)$ are sort from the smallest LLR until the highest LLR), the position of the columns in $P_{L,Z|X}(l, z|x)$ will be permuted. For the specific example used throughout this section the final order of the columns for (3.60) can be seen in Fig. 3.6-(c), where the first column and the last column are $(l, z) = (6, 4)$ and $(l, z) = (1, 2)$ respectively. For the current example, we assume that we want five levels of quantization for the variable-to-check message V.

Therefore, using $P_{L,Z|X}(l, z|x)$ and $|\mathcal{V}| = 5$ we use the *quantization algorithm* described in subsection 3.7.2 to find the optimal quantizer Q^*

which maximizes mutual information between X and V. The resulting quantizer Q^* has the following transpose:

$$(Q^*)^T = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \end{bmatrix}. \tag{3.61}$$

This optimal quantizer Q^* quantizes pairs (l, z) to the output variable-to-check message V, see the right-hand side of Fig. 3.6-(c). Finally, using the pairs (l, z), that map to v, we can find a mutual-information maximizing decoding lookup table, which is shown in Fig. 3.6-(d). More precisaly the lookup table Φ (or mapping function) can be constructed as $\Phi(l, z) = v$ if $(Q^*)^T(v, (l, z)) = 1$, where we can easily identify that when $v = 1$ the first row of $(Q^*)^T$ has 5 ones in the columns: $(3, 4), (5, 1), (5, 2), (6, 3)$ and $(6, 4)$. These five columns mapped to $v = 1$ are graphically represented in Fig 3.6-(c) by the first five lines connecting the first circle which corresponds to the case of $v = 1$. On the other hand, in Fig. 3.6-(d) we can observe that all positions (l, z) equal to 1 in the two-input resulting lookup table are the columns equal to one in row one of $(Q^*)^T$.

In belief propagation, if the inputs to a node are permuted, the output value usually does not change. However, in the proposed method, this property does not hold, for example a degree-4 node with inputs $(2, 0, 4)$ may produce a different output than with the input $(0, 4, 2)$. This is due to the possible asymmetries inherent to mutual-information maximizing quantization, see [25, Eq. (12)] for an example.

Note that the resulting lookup table Φ in Fig. 3.6-(d) is not symmetric because the conditional input distributions are not symmetric. Recall that the optimal quantizer results in nonuniform boundaries. Therefore, when the quantizer uses an even number of levels and the distributions are symmtetric the quantization regions are symmetric respect to the origin. This is not the case of this example.

3.10 Summary

At the beginning of this chapter, we described the nature of message-passing decoding algorithms working on a factor graph, which may or may not have cycles. Later we described the equations of a sum-product algorithm and some of the most common variations for these expressions. A decomposition of the nodes in the Tanner graph is described in this

chapter as a solution to reduce the size of lookup tables implemented by message-passing decoding algorithms.

In the middle of the chapter, we presented an optimal quantizer that maximizes mutual information and we proceeded to its explanation.

At the end of the chapter, we connect all ideas to clearly, and by an example describe the max-LUT method, which constructs a lookup table or mapping function that maximizes mutual information; such a lookup table can be used as a channel quantizer or as a local decoding operation (message-passing decoding).

Chapter 4

Discretized density evolution

In this chapter, we describe the construction of decoding mapping functions using the max-LUT method presented in Chapter 3. The proposed construction technique is based on a discretized density evolution algorithm which only uses the probability distribution from the channel. Given a probability distribution, our density evolution algorithm builds decoding mapping functions that maximizes mutual information and are applied systematically during the decoding process. Before fully describing the proposed discretized density evolution algorithm, first we give the idea behind general density evolution using the Gaussian approximation.

4.1 Density evolution

The density evolution algorithm (DEA) is a technique which allows analyzing the decoding performance of the message-passing algorithm on ensembles of LDPC codes with specified degree distributions. Since in this research we work with regular (d_v, d_c)-LDPC codes, the ensembles are defined by constant degrees d_c and d_v for each row and column respectively.

The analysis made by the density evolution algorithm shows that the messsage-passing decoding for a sufficiently long LDPC code presents a threshold effect, this is, communication is reliable "beyond" this threshold and unreliable below it.

We denote the decoding threshold as α^*, in the case of the binary symmetric channel α^* is the cross-over probability ε and in the case of the BI-AWGNC α^* is the standard deviation σ. Note that for a regular (d_v, d_c)-LDPC code it is possible to design different parity check matrices. This was considered in [9] where it was shown that the decoding per-

formance of ℓ rounds of message-passing decoding on a randomly chosen (d_v, d_c)-LDPC code ensemble converges to the ensemble average as the length of the code $n \to \infty$.

4.2 Density evolution for regular LDPC codes via Gaussian aproximation

The name of density evolution algorithm was used because this algorithm tracks the probability density functions (pdf) of messages passed during iterative decoding. Thus, the task of the DEA is to predict under which channel value α (ε for the BSC and σ for the BI-AWGNC) the error probability of the decoding process will converge to zero. Clearly there is more than one channel value α in which the error probability goes to zero, therefore the largest channel value α^* is the decoding threshold of the previously specified (d_v, d_c)-LDPC code when $n \to \infty$.

In this section, we describe how to compute the decoding threshold α^* of a long regular (d_v, d_c)-LDPC code. This process applies to various binary-input channels with symmetric outputs, but we restrict the analysis to the BI-AWGNC, therefore we want to compute the decoding threshold σ^*.

The Gaussian approximation of the DEA uses the idea of approximating the pdfs of the messages used during the iterative decoding process with Gaussians. Gaussians can be specified with only two parameters; the mean and the variance. As a result we only have to analyze the evolution of these two values. In this section we will use a further simplification that allows to represent the messages using only the mean of the Gaussian. This is possible by assuming *consistent normal densities*, this is, the variance of the Gaussian is twice the value of the mean. Further explanation and proofs are given in [35].

We assume that the all-zeros codeword $\mathbf{u} = [0, 0, \ldots, 0]$ is mapped to $\mathbf{x}$ using $x_j = (-1)^{u_j}$, later $\mathbf{x}$ is transmitted into the channel. Therefore, the initial channel message L_0 from the BI-AWGNC is $L_0 = 2y/\sigma^2$. Besides, $L_0 \sim \mathcal{N}(m_0 = 2/\sigma^2, 2m_0 = 4/\sigma^2)$, where m_0 is the initial mean of the initial channel message L_0. Continuing with the iterative decoding process, we track the messages $V_{j\to i} \sim \mathcal{N}(m^{(\ell)}, 2m^{(\ell)})$ from the variables to the check nodes. To obtain the message $V_{j\to i}$ we compute its mean $m^{(\ell)}$ as

$$m^{(\ell)} = m_0 + (d_v - 1)\mu^{(\ell-1)}, \tag{4.1}$$

where we considered that the means $\mu^{(\ell)}$ of the check-to-variable node messages are the same at each iteration due to the regularity of the LDPC code. For the check-to-variable node message $L_{i\to j} \sim \mathcal{N}(\mu^{(\ell)}, 2\mu^{(\ell)})$ we compute its mean $\mu^{(\ell)}$ by

$$\mu^{(\ell)} = \Upsilon^{-1}\left(1 - \left[1 - \Upsilon\left(m_0 + (d_v - 1)\mu^{(\ell-1)}\right)\right]^{d_c - 1}\right), \tag{4.2}$$

where for $\mu \geq 0$, we define

$$\Upsilon(\mu) = 1 - \frac{1}{\sqrt{4\pi\mu}} \int_{-\infty}^{\infty} \tanh(\tau/2) \exp\left[-(\tau - \mu)^2/(4\mu)\right] d\tau, \tag{4.3}$$

with approximation $\Upsilon(\mu) = \exp(-0.4527\mu^{0.86} + 0.0218)$. At this point we are enabled to track the pdfs of the messages $V_{j\to i}$ and $L_{i\to j}$ using (4.1) and (4.2) respectively. The iterative process is initialized with $\mu^{(0)} = 0$. Note that the evolution of the pdfs is governed by the degrees of the regular LDPC code (d_v, d_c) and the channel parameter σ (for the BI-AWGNC case). Since the degrees are fixed, the value that dominates the process is the channel parameter σ. As a result, two possible behaviours of the densities can be seen. In one situation when the channel parameter σ is quite large i.e. E_b/N_0 is small, the mean $\mu^{(\ell)}$ converges to a small fixed point. This means that the outgoing variable node message $V_{j\to i}$ has a certain probability to have negative values which indicates a non-zero probability of errors. On the other hand, for some smaller values of σ i.e. E_b/N_0 is large, the mean $\mu^{(\ell)}$ goes go infinity. Therefore, the pdf of the message $V_{j\to i}$ has all of its probability on positive values, this is,

$$\lim_{\ell\to\infty} \int_{-\infty}^{0} p\big(V_{j\to i}^{(\ell)}(\tau)\big)\, d\tau = 0, \tag{4.4}$$

where $p\big(V_{j\to i}^{(\ell)}(\tau)\big)$ is the pdf of the message $V_{j\to i}$ which depends on the channel parameter σ. Then the decoding threshold σ^* is given by

$$\sigma^* = \sup\left\{\sigma : \lim_{\ell\to\infty} \int_{-\infty}^{0} p\big(V_{j\to i}^{(\ell)}(\tau)\big)\, d\tau = 0\right\}. \tag{4.5}$$

We use the following example to graphically describe the above interpretation of the DEA using the Gaussian approximation. Consider

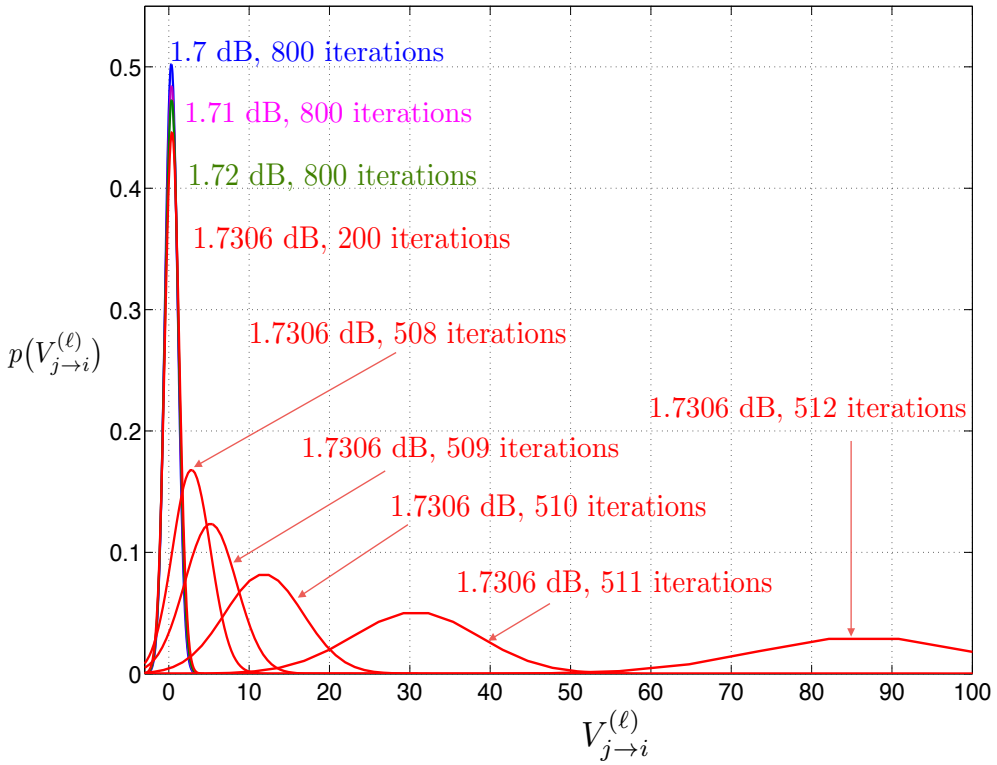


Figure 4.1: Evolution of the Gaussian pdfs for the variable-to-check message $V_{j\to i}^{(\ell)}$. Different values of E_b/N_0 are used.

a regular $(d_v = 4, d_c = 6)$-LDPC code with $R = 1/3$ and recall that $E_b/N_0(\text{dB}) = 10\log_{10}(1/(2R\sigma^2))$. Thus, in Fig 4.1, we can see that when $E_b/N_0 = 1.7$ dB, 1.71 dB and 1.72 dB the means of the pdfs after 800 iterations are small, that is, there is a high probability that the message $V_{j\to i}^{(\ell)}$ is negative for any iteration; as a result decoding errors are likely. On the other hand, when $E_b/N_0 = 1.7306$ dB the mean for the pdf of the message $V_{j\to i}^{(\ell)}$ increases as the number of iterations increases. The evolution of the pdf for iterations $508 - 512$ is plotted. Note that at iteration 512 the Gaussian is completely on the positive side. Meaning that the error probability is negligibly small and goes to zero as $\ell \to \infty$

Using the same example, in Fig 4.2 we show the behaviour of the means $\mu^{(\ell)}$ as a function of the number of iterations ℓ. In this figure we can see that when the channel value σ is too large the mean $\mu^{(\ell)}$ converges to a fixed point that cannot guarantee free error decoding, i.e it does not satisfy (4.4). This happens for $E_b/N_0 = 1.7$ dB, 1.71 dB and 1.72 dB. On the other hand, for $E_b/N_0 = 1.7306$ dB, 1.756 dB and 1.77 dB,

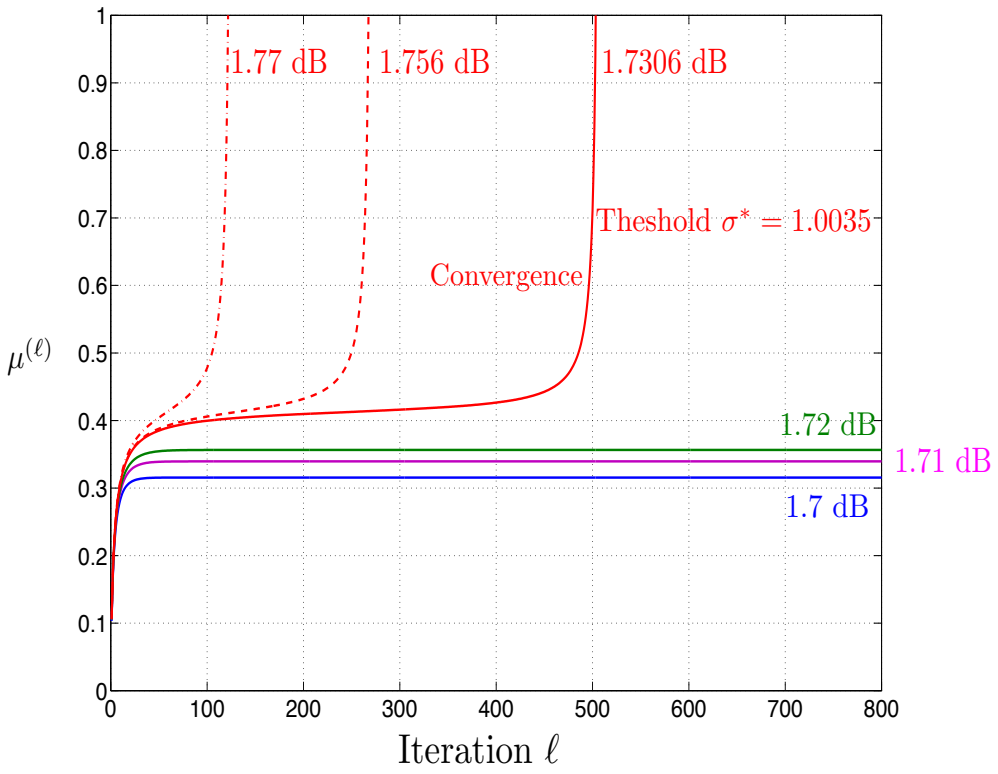


Figure 4.2: Behavior of the mean $\mu^{(\ell)}$ as a function of the number of iterations ℓ.

the mean $\mu^{(\ell)}$ goes to infinity as the number of iterations $\ell \to \infty$. If we continue decreasing the value of σ we will have convergence for all the cases, but we are interested in the higher value of σ that allows a successful decoding performance. For this specific example the decoding threshold is $\sigma^* = 1.0035$.

The density evolution algorithm via Gaussian approximation has a penalty in the resulting values of the thresholds of around 0.1 dB (in the BI-AWGNC case) due the assumption that the messages follows a Gaussian distribution and additionally these Gaussian are assumed to be consistent.

Note that the Gaussian approximation does not consider any kind of quantization in the messages. In such a case the convolution of discretized pdf will be performed. On the other hand, if the output of the channel is not symmetric the DEA has to be modified. In the following section, we will describe the proposed density evolution algorithm which can work under the above two conditions since it applies the max-LUT method to

compute the evolution of quantized pdfs.

4.3 Proposed discretized density evolution algorithm

The Shannon capacity defines the maximum rate at which we can transmit information reliably through a noisy channel. In a similar manner, for an LDPC code ensemble, it is possible to compute a noise threshold which acts as a capacity for a given LDPC code ensemble. Such threshold, represents an average value over the ensemble and not a value for an individual finite-length code with a certain number of girths in its Tanner graph representation nor considering any quantization scheme.

We were interested in the theoretical decoding thresholds of our decoding mapping functions for a given number of quantization levels. For this reason, we derive a density evolution algorithm with quantization and having the proposed decomposition in the on Subsection 3.6.2. The proposed DEA allows tracking quantized densities of the messages that can follow any symmetric and non-symmetric distribution.

Now, we describe an explicit method to compute the decoding mappings functions for a (d_v, d_c)-regular LDPC code. The construction of the mapping functions considers the decomposition presented in Section 3.6, where each node is decomposed into a set of 2-input mapping functions. The quantization algorithm presented in Subsection 3.7.2, finds an optimal quantizer Q^* for a DMC in the sense of maximization of mutual information. In this research, we not only use this algorithm to quantize the channel, we also use it to quantize the conditional probability distributions during the proposed density evolution algorithm. In essence, we will apply the max-LUT method during the proposed discretized density evolution algorithm.

Classical density evolution is restricted to channels with certain symmetry properties. But here, asymmetrical binary-input DMCs are allowed, and the optimized decoding lookup tables, and thus the distributions, may be asymmetrical even if the channel was symmetrical. Wang et al. generalized density evolution to asymmetric channels [36]. They showed that while error rates are codeword-dependent, it is sufficient to consider the evolution of densities only for the two code bits, that is densities conditioned on $X = 0$ and $X = 1$. The same method will be used here.

When density evolution is used, a channel parameter is needed [10]. As in the previous section, we will use α as the channel parameter. Therefore,

the decoding threshold is indicated as α^*.

4.3.1 Discretized density evolution algorithm with quantization

In this section, we describe the density evolution algorithm using the optimal quantizer presented in Subsection 3.7. An arbitrary, binary-input DMC is used for transmission. It has a binary input X and $|\mathcal{Z}|$ outputs $Z \in \mathcal{Z} = \{1, \ldots, |\mathcal{Z}|\}$. The channel transition probabilities are denoted by $r^{(0)}$:

$$r^{(0)}(x_0, y_0) = \Pr(Z = y_0 | X = x_0), \tag{4.6}$$

where $x_0 \in \{0, 1\}$ and $y_0 \in \{1, \ldots, |\mathcal{Z}|\}$. On iteration ℓ, the probability distribution for V is:

$$r^{(\ell)}(x, y) = \Pr(V = y | X = x) \tag{4.7}$$

with $y \in \mathcal{V}$, and the probability distribution for L is:

$$l^{(\ell)}(x, y) = \Pr(L = y | X = x) \tag{4.8}$$

with $y \in \mathcal{L}$.

The following method finds the probability distributions r and l employing the *quantization algorithm* described in Subsection 3.7. In particular, for each iteration and each node type, there are three steps:

(a) Given the node input distributions, a conditional distribution is found.

(b) The quantization algorithm produces a quantizer to K levels.

(c) The reduced distribution is found, which is used in the next step of the density evolution.

Two functions f_c and f_v are of interest when decoding LDPC codes. At the check node:

$$f_c(x_1, \ldots, x_{d_c - 1}) = x_1 + \cdots + x_{d_c - 1} \pmod 2 \tag{4.9}$$

and at the variable node:

$$f_v(x_0,\ldots,x_{d_v-1}) = \begin{cases} 0 & \text{if } x_0 = x_1 = \cdots = 0 \\ 1 & \text{if } x_0 = x_1 = \cdots = 1 \\ & \text{otherwise undefined} \end{cases}$$

where x_i are binary values. It is useful to use a single symbol that is a concatenation of the component messages in the joint distribution. In the context of the check node, let $\mathbf{y}'$ denote the concatenation.

$$\mathbf{y}' = (y_1, y_2, \ldots, y_{d_c-1}) \tag{4.10}$$

where $\mathbf{y}' \in \mathcal{V}^{d_c-1}$. And in the context of the variable node, let $\mathbf{y}'$ denote the concatenation:

$$\mathbf{y}' = (y_0, y_1, y_2, \ldots, y_{d_v-1}) \tag{4.11}$$

where $\mathbf{y}' \in \mathcal{Z} \times \mathcal{L}^{d_v-1}$.

Step (a) is to find the joint distributions $\tilde{l}^{(\ell)}(x, \mathbf{y}')$ and $\tilde{r}^{(\ell)}(x, \mathbf{y}')$, given by:

$$\tilde{l}^{(\ell)}(x, \mathbf{y}') = \left(\frac{1}{2}\right)^{d_c-2} \sum_{\mathbf{x}: f_c(\mathbf{x})=x} \prod_{i=1}^{d_c-1} r^{(\ell-1)}(x_i, y_i) \tag{4.12}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_{d_c-1})$, and

$$\tilde{r}^{(\ell)}(x, \mathbf{y}') = \sum_{\mathbf{x}: f_v(\mathbf{x})=x} r^{(0)}(x_0, y_0) \prod_{i=1}^{d_v-1} l^{(\ell-1)}(x_i, y_i) \tag{4.13}$$

where $\mathbf{x} = (x_0, x_1, \ldots, x_{d_v-1})$.

Step (b). The quantizers $Q_c^{(\ell)}$ and $Q_v^{(\ell)}$ are generated at each iteration ℓ using the joint distributions $\tilde{l}^{(\ell)}$ and $\tilde{r}^{(\ell)}$ and the integer K:

$$Q_c^{(\ell)} = \text{Quant}(\tilde{l}^{(\ell)}, K) \text{ and} \tag{4.14}$$

$$Q_v^{(\ell)} = \text{Quant}(\tilde{r}^{(\ell)}, K). \tag{4.15}$$

Step (c) is to find the reduced distributions by the following matrix multiplications:

$$l^{(\ell)} = \tilde{l}^{(\ell)}\ Q_c^{(\ell)} \text{ and} \tag{4.16}$$

$$r^{(\ell)} = \tilde{r}^{(\ell)}\ Q_v^{(\ell)}. \tag{4.17}$$

More precisely, density evolution is as follows:

1. Initialization: Initialize $\ell = 0$ and the channel message given by (4.6).

2. Check node: Compute (4.12), followed by (4.14), followed by (4.16).

3. Variable node: Compute (4.13), followed by (4.15), followed by (4.17).

4. If the mutual information *I(X; V)* approaches 1, then declare convergence and increase the channel parameter α and start again from initialization. If a fixed number of iterations is exceeded, declare non-convergence and the last channel parameter α where convergence was declared is the decoding threshold α^*. Otherwise, increment ℓ and iterate from the check node step.

Using the four steps described above, we find the decoding threshold α^* for a given regular (d_v, d_c)-LDPC code with $|\mathcal{Z}|$ levels for channel message quantization and K levels for decoder message quantization. Note that the decomposition of the nodes is not considered yet. In the following subsection, we use the threshold α^* computed by the above discretized DEA and the decomposition of the nodes presented in subsection 3.6.2 to construct the decoding mapping functions.

4.3.2 Discretized density evolution algorithm in a decomposed Tanner graph

Once we have the decoding threshold α^* for a given regular (d_v, d_c)-LDPC code, in this section we describe how to construct the decoding mapping functions $\psi_1^{(\ell)}, \ldots, \psi_{d_c-2}^{(\ell)}$, $\phi_1^{(\ell)}, \ldots, \phi_{d_v-1}^{(\ell)}$ and $\gamma_1^{(\ell)}, \ldots, \gamma_{d_v}^{(\ell)}$ for each iteration ℓ in the decomposed Tanner graph presented in 3.6.2.

1. **Initialization**.The channel parameter α^* is used to compute the channel transition probability $r^{(0)}$. In case that a channel parameter α^* is not available, pass directly the channel transition probability matrix $r^{(0)}$. Then, Initilize $\ell = 1$ and $t = r^{(0)}$.

2. **Check Node**. At iteration ℓ find successively each check node decoding mapping function $\psi_i^{(\ell)}$ for $i \in \{1, \ldots, d_c - 2\}$ as follows:

a) Compute the conditional distribution $\tilde{t}_i$ as (equivalent to **Step (a)** in Subsection 4.3.1):

$$\tilde{t}_i = \begin{bmatrix} 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 1/2 & 0 \end{bmatrix} (t_i \otimes r_i^{(\ell-1)}), \tag{4.18}$$

where $\otimes$ denotes the Kronecker product.

b) Generate the optimal quantizer $Q_i^{(\ell)}$ using the distribution $\tilde{t}_i$ and the size of one of the alphabets $\mathcal{S}$ or $\mathcal{L}$ as appropriate (equivalent to **Step (b)** in Subsection 4.3.1):

$$Q_i^{(\ell)} = \begin{cases} \text{Quant}(\tilde{t}_i, |\mathcal{L}|) & \text{if } i = d_c - 2 \\ \text{Quant}(\tilde{t}_i, |\mathcal{S}|) & \text{otherwise,} \end{cases} \tag{4.19}$$

recall that the alphabet $\mathcal{S}$ is for the interconnection of the decoding mapping functions $\psi_i^{(\ell)}$ while the alphabet $\mathcal{L}$ is to generate the check-to-variable node messages.

c) Reduce the distribution $\tilde{t}_i$ as (equivalent to **Step (c)** in Subsection 4.3.1):

$$t_i = \tilde{t}_i \; Q_i^{(\ell)}. \tag{4.20}$$

d) Fill in the two-input lookup table $\psi_i^{(\ell)}$ as:

$$\psi_i^{(\ell)}(\mathbf{y}') = y \qquad \text{if} \qquad Q_i^{(\ell)}(y, \mathbf{y}') = 1, \tag{4.21}$$

where $\mathbf{y}' \in \mathcal{V} \times \mathcal{V}$ or $\mathcal{V} \times \mathcal{S}$.

e) If $i = d_c - 2$ continue to **3) Variable Node**, otherwise construct the next $\psi_i^{(\ell)}$ decoding mapping function by going to step a).

3. **Variable Node**. First initilize $l_i^{(\ell)} = t_{d_c-2}$ and $t_0 = r^{(0)}$, then for the current iteration ℓ find successively each $\phi_i^{(\ell)}$ for $i \in \{1, \ldots, d_v - 1\}$:

a) Compute conditional distribution $\tilde{t}_i$ as (equivalent to **Step (a)** in Subsection 4.3.1):

$$\tilde{t}_i = \begin{bmatrix} t_i(x = 0, y) \otimes l_i^{(\ell)}(x = 0, y)) \\ t_i(x = 1, y) \otimes l_i^{(\ell)}(x = 1, y)). \end{bmatrix} \tag{4.22}$$

b) Compute the optimal quantizer $Q_i^{(\ell)}$ using the distribution $\tilde{t}_i$ and the size of one of the alphabets $\mathcal{T}$ or $\mathcal{V}$ as appropriate (equivalent to **Step (b)** in Subsection 4.3.1):

$$Q_i^{(\ell)} = \begin{cases} \text{Quant}(\tilde{t}_i, |\mathcal{V}|) & \text{if } i = d_v - 1 \\ \text{Quant}(\tilde{t}_i, |\mathcal{T}|) & \text{if } i < d_v - 1, \end{cases} \tag{4.23}$$

recall that the alphabet $\mathcal{T}$ is for the interconnection of the decoding mapping functions $\phi_i^{(\ell)}$ while the alphabet $\mathcal{V}$ is to generate the variable-to-check node messages.

c) Reduce the distribution $\tilde{t}_i$ as (equivalent to **Step (c)** in Subsection 4.3.1):

$$t_i = \tilde{t}_i \; Q_i^{(\ell)}. \tag{4.24}$$

d) Fill in the two-input lookup table $\phi_i^{(\ell)}$ by:

$$\phi_i^{(\ell)}(\mathbf{y}') = y \qquad \text{if} \qquad Q_i^{(\ell)}(y, \mathbf{y}') = 1. \tag{4.25}$$

If $i = d_v - 1$ continue, otherwise go to a).

e) Using the distribution $\tilde{t}_i$ and the size of the alphabet $\mathcal{T}$ construct the last two mapping functions $\gamma_{d_v-1}^{(\ell)}$ and $\gamma_{d_v}^{(\ell)}$ for the hard decision operation as:

$$Q_i^{(\ell)} = \text{Quant}(\tilde{t}_i, |\mathcal{T}|), \tag{4.26}$$

$$s = \tilde{t}_i \; Q_i^{(\ell)}, \tag{4.27}$$

$$\gamma_{d_v-1}^{(\ell)}(\mathbf{y}') = y \qquad \text{if} \qquad Q_i^{(\ell)}(y, \mathbf{y}') = 1, \tag{4.28}$$

$$\tilde{s} = \begin{bmatrix} s(x=0,y) \otimes l^{(\ell)}(x=0,y)) \\ s(x=1,y) \otimes l^{(\ell)}(x=1,y)) \end{bmatrix}, \tag{4.29}$$

$$Q_i^{(\ell)} = \text{Quant}(\tilde{s}, 2), \tag{4.30}$$

$$\gamma_{d_v}^{(\ell)}(\mathbf{y}') = y \qquad \text{if} \qquad Q_i^{(\ell)}(y, \mathbf{y}') = 1. \tag{4.31}$$

Note that the decoding mapping functions for the variable node operation $\phi_1^{(\ell)}, \ldots, \psi_{d_v-2}^{(\ell)}$ are the same as the mapping functions $\gamma_1^{(\ell)}, \ldots, \gamma_{d_v-2}^{(\ell)}$ (we remove the superscript ℓ from t and $\tilde{t}$ to avoid overloading notation,

also the subscript and superscript were removed from s and $\tilde{s}$). At this point, we use $\tilde{t}_i$ to find the mutual information for iteration ℓ and verify the convergence. If the mutual information *I(X; V)* approaches 1, then declare convergence and all the mapping functions constructed for the channel parameter α^* will be used to decode the LDPC code with degrees (d_v, d_c) . Otherwise, increment ℓ, update $r_1^{(\ell)-1} = t_{d_v-1}$ and go to a) in **2) Check Node**. Since through this process we use the decoding threshold α^* to initialize $r^{(0)}$ at some point we will find convergence. Therefore, the final number of mapping functions depends of the decoding threshold α^* computed for a given pair of degrees (d_v, d_c), and the size of the alphabets $\mathcal{V}, \mathcal{S}, \mathcal{L}$ and $\mathcal{T}$.

An appealing aspect of the proposed technique is that the LUTs are generated off-line. The offline complexity, while not as important as the size of the lookup tables, is no worse than K^6, with K representing the size of the alphabet of one incoming message. Since each LUT has two incoming messages the resulting input to the quantization algorithm is size K^2, and the quantization algorithm has no worse than cubic complexity.

4.4 Decoding thresholds for BI-AWGNC

Using the method of the previous section, we present the decoding thresholds considering a regular $(d_v = 3, d_c = 6)$-LDPC code, under different channel and decoder message quantization values.

In this research the message-passing decoding algorithms use two quantization resolutions (bits per message), one for the channel message quantization when a continuous channel such as the BI-AWGNC is considered, and the other for the decoder message quantization.

In the proposed quantized density evolution algorithm, we are able to specify different numbers of quantization levels for different parts of the process, this is, we can define a certain number of quantization levels $|\mathcal{Z}|$ for the quantization of the channel messages and define another number of quantization levels for the decoder messages (i.e., $|\mathcal{L}|, |\mathcal{T}|, |\mathcal{V}|$ and $|\mathcal{S}|$. Note that the quantization levels used for the decoder message can be different among them or can be the same. In our research, we set all quantization levels to either 3 or 4 bits per message, that is, the cardinality of all alphabets is 8 or 16 respectively. However, different variations can be used, see Appendix A.

Consider the following example where the quantization levels for the channel message is $|\mathcal{Z}| = 16$, which differs from those used for decoder message as $|\mathcal{L}|, |\mathcal{T}|, |\mathcal{V}|$ and $|\mathcal{S}| = 8$. For the above example the channel

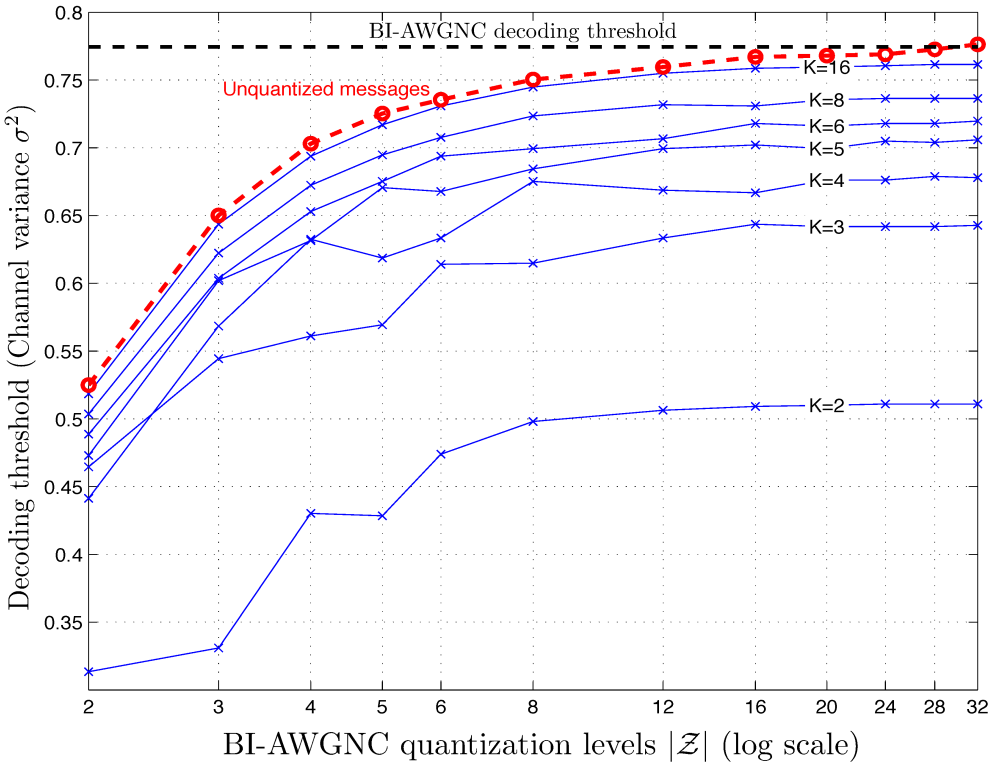


Figure 4.3: Noise decoding thresholds for a regular $(3, 6)$-LDPC code with rate $1/2$ and using different number of levels K for the decoder message quantization. The term $\log_2(K)$ is the number of bits to represent the decoder messages while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the BI-AWGNC message.

message is represented using 4 bits, while the decoder messages are represented with 3 bits, in this way the resolution in bits for the channel is 4 and 3 for the decoder. Note that a quantization resolution which is not a power of two can be used, if required.

Note that the construction of the proposed decoding mapping functions presented in Subsection 4.3.2 can be adapted by adjusting the quantization levels (size of the alphabets).

The discretized density evolution analysis proposed in Subsection 4.3.1 is useful because we can know the error correction capability of a (d_v, d_c)-LDPC code whose channel and decoder messages are quantized to K levels without resorting to finite-length simulations.

Resulting thresholds under the proposed density evolution algorithm, considering a regular $(d_v = 3, d_c = 6)$-LDPC code and different quantiza-

Table 4.1: Noise decoding thresholds for a regular $(d_v = 3, d_c = 6)$-LDPC with rate $R = 1/2$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.3078	0.4637	0.4414	0.4711	0.4859	0.4934	0.5008	0.5156	0.5156
3	0.3301	0.5379	0.5676	0.5973	0.5973	0.6195	0.6195	0.6418	0.6418
4	0.4266	0.5602	0.6270	0.6270	0.6492	0.6641	0.6715	0.6938	0.6938
5	0.4266	0.5676	0.6121	0.6641	0.6715	0.6863	0.6938	0.7160	0.7160
6	0.4711	0.6121	0.6270	0.6641	0.6938	0.6938	0.7012	0.7309	0.7309
8	0.4934	0.6121	0.6715	0.6789	0.6938	0.7160	0.7234	0.7383	0.7457
12	0.5008	0.6270	0.6641	0.6938	0.7012	0.7234	0.7309	0.7531	0.7605
16	0.5082	0.6418	0.6641	0.7012	0.7160	0.7234	0.7309	0.7531	0.7605
20	0.5082	0.6418	0.6715	0.6938	0.7160	0.7234	0.7309	0.7531	0.7605
24	0.5082	0.6418	0.6715	0.7012	0.7160	0.7309	0.7309	0.7605	0.7605
28	0.5082	0.6418	0.6789	0.7012	0.7160	0.7234	0.7309	0.7605	0.7605
32	0.5082	0.6418	0.6715	0.7012	0.7160	0.7309	0.7309	0.7605	0.7605

tion values for the channel and decoder messages, shown in Fig. 4.3.

The quantization levels for the BI-AWGNC were found by finely and uniformly quantizing the BI-AWGNC, and then applying the quantization algorithm presented in Subsection 3.7.2. Note that in Fig. 4.3, K is equal to the cardinality of the alphabets of the decoding messages, that is:

$$|\mathcal{L}| = |\mathcal{V}| = |\mathcal{T}| = |\mathcal{S}| = K.$$

Figure 4.3 along its vertical axis shows the variance σ^2 versus the channel quantization levels $|\mathcal{Z}|$ in its horizontal axis. For this example in particular, we plotted different values of $|\mathcal{Z}|$;

$$|\mathcal{Z}| \in \{2, 3, 4, 5, 6, 8, 12, 16, 20, 24, 28, 32\},$$

while decoding messages were considered to be quantized as

$$K \in \{2, 3, 4, 5, 6, 8, 16\}.$$

In Figure 4.3, each valid combination of K and $|\mathcal{Z}|$ gives a decoding threshold for the BI-AWGNC quantized to $|\mathcal{Z}|$ levels and decoder messages quantized to K levels.

One of the most significant observations is the point reached when $|\mathcal{Z}| = 16$ and $K = 16$ (4-bits per message for both channel and decoder

messages), corresponding to $\sigma^2 = 0.7531$ in the vertical axis of the graph. This threshold is in fact difficult to distinguish from that achieved when unquantized decoder messages are considered (see $-\circ-$ line on the graph for unquantized decoding messages).

Numerically the noise thresholds are listed in Table 4.1. In this table, the number of quantization levels $|\mathcal{Z}|$ for the channel message is indicated in the rows on the table, while the quantization levels K for the decoder message are listed in an arranged manner in the columns. Any pair $|\mathcal{Z}|$, K defines the theoretical decoding threshold for a $(d_v = 3, d_c = 6)$-LDPC code.

Looking at the Table 4.1, the combination $|\mathcal{Z}| = 2$, $K = 16$ corresponds to 1 bit per message for the channel message and 4 bits per message for the decoder message, the value in this intersection is $\sigma^2 = 0.5156$. Considering $|\mathcal{Z}| = 2$ is equivalent to have a binary symmetric channel (BSC), therefore the resulting lookup tables for $|\mathcal{Z}| = 2$ may be used to decode assuming a BSC with crossover probability ε. To know the equivalent decoding threshold ε^* for a specific σ^2, we can apply the Q-function in (2.21).

Another interesting example is the combination $|\mathcal{Z}| = 16$, $K = 8$, which gives $\sigma^2 = 0.7309$. This case corresponds to 4 bit per message for the channel message and 3 bits per message for the decoder message. Note that, if we increment the channel resolution in one bit ($|\mathcal{Z}| = 32$), there is no improvement in the theoretical decoding threshold. On the other hand, if we increment the resolution of the decoder message in one bit, the decoding threshold increases to $\sigma^2 = 0.7605$.

Another important observation in Fig. 4.3 is the decoding threshold for the combination $|\mathcal{Z}| = 5$, $K = 4$. The corresponding threshold in this intersection is worst than that achieved when $|\mathcal{Z}| = 4$, $K = 4$. The conjecture that we can give is that when an odd number is chosen for the channel quantization, the symmetry of the regions respect to the origin is lost, that is, when $|\mathcal{Z}| = 5$ the third quantization level starts before zero and finishes after zero. This causes that at the time to perform the hard decision this quantization region has to be zero or one. Thus the asymmetry of the quantization regions affects one bit or another. This implies that the hard decisions are not balanced.

In Fig. 4.4 we can see the decoding thresholds for a regular $(4, 6)$-LDPC code for different combination of quantization values $|\mathcal{Z}|$, and K. In this figure we can also appreciate a not smooth performance in the decoding threshold curves. The combinations that present more drastic changes are $|\mathcal{Z}| = 5$, $K = 3$ and $|\mathcal{Z}| = 4$, $K = 2$. The first combination apparently

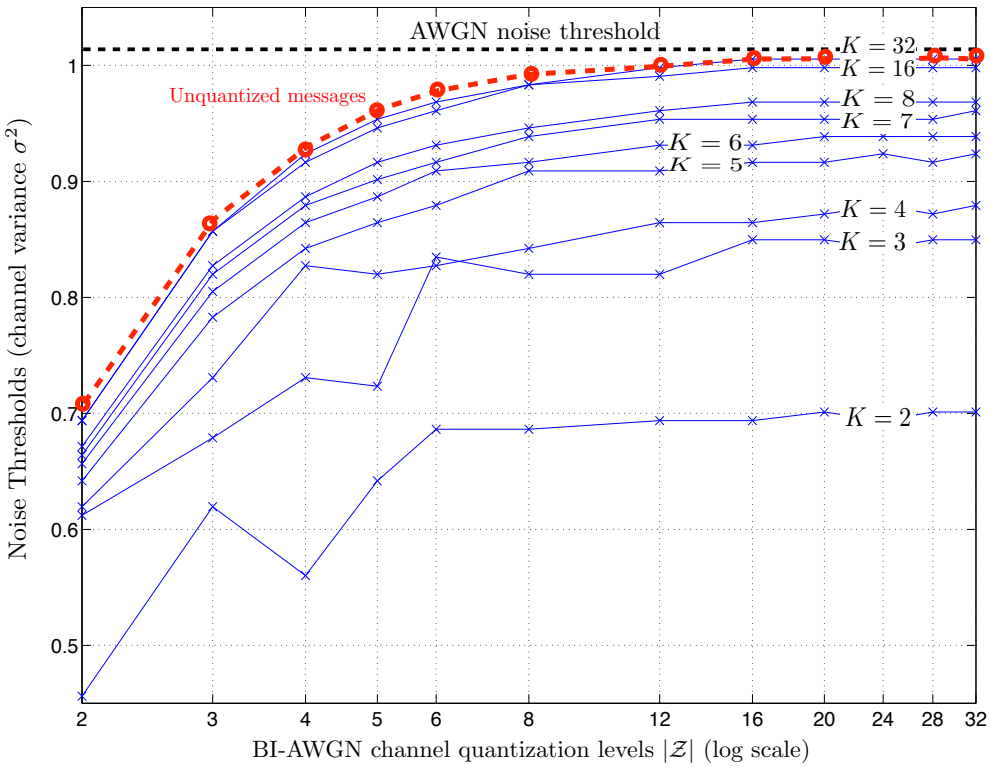


Figure 4.4: Noise decoding thresholds for a $(4, 6)$ regular LDPC code with rate $1/3$ and using different number of levels K for the decoder message quantization. The term $\log_2(K)$ is the number of bits to represent the decoder messages while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the BI-AWGNC message.

present a decrement in the decoding threshold due to the same reasoning that we explained for the case of the Figure 4.3. The second combination $|\mathcal{Z}| = 4$, $K = 2$ is a new case where a decrement on the decoding threshold σ^2 is shown when an increment is expected. For this case an apparent reason due to the symmetries of the quantization is not obvious. One hypothesis could be that this performance is due to the code ensemble itself. An analysis of the possible reasons of these unexpected decrements could lead to an improvement of decoding thresholds. Therefore, this is a tangible extension of the research presented in this work.

On the other hand, in Fig. 4.4 the SPA approaching performance of the proposed decoding mapping functions using 4 bits per channel and decoder message can also be observed for this code. In this figure we also plotted the resulting decoding thresholds when 32 quantization levels are

used to represent the decoder messages. From this, we can appreciated that the gain in the decoding threshold by increasing from 4 bits to 5 bits is not significant.

4.5 Lookup table arrangement and its representation in a tree

In section 3.5 on page 36, we briefly mentioned that the decomposition of the variable and check nodes in smaller degree-3 nodes leads to an increment on the decoding time which can be overcome or at least mitigated by using a semi parallel or pipeline architecture for the implementation of the message passing decoding.

In this work, we have only focused our attention in a pipeline architecture which in fact is used for all the finite-length results presented in Chapter 5.

Considering the hard decision operation in a variable node with degree five, i.e. $d_v = 5$, which is decomposed into a set of degree-3 nodes, the hard decision operation needs to use five two-input lookup tables ϕ to perform the hard decision estimation. One of the possible lookup table arrangement to calculate the estimation may be the following

$$\phi\Bigg(\phi\Bigg(\phi\big(\phi(L,L),\phi(L,L)\big),L\Bigg),Z\Bigg)\in\{0,1\}. \tag{4.32}$$

Remember that the channel message Z also has to be considered in this estimation. The flow of the above operation can be seen more clearly using a tree representation, see the tree χ_4 on the lower right part on Fig 4.5 which corresponds to the lookup table processing flow in (4.32). In the same figure another arrangements to perform the same operation are depicted.

We are interested in knowing which tree that correspond to a specific processing flow of the hard decision is better. In this matter, a reasonably heuristic method has been previously proposed in [37]. Such metric does not distinguish between the check-to-variable incoming messages L and the channel message Z.

Since the data processing inequality states that processing only reduces mutual information, then, the cumulative depth $\lambda(\chi)$ of a tree χ which is the sum of all distances of all leaf nodes to the root node has to be the

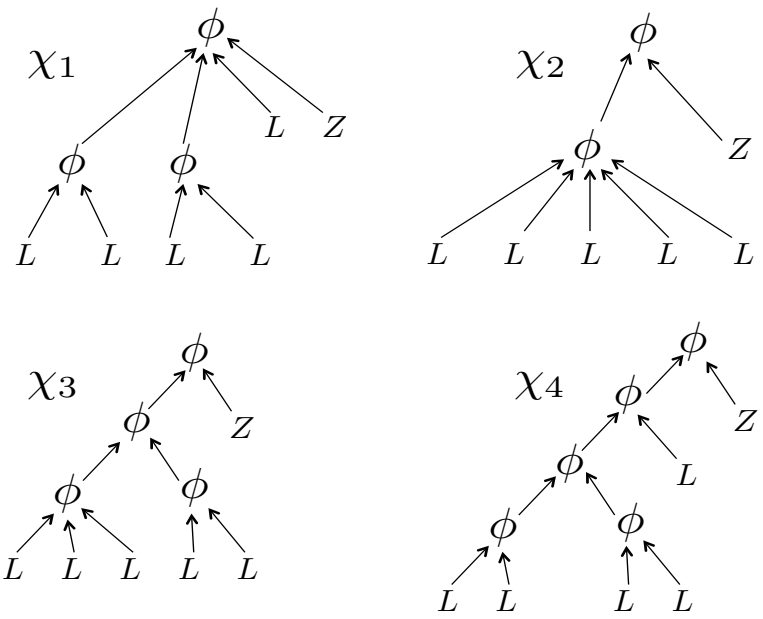


Figure 4.5: Tree representation of the implementation of a hard decision operation using lookup table. The node has six inputs including the channel message.

smallest. Looking at Fig 4.5, a distance between a leaf node and the root node is equal to the number of arrows that exist between them.

Using the tree examples in Fig 4.5, the authors of [37] present the table 4.2. In this table, we can appreciate that effectively there is a relationship between the summatory of the paths $\lambda(\chi)$ and the decoding threshold σ^* computed via density evolution. Although, a variation exist in the decoding threshold σ^* among all four arrangements, that is not significant to drastically chose one arrangement over the other. On the other hand, this result is interesting for hardware implementation point of view since

Tree	χ_1	χ_2	χ_3	χ_4
λ	10	11	16	19
σ^*	0.5330	0.5327	0.5309	0.5305

Table 4.2: Comparison for different arrangements (trees) to implement a hard decision operation with six inputs including the channel message. The decoding thresholds σ^* were computed considering that the incoming messages have a resolution of 3 bits.

complete binary trees (i.e. a binary tree in which all interior nodes have two children and all leaves have the same depth or same level) are desirable due to they have short critical paths producing low complexity lookup table implementations.

4.6 Summary

In this chapter, we first described the idea behind the density evolution algorithm using the Gaussian approximation. Later we introduced the proposed discretized density evolution algorithm. Such algorithm finds a decoding threshold assuming a regular (d_v, d_c)-LDPC code with $|\mathcal{Z}|$ quantization levels for the channel message and K quantization levels for the decoder message.

Secondly, we described a decomposed density evolution algorithm which performs the same operation as that above, but with the difference that, this constructs the decoding mapping functions during the iterative decoding process. A decomposed Tanner graph is considered.

In this chapter, we also presented some numerical decoding thresholds for a pair of regular (d_v, d_c)-LDPC codes using different combinations of quantization values for the channel and decoder message quantization. At the end of the chapter, a brief discussion about the implementations of the decoding lookup tables was introduced.

Chapter 5

Finite-length LDPC decoding via mapping functions

An infinite code length and a cycle-free graph are central assumptions made by density evolution algorithm. Therefore, in this chapter, finite-length simulations are presented to corroborate theoretical results obtained via density evolution, which situate the proposed decoding mapping functions theoretically close to the unquantized sum-product algorithm decoding performance. In this chapter, our decoding mapping functions use 3 and 4 bits per message for both channel and decoder message quantization. Three aspects are considered to carry out the performance analysis of the proposed decoding mapping functions: bit-error rate (BER), word-error rate (WER) and average number of executed iterations. The legends for the proposed decoding maps appearing on the figures in this section are referred as *max-LUT*, since a natural way to implement the decoding mapping functions can be using lookup tables (LUT) which maximize (*max*) mutual information during the iterative decoding process.

Since intensive computer-based simulations for different codes were perform, we divided the finite-length results in this section into three subsections: simulation results for low rate codes, simulation results for medium rate codes and simulation results for high rate codes. At the end of this chapter, a summary about the most important observations from the finite-length simulations is presented.

5.1 Finite-length results for LDPC codes

For finite-length simulations of LDPC codes under message-passing decoding, the bit-error rate (or word-error rate) performance curves, can be divided into two regions: the *waterfall region* and the *error-floor* region. In the waterfall region the error-rate performance curve goes down rapidly and smoothly, while in the error-floor region, it is less hasty and it often happens in the high bit signal-to-noise ratio (E_b/N_0) region [22].

The waterfall region is caused due to a large number of unsuccessful corrections and the slope of the error probability lowers exponentially in the blocklength of the code. On the other hand, the error-floor region whose decay is polynomial [38] represents a small number of errors generated mainly by near-codewords [39], trapping sets [40], or absorbing sets [41].

From this overview on finite-length error-rate perfromance curves, we can say that, the waterfall region is dominated by the implemented message-passing decoding algorithm, while the error-floor mostly depends on the code construction, although we will see that the proposed decoding mapping functions exhibits good performance in both regions.

In a simple manner, the proposed decoding mapping functions, can be seen as the optimal local recorded partitions (i.e, LUTs) of the tracked probability distributions during the discretized density evolution process presented in Section 4.3 on page 57.

In this section, finite-length results for our decoding mapping functions using 3 and 4 bits per message are presented. The channel and decoder message quantization use the same number of bits per message. In all cases, we use the sum-product algorithm for comparison without both channel and decoder message quantization.

Since we are interested in channel and decoder message quantization, we mainly used the BI-AWGNC with variance value σ^2 as a channel model. Results presented in Fig. 5.5 on page 79, are a special case because of a request for [42]. In table 5.1, we list a total of 12 different codes used during the finite-length computer-based simulations. In this table, the simulation parameters such as degrees of the nodes (d_v, d_c), code rate R, code block length N, maximum number of iterations abbreviated as "Max. iter.", and type of code construction are listed.

In all BER/WER graphs, proposed decoding mapping functions are labeled as "3 or 4 bit/msg max-LUT", the digit indicates the number of bits for the channel and decoder message. Next to each BER/WER curve, the average number of iterations for each corresponding simulation point

is indicated.

We proceed through the presentation of the finite-length simulation results grouping them by the rate of the code. Under this consideration, we define three subsections: simulation results for low rate codes, medium rate codes, and high rate codes. This division was considered to carry out the analysis of the results in a more structured and efficient way, e.g. high rate codes requires less number of iterations than low rate codes. Codes with medium rates commonly give more insights about error floors.

Table 5.1: Simulation parameters for the proposed decoding mapping functions.

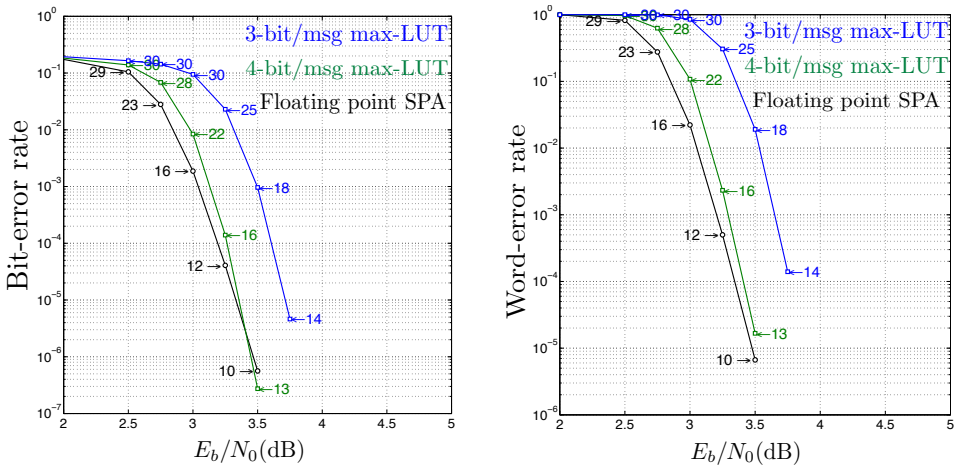
Figure	d_v	d_c	R	N	Max. iter.	Code
Fig. 5.1 on page 74	4	5	0.2	6535	30	Array code [43]
Fig. 5.2 on page 75	4	6	0.33	816	30	Gallager [44]
Fig. 5.3 on page 77	3	6	0.5	2640	25	Gallager [44]
Fig. 5.4 on page 78	4	8	0.5	10456	25	Array code [43]
Fig. 5.5 on page 79	3	12	0.75	2388	60	Gallager [44]
Fig. 5.6 on page 81	6	32	0.84	2048	30	Quasi-cyclic code [45]
Fig. 5.7 on page 82	4	36	0.89	1998	30	Gallager [44]
Fig. 5.8 on page 83	4	69	0.94	8970	20	Array code [46]

5.1.1 Simulation results for low rate codes

In this Subsection, we carry out the finite-length simulation analysis (i.e., BER and WER) for our decoding mapping functions using 3 and 4 bits per message using the BI-AWGNC and cosidering low rate codes (lower than 0.35). To do so, we consider two different codes with the following rates: $R = 0.2$ and $R = 0.33$. The noise decoding thresholds σ^2 for different combinations of channel and decoder message quantization over a BI-AWGNC for the chosen codes are deployed in tables A.4 and A.5 on page 91. The noise decoding thresholds used to construct the mapping functions for the $(d_v = 4, d_c = 5)$-LDPC code with $R = 0.2$ and $(d_v = 4, d_c = 6)$-LDPC code with $R = 0.33$ are shown in table 5.2.

Table 5.2: Noise decoding thresholds for channel and decoder message quantization using 3 and 4 bits per message.

Code	σ^2 (3 bits)	σ^2 (4 bits)
$(d_v = 4, d_c = 5)$-LDPC code	1.2875	1.3617
$(d_v = 4, d_c = 6)$-LDPC code	0.9461	0.9980



(a) Bit-error rate.

(b) Word-error rate.

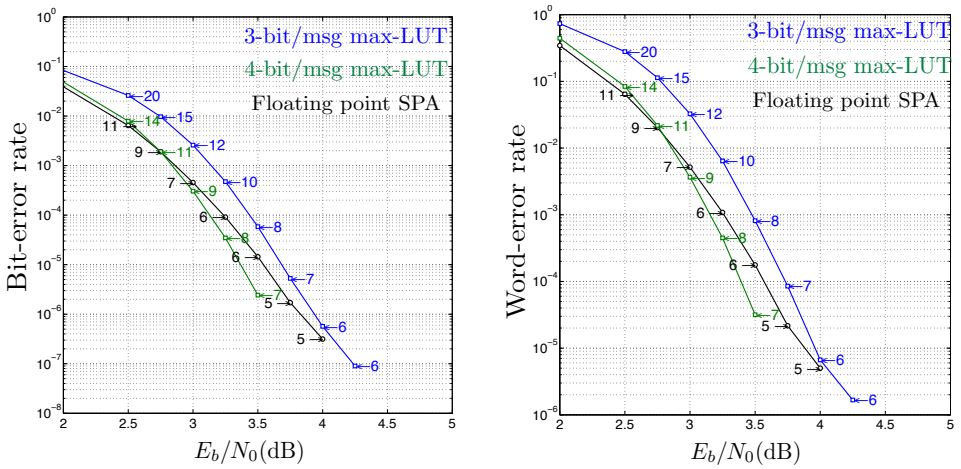
Figure 5.1: BER and WER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 4$, $d_c = 5$, $R = 0.2$, and $N = 6535$. The maximum number of Iterations was set to 25. The numbers next to the curves represent the average number of iterations for each simulation point.

At the beginning of the design of the proposed decoding mapping functions for low rate codes, our main concern was the average number of iterations for successful decoding. The reason of this concern, was a considerable amount of iterations computed by the discretized density evolution algorithm presented in Section 4.3, meaning that, the set of decoding mapping functions is large, which could lead to a latency problem during the decoding process.

Fortunately, the average number of iterations achieved by the chosen low rate codes is around 14 iterations for the lowest BER point achieved. This result was unexpected but at the same time opened the application of the proposed decoding mapping functions to decode efficiently low rate codes, i.e. perform siccessuful decoding using a few number of iterations.

In Fig 5.1, BER and WER results are presented for a $(d_v = 4, d_c = 5)$-LDPC code with rate $R = 0.2$. The length of such a code is $N = 6535$ and the maximum number of iterations was set to 30 iterations.

In Fig 5.1a, at 10^{-5} the proposed mapping functions using 3 bits per message achieve a small gap of around 0.35 dB, comparing with the curve drawn by SPA. In the same graph, but using 4 bits per message the gap practically disappear and following the curve down, we start seeing a small crossing over the bit-error rate curve of SPA that seems that is going to be



(a) Bit-error rate.

(b) Word-error rate.

Figure 5.2: BER and WER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 4$, $d_c = 6$, $R = 0.33$, and $N = 816$. The maximum number of Iterations was set to 25. The numbers next to the curves represent the average number of iterations for each simulation point.

surpassed in a lower error floor region. This performance is of substantial interest since our decoding mapping functions can in fact achieve the BER curve of a full sum-product algorithm.

In Fig 5.1b, the corresponding WER results for the same code used in Fig. 5.1a. Even though a tiny gap of less than 0.1 dB exists at 10^{-4} using 4 bits per message, the decoding results of our maps still are an interesting result for discrete LDPC decoders whose performance penalty is around 1 dB in the better cases using 6 or more bits per message. On the other hand, thinking about a good tradeoff between complexity and performance the proposed mappings using 3 bits per message fits really well into this target.

The second low rate code is a $(d_v = 4, d_c = 6)$-LDPC code with a small length of $N = 816$ and rate $R = 0.33$. The maximum number of iterations for this code is 30. The previous code had a length of a few thousands, here his counterpart is presented.

In Fig.5.2a, at 10^{-5} the proposed maps using 4 bits per message have a gain over SPA, while using 3 bits per message a small gap of less than 0.1 dB is shown. Although long codes mainly dominate most of the error-correcting code applications, we wonder about the performance of our decoding mapping functions in different scenarios. Even though the length

of the code is small, there is no error-rate penalty associated to this issue. Similar performance to that described above is achieved for WER results shown in Fig.5.2b.

Another observation from simulation results shown in Fig. 5.1 and Fig.5.2, is that the average number of iterations seen by the proposed decoding mapping functions is close to that obtained by the SPA. In the case of $R = 0.2$ code, at the lowest BER point the average number of iterations achieved by SPA is 10, while our maps using 3 bits per message achieved 14 and using 4 bits per messages achieved 13. In the second simulation where the code rate is $R = 0.33$, at the lowest BER point the average number of iterations achieved by SPA is 5, while our maps using 3 bits per message achieved 6 and using 4 bits per messages achieved 7. This is significant as well as the error-rate performance since it dominates the latency in an iterative decoding algorithm.

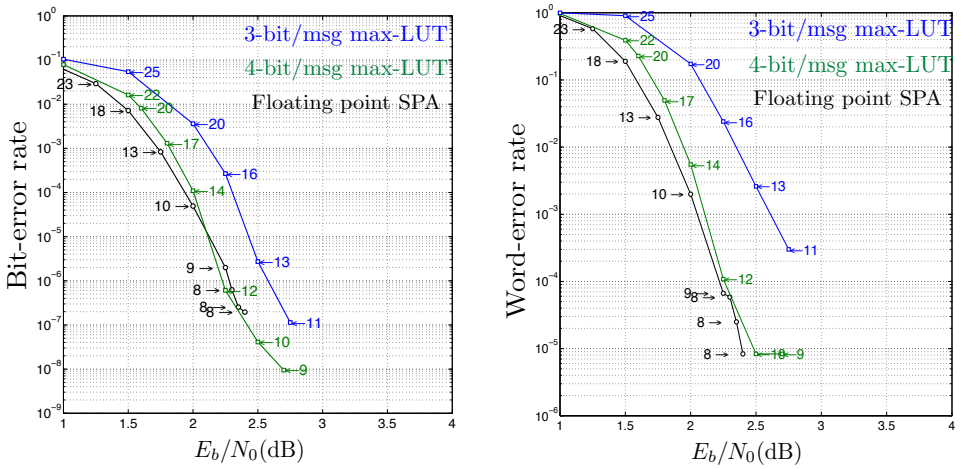
Table 5.3: Noise decoding thresholds for channel and decoder message quantization using 3 and 4 bits per message. In the case of the $(d_v = 3, d_c = 12)$-LDPC code, its variance noise thresholds σ^2 were used to calculate the corresponding crossover probabilities ε for the BSC via the Q-function.

Code	σ^2 (3 bits)	σ^2 (4 bits)
$(d_v = 3, d_c = 6)$-LDPC code	0.7531	0.7234
$(d_v = 4, d_c = 8)$-LDPC code	0.6863	0.6566
$(d_v = 3, d_c = 12)$-LDPC code	0.2680	0.2625
	$\varepsilon = 0.0267$	$\varepsilon = 0.0255$

5.1.2 Simulation results for medium code rates

Throughout this Subsection, we analyze the performance of three regular LDPC codes with rates $R = 0.5$ and $R = 0.75$. In the case of $R = 0.75$ code, the channel model is the BSC. Therefore, we use the Q-function presented in (2.21) to compute the noise decoding threshold ε for the BSC from the σ^2 threshold obtained by setting the channel quantization of a BI-AWGNC to 2 levels. The set of decoding mapping functions for each one of the three codes used in this Section were constructed using the variance values σ^2 presented in table 5.3.

Noise decoding thresholds for different combinations of channel and decoder message quantization for the $R = 0.5$ code are included in table A.3 on page 90.



(a) Bit-error rate.

(b) Word-error rate.

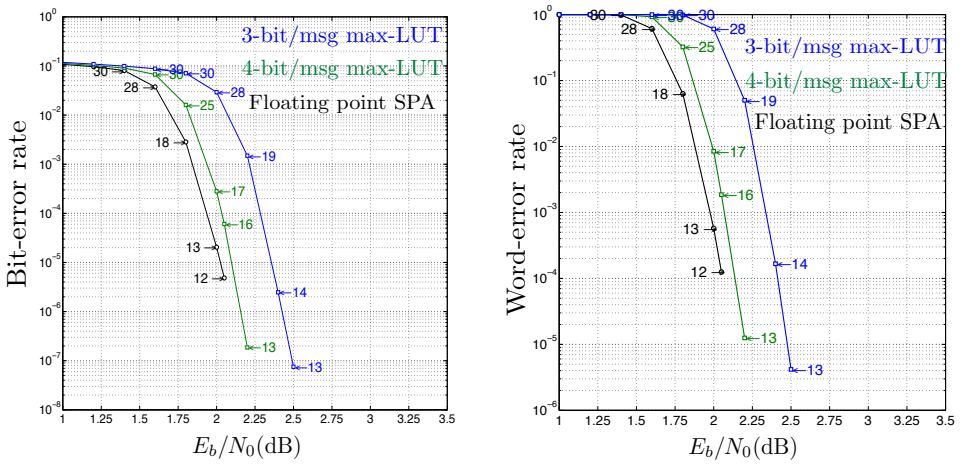
Figure 5.3: BER and WER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 3$, $d_c = 6$, $R = 0.5$, and $N = 2640$. The maximum number of Iterations was set to 25. The numbers next to the curves represent the average number of iterations for each simulation point.

In Fig. 5.3, BER and WER simulation results over a BI-AWGNC for a $(d_v = 3, d_c = 6)$-LDPC code are shown.

In Fig. 5.3a, at 10^{-6} we can observe that, the proposed decoding mapping functions using 3 bits per message perform around 0.35 dB away from full sum-product algorithm performance. While the average number of iterations for SPA is 8, the proposed mapping functions achieved 11. Even though we set the maximum number of iterations to 25, this result is significant since other quantized decoding algorithms require a range between 50 to 200 iterations to approach close to SPA performance, e. g. 50 iterations in [24], and 200 iterations in [22] and [47]. Even better, looking at the BER curve for the proposed decoding maps using 4 bits per message, we appreciate that at 10^{-6} the performance of these maps is a bit better than that achieved by the floating point SPA. In this case, our maps used 12 iterations in average.

In Fig. 5.3b, WER for the same simulation are shown. Using 3 bits per message, a gap around 0.6 dB at 10^{-3} can be seen. Down at 10^{-5} the proposed decoding maps using 4 bits per message follow closely the WER performance of SPA.

Continuing with 1/2 rate codes, in Fig. 5.4, we plot the BER and WER performance for a $(d_v = 4, d_c = 8)$-LDPC code with length $N = 10456$.



(a) Bit-error rate.

(b) Word-error rate.

Figure 5.4: BER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 4$, $d_c = 8$, $R = 0.5$, and $N = 10456$. The maximum number of Iterations was set to 30. The numbers next to the curves represent the average number of iterations for each simulation point.

Corresponding noise decoding thresholds for the above code using 3 and 4 bits per message are shown in table 5.3. The set of thresholds for different combinations of channel and decoder message quantization for a $(d_v = 4, d_c = 8)$-LDPC code are presented in table A.6 on page 91.

A well known result in coding and information theory is that, error-rate performance improve as N goes larger. Both graphs in Fig. 5.4 corroborate this fact. Even though the code rate in this simulation is the same as the previous code in Fig. 5.3, the code in Fig. 5.4 shows a stronger error correction capability by an increment of around 5 times in the block length. Another advantage of the code in Fig. 5.4 over the code in Fig. 5.3, is that the code in Fig. 5.4 is an array code [43], therefore, it is free of 4-girth cycles by construction.

The decoding benefit of non existing 4-girth cycles in this code, can be seen in both Fig. 5.4a and 5.4b. Numerically, in Fig. 5.4a at 10^{-5} our decoding mappings using 3 bits per message have a gap around 0.3 dB respect to SPA performance. In the same plot, a small gap barely of 0.07 dB for 4 bits case is hardly appreciated. Note that at the lowest error-rate probability achieved by the proposed decoders, the average number of iterations is only one over that achieved by SPA. Similar gap rations in Fig. 5.4b are kept by both sets of decoding mapping functions.

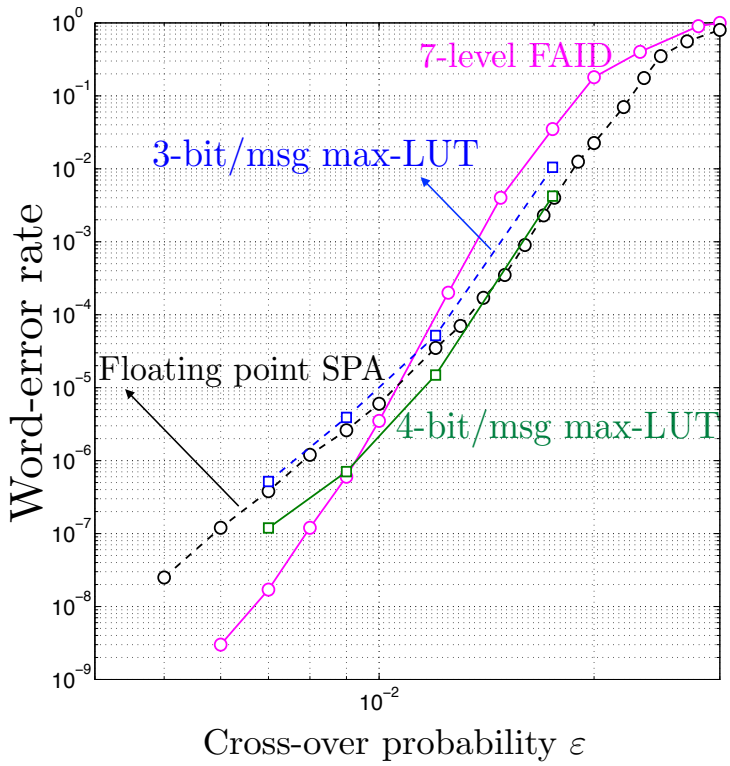


Figure 5.5: Word-error rate results for SPA using floating point numbers, FAIDs using 7 levels of quantization and the decoding mappings (*max-LUT*) using 3 and 4 bits per message. A regular $(d_v = 3, d_c = 12)$-LDPC code was used with $R = 0.75$, block length $N = 2388$ and a maximum of 60 iterations.

The last code to be analyzed in this Subsection is a $(d_v = 3, d_c = 12)$-LDPC code with rate $R = 0.751$ and block length $N = 2388$. The maximum number of iterations for our proposed decoding mapping functions is 60, while the results taken from [23] for the SPA and the 7-level finite-alphabet iterative decoder (labeled as 7-level FAID in the graph) is 100.

Looking at the error-floor region in Fig. 5.5, the proposed decoding mapping functions using 3 bits per message approach closely the SPA decoding performance, while using 4 bits per message they can surpass it though the maximum number of iterations is 40 less than the maximum of 100 for SPA. Similar to FAIDs, the proposed decoding maps have better error-rate performance than SPA in the error-floor region.

In Fig. 5.5, FAIDs show better performance in the error-floor region than the proposed decoding mapping functions since FAIDs are designed

to avoid the effects of harmful subgraphs, lowering the error floor. In the proposed technique, the mapping functions are not explicitly designed to have this property, but we believe that a similar phenomenon occurs. One advantage of the proposed technique over FAIDs, is that the proposed mapping functions can be used on a variety of channels, not only the BSC. Another issue with FAIDs is that the simulation results are restricted to weight-3 LDPC codes since there are most prone to have error-floors.

Even though FAIDs present good performance in the error-floor region, it can also be appreciated in Fig. 5.5 that FAIDs have error-rate penalty in the waterfall region, this happens because FAIDs perform the *min* operation in the check node updates, which is an approximation to the check node update performed by SPA and as a result a loss is seen.

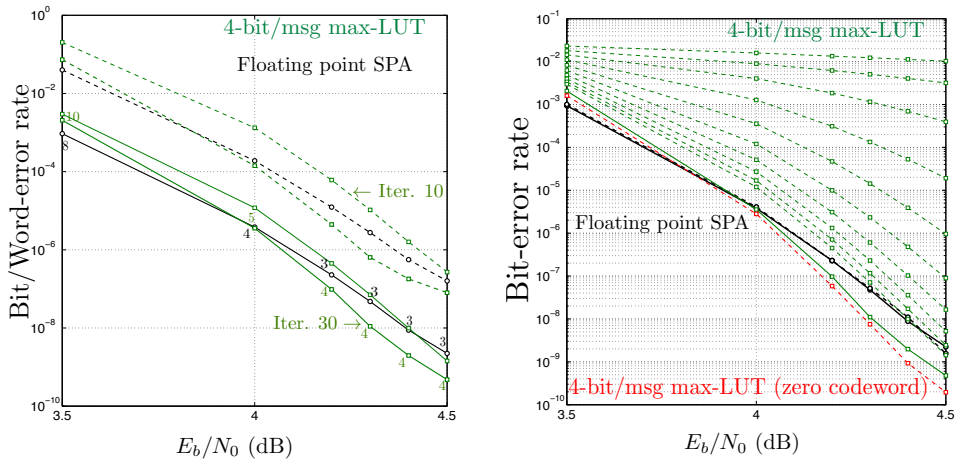
5.1.3 Simulation results for high rate codes

In this subsection, we analyze finite-length results for three different LDPC codes with rates $R = 0.84$, $R = 0.89$ and $R = 0.94$ and lengths $N = 2048$, $N = 1998$ and $N = 8970$ respectively.

Table 5.4: Decoding thresholds for channel and decoder message quantization using 3 and 4 bits per message.

Code	σ^2 (3 bits)	σ^2 (4 bits)
$(d_v = 6, d_c = 32)$-LDPC code	0.2930	0.2855
$(d_v = 4, d_c = 36)$-LDPC code	0.2410	0.2410
$(d_v = 4, d_c = 69)$-LDPC code	0.1867	0.1867

The noise decoding threshold used to construct the set of decoding mapping functions used in this section are listed in table 5.4. In Fig. 5.6, the BER and WER simulation results for the $(d_v = 6, d_c = 32)$-LDPC code are shown. The maximum number of iterations was set to 30. In table A.11 on page 94 different combinations on channel and decoder message quantization for the above code can be found. Since $(d_v = 6, d_c = 32)$-LDPC code is used in the IEEE 802.3an 10GBase-T standard producing an operation of 10 Gb/s Ethernet over up to 100 m of CAT-6a unshielded twisted-pair (UTP) cable, we are interested in the maximum number of iterations needed by the proposed maps to achieve similar decoding performance of that achieved by the SPA using 30 iterations as a maximum. In Fig. 5.6a, BER results are depicted using solid lines while the WER results are shown by dashed lines. In this graph, looking at the BER results

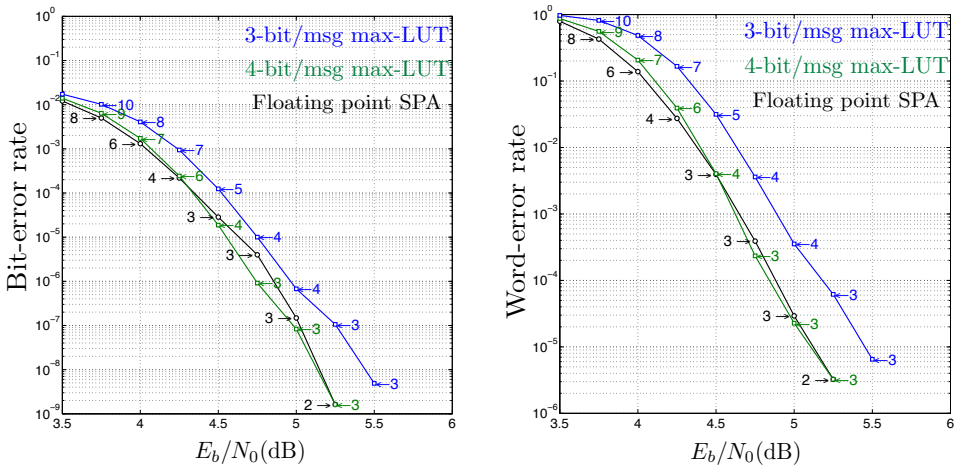


(a) Bit/Word error rate.

(b) Bit-error rate.

Figure 5.6: BER and WER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 6$, $d_c = 32$, $R = 0.84$, and $N = 2048$. The maximum number of Iterations was set to 30. The numbers next to the curves represent the average number of iterations for each simulation point.

for proposed decoding mapping functions, we can note that using 10 iterations as a maximum the proposed decoding mapping functions can achieve the BER performance of SPA using a maximum of 30 iterations. On the other hand, in Fig 5.6b a BER evolution through the maximum number of iterations of the proposed maps can be observed. Thus, in Fig 5.6b from top to the bottom, the first 10 dashed lines in green correspond to the BER performance achieved using 1–10 as the maximum number of iterations respectively. Note that the performance of the proposed maps using 30 as the maximum number of iterations is plotted by a green solid line. For this simulation, the average number of iterations achieved by SPA is $\{8, 4, 3, 3, 3, 3\}$ while $\{10, 5, 4, 4, 4, 4\}$ are those achieved by the proposed decoding mapping functions using 4 bits per message (see the numbers next to the BER curves in Fig. 5.6a), this means that for the lowest error-rate point corresponding to 4.5 dB, SPA achieved 3 and the proposed maps achieved 4. These numbers mean that most of the time the decoders perform 3 and 4 iterations respectively when the bit signal-to-noise ratio $E_b/N_0 = 4.5$ dB. On the other hand, this does not mean that if we set the maximum number of iterations to 3 or 4, we will get the same performance. In Fig. 5.6a, at 10^{-8} a gain of around 0.1 dB over SPA performance can be appreciated, this gap is setting 30 iterations as



(a) Bit-error rate.

(b) Word-error rate.

Figure 5.7: BER and WER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 4$, $d_c = 36$, $R = 0.89$, and $N = 1998$. The maximum number of Iterations was set to 30. The numbers next to the curves represent the average number of iterations for each simulation point.

a maximum number of iterations.

An important observation of the proposed decoding mapping functions is that the maps are not symmetric, this property was mentioned during the description made for the max-LUT method in Subsection 3.9. For all simulations presented in this chapter, we evaluated random codewords but for the simulation in Fig. 5.6, we also evaluated all zeros codeword to know wether an important difference due to symmetries in the mapping functions may exist. In Fig. 5.6b, the BER results for all zeros codeword also is depicted. Making a comparison between BER results achieved by the proposed decoding mapping functions using 4 bits per message, we can note that a small gap in fact exist due to symmetries but in any case both BER curves surpass the SPA performance curve.

The second high rate code is a $(d_v = 4, d_c = 36)$-LDPC code with rate $R = 0.89$ and length $N = 1998$. The maximum number of iterations also was set to 30. The noise decoding thresholds employed to compute the decoding mapping functions using 3 and 4 bits per message are presented in table 5.4.

In Fig.5.7, BER and WER results are presented. In Fig.5.7a, the proposed decoding mapping functions using 4 bits per message approach the SPA performance. On the other hand, using 3 bits per message at

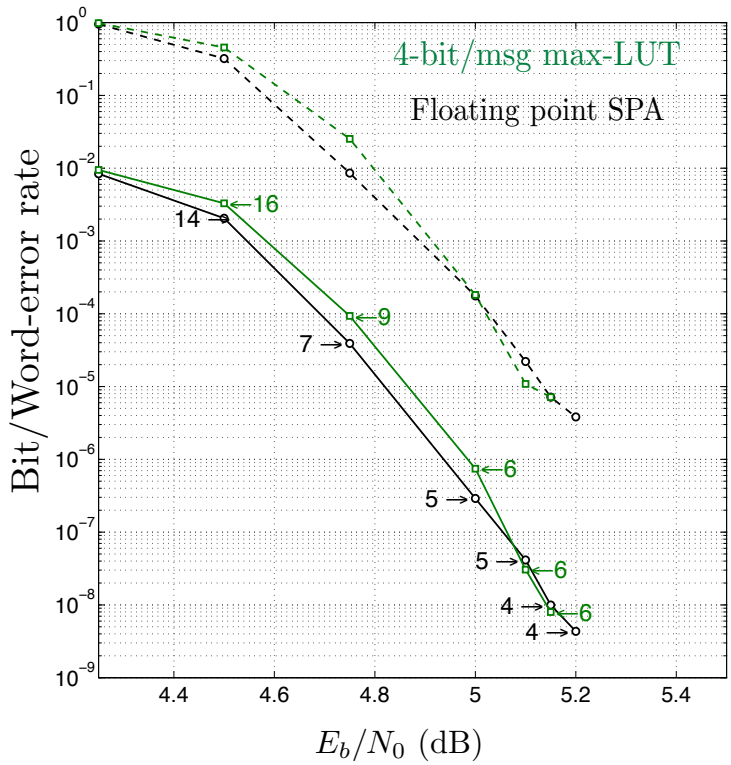


Figure 5.8: BER and WER results for the proposed decoding mapping functions, and sum-product algorithm. Parameters of the code: $d_v = 4$, $d_c = 69$, $R = 0.94$, and $N = 8970$. The maximum number of Iterations was set to 20. The numbers next to the curves represent the average number of iterations for each simulation point.

10^{-8} a gap around 0.25 dB can be seen. In Fig.5.7b, the WER results are shown, where at 10^{-5} the proposed maps using 3 bits per message have a gap around 0.35 dB. Again we continue seeing similar performance in the results to those presented for low and medium rate codes. Looking at the average number of iterations, we can note that in general the numbers have reduced compare to those in low and medium rates, this is not a surprise since the values of E_b/N_0 are bigger. Thus looking at the lowest error-rate probability for each decoding algorithm, we can see that the average number of iterations for SPA is 2 and for both decoding mapping functions is 3.

The third high rate code is a $(d_v = 4, d_c = 69)$-LDPC code with rate $R = 0.94$ and length $N = 8970$. The maximum number of iterations is set to 20. As the other codes, the noise decoding thresholds for different

channel and decoder message quantization can be found in table A.10 on page 93. Those decoding thresholds used for this specific simulation appear listen in table 5.4.

Continuing with the finite-length results for high rate codes, in Fig. 5.8 BER and WER results are shown for SPA and the proposed decoding mapping functions using 4 bits per message over a BI-AWGNC. In Fig. 5.8, we observe that even though in the waterfall region the BER curve of SPA is approximately 0.05 dB better than the proposed decoding mapping functions using 4 bits per message, at BER of 10^{-8}, SPA is reached by the proposed maps. In the same graph, WER results show a similar behaviour. From this results, we intuit that the loss in the waterfall region is because of too few bits per message were used. On the other hand, to approach SPA at low-error probability is of greater significance. In Fig. 5.8, at the lowest error-rate probability the average number of iterations of SPA is 4 while for the proposed decoding mappings is 6.

5.2 Summary

In this chapter, we analyzed the compuer-based simulation results for different regular (d_v, d_c)-LDPC codes. We divided the finite-length results into three different subsections to obtain a better understanding of the error correction capability of our proposed decoding mapping functions using 3 and 4 bits per message. Even though, the proposed mapping functions can be built for any number of bits per message (e.g., 1, 2, ..., etc.), we were interested in the sets of decoding mapping functions using a few bits per message (i.e. 3 or 4) able to reach or perform close to full sum-product algorithm using floating point values. Below, we list the most important observations observed in the simulations presented in this chapter.

- In 7 of the 8 codes simulated over the BI-AWGNC, the proposed decoding mapping functions using 4 bits per message achieved the error-rate performance of the sum-product algorithm using floating point numbers and, in some cases SPA was even surpassed by the proposed maps. In the case that the proposed decoding mapping functions using 4 bits per message had a gap respect to SPA, it was around 0.1 dB.

- Comparing the SPA with the proposed decoding mapping functions using 3 bits per message over a BI-AWGN channel, the proposed maps have a gap at most of 0.4 dB.

- Using a degree in the variable node $d_v \geq 4$ the error-rate curves look more smoothly, this aspect is due to two reasons, one that codes with $d_v = 3$ are more prone to have 4-girth cycles, and second that the chosen codes mostly are array codes (i.e., these codes by construction guarantee non 4-girth cycles [43]).

- An increment in the length of the code above 8000 generated a small gap of 0.05 dB but in the error-floor region the proposed decoding mapping functions achieved the SPA performance.

- On the average number of iterations, the proposed decoding mapping functions using 3 and 4 bits per message follow closely the numbers achieved by SPA. In all cases less than 15.

- The positive results obtained in this work for short and medium code lengths, indicates that maximization of mutual information as a metric for designing decoding mapping functions is suitable and could lead to successful decoding of regular LDPC codes. An open question is if the efficient decoding performance of the proposed maps will remain close to that achieved by the SPA for code lengths larger than 100000.

Chapter 6

Conclusions and future work

6.1 Conclusions

In this dissertation, we have developed a technique where the LDPC decoders and channel quantization implementations, including quantization of messages, are designed using only the probability distributions from the channel. Given a probability distribution, our method designs a lookup table (LUT) that maximizes mutual information, and LUTs are implemented directly in VLSI. This is the "max-LUT method".

The proposed lookup tables or sometimes referred as mapping functions are used for channel quantization and for message-passing decoding of LDPC codes. These decoding mapping functions are not derived from belief-propagation decoding or one of its approximations, instead, the decoding mapping functions are based on a channel quantizer that maximizes mutual information. More precisely, the construction technique is a systematic method which uses an optimal quantizer at each step of density evolution to generate message-passing decoding mappings.

In a simple manner, the design of LDPC decoders by maximization of mutual information is analogous to finding non-uniform quantization schemes where the quantization can vary with each iteration.

Simulation results show that the proposed decoding mapping functions using 4 bits per message can approach the error-rate performance of sum-product algorithm in the waterfall region and can surpass it in the error-floor region, this was observed in the BSC as well as in the BI-AWGNC. Originally, the proposed mapping functions/LUTs targeted data storage applications but after simulations results, the propose maps show good performance for a variety of rate codes.

6.2 Future work

During the development of the proposed decoding mapping functions for LDPC decoding, we have visualized some interesting research paths that may lead to exiting results for information and coding theory. Those are listed below.

- The proposed mapping functions are locally optimal in the sense of maximization of mutual information. Therefore, a natural problem to solve is to construct mapping function which are globally optimal.
- Density evolution algorithm assume an infinite block length for an LDPC code, it would be interesting to design a density evolution algorithm for finnite-length LDPC codes, in this way the quantizers could be optimized to produce higher decoding thresholds.
- Perform a channel quantization based on training data could give more information for our decoding mappings, unfortunately that kind of training data is only available in companies working on data storage applications.
- Extension of our approach to non-binary LDPC codes is an exciting research path, which is suitable to investigate in a near future.
- Since the optimal quantizer that we used in this research is connected to machine learning techniques, an interesting research path is to design "smart" LDPC decoders, this is, decoders that learn from unsuccessful decoding and modify the content of the lookup tables.
- Even though the proposed decoding mapping functions give some advantages in the hardware design of an LDPC decoder, the reliability of the hardware implementation itself needs more research since it is susceptible of error provoked by physical internal problems existing in VLSI.

Appendix A

Noise thresholds for a BI-AWGNC

In this appendix, decoding thresholds assuming a binary-input additive white Gaussian noise channel (BI-AWGNC) are presented. In this section there are 11 different tables, each one is related to a specific (d_v, d_c)-LDPC code. The set of channel quantization levels listed are $|\mathcal{Z}| \in \{2, 3, 4, 5, 6, 8, 12, 16, 20, 24, 28, 32\}$. The set of decoder message quantization listed are $K \in \{2, 3, 4, 5, 6, 7, 8, 16, 32\}$. An intersection between a row (number of quantization levels for the BI-AWGN) and a column (number of quantization levels for the decoder message) represents the decoding threshold of that specific (d_v, d_c)-LDPC with channel quantization levels $|\mathcal{Z}|$ and decoder quantization levels K. If other combinations for $|\mathcal{Z}|$ and K are investigated or other (d_v, d_c)-LDPC codes are investigated, apply the construction method presented in Section 4.3.

Table A.1: Noise decoding thresholds for a regular $(d_v = 2, d_c = 40)$-LDPC with rate $R = 19/20$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.0648	0.0709	0.0831	0.0831	0.0892	0.0892	0.0892	0.0892	0.0892
3	0.0831	0.0831	0.1075	0.1075	0.1075	0.1075	0.1075	0.1075	0.1075
4	0.0953	0.0953	0.1136	0.1136	0.1197	0.1197	0.1197	0.1197	0.1197
5	0.1014	0.1014	0.1197	0.1197	0.1197	0.1258	0.1258	0.1258	0.1258
6	0.1014	0.1075	0.1258	0.1258	0.1258	0.1258	0.1319	0.1319	0.1319
8	0.0953	0.1136	0.1258	0.1258	0.1319	0.1319	0.1319	0.1380	0.1380
12	0.1014	0.1197	0.1258	0.1319	0.1319	0.1380	0.1380	0.1380	0.1380
16	0.1014	0.1197	0.1258	0.1319	0.1380	0.1380	0.1380	0.1380	0.1441
20	0.1014	0.1197	0.1258	0.1319	0.1380	0.1380	0.1380	0.1380	0.1441
24	0.1014	0.1197	0.1319	0.1319	0.1380	0.1380	0.1380	0.1441	0.1441
28	0.1014	0.1197	0.1319	0.1319	0.1380	0.1380	0.1380	0.1441	0.1441
32	0.1014	0.1197	0.1319	0.1319	0.1380	0.1380	0.1380	0.1441	0.1441

Table A.2: Noise decoding thresholds for a regular $(d_v = 3, d_c = 4)$-LDPC with rate $R = 1/4$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.5973	0.8199	0.8793	0.9387	0.9758	0.9906	1.0055	1.0426	1.0574
3	0.5527	1.1094	1.0426	1.1688	1.1836	1.2355	1.2430	1.3023	1.3172
4	0.7680	1.0871	1.2504	1.2430	1.3023	1.3320	1.3469	1.4137	1.4211
5	0.7309	1.0648	1.2133	1.3320	1.3469	1.3914	1.3988	1.4582	1.4805
6	0.8273	1.1316	1.2430	1.3320	1.3914	1.4062	1.4359	1.4879	1.5102
8	0.8570	1.1688	1.2727	1.3543	1.3840	1.4359	1.4582	1.5250	1.5398
12	0.8719	1.2059	1.3098	1.3766	1.4211	1.4434	1.4730	1.5398	1.5547
16	0.8719	1.2133	1.3172	1.3840	1.4211	1.4582	1.4730	1.5473	1.5621
20	0.8719	1.2207	1.3172	1.3914	1.4285	1.4730	1.4730	1.5473	1.5621
24	0.8719	1.2207	1.3246	1.3840	1.4285	1.4656	1.4805	1.5473	1.5695
28	0.8719	1.2281	1.3246	1.3914	1.4359	1.4656	1.4805	1.5473	1.5695
32	0.8719	1.2281	1.3246	1.3914	1.4359	1.4656	1.4805	1.5473	1.5695

Table A.3: Noise decoding thresholds for a regular $(d_v = 3, d_c = 6)$-LDPC with rate $R = 1/2$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\|\mathcal{Z}\|$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.3078	0.4637	0.4414	0.4711	0.4859	0.4934	0.5008	0.5156	0.5156
3	0.3301	0.5379	0.5676	0.5973	0.5973	0.6195	0.6195	0.6418	0.6418
4	0.4266	0.5602	0.6270	0.6270	0.6492	0.6641	0.6715	0.6938	0.6938
5	0.4266	0.5676	0.6121	0.6641	0.6715	0.6863	0.6938	0.7160	0.7160
6	0.4711	0.6121	0.6270	0.6641	0.6938	0.6938	0.7012	0.7309	0.7309
8	0.4934	0.6121	0.6715	0.6789	0.6938	0.7160	0.7234	0.7383	0.7457
12	0.5008	0.6270	0.6641	0.6938	0.7012	0.7234	0.7309	0.7531	0.7605
16	0.5082	0.6418	0.6641	0.7012	0.7160	0.7234	0.7309	0.7531	0.7605
20	0.5082	0.6418	0.6715	0.6938	0.7160	0.7234	0.7309	0.7531	0.7605
24	0.5082	0.6418	0.6715	0.7012	0.7160	0.7309	0.7309	0.7605	0.7605
28	0.5082	0.6418	0.6789	0.7012	0.7160	0.7234	0.7309	0.7605	0.7605
32	0.5082	0.6418	0.6715	0.7012	0.7160	0.7309	0.7309	0.7605	0.7605

Table A.4: Noise decoding thresholds for a regular $(d_v = 4, d_c = 5)$-LDPC with rate $R = 1/5$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\|\mathcal{Z}\|$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.5527	0.8125	0.8273	0.8570	0.8867	0.8941	0.9090	0.9387	0.9461
3	0.8125	0.9016	0.9906	1.0500	1.0797	1.1020	1.1242	1.1613	1.1688
4	0.7160	0.9832	1.1094	1.1168	1.1613	1.1910	1.2059	1.2504	1.2578
5	0.8125	0.9387	1.1094	1.1762	1.2059	1.2281	1.2430	1.2949	1.3023
6	0.8867	1.0723	1.1020	1.1910	1.2281	1.2504	1.2652	1.3172	1.3246
8	0.8793	1.0871	1.1242	1.2207	1.2430	1.2727	1.2875	1.3395	1.3469
12	0.8941	1.1020	1.1688	1.2355	1.2578	1.2949	1.3098	1.3617	1.3691
16	0.8941	1.1391	1.1688	1.2355	1.2652	1.2949	1.3098	1.3617	1.3766
20	0.9016	1.1391	1.1762	1.2355	1.2652	1.2949	1.3172	1.3691	1.3766
24	0.9016	1.1316	1.1836	1.2430	1.2652	1.3023	1.3172	1.3691	1.3766
28	0.9016	1.1316	1.1836	1.2430	1.2727	1.3023	1.3172	1.3691	1.3766
32	0.9016	1.1391	1.1836	1.2430	1.2652	1.3023	1.3172	1.3691	1.3766

Table A.5: Noise decoding thresholds for a regular $(d_v = 4, d_c = 6)$-LDPC with rate $R = 1/3$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.4563	0.6121	0.6195	0.6418	0.6566	0.6641	0.6715	0.6938	0.6938
3	0.6195	0.6789	0.7309	0.7828	0.8051	0.8199	0.8273	0.8570	0.8570
4	0.5602	0.7309	0.8273	0.8422	0.8645	0.8793	0.8867	0.9164	0.9238
5	0.6418	0.7234	0.8199	0.8645	0.8867	0.9016	0.9164	0.9461	0.9535
6	0.6863	0.8348	0.8273	0.8793	0.9090	0.9164	0.9313	0.9609	0.9684
8	0.6863	0.8199	0.8422	0.9090	0.9164	0.9387	0.9461	0.9832	0.9832
12	0.6938	0.8199	0.8645	0.9090	0.9313	0.9535	0.9609	0.9906	0.9980
16	0.6938	0.8496	0.8645	0.9164	0.9313	0.9535	0.9684	0.9980	1.0055
20	0.7012	0.8496	0.8719	0.9164	0.9387	0.9535	0.9684	0.9980	1.0055
24	0.6938	0.8422	0.8793	0.9238	0.9387	0.9535	0.9684	0.9980	1.0055
28	0.7012	0.8496	0.8719	0.9164	0.9387	0.9535	0.9684	0.9980	1.0055
32	0.7012	0.8496	0.8793	0.9238	0.9387	0.9609	0.9684	0.9980	1.0055

Table A.6: Noise decoding thresholds for a regular $(d_v = 4, d_c = 8)$-LDPC with rate $R = 1/2$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.3598	0.4340	0.4340	0.4488	0.4637	0.4637	0.4711	0.4785	0.4859
3	0.4488	0.4785	0.5230	0.5527	0.5602	0.5750	0.5750	0.5898	0.5973
4	0.4340	0.5230	0.5750	0.5898	0.6047	0.6121	0.6195	0.6344	0.6344
5	0.4785	0.5305	0.5750	0.6047	0.6195	0.6270	0.6344	0.6492	0.6566
6	0.5008	0.5898	0.5898	0.6195	0.6344	0.6418	0.6492	0.6641	0.6641
8	0.5082	0.5750	0.5898	0.6344	0.6418	0.6492	0.6566	0.6715	0.6789
12	0.5082	0.5898	0.6047	0.6344	0.6492	0.6566	0.6641	0.6789	0.6863
16	0.5082	0.5973	0.6121	0.6344	0.6492	0.6641	0.6641	0.6863	0.6863
20	0.5156	0.5973	0.6047	0.6344	0.6492	0.6641	0.6715	0.6863	0.6863
24	0.5156	0.5973	0.6121	0.6418	0.6492	0.6641	0.6715	0.6863	0.6938
28	0.5156	0.6047	0.6195	0.6418	0.6492	0.6641	0.6715	0.6863	0.6938
32	0.5156	0.5973	0.6121	0.6418	0.6492	0.6641	0.6715	0.6863	0.6938

Table A.7: Noise decoding thresholds for a regular $(d_v = 4, d_c = 9)$-LDPC with rate $R = 5/9$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.3301	0.3820	0.3820	0.4043	0.4117	0.4117	0.4191	0.4266	0.4266
3	0.4043	0.4340	0.4711	0.4859	0.5008	0.5082	0.5082	0.5230	0.5230
4	0.3895	0.4711	0.5156	0.5230	0.5305	0.5453	0.5453	0.5602	0.5602
5	0.4340	0.4711	0.5156	0.5379	0.5527	0.5602	0.5676	0.5750	0.5824
6	0.4488	0.5230	0.5305	0.5453	0.5602	0.5676	0.5750	0.5898	0.5898
8	0.4563	0.5156	0.5305	0.5602	0.5676	0.5750	0.5824	0.5973	0.5973
12	0.4563	0.5305	0.5379	0.5602	0.5750	0.5824	0.5898	0.6047	0.6047
16	0.4637	0.5305	0.5453	0.5676	0.5750	0.5824	0.5898	0.6047	0.6047
20	0.4637	0.5305	0.5379	0.5676	0.5750	0.5898	0.5898	0.6047	0.6121
24	0.4637	0.5305	0.5453	0.5676	0.5750	0.5898	0.5898	0.6047	0.6121
28	0.4637	0.5379	0.5453	0.5676	0.5750	0.5898	0.5898	0.6047	0.6121
32	0.4637	0.5305	0.5453	0.5676	0.5750	0.5898	0.5898	0.6047	0.6121

Table A.8: Noise decoding thresholds for a regular $(d_v = 4, d_c = 36)$-LDPC with rate $R = 8/9$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.1445	0.1594	0.1668	0.1742	0.1742	0.1742	0.1742	0.1816	0.1816
3	0.1816	0.1965	0.2039	0.2113	0.2113	0.2113	0.2113	0.2188	0.2188
4	0.1891	0.2039	0.2188	0.2188	0.2262	0.2262	0.2262	0.2336	0.2336
5	0.1965	0.2113	0.2188	0.2262	0.2262	0.2336	0.2336	0.2336	0.2336
6	0.2039	0.2262	0.2262	0.2336	0.2336	0.2336	0.2336	0.2410	0.2410
8	0.2113	0.2188	0.2262	0.2336	0.2336	0.2336	0.2410	0.2410	0.2410
12	0.2113	0.2262	0.2262	0.2336	0.2336	0.2410	0.2410	0.2410	0.2410
16	0.2113	0.2262	0.2262	0.2336	0.2410	0.2410	0.2410	0.2410	0.2484
20	0.2113	0.2262	0.2262	0.2336	0.2410	0.2410	0.2410	0.2484	0.2484
24	0.2113	0.2262	0.2262	0.2336	0.2410	0.2410	0.2410	0.2484	0.2484
28	0.2113	0.2262	0.2262	0.2336	0.2410	0.2410	0.2410	0.2484	0.2484
32	0.2113	0.2262	0.2262	0.2336	0.2410	0.2410	0.2410	0.2484	0.2484

Table A.9: Noise decoding thresholds for a regular $(d_v = 4, d_c = 42)$-LDPC with rate $R = 19/21$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.1371	0.1520	0.1594	0.1594	0.1668	0.1668	0.1668	0.1668	0.1668
3	0.1742	0.1891	0.1891	0.1965	0.1965	0.1965	0.2039	0.2039	0.2039
4	0.1816	0.1965	0.2039	0.2113	0.2113	0.2113	0.2113	0.2188	0.2188
5	0.1891	0.1965	0.2039	0.2113	0.2188	0.2188	0.2188	0.2188	0.2188
6	0.1891	0.2113	0.2113	0.2188	0.2188	0.2188	0.2188	0.2262	0.2262
8	0.1965	0.2039	0.2113	0.2188	0.2188	0.2188	0.2262	0.2262	0.2262
12	0.1965	0.2113	0.2113	0.2188	0.2188	0.2262	0.2262	0.2262	0.2262
16	0.1965	0.2113	0.2113	0.2188	0.2262	0.2262	0.2262	0.2262	0.2262
20	0.1965	0.2113	0.2113	0.2188	0.2262	0.2262	0.2262	0.2262	0.2262
24	0.1965	0.2113	0.2113	0.2188	0.2262	0.2262	0.2262	0.2262	0.2262
28	0.1965	0.2113	0.2113	0.2188	0.2262	0.2262	0.2262	0.2262	0.2336
32	0.1965	0.2113	0.2113	0.2188	0.2262	0.2262	0.2262	0.2262	0.2336

Table A.10: Noise decoding thresholds for a regular $(d_v = 4, d_c = 69)$-LDPC with rate $R = 65/69$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.1136	0.1319	0.1313	0.1312	0.1312	0.1406	0.1380	0.1380	0.1406
3	0.1441	0.1562	0.1625	0.1594	0.1594	0.1688	0.1684	0.1684	0.1688
4	0.1502	0.1623	0.1688	0.1688	0.1781	0.1781	0.1745	0.1806	0.1781
5	0.1562	0.1688	0.1750	0.1781	0.1781	0.1781	0.1806	0.1806	0.1781
6	0.1623	0.1750	0.1750	0.1781	0.1781	0.1781	0.1867	0.1867	0.1875
8	0.1675	0.1688	0.1750	0.1781	0.1781	0.1875	0.1867	0.1867	0.1875
12	0.1675	0.1750	0.1750	0.1781	0.1875	0.1875	0.1867	0.1867	0.1875
16	0.1675	0.1750	0.1750	0.1781	0.1875	0.1875	0.1875	0.1867	0.1875
20	0.1675	0.1750	0.1750	0.1781	0.1875	0.1875	0.1875	0.1928	0.1875
24	0.1675	0.1750	0.1750	0.1781	0.1875	0.1875	0.1875	0.1928	0.1875
28	0.1675	0.1750	0.1750	0.1781	0.1875	0.1875	0.1875	0.1928	0.1875
32	0.1675	0.1750	0.1750	0.1781	0.1875	0.1875	0.1875	0.1875	0.1875

Table A.11: Noise decoding thresholds for a regular $(d_v = 6, d_c = 32)$-LDPC with rate $R = 13/16$ over a BI-AWGNC using different quantization levels K and $|\mathcal{Z}|$ for the decoder message and the channel message respectively. The term $\log_2(K)$ is the number of bits to represent the decoder message, while $\log_2(|\mathcal{Z}|)$ is the number of bits to represent the channel message.

$\lvert\mathcal{Z}\rvert$	K (All noise thresholds are the variance value σ^2)								
	2	3	4	5	6	7	8	16	32
2	0.1891	0.2039	0.2039	0.2113	0.2113	0.2188	0.2188	0.2188	0.2188
3	0.2336	0.2410	0.2484	0.2559	0.2559	0.2559	0.2559	0.2633	0.2633
4	0.2410	0.2484	0.2559	0.2707	0.2707	0.2707	0.2707	0.2781	0.2781
5	0.2484	0.2559	0.2633	0.2707	0.2707	0.2781	0.2781	0.2781	0.2855
6	0.2410	0.2707	0.2633	0.2707	0.2781	0.2781	0.2781	0.2855	0.2855
8	0.2484	0.2559	0.2707	0.2781	0.2781	0.2855	0.2855	0.2855	0.2855
12	0.2559	0.2707	0.2707	0.2781	0.2855	0.2855	0.2855	0.2930	0.2930
16	0.2559	0.2707	0.2707	0.2781	0.2855	0.2855	0.2855	0.2930	0.2930
20	0.2559	0.2707	0.2707	0.2781	0.2855	0.2855	0.2855	0.2930	0.2930
24	0.2559	0.2707	0.2707	0.2781	0.2855	0.2855	0.2855	0.2930	0.2930
28	0.2559	0.2707	0.2707	0.2781	0.2855	0.2855	0.2855	0.2930	0.2930
32	0.2559	0.2707	0.2707	0.2781	0.2855	0.2855	0.2855	0.2930	0.2930

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July/October 1948.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proceedings IEEE International Conference on Communications*, vol. 2, (Geneva, Switzerland), pp. 1064–1070, IEEE, May 1993.

[3] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: The M.I.T. Press, 1963.

[4] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, 1991.

[5] R. W. Hamming, *Coding and Information Theory*. Prentice-Hall, Inc., 1986.

[6] D. J. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge university press, 2003.

[7] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[8] S.-Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Communications Letters*, vol. 5, pp. 58–60, February 2001.

[9] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, pp. 599–618, February 2001.

[10] W. Ryan and S. Lin, *Channel codes: classical and modern*. New York, NY, USA: Cambridge University Press, 1st ed., 2009. ISBN 978-0-521-84868-8.

[11] M. Baldi, G. Cancellieri, and F. Chiaraluce, "Array convolutional low-density parity-check codes," *IEEE Communications Letters*, vol. 18, pp. 336–339, February 2014.

[12] L. Ping and W. K. Leung, "Decoding low density parity check codes with finite quantization bits," *IEEE Communications Letters*, vol. 4, pp. 62–64, February 2000.

[13] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *IEEE Communications Letters*, vol. 9, pp. 814–816, September 2005.

[14] T. Magloire, N. Ngatched, M. Bossert, A. Fahrner, and F. Takawira, "Two bit-flipping decoding algorithms for low-density parity-check codes," *IEEE Transactions on Communications*, vol. 57, pp. 591–596, March 2009.

[15] 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*, vol. 47, pp. 673–680, May 1999.

[16] N. Wiberg, *Codes and decoding on general graphs*. PhD thesis, Dept. Elec. Eng., Linköping Univ., Linköping, Sweden (Linköping Studies in Science and Technology, Dissertation 440), 1996.

[17] J. Chen and M. P. C. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," in *Proceedings IEEE Global Telecommunications Conference*, (Taipei, Taiwan), IEEE, November 2002.

[18] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes," *IEEE Transactions on Communications*, vol. 53, pp. 549–554, April 2005.

[19] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, pp. 1288–1299, August 2005.

[20] J.-S. Lee and J. Thorpe, "Memory-efficient decoding of LDPC codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Adelaide, Australia), pp. 459–463, September 2005.

[21] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Transactions on Communications*, vol. 57, pp. 3258–3268, November 2009.

[22] X. Zhang and P. H. Siegel, "Quantized min-sum decoders with low error floor for LDPC codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Cambridge, MA, USA), pp. 2871–2875, July 2012.

[23] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders–part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Transactions on Communications*, vol. 61, pp. 4033–4045, October 2013.

[24] J. Lewandowsky, M. Stark, and G. Bauch, "Optimum message mapping LDPC decoders derived from the sum-product algorithm," in *Proceedings IEEE International Conference on Communications*, pp. 1–6, May 2016.

[25] B. M. Kurkoski and H. Yagi, "Quantization of binary-input discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 60, pp. 4544–4552, August 2014.

[26] K. Moon Todd, *Error Correction Coding, Mathematical Methods and Algorithms*. John Wiley & Sons, Ltd, 2005.

[27] S. M. Aji and R. J. McEliece, "A general algorithm for distributing information in a graph," in *Proceedings of IEEE International Symposium on Information Theory*, p. 6, June 1997.

[28] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Transactions on Information Theory*, vol. 46, pp. 325–343, March 2000.

[29] T. Etzion, A. Trachtenberg, and A. Vardy, "Which codes have cycle-free tanner graphs?," *IEEE Transactions on Information Theory*, vol. 45, pp. 2173–2181, September 1999.

[30] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, March 1999.

[31] D. J. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[32] D. Burshtein, V. D. Pietra, D. Kanevsky, and A. Nadas, "Minimum impurity partitions," *The Annals of Statistics*, vol. 20, no. 3, pp. 1637–1646, 1992.

[33] P. A. Chou, "Optimal partitioning for classification and regression trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 340–354, April 1991.

[34] B. M. Kurkoski, "Source code-quantDMC, version 3." `http://www.jaist.ac.jp/is/labs/bits/source`, June 2002. Accessed: 2016-04-13.

[35] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, pp. 657–670, February 2001.

[36] C.-C. Wang, S. R. Kulkarni, and H. V. Poor, "Density evolution for asymmetric memoryless channels," *IEEE Transactions on Information Theory*, vol. 51, pp. 4216–4236, December 2005.

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

[38] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, March 2008.

[39] D. J. MacKay and M. S. Postol, "Weaknesses of margulis and ramanujan-margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, 2003.

[40] T. Richardson, "Error floors of LDPC codes," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, pp. 1426–1435, The University; 1998, 2003.

[41] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Transactions on Information Theory*, vol. 56, pp. 181–201, January 2010.

[42] F. J. C. Romero and B. M. Kurkoski, "LDPC decoding mappings that maximize mutual information," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 2391–2401, September 2016.

[43] O. Milenkovic, N. Kashyap, and D. Leyba, "Shortened array codes of large girth," *Information Theory, IEEE Transactions on*, vol. 52, no. 8, pp. 3707–3722, 2006.

[44] D. J. C. MacKay, "Encyclopedia of sparse graph codes." `http://www.inference.phy.cam.ac.uk/mackay/codes/data.html`. Accessed: 2016-04-13.

[45] IEEE Std. 802.3an, *IEEE Standard for Information Technology-Telecommunications and Information Exchange between Systems- Local and Metropolitan Area Networks-Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, September 2006.

[46] J. Li, K. Liu, S. Lin, and K. Abdel-Ghaffar, "Algebraic quasi-cyclic LDPC codes: Construction, low error-floor, large girth and a reduced-complexity decoding scheme," *IEEE Transactions on Communications*, vol. 62, pp. 2626–2637, August 2014.

[47] Z. Liu and D. A. Pados, "A decoding algorithm for finite-geometry LDPC codes," *IEEE Transactions on Communications*, vol. 53, pp. 415–421, March 2005.

Publications

[1] **F. J. C. Romero**, B. M. Kurkoski, "Decoding mapping functions for high-rate LDPC codes ," *The 8th Annual Non-Volatile Memories Workshop (NVMW 2017)*, March 2017.

[2] **F. J. C. Romero**, B. M. Kurkoski, "Decoding mapping functions that maximize mutual information for LDPC codes," *Proceedings of the 39th Symposium on Information Theory and Its Applications*, pp. 187–192, December 2016.

[3] **F. J. C. Romero**, B. M. Kurkoski, "LDPC decoding mappings that maximize mutual information," *IEEE Journal on Selected Areas in Communications*, vol. 34, No. 9, pp. 2391-2401, September 2016.

[4] **F. J. C. Romero**, B. M. Kurkoski, "Decoding LDPC codes with mutual Information-maximizing lookup tables," *Proceedings of IEEE International Symposium on Information Theory*, vol. 34, No. 9, pp. 426–430, June 2015.

[5] **F. J. C. Romero**, B. M. Kurkoski, "Low complexity 3-bit LDPC-LUT decoding algorithm," *Proceedings of the 37th Symposium on Information Theory and Its Applications*, vol. 34, No. 9, pp. 409–414, December 2014.

[6] **F. J. C. Romero**, B. M. Kurkoski, H. Yagi, "Two-bit LDPC–LUT decoder based on maximization of mutual information," *Proceedings of the 36th Symposium on Information Theory and Its Applications*, vol. 34, No. 9, pp. 232–237, November 2013.