ANR-15-CE25-0006-01 (NAND)

Deliverable D1.1

Noisy Density Evolution for NAND Decoders

Franklin Cochachin
David Declercq
Lounis Kessal
Emmanuel Boutillon

Abstract

In this deliverable we will present the density evolution (DE) framework to analyze noisy message passing decoders in the asymptotical limit of the codeword length. The concept of DE is a classical tool that is used to study the performance of LDPC code ensembles, and predict the waterfall of the considered decoders. In this deliverable, we use DE to compare different decoder structures on the same LDPC ensemble, and we generalize the study to noisy decoders which use symmetric and memoryless error models. The noise models could have any characteristic in terms of transition probability, and we show a simple example to validate the noisy-DE approach.

The update rules of noiseless and noisy decoders will be derived, with a focus on Min-Sum-Based decoders which have small message alphabets constructed from 3 and 4 quantization bits, and for regular LDPC code ensembles with $d_v = 3$ and $d_v = 4$.

For the simple error model considered, we analyze the noisy decoder performance and show that the DE noisy-thresholds results are better than the noiseless versions of the same decoder. The injected noise is then beneficial to the convergence of the decoders in the waterfall region.

As a conclusion, the results of this work can serve as general tools to analyze the quantized noisy Min-Sum decoders within the NAND project. More elaborate error models and generalizations of the DE thresholds for the BI-AWGN will be studied in future works.

November 2016

Contents

Chapter 1

Generalities on LDPC codes

In a communication system, a transmitter sends information through a noisy channel to one or more receivers. The channel adds random noise and corrupts the information, and the receiver has the purpose to retrieve the information with the least possible loss. In order to protect the information against the channel noise, the transmitter adds redundancy to the information such that the receiver can detect and correct the errors. Such process is called error correcting coding and decoding.

In the coding process, an error-correcting code converts a sequence of K information bits into a longer sequence of N bits using a coding function which defines how to build the $N - K$ redundancy bits. Examples of coding processes include convolutional codes, block turbo-codes, LDPC codes, algebraïc codes, etc.

In convolutional codes, the coding function uses individually each bit of the sequence of K bits to build the redundancy bits, through a discrete linear filter. The classical algorithms used for the decoding are the BCJR algorithm and the Viterbi algorithm.

In block codes the encoding is made by block of bits, and in this case the sequence of K bits is used altogether to build the redundancy bits, through a binary generator matrix.

Nowadays, the most popular error-correcting codes are LDPC codes [1–3] and the turbo-codes, because they have a high performance and practical decoding algorithms. LDPC codes are used for communications standards like DVB-S2 [4], DVB-S2X [5], IEEE 802.3an, etc. Turbo codes are also used for communications standards like LTE, DVB-RCS, WiMAX (IEEE 802.16), etc.

In this deliverable, we will only study LDPC codes and their associated decoders.

1.1 Representations of LDPC codes

LDPC codes can be represented either with a matrix representation or with a graphical representation [3, 6–8].

1.1.1 Matrix Representation

An LDPC code is a linear block code defined by a sparse parity-check matrix $H = [h_{ij}]$ of M rows by N columns. Each column in H is associated with a bit in the codeword while each row is associated with a parity-check equation. The following matrix is an example of a binary parity-check matrix:

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.1}$$

Let further $\boldsymbol{x}$ denote a codeword which can be represented as a vector $\boldsymbol{x} = (\hat{x}_1, ..., \hat{x}_N) \in \{0,1\}^N$. The matrix H is used to check the parity of the codeword bits, i.e. $H\boldsymbol{x}^T = 0$. For example, using the matrix presented in (1.1), a parity-check equation is given by $\hat{x}_1 \oplus \hat{x}_6 \oplus \hat{x}_7 \oplus \hat{x}_{12} = 0$, where $\oplus$ represents the modulo-two sum.

Let d_H be the density of H defined as the number of non-zero elements in H divided by the matrix size. For LDPC codes, the amount of non-zero elements in H is very small compared to the amount of 0's, and d_H converges to zero when N increases to infinity [3, 9].

1.1.2 Graphical Representation

LDPC codes can also be represented by a Tanner graph [6]. A Tanner graph is a bipartite graph whose nodes are divided into two disjoint sets. One set is made of variable-nodes (VN) and the other set by check-nodes (CN). In the graph, only check-nodes are connected to variable-nodes and vice versa, other types of connection are not allowed. Furthermore, a variable-node is typically represented with a circle, while a check-node is represented with a square.

The Tanner graph and H are closely related. The VN in the graph correspond to the columns of H and the CN correspond to the rows of H, with an edge connecting CN i to VN j exists if and only if $h_{ij} \neq 0$.

1.2 Classification of LDPC codes

LDPC codes are classified according to their structural properties as regular or irregular LDPC codes. Additionally, the LDPC codes can also be classified as non-structured LDPC codes and structured LDPC codes. Non-structured LDPC codes do not exhibit a specific structure, while in structured LDPC codes, H is generated with algebraic equations, or contrained by specific topological properties. Usually, those constraints are introduced in order to help the decoder to have a low cost hardware implementation. One could further categorize the structured LDPC codes in three types: *(i)* quasi-cyclic LDPC codes, *(ii)* convolutional LDPC codes, and *(iii)* algebraic constructions of LDPC codes.

1.2.1 Regular LDPC codes

A (dv, dc)-regular LDPC code has all its variable-nodes with the same degree and all its check-nodes of a fixed degree. This means that the number of edges incident to each variable-node is constant, denoted by d_v (variable node degree), and the number of edges incident to each check-node is also constant, denoted by d_c (check node degree). In the corresponding sparse parity-check matrix, d_v is the amount of non-zero elements per column, and d_c is the amount of non-zero elements per row.

For a regular LDPC code, the density of H is equal to $d_H = d_v/M = d_c/N$. For example, the matrix in (1.1) has $d_v = 3$, $d_c = 4$ and $d_H = 1/3$.

Let E denote the number of edges in the Tanner graph, or equivalently E is the amount of non-zero elements in H. For a regular LDPC code we have $E = d_v\, N = d_c\, M$. The code rate R can be calculated as $R = K/N \geq \frac{N-M}{N}$ [3], and if the rows of H are linearly independent, we can write $R = 1 - (d_v/d_c)$, which is usually defined as the *design rate* [10].

In figure 1.1, we show a bipartite graph which is related to the sparse parity-check matrix presented in equation (1.1).

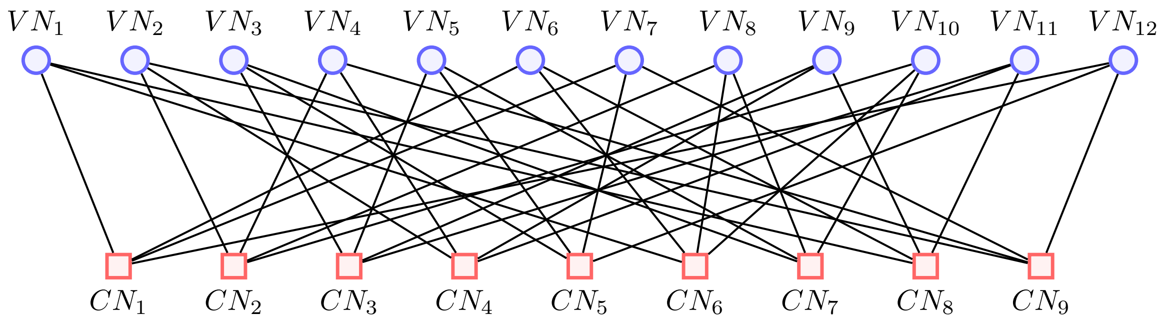


Figure 1.1: A Tanner graph for a $(3, 4)$-regular LDPC code of length 12. There are 12 VN, 9 CN, and 36 edges.

An LDPC code is uniquely defined by the choice of a parity-check matrix H, but for the theoretical analysis of LDPC codes, we need to introduce the concept of code ensembles.

Definition 1.2.1. *An ensemble or family $\mathcal{C}^N(d_v, d_c)$ of LDPC codes is composed of all the Tanner graphs with N VNs and regular degrees d_v and d_c, generated from all the possible permutations π of the edges [8, 10–13]. π is a permutation of size E.*

1.2.2 Irregular LDPC codes

For irregular LDPC codes [12], not all variable-nodes and/or check-nodes have the same degree, i.e. in H, the amount of non-zero elements in at least one row (and/or one column) is different from the amount of non-zero elements from the other rows (and/or columns).

Let $\lambda(x)$ denote the edge-wise VN degree distribution. Let $\rho(x)$ denote the edge-wise CN degree distribution. Similarly, let $\tilde{\lambda}(x)$ denote the node-wise VN degree distribution, and let $\tilde{\rho}(x)$ denote the node-wise CN degree distribution. Those polynomials are expressed the following way:

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

$$\rho(x) = \sum_{j=2}^{d_{c_{max}}} \rho_j x^{j-1},$$

where $\lambda_i \in$ [0,1] denotes the proportion of edges connected to variable-nodes of degree i, and $\rho_j \in$ [0,1] denotes the proportion of edges connected to check-nodes of degree j.

$$\tilde{\lambda}(x) = \sum_{i=2}^{d_{v_{max}}} \tilde{\lambda}_i x^{i-1},$$

$$\tilde{\rho}(x) = \sum_{j=2}^{d_{c_{max}}} \tilde{\rho}_j x^{j-1},$$

where $\tilde{\lambda}_i \in$ [0,1] denotes the proportion of variable-nodes of degree i, and $\tilde{\rho}_j \in$ [0,1] denotes the proportion of check-nodes of degree j.

As for the case of regular LDPC codes, E denotes the number of edges in the Tanner graph. Considering H, the amount of non-zero elements in columns of degree i is calculated as $E\lambda_i$ or $i\tilde{\lambda}_i N$. Similarly, the amount of non-zero elements in rows of degree j is given by $E\rho_j$ or $j\tilde{\rho}_j M$. Then, it is easy to obtain

$$E = \frac{i\tilde{\lambda}_i N}{\lambda_i} = \frac{j\tilde{\rho}_j M}{\rho_j}$$

The relationship between λ_i and $\tilde{\lambda}_i$ is given by

$$\tilde{\lambda}(x) = \sum_{i=2}^{d_{v_{max}}} \frac{\lambda_i E}{iN} x^{i-1},$$

Similarly, the relationship between ρ_j and $\tilde{\rho}_j$ is

$$\tilde{\rho}(x) = \sum_{j=2}^{d_{c_{max}}} \frac{\rho_j E}{jM} x^{j-1},$$

The design rate $R(\lambda, \rho)$

The design rate can be computed as $R(\lambda, \rho) = \frac{N-M}{N}$ [11].

We can calculate the number of variable-nodes as

$$E \int_0^1 \lambda(x) dx.$$

Similarly, the number of check-nodes is equal to

$$E \int_0^1 \rho(x) dx.$$

Using $\lambda(x)$ we have

$$\int_0^1 \lambda(x) dx = \int_0^1 \sum_{i=2}^{d_{v_{max}}} \lambda_i x^{i-1} = \sum_{i=2}^{d_{v_{max}}} \frac{\lambda_i}{i},$$

And using $\rho(x)$ we get

$$\int_0^1 \rho(x) dx = \int_0^1 \sum_{j=2}^{d_{c_{max}}} \rho_j x^{j-1} = \sum_{j=2}^{d_{c_{max}}} \frac{\rho_j}{j},$$

Therefore $R(\lambda, \rho)$ can also be computed as

$$R(\lambda, \rho) = 1 - \frac{\int_0^1 \rho(x) dx}{\int_0^1 \lambda(x) dx},$$

or equivalently

$$R(\lambda, \rho) = 1 - \frac{\sum_{j=2}^{d_{c_{max}}} \frac{\rho_j}{j}}{\sum_{i=2}^{d_{v_{max}}} \frac{\lambda_i}{i}}.$$

In figure 1.2, we show a Tanner graph for an irregular LDPC code. For example, for this graph we have $\lambda(x) = \frac{2}{3}x + \frac{1}{3}x^2$, $\rho(x) = \frac{1}{2}x^2 + \frac{2}{9}x^3 + \frac{5}{18}x^4$, $\tilde{\lambda}(x) = \frac{3}{4}x + \frac{1}{4}x^2$, $\tilde{\rho}(x) = \frac{3}{5}x^2 + \frac{1}{5}x^3 + \frac{1}{5}x^4$ and $R(\lambda, \rho) = \frac{3}{8}$.

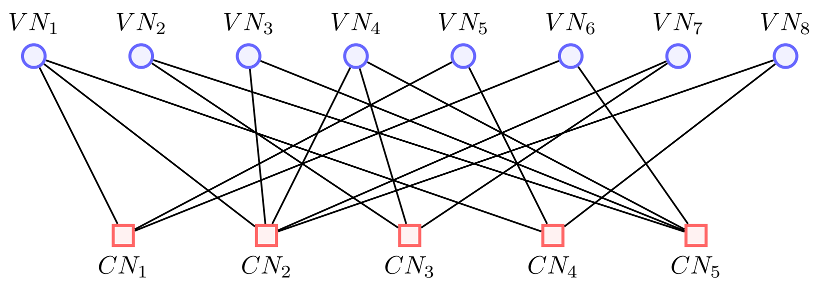


Figure 1.2: A Tanner graph for an irregular LDPC code of length 8. There are 9 VN, 5 CN, and 18 edges.

1.2.3 Quasi-cyclic LDPC codes

Among all types of LDPC codes that we have mentioned, we describe only quasi-cyclic (QC) LDPC codes in details, because these are the codes that will be used in the NAND project.

A QC-LDPC code is characterized by its parity-check matrix which is organized in small square sub-matrices which are either the all-zero matrix or circulant permutation matrices [14, 15]. Let P^i denote the $L \times L$ permutation matrix which shifts the identity matrix I to the left by i times $0 \leq i < L$. We consider that P^∞ denotes the zero matrix, and P^0 is equal to I. For example P^1 is given by

$$P^1 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \end{pmatrix}$$

Using circulant permutation matrices, H can be built as follows

$$H = \begin{pmatrix} P^{a_{11}} & P^{a_{12}} & \dots & P^{a_{1(n-1)}} & P^{a_{1n}} \\ P^{a_{21}} & P^{a_{22}} & \dots & P^{a_{2(n-1)}} & P^{a_{2n}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ P^{a_{m1}} & P^{a_{m2}} & \dots & P^{a_{m(n-1)}} & P^{a_{mn}} \end{pmatrix} \tag{1.2}$$

Where H has $M = mL$ rows and $N = nL$ columns, and $a_{ij} \in \{0, 1, ..., L-1, \infty\}$.

A QC-LDPC code may be regular or irregular depending on the choice of a_{ij}'s of H in (1.2). When $a_{ij} \neq \infty$, a QC-LDPC code is a $(M/L, N/L)$-regular LDPC code. For example, the matrix in (1.1) represents a QC-LDPC code which can be conveniently written as follows:

$$H = \left(\begin{array}{ccc|ccc|ccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right) = \begin{pmatrix} P^0 & P^1 & P^0 & P^1 \\ P^2 & P^0 & P^1 & P^2 \\ P^1 & P^2 & P^2 & P^0 \end{pmatrix}$$

Where $M = 9$, $N = 12$, and $L = 3$. Some values of a_{ij} are $a_{11} = 0$, $a_{12} = 1$ and $a_{32} = 2$.

Remark. *The QC-LDPC codes presented in this section are a special case of type-I protograph codes [16], in which the graph expansion is constrained to be quasi-cyclic. For type-II protographs, one square sub-matrix could contain two or more circulants.*

1.3 File Format for LDPC Codes

In this section, we present two file formats that will be used to store the parity-check matrices of LDPC codes. The firts one is called *Alist* [17] and the other one stores only the exponents of the quasi-cyclic form of (1.2).

1.3.1 Alist format

A file in Alist format for a (dv, dc)-regular LDPC code contains in its 1st row the dimension of H, in its 2nd row both values d_v and d_c, in its 3rd and 4th rows the degrees of the N variable-nodes and the M check-nodes, respectively. Then the next rows indicate the positions of the non-zero elements in each row of H.

For example, considering the matrix in (1.1), a file in Alist format for a $(3, 4)$-regular LDPC code is shown in Table 1.1. In this file we have $M = 9$ and $N = 12$, then $d_v = 3$ and $d_c = 4$, then the degrees of the 12 VNs and the 9 CNs. Then the position of non-zero elements in the 1st row of H, i.e. 0, 5, 6, and 11; then the position of non-zero elements in the 2nd row of H, i.e. 1, 3, 7, and 9; and so on.

```
%=========================================
9     12
3     4

3   3   3   3   3   3   3   3   3   3   3   3
4   4   4   4   4   4   4   4   4

0    5    6    11
1    3    7    9
2    4    8    10
1    3    8    10
2    4    6    11
0    5    7    9
2    4    7    9
0    5    8    10
1    3    6    11
%=========================================
```

Table 1.1: Alist format for the LDPC code of equation (1.1)

1.3.2 Quasi-cyclic format

The quasi-cyclic format consists in storing the exponents of the matrix P in equation (1.2), with the following format.

$$\begin{array}{llllll}
M & N & L & & & \\
\nu_{11} & \nu_{12} & \nu_{13} & \cdots & \nu_{1(n-1)} & \nu_{1n} \\
\nu_{21} & \nu_{22} & \nu_{13} & \cdots & \nu_{2(n-1)} & \nu_{2n} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\nu_{m1} & \nu_{m2} & \nu_{m3} & \cdots & \nu_{m(n-1)} & \nu_{mn}
\end{array}$$

Where $\nu_{ij} = -1$ if $a_{ij} = \infty$, $\nu_{ij} = a_{ij}$ otherwise (see section 1.2.3).

For example, the Table 1.2 shows a quasi-cyclic format for the matrix presented in (1.1). The file contains in its 1st row $M = 9$, $N = 12$, and $L = 3$; in its 2nd row $\nu_{11} = 0$, $\nu_{12} = 1$, $\nu_{13} = 0$, and $\nu_{14} = 1$; in its 3rd row $\nu_{21} = 2$, $\nu_{22} = 0$, $\nu_{23} = 1$, and $\nu_{24} = 2$; and in its 4th row $\nu_{31} = 1$, $\nu_{32} = 2$, $\nu_{33} = 2$, and $\nu_{34} = 0$.

```
%========================================
9    12  3

0    1    0    1
2    0    1    2
1    2    2    0
%========================================
```

Table 1.2: Quasi-cyclic format for the LDPC code of equation (1.1)

Chapter 2

Binary LDPC decoders

In this chapter, we first introduce the notations and terminologies related to binary LDPC decoders that we use throughout this deliverable. Then we briefly review a number of important message-passing decoding algorithms.

2.1 Definitions

In figure 2.1, we depict a simple communication system. We assume that the source produces a vector $\boldsymbol{s} = (s_1, s_2, ..., s_K)$. The encoder adds redundancy to $\boldsymbol{s}$ in order to obtain an encoded vector $\boldsymbol{x} = (x_1, x_2, ..., x_N)$, which is a codeword, and which is mapped by *e.g.* a binary phase-shift keying (BPSK) modulation, to obtain $\boldsymbol{w} = (w_1, w_2, ..., w_N)$. After $\boldsymbol{w}$ is sent through the noisy channel. Based on the channel outputs $\boldsymbol{y} = (y_1, y_2, ..., y_N)$, the decoder produces the vector $\hat{\boldsymbol{x}} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_N)$ which is an estimation of $\boldsymbol{x}$. To check if $\hat{\boldsymbol{x}}$ is a valid codeword, we verify that the syndrome vector is all-zero, i.e. $H\hat{\boldsymbol{x}}^T = 0$, where H is the sparse parity-check matrix.

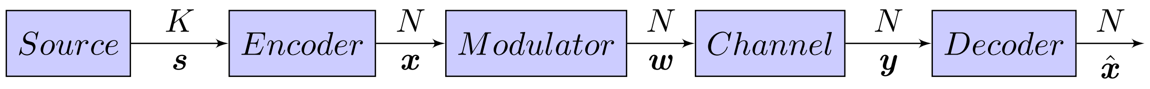


Figure 2.1: A simple communication system

We denote the channel output alphabet by $\mathcal{A}_y$, while the channel input alphabet is denoted $\mathcal{A}_x$. For binary LDPC decoders we have $\boldsymbol{y} \in \mathcal{A}_y^N$, $\boldsymbol{s} \in \{0,1\}^K$, $\boldsymbol{x} \in \{0,1\}^N$ and $\hat{\boldsymbol{x}} \in \{0,1\}^N$.

The channel output alphabet depends on the channel model. We consider in this deliverable two models of binary memoryless channels: the first one, the Binary Symmetric Channel (BSC); and the second one, the Binary-Input Additive White Gaussian Noise (BI-AWGN) channel [18, 19].

Binary Symmetric Channel

In the BSC, a bit transmitted $x_n \in \{0,1\}$ is flipped to y_n with probability ϵ, referred to as the error probability or crossover probability of the channel, hence $y_n \in \mathcal{A}_y = \{0,1\}$.

The BSC satisfies the following symmetry condition

$$p(y_n = \xi \mid x_n = 0) = p(y_n = -\xi \mid x_n = 1) \qquad \xi \in \{0,1\} \tag{2.1}$$

Where $p(y \mid x)$ is the likelihood (channel transition probability).

Binary-Input Additive White Gaussian Noise channel

The BI-AWGN channel is modeled by $y_n = (1 - 2x_n) + z_n$, where $1 - 2x_n \in \{+1, -1\}$, and z_n is a sequence of independent and identically distributed (i.i.d.) random variables with probability density function given by the normal (or Gaussian) distribution.

$$p_{BIAWGN}(z_n) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z_n)^2/2\sigma^2} \tag{2.2}$$

Where σ^2 is the noise variance.

Similarly to the BSC, the BI-AWGN channel satisfies the following symmetry condition

$$p(y_n = \xi \mid x_n = 0) = p(y_n = -\xi \mid x_n = 1), \xi \in \mathcal{A}_y \tag{2.3}$$

Where $p(y \mid x)$ defines the likelihood distribution.

Log-Likelihood Ratio

A log-likelihood ratio (LLR) form for the bit x_n with probability $p(x_n)$ is defined by

$$L(x_n) = \log\left(\frac{p(x_n = 0)}{p(x_n = 1)}\right). \tag{2.4}$$

Where $p(x_n = 0) + p(x_n = 1) = 1$. If $p(x_n = 0) > p(x_n = 1)$ then $L(x_n)$ is positive; if the inequality es reversed, then $L(x_n)$ is negative. Therefore, the sign of $L(x_n)$ indicates the value of the bit x_n ($x_n = (1-\text{sign}(L(x_n)))/2$), and $|L(x_n)|$ give us a measure of its reliability. We consider that the notation log denotes the logarithm with base e in the rest of the text.

The channel output can be also expressed in a LLR form as follows

$$L(y_n) = \log\left(\frac{\Pr(y_n \mid x_n = 0)}{\Pr(y_n \mid x_n = 1)}\right). \tag{2.5}$$

In the case of the BSC, we get

$$L(y_n) = (1 - 2y_n)\log\left(\frac{1-\epsilon}{\epsilon}\right), \tag{2.6}$$

and for the BI-AWGN channel with noise variance σ^2, we have

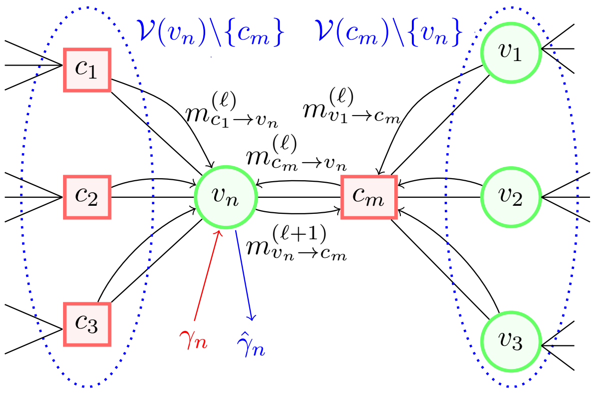
$$L(y_n) = \frac{2}{\sigma^2} y_n. \tag{2.7}$$

2.2 Message-Passing decoding algorithms

Message-Passing (MP) decoding algorithms are iterative algorithms that use a Tanner graph to pass messages along the edges, these algorithms compute new messages at each new iteration. Considering any VN v and any CN c, messages passed on the edge $\{v, c\}$, either from VN to CN, or vice versa, are computed as a function of all incoming messages, except the message passed on the edge $\{v, c\}$.

Here we present the notations used to describe different algorithms. We consider that the message alphabet is $\mathcal{M}$. Let $\ell \in \mathbb{N}$, denote the number of iterations. Let $m^{(\ell)}_{v\to c} \in \mathcal{M}$, denotes the message sent from VN v to CN c in the ℓth iteration. Let $m^{(\ell)}_{c\to v} \in \mathcal{M}$, denotes the message sent from CN c to VN v in the ℓth iteration. Let $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_N)$ denote the *a posteriori* probability, where γ_n is associated to a VN v_n, $n = 1, 2, ..., N$, note that often $\gamma_n \in \mathcal{M}$ but it is a loss of generalisation. Also, let $\mathcal{V}(v_n)$ denote the set of neighbors of a VN v_n in a Tanner graph, and let $\mathcal{V}(c_m)$ denote the set of neighbors of a CN c_m in a Tanner graph, these neighborhood definitions are required to explain MP decoding algorithms.

Fig. 2.2 depicts a Tanner graph fragment, showing the flow of messages in MP decoding algorithms. From this figure we have $\mathcal{V}(v_n) = \{c_m, c_1, c_2, c_3\}$ and $\mathcal{V}(c_m) = \{v_n, v_1, v_2, v_3\}$. At iteration zero $m^{(0)}_{v_n\to c_m} = L(y_n)$ because all the messages inside the Tanner graph are initialized to zero. At each iteration $m^{(\ell+1)}_{v_n\to c_m}$, $m^{(\ell)}_{c_m\to v_n}$ and γ_n are computed, and an estimation of the bit transmitted x_n is obtained from γ_n.



$\mathcal{V}(v_n)\backslash\{c_m\}$
$\mathcal{V}(c_m)\backslash\{v_n\}$
c_1
c_2
c_3
v_n
c_m
v_1
v_2
v_3
$m^{(\ell)}_{c_1\to v_n}$
$m^{(\ell)}_{v_1\to c_m}$
$m^{(\ell)}_{c_m\to v_n}$
$m^{(\ell+1)}_{v_n\to c_m}$
γ_n
$\hat{\gamma}_n$

Figure 2.2: A Tanner graph fragment

In MP decoding algorithms, a VN v_n sends its message to its neighbors $\mathcal{V}(v_n)$. Similarly a CN c_m sends its message to its neighbors $\mathcal{V}(c_m)$. In each iteration, the VN update (VNU) and CN update (CNU) compute outgoing messages from all incoming message.

In this deliverable, the messages will be described in the log-likelihood ratio (LLR) domain. The sign of a message represents the hard-decidion value of the VN it is connected to, and the absolute value of the message represents its reliability. As a consequence, the message alphabet $\mathcal{M}$ has to be symmetric around O. For example $\mathcal{M} = \mathbb{R}$ for a continuous alphabet and $\mathcal{M} = \{-N_q, \dots, 0, \dots, +N_q\}$ for a discrete alphabet. The LLR from the channel $L(y_n)$ is usually referred to as the *intrinsic* message for the VN v_n. The exchanged messages in the decoder are usually referred to as *extrinsic* messages. For a successful decoding, the measure of the reliability of *extrinsic* messages becomes more and more reliable at each new iteration.

We now define update functions for the VNU and the CNU. Let $\Psi_v : \mathcal{A}_x \times \mathcal{M}^{(d_v-1)} \to \mathcal{M}$ denote the function used for the update at a variable-node v of degree d_v. Let $\Psi_c : \mathcal{M}^{(d_c-1)} \to \mathcal{M}$ denote the function used for the update at a check-node c of degree d_c. We can write $\Psi_v : \mathcal{A}_x \to \mathcal{M}$ in the 0th iteration. Let $b_v \in \{\pm 1\}$. The functions Ψ_v and Ψ_c satisfy the following symmetry conditions [10].

$$\Psi_c\left(\left\{b_v m^{(\ell)}_{v\to c_m}\right\}_{v\in\mathcal{V}(c_m)\backslash\{v_n\}}\right) = \Psi_c\left(\left\{m^{(\ell)}_{v\to c_m}\right\}_{v\in\mathcal{V}(c_m)\backslash\{v_n\}}\right)\left(\prod_{v\in\mathcal{V}(c_m)\backslash\{v_n\}} b_v\right) \tag{2.8}$$

$$\begin{aligned} \Psi_v\left(-L(y_n), \left\{-m^{(\ell)}_{c\to v_n}\right\}_{c\in\mathcal{V}(v_n)\backslash\{c_m\}}\right) &= -\Psi_v\left(L(y_n), \left\{m^{(\ell)}_{c\to v_n}\right\}_{c\in\mathcal{V}(v_n)\backslash\{c_m\}}\right), \ell \geq 1 \\ \Psi_v\left(-L(y_n)\right) &= -\Psi_v\left(L(y_n)\right), \ell = 0 \end{aligned} \tag{2.9}$$

We now describe the main steps of MP decoding algorithms:

1. [**Initialization**] the LLR $L(y_n)$ is computed for each VN v_n. Then, variable-to-check messages $m^{(\ell)}_{v_n\to c_m}$ are initialized by $L(y_n)$ at the 0th iteration.

2. [**IterationLoop**] Each decoding iteration consists of the following steps:

   (a) [**CNU**] a check-node computes the outgoing message based on all the incoming messages except the one received from the outgoing message.
   $m^{(\ell)}_{c_m \to v_n} = \Psi_c\left(\left\{m^{(\ell)}_{v \to c_m}\right\}_{v \in \mathcal{V}(c_m)\setminus\{v_n\}}\right).$

   (b) [**VNU**] a variable-node computes the outgoing message from the channel observation and from all the incoming messages except the one which receives the outgoing message.
   $m^{(\ell+1)}_{v_n \to c_m} = \Psi_v\left(L(y_n), \left\{m^{(\ell)}_{c \to v_n}\right\}_{c \in \mathcal{V}(v_n)\setminus\{c_m\}}\right).$

   (c) [**APP − update**] (*a posteriori* probability update) the *a posteriori* probability is computed from the channel observation and from all the incoming messages.
   $\gamma_n = \Psi_v\left(L(y_n), \left\{m^{(\ell)}_{c \to v_n}\right\}_{c \in \mathcal{V}(v_n)}\right)$

   (d) [**hard decision**] makes an estimation of the bits transmitted from the *a posteriori* probability.
   $\hat{x}_n = (1 - \text{sign}(\gamma_n))/2$

   (e) [**syndrome check**] verifies that the syndrome vector is all-zero in order to check if $\hat{\boldsymbol{x}}$ is a valid codeword.
   $H\hat{\boldsymbol{x}}^T = 0.$

The decoding stops when either $[\hat{x}_n]_{n=1,\ldots,N}$ is a codeword or a maximum number of iteration is reached.

In binary LDPC decoders, the MP decoding can be performed either (*i*) using one bit to represent messages (hard-decision MP decoders) or (*ii*) using more than one bit to represent messages (soft-decision MP decoders). We present in the next sections the most common examples of such decoders.

2.2.1 Hard-decision MP decoders

Hard-decision MP decoders use just one bit to represent the messages propagated in the Tanner graph, i.e. $m^{(\ell)}_{v \to c}$ and $m^{(\ell)}_{c \to v} \in \{+1, -1\}$ (which is equivalent to $\{0, 1\}$). Hard-decision decoders are more interesting for the BSC channel than for the AWGN channel, and we restrict in this section the presentation of the decoders to the BSC.

Gallager-B decoder

The Gallager-B algorithm is a MP decoding algorithm [18, 19] with binary alphabet $\mathcal{M} = \{+1, -1\}$. The Gallager-B decoder is described by Algorithm 1. At the initialization step, the LLR from the BSC channel is equal to $L(y_n) = 1 - 2y_n \in \mathcal{M}$, and $m^{(0)}_{v_n \to c_m} = L(y_n)$ at iteration $\ell = 0$.

In each decoding iteration, the CNU computes the check-to-variable messages $m^{(\ell)}_{c_m \to v_n}$ as the parity (in ±1 format) of the incoming messages $m^{(\ell)}_{v \to c_m}$, where $v \in \mathcal{V}(c_m)\setminus\{v_n\}$:

$$m^{(\ell)}_{c_m \to v_n} = \Psi_c\left(\left\{m^{(\ell)}_{v \to c_m}\right\}_{v \in \mathcal{V}(c_m)\setminus\{v_n\}}\right) = \prod_{v \in \mathcal{V}(c_m)\setminus\{v_n\}} m^{(\ell)}_{v \to c_m} \tag{2.10}$$

The VNU computes the variable-to-check messages $m^{(\ell+1)}_{v_n \to c_m}$ by comparing the sum of $L(y_n)$ and

```
Input: $[y_n]_{n=1,\dots,N} \in \{0,1\}^N$,
Output: $[\hat{x}_n]_{n=1,\dots,N} \in \{0,1\}^N$,
Initialization
    for all $\{v_n\}_{n=1,\dots,N}$ do
        $L(y_n) = 1 - 2y_n$;
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do
        $m^{(0)}_{v_n \to c_m} = L(y_n)$;
Iteration Loop
    for all $\{c_m\}_{m=1,\dots,M}$ and $v_n \in \mathcal{V}(c_m)$ do /*CN-update*/
        $m^{(\ell)}_{c_m \to v_n} = \prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} m^{(\ell)}_{v \to c_m}$;
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do /*VN-update*/
        $s_{vc} = L(y_n) + \sum_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} m^{(\ell)}_{c \to v_n}$
        $m^{(\ell+1)}_{v_n \to c_m} = \begin{cases} L(y_n), & \text{if } |s_{vc}| < t \\ \text{sign}(s_{vc}), & \text{otherwise;} \end{cases}$
    for all $\{v_n\}_{n=1,\dots,N}$ do /*APP-update*/
        $s_n = L(y_n) + \sum_{c \in \mathcal{V}(v_n)} m^{(\ell)}_{c \to v_n}$
        $\gamma_n = \begin{cases} L(y_n), & \text{if } s_n = 0 \\ \text{sign}(s_n), & \text{otherwise;} \end{cases}$
    for all $\{v_n\}_{n=1,\dots,N}$ do /*hard decision*/
        $\hat{x}_n = (1 - \gamma_n)/2$;
    if $[\hat{x}_n]_{n=1,\dots,N}$ is a codeword then
        exit the iteration loop;
End Iteration Loop
```

Algorithm 1: Gallager-B decoder

the incoming messages $m^{(\ell)}_{c \to v_n}$ to a pre-determined threshold t:

$$\begin{aligned} s_{vc} &= L(y_n) + \sum_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} m^{(\ell)}_{c \to v_n} \\ m^{(\ell+1)}_{v_n \to c_m} &= \begin{cases} L(y_n), & \text{if } |s_{vc}| < t \\ \text{sign}(s_{vc}), & \text{otherwise;} \end{cases} \end{aligned} \tag{2.11}$$

The *a posteriori* probability γ_n is calculated from the sign of s_n which is the sum of $L(y_n)$ and all incoming messages $m^{(\ell)}_{c \to v_n}$, where $c \in \mathcal{V}(v_n)$. If $s_n = 0$ then $\gamma_n = L(y_n)$. The hard decision $[\hat{x}_n]_{n=1,\dots,N} \in \{0,1\}^N$ is defined as the binary equivalent of $[\gamma_n]_{n=1,\dots,N} \in \{+1,-1\}^N$, i.e. $\hat{x}_n = (1-\gamma_n)/2$. The decoder stops if either $[\hat{x}_n]_{n=1,\dots,N}$ is a codeword or a maximum number of iteration is reached.

To improve the performance of the Gallager-B decoder, the threshold t can be optimized, could take different values at each iteration, and may vary from a VNU to another. However, the value of t is most of the time considered as constant. Two examples of Gallager-B decoders are given in the next subsections, and we can refer to [18, 19] for more details.

Gallager-A decoder

The Gallager-A decoder can be seen as a particular case of the Gallager-B decoder. In the Gallager-A decoder, the threshold t is equal to $deg(v_n) - 2$, where $deg(v_n)$ is the degree of the variable-node v_n. Note that for regular LDPC codes, t is a constant value.

Majority-Voting decoder

The Majority-Voting decoder can be obtained from the Gallager-B decoder. In this case the threshold t is equal to 1. Hence, the update rule at a variable-node is given by

$$m_{v_n \to c_m}^{(\ell+1)} = \Psi_v \left(L(y_n), \left\{ m_{c \to v_n}^{(\ell)} \right\}_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} \right) = \begin{cases} L(y_n), & \text{if } s_{vc} = 0 \\ \operatorname{sign}(s_{vc}), & \text{otherwise;} \end{cases}$$

If s_{vc} is equal to zero, $m_{v_n \to c_m}^{(\ell+1)}$ is equal to $L(y_n)$; otherwise, $m_{v_n \to c_m}^{(\ell+1)}$ is the sign of s_{vc}.

Gallager-B decoder with extended alphabet (erasure decoder)

The message alphabet for Gallager-B with extended alphabet is $\mathcal{M} = \{-1, 0, 1\}$. The value 0 is added to the alphabet to deal with the ties in the VNU update of the Gallager-B and propagate 0 instead of the likelihood in such case. This decoder is also named *erasure decoder*. Strictly speaking this decoder is not a hard decision decoder, and a least two bits of precision must be used in the hardware implementation. Algorithm 2 presents the details of the erasure decoder.

```
... same as Gallager-B decoder
Iteration Loop
    ... same as Gallager-B decoder
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do /*VN-update*/
        $m_{v_n \to c_m}^{(\ell+1)} = \operatorname{sign}\left( L(y_n) + \sum_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} m_{c \to v_n}^{(\ell)} \right);$
... same as Gallager-B decoder
End Iteration Loop
```

Algorithm 2: Erasure decoder

Some authors have proposed to improve the performance of the erasure decoder by weighting differently the channel value and the extrinsic contribution. Refer to [18, 19] for more details.

2.2.2 Soft-decision MP decoders

Soft-decision MP decoders use more than one bit to represent messages in the Tanner graph. These decoders propagate more information than hard-decision decoders, and they usually work in the LLR domain using real valued message, i.e. $m_{v \to c}^{(\ell)}$ and $m_{c \to v}^{(\ell)} \in \mathbb{R}$. The performance of soft-decision MP decoders is much better than the performance of the hard-decision MP decoders, but their hardware implementation is more complex. The sum-product algorithm, the Min-Sum (MS) algorithm are some examples of algorithms used for soft-decision MP decoders.

Belief Propagation decoder

One important kind of message-passing decoding algorithm is the sum-product algorithm [20], also called Belief-Propagation (BP) algorithm. This algorithm is commonly used in different applications like artificial intelligence, information theory, etc.

For the BP decoder presented in this section, we consider that the message alphabet is continuous, i.e. $\mathcal{M} = \mathbb{R}$, and the channel output is discrete or continuous. For the BSC, the channel likelihood is given by $L(y_n) = (1 - 2y_n) \log \left(\frac{1 - \epsilon}{\epsilon} \right)$, and for the BI-AWGN channel, the LLR is equal to $L(y_n) = 2 \frac{y_n}{\sigma^2}$.

The update rule at a variable-node is given by

$$m_{v_n \to c_m}^{(\ell+1)} = \Psi_v \left(L(y_n), \left\{ m_{c \to v_n}^{(\ell)} \right\}_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} \right) = L(y_n) + \sum_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} m_{c \to v_n}^{(\ell)} \tag{2.12}$$

And the update rule at a check-node is given by

$$m_{c_m \to v_n}^{(\ell)} = \Psi_c \left(\left\{ m_{v \to c_m}^{(\ell)} \right\}_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \right) = \log \left(\frac{1 + \prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \tanh \frac{m_{v \to c_m}^{(\ell)}}{2}}{1 - \prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \tanh \frac{m_{v \to c_m}^{(\ell)}}{2}} \right) \tag{2.13}$$

Considering the function

$$\Phi(x) = \log \left(\tanh \frac{x}{2} \right) = \log \left(\frac{1 + e^{-x}}{1 - e^{-x}} \right), \forall x > 0,$$

equation (2.13) can be rewritten as:

$$\Psi_c \left(\left\{ m_{v \to c_m}^{(\ell)} \right\}_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \right) = \left(\prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \text{sign} \left(m_{v \to c_m}^{(\ell)} \right) \right) . \Phi \left(\sum_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \Phi \left(\left| m_{v \to c_m}^{(\ell)} \right| \right) \right) \tag{2.14}$$

The BP decoder is quite tedious to implement because of the function Φ used to compute check-to-variable messages. Algorithm 3 describes the BP decoder.

```
Input: $[y_n]_{n=1,\dots,N} \in \mathcal{A}_y^N$,
Output: $[\hat{x}_n]_{n=1,\dots,N} \in \{0,1\}^N$,
Initialization
    for all $\{v_n\}_{n=1,\dots,N}$ do
        $L(y_n) = \log \left( \frac{\Pr(y_n \mid x_n = 0)}{\Pr(y_n \mid x_n = 1)} \right)$;
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do
        $m_{v_n \to c_m}^{(0)} = L(y_n)$;
Iteration Loop
    for all $\{c_m\}_{m=1,\dots,M}$ and $v_n \in \mathcal{V}(c_m)$ do /*CN-update*/
        $m_{c_m \to v_n}^{(\ell)} = \left( \prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \text{sign} \left( m_{v \to c_m}^{(\ell)} \right) \right) . \Phi \left( \sum_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \Phi \left( \left| m_{v \to c_m}^{(\ell)} \right| \right) \right)$;
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do /*VN-update*/
        $m_{v_n \to c_m}^{(\ell+1)} = L(y_n) + \sum_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} m_{c \to v_n}^{(\ell)}$;
    for all $\{v_n\}_{n=1,\dots,N}$ do /*APP-update*/
        $\gamma_n = L(y_n) + \sum_{c \in \mathcal{V}(v_n)} m_{c \to v_n}^{(\ell)}$;
    for all $\{v_n\}_{n=1,\dots,N}$ do /*hard decision*/
        $\hat{x}_n = (1 - \text{sign}(\gamma_n))/2$;
    if $[\hat{x}_n]_{n=1,\dots,N}$ is a codeword then
        exit the iteration loop;
End Iteration Loop
```

Algorithm 3: Belief-Propagation (BP) decoder

Min-Sum decoder

The Min-Sum (MS) decoder is derived from the BP decoder. The MS decoder reduces the computational complexity at the CNU and is less sensitive than BP decoder to message quantization effects.

In the MS decoder, the update rule for the VNU is the same as the BP decoder, equation (2.12). To obtain the update rule at the CNU, the following relation is used

$$\Phi\left(\Phi(a)+\Phi(b)\right) \leq \min(a,b), \qquad \forall a>0 \text{ and } b>0 \tag{2.15}$$

Using this relation, one could replace the CNU of the BP (2.14) by:

$$\Psi_c\left(\left\{m^{(\ell)}_{v\to c_m}\right\}_{v\in\mathcal{V}(c_m)\setminus\{v_n\}}\right) = \left(\prod_{v\in\mathcal{V}(c_m)\setminus\{v_n\}} \text{sign}\left(m^{(\ell)}_{v\to c_m}\right)\right)\cdot\left(\min_{v\in\mathcal{V}(c_m)\setminus\{v_n\}}\left(\left|m^{(\ell)}_{v\to c_m}\right|\right)\right) \tag{2.16}$$

Algorithm 4 describes the MS decoder [19].

```
Input: $[y_n]_{n=1,\dots,N} \in \mathcal{A}_y^N$,
Output: $[\hat{x}_n]_{n=1,\dots,N} \in \{0,1\}^N$,
Initialization
    for all $\{v_n\}_{n=1,\dots,N}$ do
        $L(y_n) = \log\left(\frac{\Pr(y_n \mid x_n = 0)}{\Pr(y_n \mid x_n = 1)}\right)$;
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do
        $m^{(0)}_{v_n\to c_m} = L(y_n)$;
Iteration Loop
    for all $\{c_m\}_{m=1,\dots,M}$ and $v_n \in \mathcal{V}(c_m)$ do /*CN-update*/
        $m^{(\ell)}_{c_m\to v_n} = \left(\prod_{v\in\mathcal{V}(c_m)\setminus\{v_n\}} \text{sign}\left(m^{(\ell)}_{v\to c_m}\right)\right)\cdot\left(\min_{v\in\mathcal{V}(c_m)\setminus\{v_n\}}\left(\left|m^{(\ell)}_{v\to c_m}\right|\right)\right)$;
    for all $\{v_n\}_{n=1,\dots,N}$ and $c_m \in \mathcal{V}(v_n)$ do /*VN-update*/
        $m^{(\ell+1)}_{v_n\to c_m} = L(y_n) + \sum_{c\in\mathcal{V}(v_n)\setminus\{c_m\}} m^{(\ell)}_{c\to v_n}$;
    for all $\{v_n\}_{n=1,\dots,N}$ do /*APP-update*/
        $\gamma_n = L(y_n) + \sum_{c\in\mathcal{V}(v_n)} m^{(\ell)}_{c\to v_n}$;
    for all $\{v_n\}_{n=1,\dots,N}$ do /*hard decision*/
        $\hat{x}_n = (1 - \text{sign}(\gamma_n))/2$;
    if $[\hat{x}_n]_{n=1,\dots,N}$ is a codeword then
        exit the iteration loop;
End Iteration Loop
```

Algorithm 4: Min-Sum (MS) decoder

Min-Sum-based decoders

There are several Min-Sum-based decoders proposed in the literature which have been proposed to improve the performance of the MS decoder. We introduce briefly three of them.

(i) The Normalized-Min-Sum (NMS) decoder: in this decoder a factor $\lambda \in]0,1[$ is used to weight the messages at the output of the CN update. Since the relation (2.15) leads to a systematic over-estimation of the amplitude of the CNU output message, it makes sense to shrink it with $\lambda \in]0,1[$. λ could be fixed to a constant value, or vary according to the check-node degree. This factor can be optimized by Monte-Carlo simulation, or using density evolution analysis. Algorithm 5 describes the difference with the MS decoder presented in algorithm 4.

```
... same as MS decoder
Iteration Loop
    for all $\{c_m\}_{m=1,\ldots,M}$ and $v_n \in \mathcal{V}(c_m)$ do /*CN-update*/
      $m^{(\ell)}_{c_m \to v_n} = \lambda . \left( \prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \text{sign}\left(m^{(\ell)}_{v \to c_m}\right) \right) . \left( \min_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \left( \left| m^{(\ell)}_{v \to c_m} \right| \right) \right)$;
... same as MS decoder
End Iteration Loop
```

Algorithm 5: Normalized-Min-Sum (NMS) decoder

(ii) The Offset-Min-Sum (OMS) decoder: a variant of the NMS is proposed by using an offset $\lambda > 0$ to diminish the message amplitude at the output of the CNU. As in the NMS decoder, the offset λ has the objective of compensating the over-estimation of the MS outputs. λ can be a constant value, or vary according to the check-node degree, and can be optimized by Monte-Carlo simulation, or using density evolution analysis.

```
... same as MS decoder
Iteration Loop
    for all $\{c_m\}_{m=1,\ldots,M}$ and $v_n \in \mathcal{V}(c_m)$ do /*CN-update*/
      $m^{(\ell)}_{c_m \to v_n} = \left( \prod_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \text{sign}\left(m^{(\ell)}_{v \to c_m}\right) \right) . \max \left\{ \left( \min_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \left( \left| m^{(\ell)}_{v \to c_m} \right| \right) \right) - \lambda, 0 \right\}$;
... same as MS decoder
End Iteration Loop
```

Algorithm 6: Offset-Min-Sum (OMS) decoder

(iii) The Self-Corrected Min-Sum (SCMS) decoder: this decoder modifies the VNU of the MS decoder, and uses a 1-bit extra memory to store the sign of the previous variable-to-check message, i.e sign($m^{(\ell)}_{v_n \to c_m}$). The current variable-to-check message $m^{(\ell+1)}_{v_n \to c_m}$ is updated according to Algorithm 7. This algorithm helps the decoder to detect oscillating behaviors in the decoder and propagates an erasure to break the oscillation [19].

```
... same as MS decoder
Iteration Loop
    ... same as MS decoder
    for all $\{v_n\}_{n=1,\ldots,N}$ and $c_m \in \mathcal{V}(v_n)$ do /*VN-update*/
      $m^{(\ell+1)}_{v_n \to c_m} = L(y_n) + \sum_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} m^{(\ell)}_{c \to v_n}$;
      if sign $\left(m^{(\ell+1)}_{v_n \to c_m}\right) \neq$ sign $\left(m^{(\ell)}_{v_n \to c_m}\right)$ and $m^{(\ell+1)}_{v_n \to c_m} \neq 0$
        $m^{(\ell+1)}_{v_n \to c_m} = 0$;
      end if
... same as MS decoder
End Iteration Loop
```

Algorithm 7: Self-Corrected Min-Sum (SCMS) decoder

2.3 Conclusions

In this chapter we have presented a variety of Message-Passing decoding algorithms which some of them can be implemented with small hardware resources, whereas other do not. Hard-decision MP decoders have a low cost of hardware implementation because the message alphabet only needs one bit, but the performance of these decoders is less good than for soft-decision MP decoders. Hard decision decoders can be used in systems which do not require high decoding performance.

Soft-decision MP decoders have the message alphabet equal to the real numbers, for these reason, these decoders are more complex to be implemented in hardware. The BP decoder has high computational complexity due to the function Φ, but has high decoding performance. In the class of soft-decision MP decoders, the Min-Sum decoder and Min-Sum-based decoders have lower computational complexity than the BP decoder, with a slight, sometimes negligible, performance loss.

For soft-decision MP decoders, in order to make a hardware implementation, the message alphabet and the channel output alphabet have to be discrete and finite.

We present in the next chapter the Min-Sum-based decoders using quantized messages, and describe the density evolution tools used to analyze them.

Chapter 3

Quantized Min-Sum Decoders and Density Evolution

This chapter is dedicated to the presentation of the main theoretical tool that is used to analyze the performance of LDPC codes and decoders, called Density Evolution (DE). The concept of DE is to track the evolution of the probability mass function of the messages during the iterations of LDPC decoders. Although DE has been introduced as a theoretical approach, it turns out to be a very efficient tool to predict the performance of LDPC decoders in the waterfall region. This is especially true when the DE can follow the exact density of the messages (under the independance assumption), which is the case of quantized decoders, when the message alphabet is small enough.

From now on, and throughout the rest of the deliverable, we assume that the message alphabet is finite, composed of $Ns = 2N_q + 1$ states, with $N_q = 2^{(q-1)} - 1$. The message alphabet is denoted by $\mathcal{M} = \{-N_q, -(N_q - 1), ..., -1, 0, +1, ..., +(N_q - 1), +N_q\}$, i.e. the messages are quantized on q bits.

3.1 Channel Value Quantization

3.1.1 Quantization for the Binary Symmetric Channel

According to Chapter 2, for the BSC we have $L(y_n) = (1 - 2y_n)\log((1 - \epsilon)/\epsilon) \in \mathbb{R}$, with $y_n \in \mathcal{A}_y = \{0, 1\}$. For a given BSC probability ϵ, the decoder input alphabet is composed of two values in $\mathcal{A}_x = [+|L(y_n)|, -|L(y_n)|]$. Figure 3.1 shows the LLR $L(y_n)$ as a function of the cross-over probability ϵ, we can see that $L(y_n)$ is a real number.

As we deal with quantized decoders with message alphabet $\mathcal{M}$, we can consider without loss of generality that $\mathcal{A}_x \subseteq \mathcal{M}$. In the sequel, we denote by C the *channel value* which is a positive integer, and consider that $|L(y_n)|$ is mapped to C. As a result, the decoder input alphabet becomes $\mathcal{A}_x = \{+C, -C\}$, with $C \in \{+1, +2, ..., +N_q\}$. In other words $y_n = 0$ is mapped to $+C$ and $y_n = 1$ is mapped to $-C$. The value of C can be seen as an extra degree of freedom in the decoder definition. For example, a MS decoder with $C = 1$ and a MS decoder with $C = 2$ are interpreted as two different decoders. C can be optimized in order to improve the decoder performance.

3.1.2 Quantization for the Binary-Input Additive White Gaussian Noise channel

According to Chapter 2, for the BI-AWGN channel we have $L(y_n) = 2\,y_n/\sigma^2$, with $y_n \in \mathcal{A}_y = \mathbb{R}$. In the initialization step, variable-to-check messages are initialized by integer numbers in quantized decoders, hence, $L(y_n)$ has to be quantized on θ bits.

We denote by $\mathcal{Q} : \mathbb{R} \to \mathbb{Z}$, the quantizer, defined by:

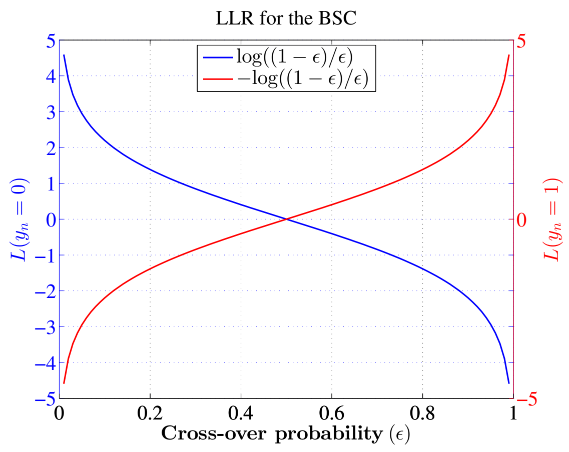


Figure 3.1: $L(y_n)$ for the BSC.

$$\mathcal{Q}\left(L(y_n)\right) = \left\lfloor L(y_n)\frac{2^{(\theta-1)}-1}{\alpha} + 0.5 \right\rfloor, L(y_n) \in \mathbb{R} \tag{3.1}$$

where the notation $\lfloor . \rfloor$ depicts the floor function, θ denotes the numbers of bits used to quantize $L(y_n)$.

The decoder input alphabet is $\mathcal{A}_x = \mathbb{Z}$ because $\mathcal{Q}\left(L(y_n)\right)$ is an integer. The parameter α in equation (3.1) is a factor used to enlarge or decrease the standard deviation of quantized values. Figure 3.2 shows the $f\left(\mathcal{Q}(L(y_n))\right)$ as a function of $\mathcal{Q}\left(L(y_n)\right)$, the quantized values presented in figure 3.2a were obtained considering $\theta = 4$, $\sigma = 0.8$, and $\alpha = 2$; whereas $\theta = 4$, $\sigma = 0.8$, and $\alpha = 8$ were used to compute the quantized values shown in figure 3.2b.

We can see that if α is large, most of the quantized values are close to zero, see figure 3.2b. On the other hand, for a small value of α, the quantized values are spread along a wider interval, see figure 3.2a.

Similar to C for the BSC channel, α represents a degree of freedom in the decoder definition that can be analyzed and optimized for quantized decoders on the BI-AWGN channel.

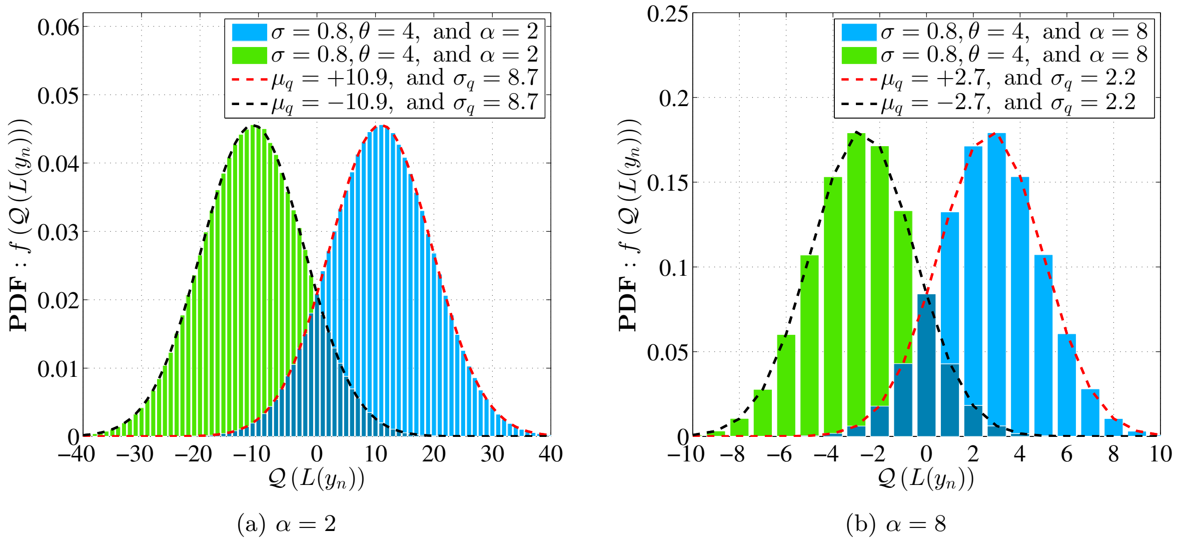


Figure 3.2: PDF of quantized values using $\theta = 4$ and $\sigma = 0.8$

Saturation

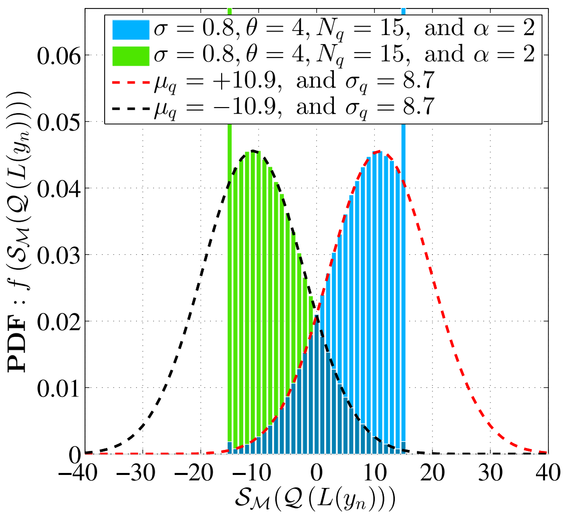
We denote by $\mathcal{S}_{\mathcal{M}} : \mathbb{Z} \to \mathcal{M}$, the saturation, defined by:

$$\mathcal{S}_{\mathcal{M}}(\beta) = \begin{cases} -N_q & \text{if} \quad \beta < -N_q \\ \beta & \text{if} \quad \beta \in \mathcal{M} \\ +N_q & \text{if} \quad \beta > +N_q \end{cases} \tag{3.2}$$

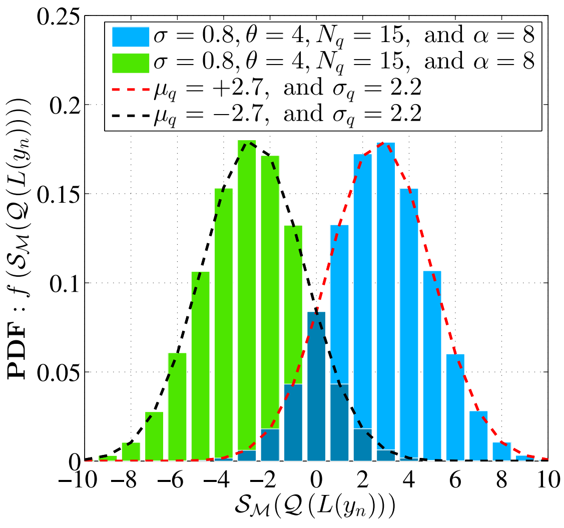
Assumptions for the AWGN channel

We consider that $\mathcal{A}_x \subseteq \mathcal{M}$. This assumption is a loss of generalization, but generalizations to $\mathcal{M} \subseteq \mathcal{A}_x$ could be derived easily from the analysis presented in this chapter.

To obtain quantized values belonging to $\mathcal{M}$ for the BI-AWGN channel, $L(y_n)$ has to be quantized and saturated, i.e. $\mathcal{S}_{\mathcal{M}}(\mathcal{Q}(L(y_n)))$. Distributions of quantized and saturated values presented in figure 3.3 are obtained from the quantized values shown in figure 3.2, doing the saturation with $N_q = 15$. Note that if α is too small, most of quantized values are saturated to N_q.



(a) $\alpha = 2$, $N_q = 15$



(b) $\alpha = 8$, $N_q = 15$

Figure 3.3: Quantization with saturation of $L(y_n)$ using $q = 5$, $\theta = 4$, and $\sigma = 0.8$.

3.2 Quantized Min-Sum-Based Decoders

3.2.1 Notations

We keep the notations presented in Chapter 2, i.e. $\ell \in \mathbb{N}$ denotes the number of iterations, $m^{(\ell)}_{v \to c} \in \mathcal{M}$ denotes the message sent from VN v to CN c in the ℓth iteration, $m^{(\ell)}_{c \to v} \in \mathcal{M}$ denotes the message sent from CN c to VN v in the ℓth iteration. Also, $\mathcal{V}(v_n)$ is the set of neighbors of a VN v_n in a Tanner graph, and $\mathcal{V}(c_m)$ is the set of neighbors of a CN c_m in a Tanner graph.

Following the definitions of the VNU and CNU presented in Chapter 2, in this section we present the discrete update functions for quantized Min-Sum-Based decoders. $\Psi_v : \mathcal{A}_x \times \mathcal{M}^{(d_v-1)} \to \mathcal{M}$ denotes the discrete function used for the update at a variable-node v of degree d_v, and $\Psi_c : \mathcal{M}^{(d_c-1)} \to \mathcal{M}$ denotes the discrete function used for the update at a check-node c of degree d_c. The functions Ψ_v and Ψ_c also satisfy the symmetry conditions presented in equations (2.8) and (2.9), respectively.

We also consider that $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_N)$ denote the *a posteriori* probability, where $\gamma_n \in \mathcal{M}$ is associated to a VN v_n. Also, $\boldsymbol{L} = (L(y_1), L(y_2), ..., L(y_N))$ denote the likelihood vector. The likelihoods are computed as explained in the previous sections, *i.e.* $L(y_n) = \pm C$ for the BSC, and $L(y_n) = \mathcal{S}_{\mathcal{M}}(\mathcal{Q}(L(y_n)))$ for the BI-AWGN channel.

3.2.2 Update Rules

Two quantized decoders are considered, a quantized Min-Sum decoder and a quantized offset Min-Sum decoder with offset value $\lambda \in \{1, ..., (N_q - 2)\}$. The update rule at a check-node (CNU) is the same for the MS and the OMS decoders, and is given by

$$\begin{aligned} m^{(\ell)}_{c_m \to v_n} &= \Psi_c\left(\left\{m^{(\ell)}_{v \to c_m}\right\}_{v \in \mathcal{V}(c_m)\setminus\{v_n\}}\right) \\ &= \left(\prod_{v \in \mathcal{V}(c_m)\setminus\{v_n\}} \operatorname{sign}\left(m^{(\ell)}_{v \to c_m}\right)\right) \cdot \left(\min_{v \in \mathcal{V}(c_m)\setminus\{v_n\}} \left(\left|m^{(\ell)}_{v \to c_m}\right|\right)\right). \end{aligned} \tag{3.3}$$

As for the update rules at a variable-node (VNU), we can use the same expression for both MS and OMS decoders, since OMS with offset value $\lambda = 0$ becomes the MS decoder. The VNU expression is given by:

$$m^{(\ell+1)}_{v_n \to c_m} = \Psi_v\left(L(y_n), \left\{m^{(\ell)}_{c \to v_n}\right\}_{c \in \mathcal{V}(v_n)\setminus\{c_m\}}\right) = \mathcal{S}_{\mathcal{M}}\left(\Lambda\left(L(y_n) + \sum_{c \in \mathcal{V}(v_n)\setminus\{c_m\}} m^{(\ell)}_{c \to v_n}\right)\right). \tag{3.4}$$

Where the function $\Lambda(s_{vc})$ is defined by

$$\Lambda(s_{vc}) = \begin{cases} \min(s_{vc} + \lambda, 0), \text{if} & s_{vc} < 0 \\ \max(s_{vc} - \lambda, 0), \text{if} & s_{vc} \geq 0 \end{cases} \tag{3.5}$$

and

$$s_{vc} = L(y_n) + \sum_{c \in \mathcal{V}(v_n)\setminus\{c_m\}} m^{(\ell)}_{c \to v_n}. \tag{3.6}$$

The *a posteriori* probability (APP) update at a variable-node is given by

$$\gamma_n = \mathcal{S}_{\mathcal{M}}\left(L(y_n) + \sum_{c \in \mathcal{V}(v_n)} m^{(\ell)}_{c \to v_n}\right) = \mathcal{S}_{\mathcal{M}}\left(s_{vc} + m^{(\ell)}_{c_m \to v_n}\right). \tag{3.7}$$

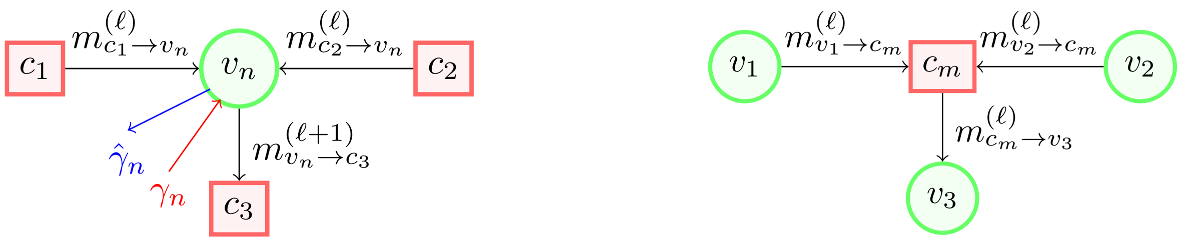
And the decision on the estimated bits $\hat{x}_n$ is taken according to the sign of γ_n.

As we can see, the is not much difference with the decoders presented in chapter 2, except that the messages belong to a discrete alphabet, and then the update rules require a saturation function $\mathcal{S}_{\mathcal{M}}$.

3.2.3 Look-up Table Representation of Quantized Min-Sum

For quantized decoders, it is sometimes convenient to use a look-up table (LUT) or Boolean map representation, to illustrate more clearly their behaviors. This will be for example helpful for the interpretation of noisy MS decoders.

Let us take the example of a decoder quantized on $q = 3$ bits, with $N_q = 3$, and $C = +1$, and the simplest case of VNU and CNU with connexion degrees $d_v = 3$ and $d_c = 3$, as depicted on figures 3.4a and 3.4. The message alphabet is $\mathcal{M} = \{-3, -2, -1, 0, +1, +2, +3\}$ and $\mathcal{A}_x = \{+1, -1\}$.



c_1
$m^{(\ell)}_{c_1\to v_n}$
v_n
$m^{(\ell)}_{c_2\to v_n}$
c_2
$\hat{\gamma}_n$
γ_n
$m^{(\ell+1)}_{v_n\to c_3}$
c_3
v_1
$m^{(\ell)}_{v_1\to c_m}$
c_m
$m^{(\ell)}_{v_2\to c_m}$
v_2
$m^{(\ell)}_{c_m\to v_3}$
v_3

(a) Variable-node v_n and its neighbors $\mathcal{V}(v_n) = \{c_1, c_2, c_3\}$

(b) Check-node c_m and its neighbors $\mathcal{V}(c_m) = \{v_1, v_2, v_3\}$

Figure 3.4: The VN v_n and the CN c_m used to obtain the LUT representations for the functions Ψ_v and Ψ_c.

$m^{(\ell)}_{v_2\to c_m} \setminus m^{(\ell)}_{v_1\to c_m}$	-3	-2	-1	0	$+1$	$+2$	$+3$
-3	$+3$	$+2$	$+1$	0	-1	-2	-3
-2	$+2$	$+2$	$+1$	0	-1	-2	-2
-1	$+1$	$+1$	$+1$	0	-1	-1	-1
0	0	0	0	0	0	0	0
$+1$	-1	-1	-1	0	$+1$	$+1$	$+1$
$+2$	-2	-2	-1	0	$+1$	$+2$	$+2$
$+3$	-3	-2	-1	0	$+1$	$+2$	$+3$

Table 3.1: LUT representation of $\Psi_c(m^{(\ell)}_{v_1\to c_m}, m^{(\ell)}_{v_2\to c_m})$.

Table 3.1 shows the LUT representation of the discrete function Ψ_c (equation (3.3)), for a CN c_m connected to three variable-nodes v_1, v_2, and v_3. The output message v_3 is computed from v_1 and v_2 according to $m^{(\ell)}_{c_m\to v_3} = \Psi_c(m^{(\ell)}_{v_1\to c_m}, m^{(\ell)}_{v_2\to c_m}) \in \mathcal{M}$.

Table 3.2 shows the LUT representation of the discrete function Ψ_v (equations (3.4)), for a VN v_n connected to three check-nodes c_1, c_2, and c_3. The LUT outputs are computed using $m^{(\ell+1)}_{v_n\to c_3} = \Psi_v(L(y_n), m^{(\ell)}_{c_1\to v_n}, m^{(\ell)}_{c_2\to v_n}) \in \mathcal{M}$.

$L(y_n)$	$+1$							-1						
$m^{(\ell)}_{c_2\to v_n} \setminus m^{(\ell)}_{c_1\to v_n}$	-3	-2	-1	0	$+1$	$+2$	$+3$	-3	-2	-1	0	$+1$	$+2$	$+3$
-3	-3	-3	-3	-2	-1	0	$+1$	-3	-3	-3	-3	-3	-2	-1
-2	-3	-3	-2	-1	0	$+1$	$+2$	-3	-3	-3	-3	-2	-1	0
-1	-3	-2	-1	0	$+1$	$+2$	$+3$	-3	-3	-3	-2	-1	0	$+1$
0	-2	-1	0	$+1$	$+2$	$+3$	$+3$	-3	-3	-2	-1	0	$+1$	$+2$
$+1$	-1	0	$+1$	$+2$	$+3$	$+3$	$+3$	-3	-2	-1	0	$+1$	$+2$	$+3$
$+2$	0	$+1$	$+2$	$+3$	$+3$	$+3$	$+3$	-2	-1	0	$+1$	$+2$	$+3$	$+3$
$+3$	$+1$	$+2$	$+3$	$+3$	$+3$	$+3$	$+3$	-1	0	$+1$	$+2$	$+3$	$+3$	$+3$

Table 3.2: LUT representation of $\Psi_v(L(y_n), m^{(\ell)}_{c_1\to v_n}, m^{(\ell)}_{c_2\to v_n})$.

The LUT representation is convenient only for small values of d_v and for the BSC channel. For $d_v > 3$ or the BI-AWGN channel, the LUTs are multidimensional.

3.3 Density Evolution for Quantized Min-Sum-Based decoders

3.3.1 General Principle of Density Evolution

Density evolution (DE) is a tool which describes the asymptotic behavior of an iterative MP decoder as a dynamical system, and follows the probability mass function (PMF) of the messages in the Tanner graph along the iterations. With DE, one can predict if an ensemble of LDPC codes, parametrized by its degree distributions, decoded with a given MP decoder, converges to zero error probability in the limit of infinite block length. This property gives rise to the definition of a *density evolution threshold* [10, 21, 22].

The DE threshold δ is expressed as a crossover probability ($\delta = \epsilon^*$) for the BSC or as a standard deviation ($\delta = \sigma^*$) for the BI-AWGN channel, with the objective of separating two regions of channel noise parameters. The first region composed of values smaller than δ corresponds to when the DE converges to the zero error probability fixed point. The second region composed of values greater than δ corresponds to when the DE does not converge. In this later case, the DE converges to a fixed point which does not represent the zero error probability. Then the DE threshold can be considered as a point of discontinuity between these two regions.

The value of the threshold δ is then an indicator of whether the couple "LDPC ensemble + MP decoder" is good or not. In particular, the DE threshold can be used to compare different systems and decide which ones are the best, in terms of error correction. The most common utilization of DE in the literature is to compare different LDPC codes ensembles using the BP decoder [16]. It is used for example to optimize the degree distributions of irregular LDPC codes, or to design protograph LDPC ensembles with good thresholds. Another less common utilization of DE is to fix the LDPC ensemble (to the same parameters d_v, d_c for example), and compare the thresholds of different decoders. This can be used for example to optimize the offset value in OMS decoders, or in the case of the NAND project to analyze and optimize the injected noise within the noisy MP decoders.

3.3.2 Density Evolution Recursion

In this section, we describe how to implement DE using the discrete update functions, equations (3.3) and (3.4).

Let $p^{(\ell)}_{ctov}(k)$, $k \in \mathcal{M}$ denote the PMF of check-to-variable messages in the ℓth iteration. Similarly, let $p^{(\ell)}_{vtoc}(k)$, $k \in \mathcal{M}$ denote the PMF of variable-to-check messages in the ℓth iteration. Also, let $p^{(0)}_{vtoc}(k)$, $k \in \mathcal{A}_x$, be the initial PMF of messages sent at $\ell = 0$.

The assumption of infinite block length is useful such that the PMF evolution does not depend on the iteration number. The infinite block length allows to consider that the messages incoming a CNU or a VNU are *independant*, which is a necessary condition to ensure that the function $p^{(\ell+1)}_{vtoc} = function\left(p^{(\ell)}_{vtoc}\right)$ is the same for all iterations ℓ. In DE, we need also to consider that the all-zero codeword is sent over the channel.

Initialization
DE is initialized with the PMF of the channel as follows.

For the BSC with crossover probability ϵ:

$$p^{(0)}_{vtoc}(k) = \begin{cases} 1-\epsilon, & \text{if} \quad k = C \\ \epsilon, & \text{if} \quad k = -C \\ 0, & \quad\ \ \text{otherwise} \end{cases} \tag{3.8}$$

For the BI-AWGN channel with noise variance σ^2

$$p_{vtoc}^{(0)}(k) = \begin{cases} F(k+0.5) & \text{if} \quad k = -N_q \\ F(k+0.5) - F\left((k-1)+0.5\right) & \text{if} \quad -N_q < k < +N_q \\ F(\infty) - F((k-1)+0.5) & \text{if} \quad k = +N_q \end{cases} \tag{3.9}$$

where $F(k)$ is given by [21, 23, 24]:

$$F(k) = \frac{1}{\sqrt{2\pi}\sigma_n} \int_{-\infty}^{k} e^{-(t-\mu_n)^2/2\sigma_n^2} dt \tag{3.10}$$

with $\sigma_n = 2/\sigma$ and $\mu_n = 2/\sigma^2$.

DE update for CNU
To compute the PMF of the output of a check-node of degree d_c, we can decompose the check-node into *elementary check-nodes*. An elementary check-node has only three edges, and its output PMF is computed with only two incoming messages which have the same PMF, because of the independance assumption. The PMF update for an elementary CN is expressed as

$$p_{ctov}^{(\ell)}(k) = \sum_{(i,j):\Psi_c(i,j)=k} p_{vtoc}^{(\ell)}(i)\, p_{vtoc}^{(\ell)}(j), \qquad \forall k \in \mathcal{M} \tag{3.11}$$

This equation is used $d_c - 2$ times in order to compute the PMF of the output of a check-node of degree d_c.

We can note that for a check-node of degree $d_c > 3$, the computational complexity of the direct PMF update would be $(\mathcal{M})^{d_c-1}$, while its implementation using elementary CN updates is $(d_c-2)(\mathcal{M})^2$. This represents a huge complexity reduction, and a large values of d_c is not a limitation to the practical computation of DE.

DE update for VNU
We compute the PMF of the output of a variable-node of degree d_v, using the following relations: For the BSC

$$\begin{aligned} p_{vtoc}^{(\ell+1)}(k) = & \sum_{(i_1,\ldots,i_{d_v-1}):\Psi_v(i_1,\ldots,i_{d_v-1},-C)=k} p_{ctov}^{(\ell)}(i_1) \ldots p_{ctov}^{(\ell)}(i_{d_v}-1)\, p_{vtoc}^{(0)}(-C) + \\ & \sum_{(i_1,\ldots,i_{d_v-1}):\Psi_v(i_1,\ldots,i_{d_v-1},+C)=k} p_{ctov}^{(\ell)}(i_1) \ldots p_{ctov}^{(\ell)}(i_{d_v}-1)\, p_{vtoc}^{(0)}(+C), \qquad \forall k \in \mathcal{M} \end{aligned} \tag{3.12}$$

For the BI-AWGN channel

$$p_{vtoc}^{(\ell+1)}(k) = \sum_{(i_1,\ldots,i_{d_v-1},t):\Psi_v(i_1,\ldots,i_{d_v-1},t)=k} p_{ctov}^{(\ell)}(i_1) \ldots p_{ctov}^{(\ell)}(i_{d_v}-1)\, p_{vtoc}^{(0)}(t), \qquad \forall k \in \mathcal{M} \tag{3.13}$$

For the VNU, we cannot rely on the decomposition in elementary variable-node updates, because Ψ_v cannot be factorized into a sequence of elementary operation. The complexity of DE implementation grows then rapidly with increasing d_v, and becomes a bottleneck of the DE analysis, especially for irregular LDPC codes. A solution that is usually proposed in the litterature is then to make use of the *Gaussian approximation* of DE, also known as the EXIT charts analysis of MP decoders [16]. We will not address the Gaussian approximation of DE in this deliverable, and will limit the illustration of our analysis to regular codes with $d_v\ leq4$.

DE recursion
By combining equations (3.10) and (3.12) for the BSC, or (3.11) and (3.13) for the BI-AWGN, one gets the so called DE recursion, which expresses the evolution of the CtoV messages PMF from one iteration to another. This recursion is then computed iteratively to obtain $p_{vtoc}^{+\infty}$, or $p_{vtoc}^{L_{max}}$ with L_{max} sufficiently large in a practical implementation. It can be shown that when the iteration number grows to infinity, the PMF should converge to a dirac mass at $+\infty$ to characterize a zero error probability [8]. For the case of a quantized MP decoder, the convergence is translated to a dirac mass at the saturation value $+N_q$. In other words, if the PMF converges to

$$\begin{aligned} &p_{vtoc}^{L_{max}}(L_{max}) = 0 \quad && \forall k \in \{-N_q, \ldots, 0, \ldots, +N_q - 1\} \\ &p_{vtoc}^{L_{max}}(L_{max}) = 1 \quad && k = +N_q \end{aligned} \tag{3.14}$$

Then sucessful decoding is declared.

3.4 Asymptotic Bit Error Probability

The asymptotic bit error probability can be deduced from the PMF of the APPs, which is obtained from the DE equations. Let $p_{app}^{(\ell)}(k)$, $k \in \mathcal{M}$ denote the PMF of the APP at the end of the ℓth iteration. To compute $p_{app}^{(\ell)}$ for the BSC we use

$$\begin{aligned} p_{app}^{(\ell)}(k) = & \sum_{(i_1,\ldots,i_{d_v}):\Psi_v(i_1,\ldots,i_{d_v},-C)=k} p_{ctov}^{(\ell)}(i_1) \ldots p_{ctov}^{(\ell)}(i_{d_v})\, p_{vtoc}^{(0)}(-C) + \\ & \sum_{(i_1,\ldots,i_{d_v}):\Psi_v(i_1,\ldots,i_{d_v},+C)=k} p_{ctov}^{(\ell)}(i_1) \ldots p_{ctov}^{(\ell)}(i_{d_v})\, p_{vtoc}^{(0)}(+C), \qquad \forall k \in \mathcal{M} \end{aligned}$$

and for the BI-AWGN channel we use

$$p_{app}^{(\ell)}(k) = \sum_{(i_1,\ldots,i_{d_v},t):\Psi_v(i_1,\ldots,i_{d_v},t)=k} p_{ctov}^{(\ell)}(i_1) \ldots p_{ctov}^{(\ell)}(i_{d_v})\, p_{vtoc}^{(0)}(t), \qquad \forall k \in \mathcal{M}$$

Let $p_e^{(\ell)}$ denote the bit error probability at iteration ℓ. Assuming the transmission of the all-zero codeword, we have

$$p_e^{(\ell)} = \frac{1}{2} p_{app}^{(\ell)}(0) + \sum_{i=-N_q}^{-1} p_{app}^{(\ell)}(i) \tag{3.15}$$

The evolution of $p_e^{(\ell)}$ with the iterations characterizes whether the MP decoder converges or diverges in the asymptotic limit of the codeword length. When the number of iterations ℓ goes to infinity, we obtain the asymptotic error probability $p_e^{(\infty)}$, and when $p_e^{(+\infty)} = 0$, the decoder converges to a zero error probability and successful decoding is declared. Note that this condition is weaker than condition (3.14) presented in the previous section, but because of the properties of the functions Ψ_v and Ψ_c, both conditions are equivalent if $L_{max} = +\infty$. We use the condition on the bit error probability to define the DE threshold, presented in the next section.

3.5 Density Evolution threshold

The DE threshold δ is defined as the point of discontinuity between these two regions: the region of channel noise $(\epsilon, \sigma) > \delta$ in which the DE recursion does not converge to a zero error probability, and the region of channel noise $(\epsilon, \sigma) < \delta$ for which the DE recursion converges to a zero error probability (see equation (3.15)) in less than L_{max} iterations of the DE recursion. The most efficient way to compute the DE threshold is to perform a dichotomic search and stop when the bisection search interval size is lower than some precision, e.g. $prec = 10^{-7}$. The procedure is described in the algorithm below.

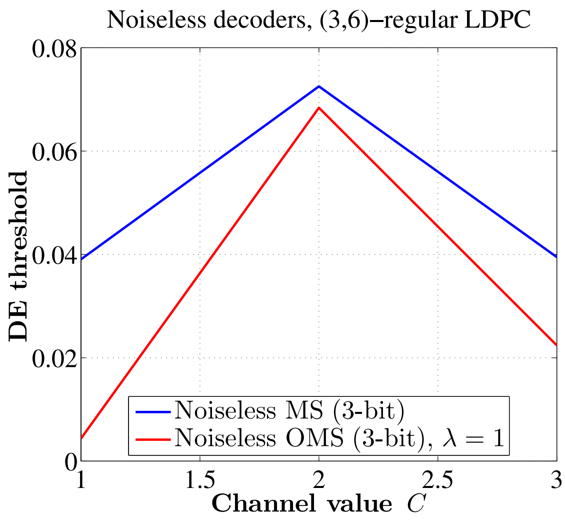
A slight modification of the DE estimation procedure is to set a small target bit error probability η instead of targetting a zero error probability, to declare that the DE recursion converged. If η is small enough, e.g. $\eta = 10^{-6}$, it does not change the threshold estimation for noiseless decoder. Having $\eta > 0$ is however necessary for noisy decoders, as will be explained in the next section.

In the rest of the deliverable, we consider that the notation δ denotes the DE threshold for both the BSC or the BI-AWGN channel, and the interpretation will depend on the context. We describe the main steps to compute the DE threshold for the BSC channel, the DE threshold for the BI-AWGN can be easily deduced.

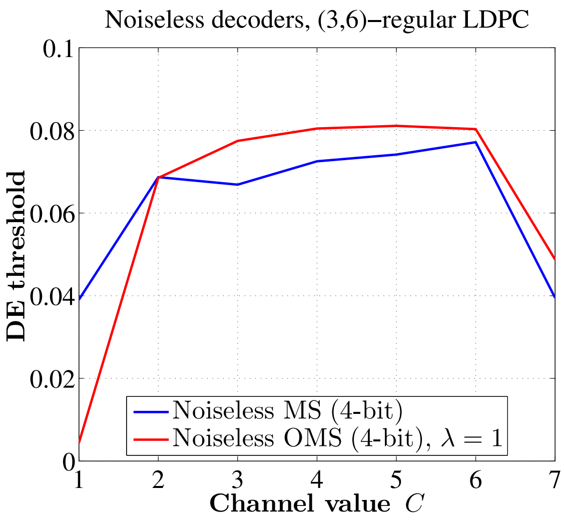
1. [**Initialization**] Initialize interval limits $[\delta_1, \delta_2]$ with $\delta_1 < \delta_2$, such that DE succeeds for $\alpha = \delta_1$ and fails for $\alpha = \delta_2$. Further define $\delta_m = (\delta_1 + \delta_2)/2$.
2. [**while** $|\boldsymbol{\delta_2} - \boldsymbol{\delta_1}| > \textbf{prec}$]
   (a) [**PerformDE**]
      i. [**InitializeDE**] DE is initialized with the equation (3.8) and $\epsilon = \delta_m$.
      ii. [**IterationLoop**]
         A. [**ComputePMF**] apply recursively the sequence of two equations (3.11) and (3.12) for L_max iterations,
         B. [**BreakIteration**] The iteration loop breaks when either the $p_e^{(\ell)} \leq \eta$ or L_max is reached.
   (b) [**DE succeeds**] if $p_e^{(\ell)} \leq \eta$, the DE has converged and we update $\delta_1 = \delta_m$, $\delta_2 = \delta_2$ and $\delta_m = (\delta_1 + \delta_2)/2$.
   (c) [**DE fails**] if $p_e^{(L_{max})} > \eta$, the DE has not converged, and we update $\delta_1 = \delta_1$, $\delta_2 = \delta_m$ and $\delta_m = (\delta_1 + \delta_2)/2$.
   (d) [**Tolerance** Compute the size of the interval $|\delta_2 - \delta_1|$ and stops the procedure if it is smaller than the threshold tolerance (e.g. 10^{-7})
3. [**Threshold**] $\delta = \delta_m$ is the DE threshold.

We further define $p_{th}^{(\ell)}$ the bit error probability at the ℓth decoding iteration when the DE threshold is used in $p_{vtoc}^{(0)} = \delta$ to initialize the DE.

We show on figures 3.5a and 3.5b two examples of the use of DE thresholds to compare different decoding algorithms. We plot the DE threshold of Min-Sum-Based decoders on the BSC channel, as a function of the channel value C. As can be seen the optimization of the channel value yields important gains foe all algorithms. We can note also that the use of an offset is beneficial for the 4-bits MS while it is not for the 3-bits MS.



(a) 3 bits of quantization.



(b) 4 bits of quantization.

Figure 3.5: DE thresholds of MS and OMS, as a function of the channel value C.

Chapter 4

Noise Models and Noisy Density Evolution

4.1 Noise Models in Quantized Decoders

In this deliverable, we will present the concept of *noisy decoders*, defined as decoders that incorporate a certain amount of random perturbation, due to hardware or circuit impreciseness, or to deliberate injected noise. This extra random noise has to be differentiated from the channel noise, which the decoder is supposed to combat. We still call *noise* the effect of those random perturbations, but will differentiate it from the channel's when the context is not clear enough.

We only consider the case of quantized decoders. To implement a noisy decoder, the more classical model is to perturb the noiseless decoder with a random noise applied at the output of the VNU, of the VNU, or both. We further assume that the random noise does not change the message alphabet, *i.e.* noisy messages belong to $\mathcal{M}$. Let $\tilde{m}_{v \to c}^{(\ell)} \in \mathcal{M}$ denote the noisy message obtained after corrupting the noiseless message $m_{v \to c}^{(\ell)} \in \mathcal{M}$, and $\tilde{m}_{c \to v}^{(\ell)} \in \mathcal{M}$ denotes the noisy message obtained after corrupting the noiseless message $m_{c \to v}^{(\ell)} \in \mathcal{M}$.

To siplify the notations in this section, we use the notation $m^{(\ell)}$ to denote any of the check-to-variable messages $m_{c \to v}^{(\ell)}$ or any variable-to-check messages $m_{v \to c}^{(\ell)}$.

4.1.1 Constraints on the Noise Models

In order to be able to perform DE analysis of noisy decoders, the considered noise models have to follow certain properties.

Memoryless
The noise models considered should be independent on the data streams processed by the decoders, since in each iteration of the DE we assume that the messages are independant of the messages at the previous iteration. For example, the perturbation of a message equal to $m^{(\ell)} = +5$ should not depend on the value of the message $m^{(\ell-1)}$.

Symmetric
The considered noise models must also satisfy a symmetry condition defined as follows:

$$\sum_{\beta_2 \in \mathcal{M}} \Pr(\tilde{m} = \beta_2 | m = \beta_1) = \sum_{-\beta_2 \in \mathcal{M}} \Pr(\tilde{m} = -\beta_2 | m = -\beta_1), \forall \beta_1, \beta_2 \in \mathcal{M}. \tag{4.1}$$

Equivalently we can write

$$\Pr(\tilde{m} = \beta_2 | m = \beta_1) = \Pr(\tilde{m} = -\beta_2 | m = -\beta_1), \forall \beta_1, \beta_2 \in \mathcal{M}. \tag{4.2}$$

Consequence on DE
Noisy decoders that use the memoryless and symmetric noise models can still be analysed with DE. The noisy-VNU and noisy-CNU will be symmetric functions, allowing to use the all-zero codeword, and will follow the independance assumption necessary in DE.

We are aware that noise models having those constraints are *not practical* and represent a loss of generality. More complicated or more precise error models could be thought of, including data dependance (current and/or previous), but those models cannot be used in DE analysis.

Let $\Upsilon : \mathcal{M} \to \mathcal{M}$ denote the noise model, which satisfies the symmetry property (equation (4.1)), defined by the conditional PDF $\Pr(\tilde{m}|m)$. Υ maps or changes a noiseless messages $m^{(\ell)} \in \mathcal{M}$ to a noisy message $\tilde{m}^{(\ell)} \in \mathcal{M}$ according to its conditional PDF $\Pr(\tilde{m}|m)$.

4.1.2 A simple Noise Model

The following noise model will be used as an example to illustrate the modifications induced in the decoder and the changes in the DE analysis. Note that this noise model is trivial and far from being practical, but is still interesting to derive some useful interpretations.

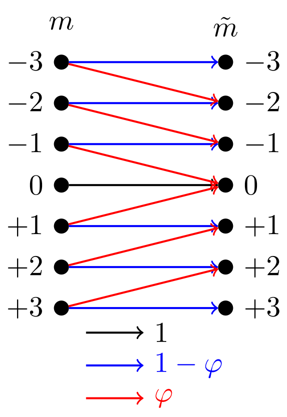
The noise model Υ_1 maps a positive integer value or a negative integer value $\beta \in \mathcal{M}\backslash\{0\}$ to the largest previous or the smallest following integer value, respectively, with probability φ. Υ_1 satisfies the symmetry property $\Pr(\tilde{m} = +\beta|m = +\beta)$ =$\Pr(\tilde{m} = -\beta|m = -\beta), \forall \beta \in \mathcal{M}$, which is a particular case of the equation (4.2) doing $\beta = \beta_1 = \beta_2$. The following equation defines Υ_1

$$\tilde{m} = \Upsilon_1(m = \beta) = \begin{cases} \beta, & \text{with probability} \quad 1-\varphi, \quad \text{if} \quad \beta \in \{+1, ..., +N_q\} \\ \beta - 1, & \text{with probability} \quad \varphi, \quad \text{if} \quad \beta \in \{+1, ..., +N_q\} \\ \beta, & \text{with probability} \quad 1, \quad \text{if} \quad \beta = 0 \\ \beta + 1, & \text{with probability} \quad \varphi, \quad \text{if} \quad \beta \in \{-N_q, ..., -1\} \\ \beta, & \text{with probability} \quad 1-\varphi, \quad \text{if} \quad \beta \in \{-N_q, ..., -1\} \end{cases} \tag{4.3}$$

Let us now asssume that we apply this noise model to the classical MS decoder after the CNU. The proposed noise model is interesting since the corresponding noisy MS implements a decoder which is a weighted combination of MS and OMS. We can note that when $\varphi = 0$, Υ_1 does not perturb the decoder, i.e. noisy messages are equal to noiseless messages and the noisy decoder is the regular MS. When $\varphi = 1$, Υ_1 performs the role of an offset with value of 1, and the noisy decoder becomes a regular OMS decoder with $\lambda = 1$.

Of course, any value $0 < \varphi < 1$ would correspond to a different decoder. This noise model could then be used to determine whether a noisy MS can outperform a noiseless decoder, could it be MS or OMS.

We depict the noise model Υ_1 in Fig. 4.1. On the left of the figure, we show the mapping used to corrupt (or add noise) to a noiseless message m in order to obtain a noisy message $\tilde{m}$. And on the right, it shows the LUT representation of the noise model Υ_1. In this figure, we use $q = 3$ bits for message quantization. It must be noted that each input in the LUT describes the probability to change a noiseless message into a noisy message. For example, a noiseless message $m = +2$ is mapped to $\tilde{m} = +2$ with probability $1 - \varphi$, and it is also mapped to $\tilde{m} = +1$ with probability φ.



$\tilde{m} \setminus m$	-3	-2	-1	0	$+1$	$+2$	$+3$
-3	$1-\varphi$	0	0	0	0	0	0
-2	φ	$1-\varphi$	0	0	0	0	0
-1	0	φ	$1-\varphi$	0	0	0	0
0	0	0	φ	1	φ	0	0
$+1$	0	0	0	0	$1-\varphi$	φ	0
$+2$	0	0	0	0	0	$1-\varphi$	φ
$+3$	0	0	0	0	0	0	$1-\varphi$

Figure 4.1: LUT representation and the mapping used for the noise model Υ_1.

4.2 Noisy Quantized Min-Sum-Based Decoders

To obtain the noisy version of a noiseless decoder, we corrupt the output of the functions Ψ_v (VNU) and Ψ_c (CNU) with random noise. The notation Υ^{p_c} (resp. Υ^{p_v}) indicates that the noise model is applied to the CNU (resp. the VNU) with probability $\varphi = p_c$ (resp. $\varphi = p_c$).

Using the equation (3.3), the update rule at a check-node is given by

$$m^{(\ell)}_{c_m \to v_n} = \Psi_c \left(\left\{ \tilde{m}^{(\ell)}_{v \to c_m} \right\}_{v \in \mathcal{V}(c_m) \setminus \{v_n\}} \right). \tag{4.4}$$

Then, the noise model Υ^{p_c} is used to corrupt each check-to-variable message

$$\tilde{m}^{(\ell)}_{c_m \to v_n} = \Upsilon^{p_c} \left(m^{(\ell)}_{c_m \to v_n} \right). \tag{4.5}$$

The update rule at a variable-node using the update function Ψ_v, equation (3.4), is given by

$$m^{(\ell+1)}_{v_n \to c_m} = \Psi_v \left(L(y_n), \left\{ \tilde{m}^{(\ell)}_{c \to v_n} \right\}_{c \in \mathcal{V}(v_n) \setminus \{c_m\}} \right). \tag{4.6}$$

Then, each variable-to-check message is corrupted by the noise model Υ^{p_v}

$$\tilde{m}^{(\ell+1)}_{v_n \to c_m} = \Upsilon^{p_v} \left(m^{(\ell+1)}_{v_n \to c_m} \right) \tag{4.7}$$

We apply recursively the sequence of four equations (4.4), (4.5), (4.6) and (4.7) to implement one iteration in the decoding process of noisy decoders.

The APP update at a variable-node of a noisy decoder is given by

$$\gamma_n = \mathcal{S}_{\mathcal{M}} \left(L(y_n) + \sum_{c \in \mathcal{V}(v_n)} \tilde{m}^{(\ell)}_{c \to v_n} \right). \tag{4.8}$$

Note that the APP calculation is performed at the VN and is assumed to be not perturbed by a noise, i.e the APP calculation is exact. A justification of this choise is that in the NAND project, our goal is to determine if deliberate perturbations of the noiseless decoders can improve the error correction performance. We can then assume that some of the computing units (including but not limited to, encoding and APP calculation) are noiseless.

Remark. *The equations presented in this section were obtained assuming that variable-to-check and check-to-variable messages were corrupted by* Υ*. When only check-to-variable messages are corrupted, we have* $\tilde{m}^{(\ell)}_{v\to c} = m^{(\ell)}_{v\to c}$*, similarly, when only variable-to-check messages are corrupted, we have* $\tilde{m}^{(\ell)}_{c\to v} = m^{(\ell)}_{c\to v}$*.*

4.3 Noisy Density Evolution

We are interested in obtaining and comparing the DE thresholds of noisy decoders with the DE threshold of noiseless decoders.

Noisy DE recursions can be easily deduced from noiseless DE recursions and the noise model description of (4.3). To deduce the noisy DE equations, let $\tilde{p}^{(\ell)}_{ctov}(k)$, $k \in \mathcal{M}$ denote the PMF of noisy check-to-variable messages in the ℓth iteration. Similarly, let $\tilde{p}^{(\ell)}_{vtoc}(k)$, $k \in \mathcal{M}$ denote the PMF of noisy variable-to-check messages in the ℓth iteration. We consider that the all-zero codeword is sent over the channel.

Initialization
The initialization of DE does not change for noisy decoders, therefore, the equations (3.8) and (3.9) are used to initialize the noisy DE for the BSC and for the BI-AWGN channel, respectively.

Noisy DE update for CNU
Similar to the noiseless DE, we make use of the decomposition of the CNU into elementary CNU. The input of an elementary CNU is the PMF of the noisy messages going out of a noisy VNU, *i.e.* $\tilde{p}^{(\ell)}_{vtoc}$.

$$p^{(\ell)}_{ctov}(k) = \sum_{(i,j):\Psi_c(i,j)=k} \tilde{p}^{(\ell)}_{vtoc}(i)\, \tilde{p}^{(\ell)}_{vtoc}(j), \qquad \forall k \in \mathcal{M} \tag{4.9}$$

Equation (4.9) is used recursively $d_c - 2$ times to compute the PMF of the output of a check-node of degree d_c.

To take into account the effect of random noise on the PMF of the output messages of a check-node, the function Ψ_c is corrupted by the noise model Υ^{p_c}, following

$$\tilde{p}^{(\ell)}_{ctov}(k) = \sum_{(i):\Upsilon^{p_c}\left(p^{(\ell)}_{ctov}(i)\right)=k} p^{(\ell)}_{ctov}(i)\, p_{\Upsilon^{p_c}}(i,k), \qquad \forall k \in \mathcal{M} \tag{4.10}$$

Where $p_{\Upsilon^{p_c}}$ is the transition probability of the CN noise.

Noisy DE update for VNU
Similarly to the calculation made at the variable-node for the noiseless DE, we compute the PMF of the output of a variable-node of degree d_v. The input of noisy DE for the VNU are the corrupted PMF computed with (4.10).

For the BSC

$$\begin{aligned} p^{(\ell+1)}_{vtoc}(k) = &\sum_{(i_1,\ldots,i_{d_v-1}):\Psi_v(i_1,\ldots,i_{d_v-1},-C)=k} \tilde{p}^{(\ell)}_{ctov}(i_1) \ldots \tilde{p}^{(\ell)}_{ctov}(i_{d_v-1})\, p^{(0)}_{vtoc}(-C) + \\ &\sum_{(i_1,\ldots,i_{d_v-1}):\Psi_v(i_1,\ldots,i_{d_v-1},+C)=k} \tilde{p}^{(\ell)}_{ctov}(i_1) \ldots \tilde{p}^{(\ell)}_{ctov}(i_{d_v-1})\, p^{(0)}_{vtoc}(+C), \qquad \forall k \in \mathcal{M} \end{aligned} \tag{4.11}$$

For the BI-AWGN channel

$$p_{vtoc}^{(\ell+1)}(k) = \sum_{(i_1,\ldots,i_{d_v-1},t):\Psi_v(i_1,\ldots,i_{d_v-1},t)=k} \tilde{p}_{ctov}^{(\ell)}(i_1) \ldots \tilde{p}_{ctov}^{(\ell)}(i_{d_v-1})\, p_{vtoc}^{(0)}(t), \quad \forall k \in \mathcal{M} \tag{4.12}$$

Then the noise effect is added at the output of the DE for VNU.

$$\tilde{p}_{vtoc}^{(\ell+1)}(k) = \sum_{(i):\Upsilon^{p_v}\left(p_{vtoc}^{(\ell+1)}(i)\right)=k} p_{vtoc}^{(\ell+1)}(i)\, p_{\Upsilon^{p_v}}(i,k), \quad \forall k \in \mathcal{M} \tag{4.13}$$

Where $p_{\Upsilon^{p_v}}$ is the transition probability of the VN noise Υ^{p_v} (which can be identical to the CN probability).

Noisy DE recursion
The rest of the DE principle is unchanged compared to the noiseless DE of section 3.3.

Remark. *The equations presented in this section were obtained assuming that variable-to-check and check-to-variable messages were corrupted by Υ. When only check-to-variable messages are corrupted, we have $\tilde{p}_{vtoc}^{(\ell)} = p_{vtoc}^{(\ell)}$, similarly, when only variable-to-check messages are corrupted, we have $\tilde{p}_{ctov}^{(\ell)} = p_{ctov}^{(\ell)}$.*

The Fig. 4.2 depicts a simple Tanner graph with a single CN, a single VN, and the PMF of noiseless and noisy messages sent from the VN to the CN, and vice versa.

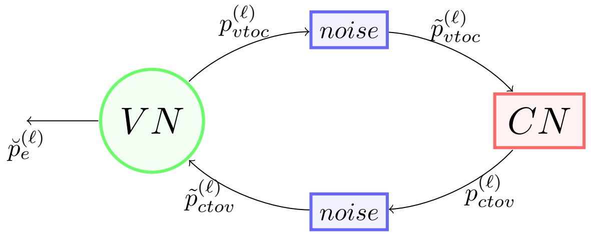


Figure 4.2: Concept of Noisy DE calculation

4.4 Bit Error Probability

Let $\tilde{p}_e^{(\ell)}$ denote the bit error probability at iteration ℓ, which is computed from the PMF of all incoming messages to a VN in the ℓth iteration, defined by

$$\tilde{p}_e^{(\ell)} = \frac{1}{2}\tilde{p}_{app}^{(\ell)}(0) + \sum_{i=-N_q}^{-1} \tilde{p}_{app}^{(\ell)}(i) \tag{4.14}$$

Where $\tilde{p}_{app}^{(\ell)}(k)$, $k \in \mathcal{M}$, denotes the PMF of the APP at the end of the ℓth iteration for noisy decoders, computed similarly than for the noiseless case. In the case of the BSC $\tilde{p}_{app}^{(\ell)}(k)$ is computed as

$$\tilde{p}_{app}^{(\ell)}(k) = \sum_{(i_1,\ldots,i_{d_v}):\Psi_v(i_1,\ldots,i_{d_v},-C)=k} \tilde{p}_{ctov}^{(\ell)}(i_1)\ldots\tilde{p}_{ctov}^{(\ell)}(i_{d_v})p_{vtoc}^{(0)}(-C) +$$
$$\sum_{(i_1,\ldots,i_{d_v}):\Psi_v(i_1,\ldots,i_{d_v},+C)=k} \tilde{p}_{ctov}^{(\ell)}(i_1)\ldots\tilde{p}_{ctov}^{(\ell)}(i_{d_v})p_{vtoc}^{(0)}(+C), \forall k \in \mathcal{M}$$

whereas in the case of the BI-AWGN channel we have

$$\tilde{p}_{app}^{(\ell)}(k) = \sum_{(i_1,\ldots,i_{d_v},t):\Psi_v(i_1,\ldots,i_{d_v},t)=k} \tilde{p}_{ctov}^{(\ell)}(i_1)\ldots\tilde{p}_{ctov}^{(\ell)}(i_{d_v})p_{vtoc}^{(0)}(t), \forall k \in \mathcal{M}$$

In the asymptotic limit of the code-length, $\tilde{p}_e^{(\ell)}$) is the bit error probability of a noisy decoder at the ℓth iteration.

Contrary to the noiseless case, $\tilde{p}_e^{(+\infty)}$ is not necessarily equal to zero when the noisy DE converges and corrects the channel noise. It depends mainly on the chosen error model and the computing units to which it is applied.

A lower bound on the asymptotic error probability can be computed. In [22], the authors have proposed two error models for a quantized noisy MS decoder, and confirmed that $\tilde{p}_e^{(+\infty)}$ is bounded away from zero.

Let $\tilde{p}_e^{(lb)}$ be the lower bound of the asymptotic bit error probability, we have $\tilde{p}_e^{(\ell)} \geq \tilde{p}_e^{(lb)} > 0$. For some noise models, $\tilde{p}_e^{(lb)}$ has a mathematical expression that we can compute, but for other noise models is very difficult to find it.

To solve this problem and still be able to compute noisy DE thresholds, one solution is to choose a target residual error probability $\eta > \tilde{p}_e^{(lb)}$, and declare convergence of the noisy DE recursion when $\tilde{p}_e^{(+\infty)}$ is less than or equal to η. With $\eta = 0$, we get the noiseless case.

Chapter 5

Asymptotic Analysis of the Noisy Quantized Min-Sum Decoder over the BSC

5.1 Case Study

For all results presented in this section, we use a ($d_v = 3, d_c = 6$)-regular ensemble of LDPC codes and Min-Sum-Based decoders with quantization $q \in \{3, 4\}$. First, we present the DE threshold values computed for noiseless and noisy quantized MS decoders. Later, we present the asymptotic bit error probabilities for these decoders, and verify the conclusions of the theoretical analysis with finite length frame error rate (FER) simulations.

5.2 Noisy Thresholds of the MS over the BSC

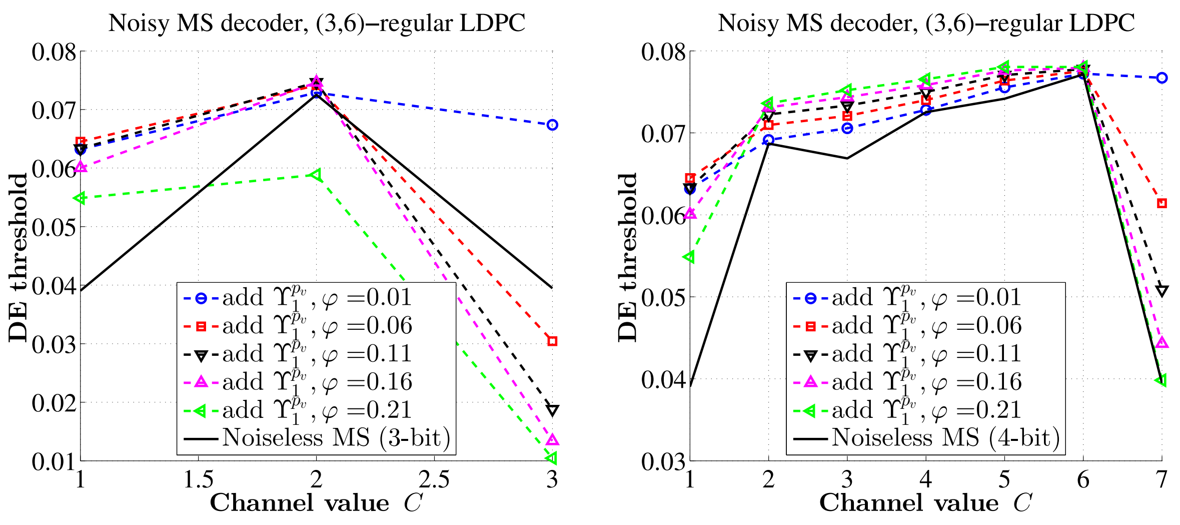
Fig. 5.1, Fig. 5.2 and Fig. 5.3 show the DE threshold as a function of C. In these figures, the solid black curve correspond to DE thresholds of noiseless decoders, and dashed curves correspond to DE thresholds of noisy decoders. We use the noise model Υ_1 presented in section 3.1.2 to perturb the noiseless decoders with different values of $\varphi \in \{0.01, 0.06, 0.11, 0.16, 0.21\}$. For the noisy threshold computation, the DE parameters were set to $L_{max} = 10000$ iterations and $\eta = 10^{-5}$.

Injected noise at the VNU
The noise model $\Upsilon_1^{p_v}$, which perturbs outgoing messages of the VN, is used to obtain the DE thresholds shown in figure 5.1.

Figure 5.1a shows that noisy decoders exhibit better DE thresholds than noiseless decoders for $C = 1$. We can also see that noisy-DE thresholds are slightly better than the noiseless-DE threshold for $C = +2$ and $\varphi \in \{0.01, 0.06, 0.11, 0.16\}$, but noisy-DE threshold is worst than the noiseless-DE threshold for $\varphi = 0.21$. In the case of $C = +3$ and $q = 3$, only the noisy-DE threshold computed for $\varphi = 0.01$ is better than the noiseless-DE threshold.

In the case of 4 bits quantization, noisy decoders exhibit better DE thresholds than noiseless decoders for all channel values. We can note that 3-bit noisy decoders and 4-bit noisy decoders exhibit similar DE threshold for $C = +1$. We also note that noisy-DE thresholds for small values of φ are better than for large values of φ using $C = +N_q$.

Injected noise at the CNU
In a second experiment, we injected the noise at the output of the CNU only. Figure 5.2 shows DE thresholds computed using the noise model $\Upsilon_1^{p_c}$. In figure 5.2a, noisy decoders exhibit better DE



(a) DE threshold using 3 quantization bits (b) DE threshold using 4 quantization bits

Figure 5.1: DE threshold as a function of C of noiseless and noisy decoders, the noise model Υ_1 perturbs the VNU.

thresholds than noiseless decoders for $C = 1$ and $C = +2$, the best DE threshold was computed for $C = +3$ and $\varphi = 0.01$. Similar to figure 5.1, we can observe that for 4 bits quantization, noisy decoders exhibit better DE thresholds than noiseless decoders for all channel values. Using either 3 or 4 bits quantization, noisy decoders exhibit similar DE thresholds for $C = +1$.

Injected noise at both the VNU and the CNU

We perturbed outgoing messages of the CNU and the VNU with $\Upsilon_1^{p_v}$ and $\Upsilon_1^{p_c}$, respectively. The results are reported in figure 5.3. In this figure is evident that noisy-DE thresholds are in general worse than in the case of noise ate the VNU only or at the CNU only, especially when the channel value is large, close to the saturation value N_p.

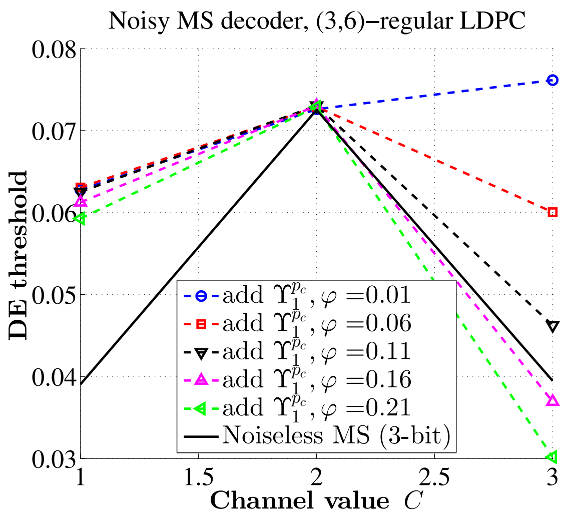
Comparison of best Thresholds

Table 5.1 shows the best DE thresholds for noiseless decoders using $q = 3$ and $q = 4$, and Table 5.2 shows some of the best thresholds for noisy decoders. Small values of φ give us the best DE thresholds for 3 bits quantization, whereas a large value of φ give us the best DE thresholds for 4 bits quantization. The best noisy-DE threshold was computed perturbing the check-to-variable messages with Υ_1 in the 3-bit noisy decoder. For 4-bit noisy decoders, the best noisy-DE threshold was computed perturbing the variable-to-check and check-to-variable messages with Υ_1. The main interesting conclusion of this study is that there exist noisy decoders which exhibit better thresholds than their noiseless counterparts. It is likely that this DE threshold improvement due to the injected noise will be reflected in the waterfall region of the FER performance.

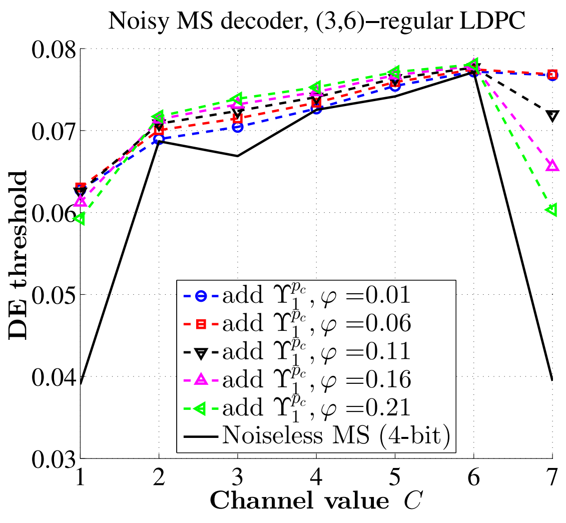
Noiseless decoder, $(3,6)$-regular LDPC code, BSC		
q	C	DE threshold δ
	$+2$	0.0725200764
3 bits	$+6$	0.0771377012

Table 5.1: Threshold values for noiseless MS decoders running these decoders for 10000 iterations.

4 bits

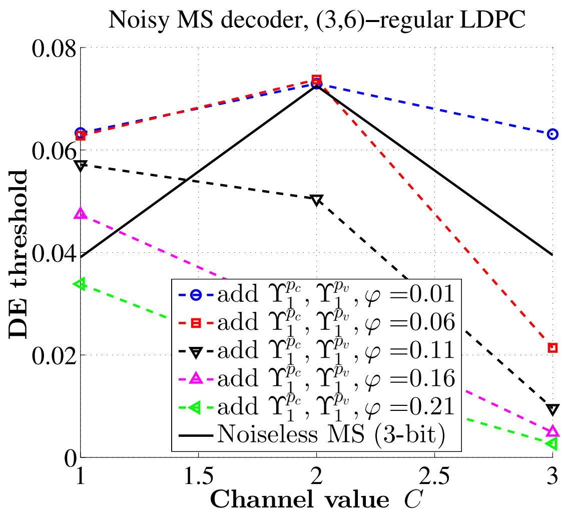


(a) DE threshold using 3 quantization bits

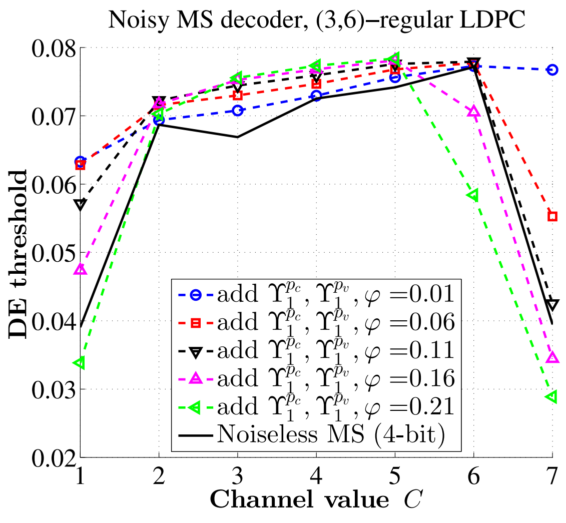


(b) DE threshold using 4 quantization bits

Figure 5.2: DE threshold as a function of C of noiseless and noisy decoders, the noise model Υ_1 perturbs the CNU.



(a) DE threshold using 3 quantization bits



(b) DE threshold using 4 quantization bits

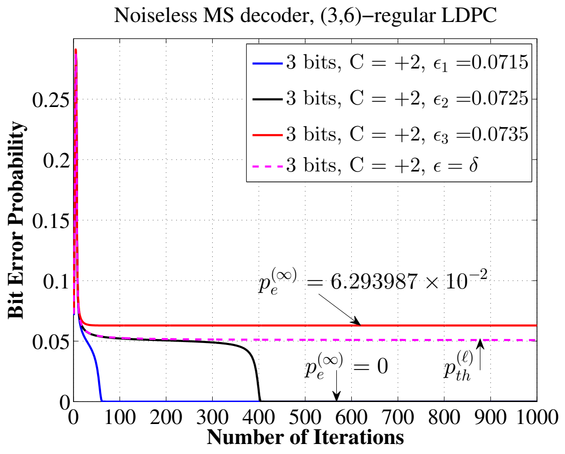
Figure 5.3: DE threshold as a function of C of noiseless and noisy decoders, the noise model Υ_1 perturbs the VN and the CN.

Noisy decoder, $(3, 6)$-regular LDPC code, BSC				
q	C	Noise model	φ	DE threshold $\tilde{\delta}$
3 bits	+2	$\Upsilon_1^{p_v}$	0.11	0.0746431052
	+3	$\Upsilon_1^{p_c}$	0.01	0.0761452462
	+2	$\Upsilon_1^{p_v}$-$\Upsilon_1^{p_c}$	0.06	0.0737253046
4 bits	+5	$\Upsilon_1^{p_v}$	0.21	0.0780435741
	+6	$\Upsilon_1^{p_c}$	0.21	0.0780683933
	+5	$\Upsilon_1^{p_v}$-$\Upsilon_1^{p_c}$	0.21	0.0783768026

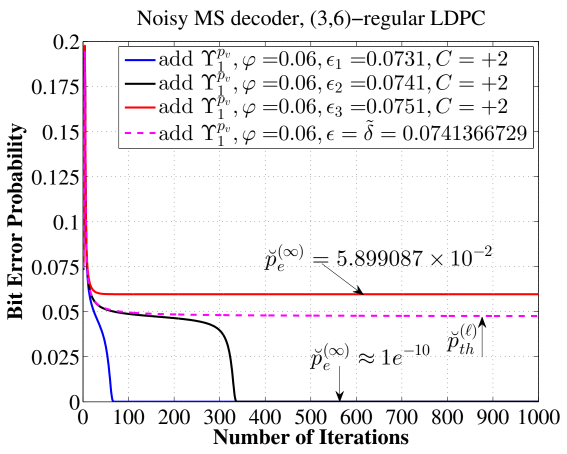
Table 5.2: Best DE thresholds of noisy quantized MS decoders.

5.3 Asymptotic Bit Error Probability of Noisy MS over the BSC

In line with results obtained in the previous section, we illustrate in this section the decoder behaviors in terms of asymptotic bit error probability, for the MS using 3 and 4 bits quantization, $C = +2$ and $C = +6$. We consider only the case of injected noise at the VNU. Figure 5.4 and figure 5.5 show the bit error probabilities, $p_e^{(\ell)}$ and $\tilde{p}_e^{(\ell)}$, as a function of the iteration number ℓ for noiseless and noisy decoders. In both figures, the solid blue, black and red curve correspond to cross-over probabilities around the DE threshold, indicated with the dashed purple curve.



(a) $p_e^{(\ell)}$ of the noiseless MS decoder

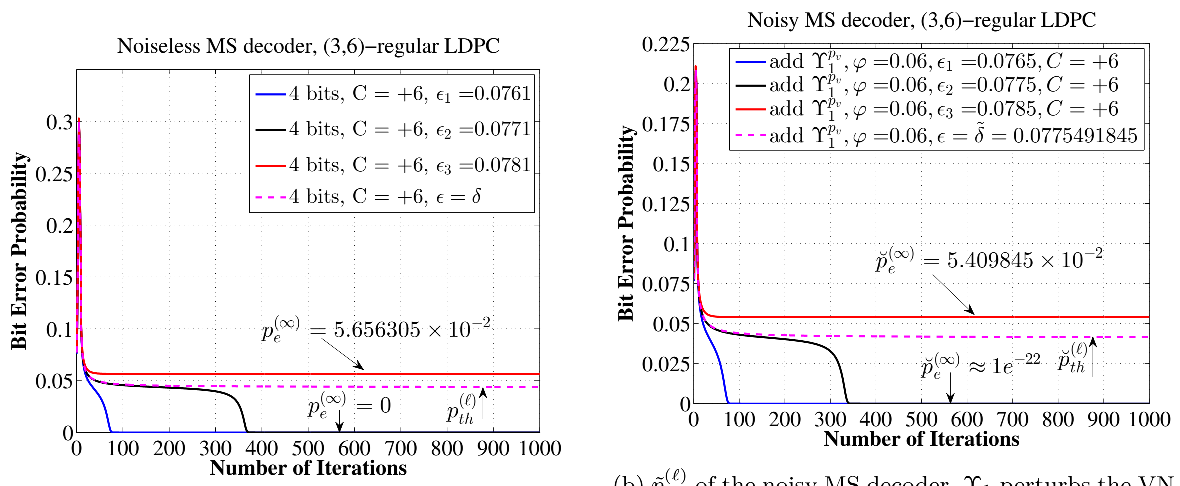


(b) $\tilde{p}_e^{(\ell)}$ of the noisy MS decoder, Υ_1 perturbs the VN with $\varphi = 0.06$

Figure 5.4: Bit error probability behavior of noiseless and noisy decoders around the DE threshold using 3 quantization bits and $C = +2$.

Let us analyze the results presented in figure 5.4. For noiseless and noisy decoders, we considered three crossover probabilities around the DE threshold, with $\epsilon_1 < \epsilon_2 < (\delta, \tilde{\delta}) < \epsilon_3$. It can be seen that $p_e^{(\ell)}$ converges much faster for ϵ_1 than for ϵ_2 to $p_e^{(\infty)} = 0$, and $p_e^{(\ell)}$ converges to $p_e^{(\infty)} = 6.293987 \times 10^{-2}$ for ϵ_3. We observe the same behaviors for the noisy decoders with $\tilde{p}_e^{(\ell)}$. We can also note that for this particular noise model Υ_1, the asymptotic error probability for $\epsilon \leq \tilde{\delta}$ is not equal to zero, which is a common property of noisy decoders, that is $\tilde{p}_{th}^{(\infty)} > 0$. The curves for $\epsilon = \tilde{\delta}$ looks flat, but the DE converges after $\ell = 9994$ iterations.

The curves of figure 5.5 for 4 quantization bits and $C = +6$ yield the same observations.



(a) $p_e^{(\ell)}$ of the noiseless MS decoder

(b) $\tilde{p}_e^{(\ell)}$ of the noisy MS decoder, Υ_1 perturbs the VN with $\varphi = 0.06$

Figure 5.5: Bit error probability behavior of noiseless and noise decoders around the DE threshold using 4 quantization bits and $C = +6$.

5.4 Finite Length FER of the Noisy MS over the BSC

In this section we present the frame error rate (FER) performance for noiseless and noisy decoders. In order to corroborate the asymptotic results obtained in the previous section, we analyze the quantized decoder performance over the BSC. For this purpose, we consider a $(3, 6)$-regular QC-LDPC code with length $N = 1296$ and circulent size $L = 54$, and the noise model Υ_1. The considered decoders are MS with $C = +1$, $C = +2$, $C = +6$, and $\varphi = 0.06$ and $\varphi = 0.21$ for the noisy versions. Additionally, in order to compare the performance of noisy quantized MS decoders, we use a floating-point Belief Propagation decoder, and a quantized offset Min-Sum decoder whose parameters are $C = +10$ offset=3 and 6 bits quantization.

Fig. 5.6 shows the FER performance comparisons between the BP decoder, the quantized OMS decoder, and the noiseless quantized MS decoders as a function of cross-over probability over the BSC. The 3-bit noiseless MS decoder achieves better FER performance than the BP decoder in the error-floor region, but its performance is far from the FER performance of the 6-bit noiseless OMS decoder.

Fig. 5.7 shows the FER performance comparisons between the BP decoder, the quantized OMS decoder, and the noisy quantized MS decoders as a function of cross-over probability over the BSC. In this figure, noisy decoders achieve slightly better FER performance results than the noiseless quantized decoders in the waterfall region, as predicted by the DE threshold analysis. In particular, the 4-bit noisy MS with $\Upsilon_1^{p_v}$ and $\varphi = 0.21$ exhibits the best waterfall of all quantized decoders, and reaches the waterfall of floating-point BP decoding. In the error-floor region, the noisy decoder which uses $\Upsilon_1^{p_v}$ with $\varphi = 0.06$, achieve better FER performance results than the floating-point BP decoder and is very close to the error floor performance of the 6-bit OMS decoder. A interesting conclusion from the results presented in this figure is that a 3-bit noisy-MS decoder has the same (or very close) error correction performance than a 6-bit OMS decoder. This is a nice conclusion in the scope of the NAND project, since it is a first evidence that the use of injected noise could yield improvements with respect to the trade-off implementation complexity vs. performance.

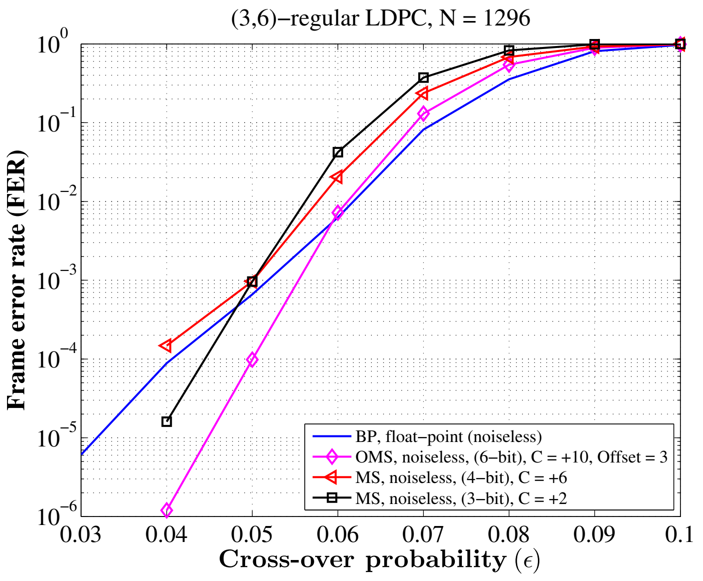


Figure 5.6: FER performance of noiseless MS decoders for $(3, 4)$-regular LDPC code.

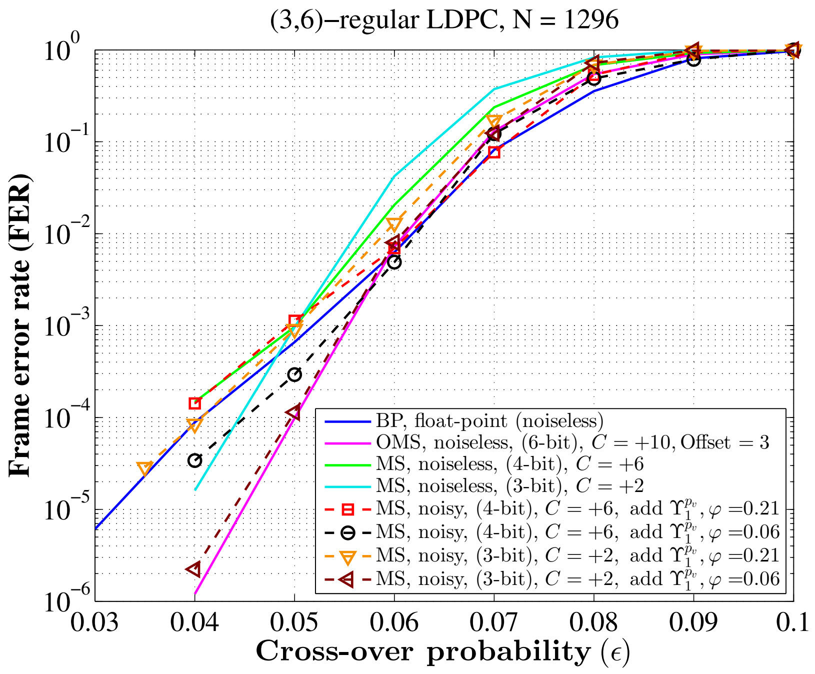


Figure 5.7: FER performance of noiseless and noisy MS decoders for $(3, 4)$-regular LDPC code and $\epsilon_p = 0.21$.

Chapter 6

Conclusions of this Deliverable

In this deliverable we have presented the density evolution framework to analyze noisy message passing decoders in the asymptotical limit of the codeword length. The concept of DE has been generalized to noisy decoders using noise models whiwh are symmetric and memoryless, but which could take any characteristic in terms of transition probability.

The update rules of noiseless and noisy decoders have been presented. The focus has been made on Min-Sum-Based decoders which have small message alphabets constructed from 3 and 4 quantization bits, and for regular LDPC code ensembles with $d_v = 3$ and $d_v = 4$.

We have analyzed the noisy decoder performance for a simple error model. The DE thresholds results have shown that the effect of the injected noise could be beneficial, as we identified noisy decoders with better noisy-DE thresholds than the noiseless ones.

As a conclusion, the results of this work can serve as general tools to analyze the quantized noisy Min-Sum decoders within the NAND project. More elaborate error models and generalizations of the DE thresholds for the BI-AWGN will be studied in future works.

Bibliography

[1] R. G. Gallager, "Low-Density Parity-Check Codes," *Cambridge, MA:*, MIT Press, 1963.

[2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEEE Transactions on Communications*, vol. Vol-62, no. 10, pp. 3385–3400, October 2014.

[3] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Transactions on Information Theory*, vol. Vol-45, no. 2, pp. 399–431, March 1999.

[4] ETSI, "ETSI EN 302 307-1 V1.4.1 (2014-07) Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2," 2014.

[5] ETSI, "ETSI EN 302 307-2 V1.1.1 (2014-10) Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 2: DVB-S2 Extensions (DVB-S2X)," 2014.

[6] M. R. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Transactions on Information Theory*, vol. Vol-27, no. 5, pp. 533–547, September 1981.

[7] N. Wiberg, *Codes and Decoding on General Graphs.* PhD thesis, Linkoping Univ., Linkoping, Sweden, 1996.

[8] T. Richardson and R. Urbanke, "Modern Coding Theory," *Cambridge University Press*, pp. 70–90, 2007.

[9] D. J. C. MacKay, "Information Theory, Inference and Learning algorithms," *Cambridge University Press*, vol. 7, pp. 555–573, 2003.

[10] 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. Vol-47, no. 2, pp. 599–618, February 2001.

[11] T. J. R. M. A. Shokrollahi and R. L. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," *IEEE Transactions on Information Theory*, vol. Vol-47, no. 2, pp. 619–637, February 2001.

[12] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved Low-Density Parity-Check Codes Using Irregular Graphs," *IEEE Transactions on Information Theory*, vol. Vol-47, no. 2, pp. 585–598, February 2001.

[13] A. Bennatan and D. Burshtein, "On the Application of LDPC Codes to Arbitrary Discrete-Memoryless Channels," *IEEE Transactions on Information Theory*, vol. Vol-50, no. 3, pp. 417–438, March 2004.

[14] S. Myung, K. Yang, and J. Kim, "Quasi-Cyclic LDPC Codes for Fast Encoding," *IEEE Transactions on Information Theory*, vol. Vol-51, no. 8, pp. 2894–2901, August 2005.

[15] M. R. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, “LDPC block and Convolutional Codes Based on Circulant Matrices,” *IEEE Transactions on Information Theory*, vol. Vol-50, no. 12, pp. 2966–2984, December 2004.

[16] E. B. E. D. Declercq, M Fossorier, “Channel Coding: Theory, Algorithms, and Applications,” *Academic Press Library in Mobile and Wireless Communications, Elsevier*, vol. ISBN: 978-0-12-396499-1, 2014.

[17] D. J. C. MacKay, “Alist format,” [Online]. Available:.

[18] C. L. K. Ngassa, *LDPC Decoders Running on Error Prone Devices: Theoretical Limits and Practical Assessment of the Error Correction Performance.* PhD thesis, Université de Cergy-Pontoise, Cergy-Pontoise, France, 2015.

[19] V. Savin, “LDPC decoders,” pp. 1–54, October 2013.

[20] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor Graphs and the Sum-Product Algorithm,” *IEEE Transactions on Information Theory*, vol. Vol-47, no. 2, pp. 498–519, February 2001.

[21] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke, “Analysis of Sum-Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation,” *IEEE Transactions on Information Theory*, vol. Vol-47, no. 2, pp. 657–670, February 2001.

[22] C. K. Ngassa, V. Savin, E. Dupraz, and D. Declercq, “Density evolution and functional threshold for the noisy min-sum decoder,” *IEEE Transactions on Communications*, vol. Vol-63, no. 5, pp. 1497–1509, May 2015.

[23] M. Fu, “On Gaussian Approximation for Density Evolution of Low-Density Parity-Check Codes,” *IEEE*, pp. 1107–1112, 2006.

[24] G. Li, I. J. Fair, and W. A. Krzymien, “Density Evolution for Nonbinary LDPC Codes Under Gaussian Approximation,” *IEEE Transactions on Information Theory*, vol. Vol-55, no. 3, pp. 997–1015, March 2009.