الجـمهوريةالجزائريةالديمقراطيةالشعبية
République Algérienne démocratique et populaire

وزارةالتعليمالــعــاليوالبحــثالعــلمــي
Ministère de l'enseignement supérieur et de la recherche scientifique

جــامعةسعددحلبالبليدة
Université SAAD DAHLAB de BLIDA

كليةالتكنولوجيا
Faculté de Technologie

قسمالإلكترونيك
Département d'Électronique

Mémoire de Master

Mention Électronique
Spécialité Systémes de Télécommunications

présenté par

Goso Kelvin Tatenda

ETUDES DES PERFOMANCES DES DECODEURS LDPC

Proposé par : Professeur Meliani Hamza

Année Universitaire 2017-2018

Remerciements

Firstly, I would like to thank the governments of Zimbabwe and Algeria for the opportunity to study under Zimbabwe-Algeria Bilateral scholarship.

Secondly, all my teachers from Language Department of The University of Annaba, Science and Technic department and Electronic Department of the University of Blida.

Eternal gratitude to my project supervisor Professor Meliani.

Lastly to all the family and friends for much appreciated support throughout my studies.

ملخص: خصصت دراسة لتقنية التشفير LDPC لا سيما فيما يتعلق بـطريقة بنيت مصفوفة تحكم التكفؤ باستخدام أسلوب غالاغر والبناء لنيل وماكاي. تستخدم بعد ذلك تقنية تحللLU للمصفوفات لتشفير البيانات قبل إرسالها عبر قناة صاخبة مثل قناة AWGN أو Rayleigh . البيانات المشفرة تفك تشفيرها عند استقبالها من طرف جهاز الإستبال باستخدام خوارزميات Bit Flipping , probability domain, log domain و simplified log domain. أخيرا يتم تحقيق على أثر إشارة إلى نسبة الضوضاء(SNR) , عدد مرات التكرار, طول الرمز , نوع القناة (قناة AWGN و قناة Rayliegh) على معدل خطأ البت.

الكلمات الرئيسية : encoding, decoding, BER, SNR, LDPC, Channel, iteration, AWGN.

Résumé : Une étude est consacrée aux codes LDPC notamment sur la façon dont la matrice de contrôle de parité est construite en utilisant la méthode Gallagher et construction de Neal et MacKay. La décomposition LU est ensuite utilisée pour coder le message avant de le passer sur les canaux AWGN et Rayleigh. Les mots codés envoyés sont ensuite décodés l'aide de l'algorithme décision Hard (retournement de bit) et les algorithme soft (probability domain, log domain et simplified log domain). Enfin une évaluation est faite sur l'effet du rapport signal sur bruit (SNR), le nombre d'itérations, la longueur du code et le type du canal (canal AWGN, canal de Rayliegh) sur le taux d'erreur BER.

Mots clés: encoding, decoding, BER, SNR, LDPC, Channel, iteration, AWGN.

Abstract: A study of Low-Density Parity Check codes particularly on how the parity check matrix is constructed using Gallagher method and using Neal and MacKay construction. The LU decomposition is then used to encode the message before passing it over AWGN and Rayleigh channel. The sent codewords are then decoded using Hard decision (Bit Flipping) and Soft Decision (probability domain, log domain and simplified log domain) decoding methods. Effects of SNR, number of iterations, code length and channel type on BER are studied.

Keywords: encoding, decoding, BER, SNR, LDPC, Channel, iteration, AWGN.

List of Acronyms and Abbreviations

BER: Bit Error Rate

SNR: Signal to Noise Ratio

LDPC: Low-Density Parity-Check

BEC: Binary Erasure Channel

BSC: Binary Symmetric Channel

AWGN: Additive White Gaussian Noise

Table of Contents

List of Figures

List of tables

General Introduction

In an era like ours, an era of information, an era in which everyone is connected and civilization is ever-rising, every business has an online presence, an era whereby unimaginable volumes of data needs to be transmitted in a fraction of a second, there needs to be a communication system in place which ensures reliable transfer of information. Digital modulations are improving, internet speeds are increasing and broad bandwidths are in place. Technological generations are evolving and as of now, the fifth generation is being tested for commercialization. That means that our information needs to be coded with sophisticated channel codes, thank goodness we have LDPC codes for exactly that.

LDPC codes were invented by Robert Gallagher in 1963 [1] in his PhD thesis but they were quickly forgotten as they were impractical to implement due to the fact that the hardware at the time was not as sophisticated. They lied dormant till Tanner designed his graphs. Still the hardware had not evolved for them to be practically applicable, so other channel codes like the convolutional codes were being used. In the 1990s when it became possible to implement the LDPC codes, MacKay rediscovered them and progressed with other notable researchers such as Urbanke. Now that the hardware has finally caught up, the LDPC codes are now used in the satellite television standard the DVB-S2, they are used in the 10GBase-T ethernet which sends data at 10 gigabits per second over a twisted cable ad they are also used in the latest Wi-Fi standards 802.11n. The other codes like turbo codes are now being relegated to moderate-high data rates transmissions.

In this thesis there are four chapters in which we will study the LDPC codes.

In chapter 1, we are looking at introduction to coding. A little background to assess why we need channel coding through studying the digital communication system and the noisy transmission lines. We go on to look at the type of codes that we have, the lock codes and the convolutional codes. We also take a look at the decoding channels and see how they affect the decoded result.

In chapter 2, we look at the basics of the LDPC code and their representation, how to create a parity check matrix using the Gallagher method and the Neal and Mackay method. We look at how to encode the information source using the systematic encoding and using the LU decomposition.

In chapter 3 we look at decoding the LDPC codes using an iterative decoding algorithm called message passing algorithm. We use the hard decision decoder (bit flipping algorithm) and the soft decoders like the probability domain, log domain and log domain simple decoder. There is also the errors and limitations of the decoders and code.

In chapter 4 we simulate the four decoders to study the evolution of Bit Error Rate (BER) as the signal to noise ratio increases, as the number of iterations increases, as the code length increases and as the channel type changes specifically for the information that has traversed the Additive White Gaussian Noise (AWGN) and the Rayleigh channels.

Chapitre 1 : Introduction

1.1 Background

1.1.1 Digital communication system

In any given type of communication, it consists of the source of information, medium being used to transmit that information and the receiver, that is where the information is intended to reach. The most basic communication is a point-to-point system depicted by the following diagram [2]:

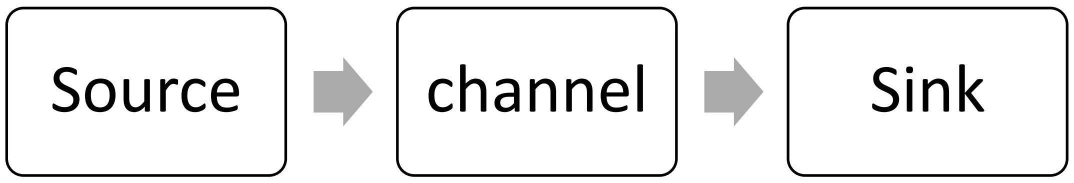

Figure 1. 1 : The General Communication System

The $source$ can be speech, audio, data etc. being transferred through a $noisy$ channel such as a telephone line, wireless line, optical line etc. to $sink$ which is the receiver. In this project, we are interested in the reliable transfer of the data. Reliability of the transfer implies a very low probability of error of the information received at the sink. A more informative diagram about digital transfer of information is shown below:

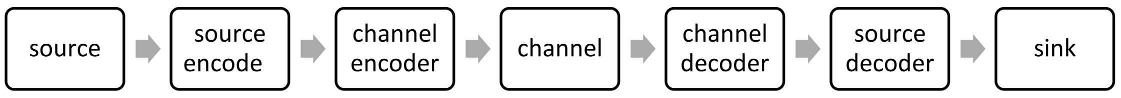

Figure 1. 2 : The Digital Communication system

Source encoder transforms a source(analog) into a bit stream(digital) by use of digitalization techniques such as the pulse coded modulation (PCM). It then removes the redundancy bits in the source in a way that we can retain the information without losing the essential part of the information. There are several techniques for this process such as the Hamming source coding. They all follow "the Shannon's source coding theorem[3] which asserts that for a given source and distortion measure, there exists a minimum rate $R = R(d)$ (bits emitted per source symbol) which is necessary and efficient to describe this source and distortion d"[2].The bit stream obtained is called the $information\ sequence$. Source decoder does exactly the opposite of this process that is transforming the bit stream into accessible analog information.

Channel encoder adds some redundancy bits to the information sequence which is used at the receiver to detect errors produced by the noisy channel and eventually to correct them. It is responsible for ensuring a reliable transmission of the information. Different techniques such as Convolutional coding, Cyclic coding, Turbo coding and LDPC coding exist and they go on to error correction at the decoder as well. They are all optimized using the Shannon Channel Theory. It asserts that "there is a maximum rate (bits per channel use), known as channel $capacity\ C$ at which information can be transmitted reliably, that is to say with vanishingly probability of an error over a channel[2]."

Virtually all modern systems are based on Shannon's $source - channel\ theorem$[3], which asserts that the source can be reconstructed with a distortion of at least d at the receiver if the required rate to represent a given source is less than the channel capacity $R(d) < C$. It follows that a good channel coding can be used for virtually any source. The capacity limit $C\ in\ bits/seconds$ is given by the formula below where B is the bandwidth in Hz and S/N is the signal to noise ratio

$$C = B \log_2\left(1 + \frac{S}{N}\right) \tag{1.1}$$

The channel is the physical medium that is used to send information from the transmitter to the receiver[4]. In the case of wireless communications, wireless broadcast channels, mobile radio channels and satellite channels are used. In case of wired communications, telephone lines, coaxial cables and optical fibers are used. Whichever channel is used, the information sent is going to be corrupted in a random manner. Be it interference from man-made noise, thermal currents in the cables or even any other physical random phenomenon like lightning. It is for the channel decoder to try to detect and correct those errors caused by the channel noises

1.1.2 Noisy Channels

We have so far seen that in the channels, there are random noises. now we are modeling the type of noise associated with the channels. There are three main models namely the Gaussian channel, the Rician channel and the Rayleigh channel.

The Gaussian Channel

This is used to model the communication where there is a direct Line of Sight (LOS) between the transmitter and the receiver. The signal received is equal to the signal transmitted plus some noise. The channel is assumed to be linear, time invariant and frequency non-selective. It is a model for thermal noise in communication channels. The noise used in simulation is the Additive White Gaussian Noise (AWGN). It is white noise because its power spectral density is flat, so the autocorrelation of the noise in time domain is zero for any non-zero time-offset. The noise samples follow a Gaussian distribution. This kind of channel is next to ideal for it disregards any external noise except for that caused by the heating of the circuit components of the system[5].

The Rician Channel

This channel is a wireless channel and has a line of sight in addition to all other multipath signals. So, there is a dominant stationary signal. Rician model is used in modeling satellite channels, indoor channels, microcellular channels, etc.

The Rayleigh Channel

This channel is also a wireless channel and has no direct LOS. The received signal is a sum of multipath signals. It is characterized by multipath. Multipath is when a signal is reflected refracted and diffracted by various objects between emitter and receiver. As a result, it arrives at the receiver as sums of multipath components with various delays from the first component. It is a more practical channel model of the three discussed. An example of this communication is a device connected to a Wi-Fi network.

1.2 Types of codes

Any system with digitally presented data such as Internet, television, CD player, mobile telephones, FAX machines, satellites etc. all use the digital communication system. They all succumb to noises. For a reliable communication, they need error correcting codes and the codes must ensure fast encoding of information, easy transmission of the encoded information, fast decoding of the received information, reliable error correction and transfer of maximum information per unit time.

In search for a code as powerful as described above among the most known channel codes which are the Block codes and the Convolutional codes. The most efficient are codes are : the LDPC code which is a block linear code, the Convolutional codes and the Turbo codes which are a hybrid of block and convolutional codes[6].

1.2.1 Linear Block codes

An (n, k) block linear code $\boldsymbol{C}$ is a code in which n is the number of bits of an output which has an input of k bits and d the Hamming distance[6]. In general, our input is associated with 2^k distinct messages. Each message has n symbols(bits) and is known as a $codeword$ denoted c. The code is said to be linear if and only if it forms k-dimensional subspace of $\{0,1\}^n$.

Input $\boldsymbol{u} = (u_0, u_1, \dots, u_{k-1})$ of length k. (Information bits)

Output $\boldsymbol{c} = (c_1, c_2, \dots, c_{n-1})$ of length n.

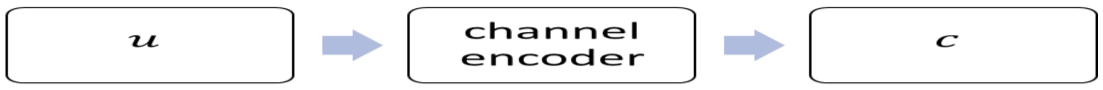

Figure 1. 3 : Block Encoding

Since we are working in binary, the number of possible of information bits and codewords becomes2^k.

The rate at which information passes through the channel is then reduced, because $k < n$, by a factor R, called the $code\ rate$

$$r = \frac{k}{n} \tag{1.2}$$

Another useful property of these block codes is the $Hamming$ distance. If we have two codewords $c_1 = (c_{11}, c_{12}, \dots, c_{1n})$ and $c_2 = (c_{21}, c_{22}, \dots, c_{2n})$, the Hamming distance is the number of positions at which the corresponding elements of c_1 and c_2 are different. Meaning that the minimum number of substitutions in c_1 to make it exactly like c_2 or vice versa.

When a codeword is sent over a noisy channel, the receiver might receive any of the 2^n codewords. The receiver may detect only $2^{(n-k)}$ erroneous codewords and the remaining 2^k are undetectable erroneous codewords.

A generator matrix $\boldsymbol{G}$ is the one used to encode the information word (sequence), $\boldsymbol{u}$ into a codeword $\boldsymbol{c}$. $\boldsymbol{G}$ of a block linear code is constructed by a suitable set of k linearly independent basis vectors as follows:

$$G = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{pmatrix} = \begin{pmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{0,1} & \cdots & g_{k-1,n-1} \end{pmatrix}$$

So, the k-dimensional$\boldsymbol{u}$is encoded into n-dimensional $\boldsymbol{c}$ using the equation below:

$$\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G} \tag{1.3}$$

For any linear block code C, there is an equivalent generator matrix G of dimensions $k \times n$ to define it such that:

$$G = \left(I_k \middle| A_{k,n-k}\right) \tag{1.4}$$

The encoding equation hence becomes:

$$c = \left(u \middle| uA_{k,n-k}\right) \tag{1.5}$$

The first k bits of the code are the information sequence. The remaining bits are parity check bits $m = n - k$. They serve a purpose of error detection and or correction.

The parity check matrix of the aforementioned block linear code is defined as follows:

$$H = \left(-A_{k,n-k}^T \middle| I_{n-k}\right) \tag{1.6}$$

The generator matrix and the Parity check matrix are orthogonal to each other, hence

$$HG^T = 0_{n-k,k} \tag{1.7}$$

At the reception,$\hat{c}$ the codeword received is of the same dimensions as the one sent c. If the received codeword is correct, then the parity check equations is given by 1.8 below is satisfied:

$$H\hat{c}^T = 0 \tag{1.8}$$

H is an $m \times n$ matrix. For binary matrices, the negative sign is inutile, hence it is not used.

Examples of block linear codes are Low-Density Parity Check codes, Reed-Muller Codes, Hamming Codes etc.

1.2.2 Convolutional codes

This type of codes is the one in which a block of $k - message$ bits **u**, is encoded into an $n - bits$codeword **v**. The encoded block depends on the corresponding k bits at the time and also on m previous message blocks. This is to say that the convolutional encoder has got a memory of the order m.

A set of k bits input, n output its encoded by an encoder of memory order m is called an (n, k, m) convolutional code of rate $r = \frac{k}{n}$.

To have an increased redundancy, there needs to simply increase the memory order m.[6][7]

1.3 Decoding Channels

It is important to understand the impact of channel impairments on compressed data which passes over them for they play a crucial role in the designing and development of multimedia applications. The simple channel models are considered as links which can be developed to become routes and consequently networks. The two basic classes of these channels are the Memoryless channel and the channel with memory. Here we will be focusing on discreet memoryless channels only. A discreet memoryless channel is a statistical model with an input **X** and an output **Y**, which is a noisy version of **X**. Both are random variables. It is called discreet because **X** and **Y** are both from alphabets of finite sizes $\mathcal{X}$ and $\mathcal{Y}$ respectively; and memoryless because at any given time (t), the output $Y = y$ depends only on the input X=x at that time. It is represented by conditional probabilities as the figure below[8] shows

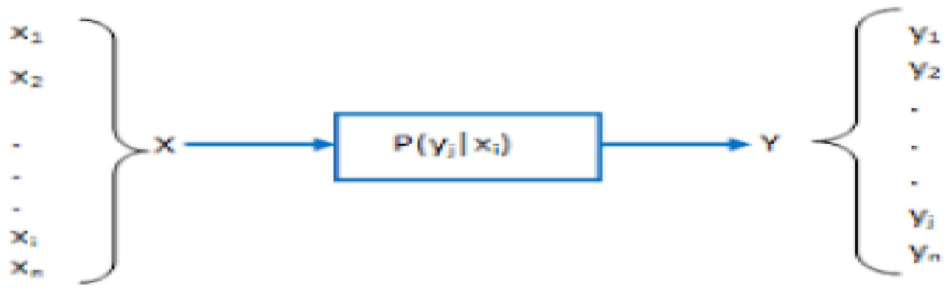

Figure 1. 4 : Memory-less Channel

1.3.1 Binary erasure channel (BEC)

BECs are primarily used in cases which some data will be lost[8]. Internet links and routes are examples of BEC models because data packets either arrive correctly or are lost due to buffer overflows or excessive delays. So, in BEC, bits are either correctly received or they are known to be lost. The figure below[2] shows the BEC(ε) model with probabilities; ε being the probability that the transmitted bit at a particular instant is erased. The conditional probabilities in play are:

$$P[Y = \text{"erasure"} \mid X=0]=\varepsilon \tag{1.9}$$

$$P[Y = \text{"erasure"} \mid X=1]=\varepsilon \tag{1.10}$$

$$P[Y = 0 \mid X=0]=1-\varepsilon \tag{1.11}$$

$$P[Y = 1 \mid X=1]=1-\varepsilon \tag{1.12}$$

It follows that the probability of bit error is zero

$$P[Y = 1 \mid X=0]=0 \tag{1.13}$$

$$P[Y = 0 \mid X=1]=0 \tag{1.14}$$

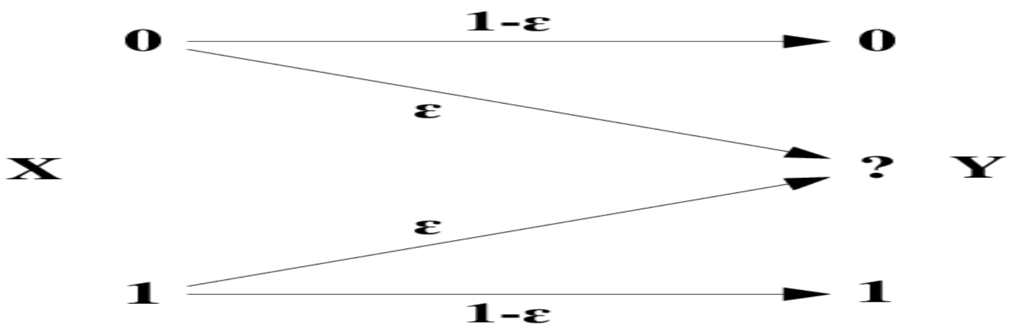

Figure 1. 5 : The Binary Erasure Channel

The total probability of an erasure thus becomes:

$$P[Y = "erasure"] = \sum p(x, Y = "erasure") \tag{1.15}$$

$$P[Y = "erasure"] = \sum p[Y = "erasure"|x]p(x) \tag{1.16}$$

$$\begin{aligned} P[Y = "erasure"] \\ &= p[Y = "erasure"|X = 0]p(X = 0) \\ &+ p[Y = "erasure"|X = 1]p(X = 1) \end{aligned} \tag{1.17}$$

$$P[Y = "erasure"] = \varepsilon p(X = 0)+\varepsilon p(X = 1)=\varepsilon\{p(X = 0)+p(X = 1)\} \tag{1.18}$$

$$P[Y = "erasure"]=\varepsilon \tag{1.19}$$

Cascaded BEC channel

Network routes comprises of multiple links which can be modeled as sets of cascaded channels. Therefore, the output of a first channel serves as the input of the second channel. Now consider the first link BEC(ε) and the second link BEC(δ) such that there is input X, output Y_1 which is the input of the second channel and the output Y_2. Assuming that all these channels are independent to each other.
The probability that the communication is reliable is then:

$$P[Y_2 \neq "erasure"] = (1-\varepsilon)(1-\delta) \qquad (1.19)$$

Thus, the probability of an erasure is given by:

$$P[Y_2 = "erasure"] = 1 - P[Y_2 \neq "erasure"] \qquad (1.20)$$

$$P[Y_2 = "erasure"] = 1 - (1-\varepsilon)(1-\delta) \qquad (1.21)$$

1.3.2 Binary Symmetric channel

This type of channel is widely used to model the channels that exhibit errors. Wireless links and low-quality wired links are prime examples of BSCs[8]. It consists of a binary input X and a binary output Y as shown in the Figure 1.6.

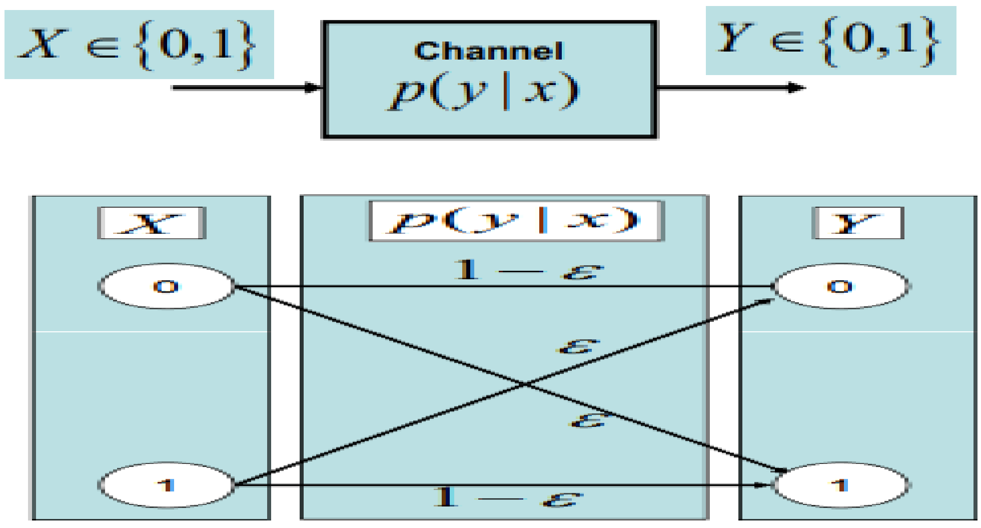

Figure 1. 6 : The Binary Symmetric Channel

A BSC is characterized by the following probabilities:

$$P[Y = 0 \,|X = 0] = 1 - \varepsilon \tag{1.22}$$

$$P[Y = 1 \,|X = 1] = 1 - \varepsilon \tag{1.23}$$

$$P[Y = 0 \,|X = 1] = \varepsilon \tag{1.24}$$

$$P[Y = 1 \,|X = 0] = \varepsilon \tag{1.25}$$

a Cascaded BSC channel

Two or more independent BSC links are joined together. Now consider two links with error probabilities ε_1 and ε_2 shown in the fig below[8]

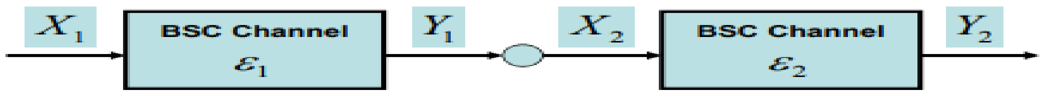

Figure 1. 7 : Cascaded BSC channel

The overall error probability is given by:

$$P[\text{error}] = P[Y_2 \neq X_1] \tag{1.26}$$

So, there is an error if it occurs over the first BSC channel ε_1 and no occurrence over the second BSC channel, ε_2; or if no error occurrence over the first BSC channel and an occurrence over the second BSC channel. It is clear that there is no error if it occurs over both channels. The probability of an error thus becomes :

$$P[Y_2 \neq X_1] = \varepsilon_1(1 - \varepsilon_2) + \varepsilon_2(1 - \varepsilon_1) \tag{1.27}$$

In the next chapter LDPC encoder will be studied.

Chapitre 2 : LDPC Encoder

2.1 Introduction

The basic processes that take place at the LDPC encoder are the creation of parity check matrix, encoding the message and passing the coded information on to the channel.

2.1.1 LDPC Codes

A Low-Density Parity Check code is a linear block code whose control matrix (parity-check matrix H) is sparse, meaning that it has very few non-zero elements in each row and column. There exist regular and irregular LDPC codes. When the weight of the column w_c, that is the number of ones in a column, and the weight of the row w_r are constant, it's a regular LDPC code; otherwise it is an irregular LDPC code. Also, the following conditions hold for a regular LDPC code: $w_c \ll n$, $w_r \ll m$,$w_c \leq w_r$ and $w_r = w_c \frac{n}{m}$ where n is the column number of H and m is its row number.

LDPC codes are represented using a matrix or a bipartite graph called the Tanner Graph[11].

2.1.2 Tanner graphs

Tanner graphs are called bipartite graphs because they are made up of two disjoint and independent sets called check nodes and variable nodes. It has m check nodes (also known as function nodes) which correspond to the parity check constraints, that is the rows of H, and n variable nodes corresponding to the elements of the codeword (also known as message node). These sets can only be connected by an edge and no intra-connection in the sets. The i-th variable node is connected to the j-th check node if and only if $H_{ij} = 1$, with $0 \leq i < n$ and $0 \leq j < m$. Node degree is the number of connections associated with a node

Figure 2.1 [13] is an example of a (10,5) regular LDPC code of H,$w_c = 2$; consequently $w_r = w_c \frac{n}{m} = 4$

As explained above, the check node f_0 corresponds to the first row and is connected to the message nodesy_0, y_1, y_2 and y_3. It is all because $H_{0,0} = H_{0,1} = H_{0,2} = H_{0,3} = 1$.

The logic sum of all bit values connected to a check node is zero;

$$f_0 = y_0 \oplus y_1 \oplus y_2 \oplus y_3 = 0 \text{ and } f_1 = y_0 \oplus y_4 \oplus y_5 \oplus y_6 = 0$$

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

$W_c = 2, W_r = 4,$

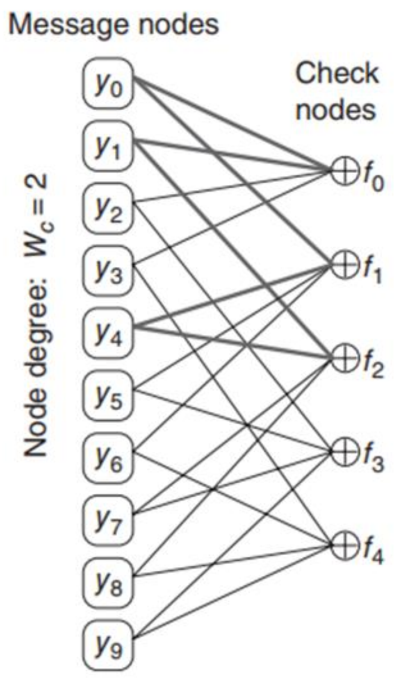

Figure 2. 1 : The Tanner Graph

A sequence of connected nodes which start and end at the same node containing no other node more than once in a Tanner graph is called a $cycle$. The length of a cycle is defined as the number of edges that it has. The size of the smallest cycle is called $girth$ of the graph. The bold edges in the graph above show a cycle of length 6.

The main purpose of the Tanner graph is to serve in a decoding technique based on the $MessagePassingAlgorithm$(MPA). This is an efficient $iterative$ process involving message being passed along the edges from variable nodes to check nodes and vice versa[11]. In the iterative process, if the syndrome of the estimated decoded vector is an all-zero at the check node level, this estimated vector is then our output and it is said the algorithm has converged. However, if the algorithm has not converged for a given maximum number of iterations, it is noted that there has been a decoding failure.

Due to the fact that the LDPC code is sparse, its Tanner graph is consequently sparse and the number of traversed edges is small, the running time of the MPA becomes importantly short. In addition, for a constant number of iterations, each edge is traversed a constant number of times, the number of operations in the message nodes becomes linear, thus linear decoding. It all renders the algorithm efficient.

To achieve this kind of efficiency, the LDPC code has to be well-constructed. There are basically two ways in which the LPDC codes are constructed, Pseudo-random way and the Algebraic way. Pseudo-random codes generally perform better than the algebraically constructed ones but they pose an encoding complexity and a memory problem as the code has to be stored at both the transmitter and the receiver.

2.2 Construction of H parity check matrix

There are two basic ways of creating an LDPC parity check matrix namely the random construction and the structured construction. The random entails the pseudo-random methods mainly done by Gallagher[1] and Neal and Mackay[13] whereas the structured construction refers mainly to the algebraic construction carried out in the 2000s.

The structured constructions are easier to implement in hardware and have regular interconnections that give them good performance. However, they are limited with respect to the length of the code performance-wise, also the rate and the girth of the code.

The randomly constructed codes have a higher encoding complexity because they are difficult to implement in hardware, they suffer more errors than the structured codes. However, given that there are no 4-cycles in the code, they outperform the structured code easily and they produce higher rates and girths. They are also handy when a good code of a long length is required. In this project, they are the ones used.

2.2.1 Gallagher construction

A Gallagher parity check matrix is defined as (n, w_c, w_r) matrix. It consists of w_c sub-matrices of m/w_c rows such that the final matrix is in the form:

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_{w_c} \end{bmatrix} \qquad (2.1)$$

The first submatrix H_1 contains w_r consecutive ones from the left right across all the columns and the rest of the sub-matrices are just random column permutations of the H_1. Below is an example of a (12,3,4) H-matrix created using the Gallagher method.

$$\text{H=} \quad \begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\hline
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1
\end{array}$$

This is the construction of a regular Gallagher LDPC code.

A simulation result of an irregular LDPC code of length 10^6 using a code rate of 0.5 yielded a bit-error probability of 10^{-6}, which is just $0.13dB$ away from the Shannon Capacity [12]. This is better than any Turbo code in existence.
In favor of performance, in this project, the irregular codes are going to be used. A little modified construction by R. Neal is the one to be used specifically due to its simplicity to understand. R. Neal and D Mac Kay rediscovered the LDPC codes and showed their good performances in the mid-1990s [12].

2.2.2 Neal and Mackay construction

This is the method of creating very sparse matrix as described by MacKay in their work [13]. The code is created in four main steps as described below:

- Creating a preliminary parity check matrix which is essentially an all-zero matrix by randomly putting a specified number of ones in every column, w_c. This is to say flipping w_c zeros to 1s. At the same time, it tries to maintain a constant number of 1s in a row throughout the whole matrix. Initially, it creates indicators for all the 1s that will be required and assigns these 1s to rows as evenly as it can, favoring earlier rows if an exactly even split is not possible. It then assigns 1s to successive columns by selecting randomly, without replacement, from this initial supply of 1s, subject only to the constraint that the 1s assigned to a column must be in distinct rows. If at some point it is impossible to put the required number of 1s in a column by picking from the 1s remaining, a 1 is set in that column without reference to other columns, creating a possible unevenness.

This is how $w_c$ones are put in a column, from the first column to the n-th column; the indices of the columns are randomly permuted.

Example of generating a 4 by 8 H-matrix using the MacKay method with w_c=1

- The first step of the algorithm is distributing randomly the indices (the row numbers) 1 to 4 in the first column of the H matrix and so on till 8-th column and the matrix appears as given below:

$$\begin{bmatrix} 1 & 1 & 4 & 4 & 1 & 3 & 3 & 1 \\ 2 & 2 & 2 & 3 & 4 & 1 & 1 & 3 \\ 4 & 4 & 3 & 2 & 2 & 2 & 4 & 2 \\ 3 & 3 & 1 & 1 & 3 & 4 & 2 & 4 \end{bmatrix}$$

- The second step is re-arranging the first w_c elements of each column into a column vector

$$r = \begin{matrix} 1 \\ 1 \\ 4 \\ 4 \\ 1 \\ 3 \\ 3 \\ 1 \end{matrix}$$

- Repeat a row vector of 1 to n elements into a block of w_c rows by 1 column to get:

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8$$

- Reshape the results above into a n*w_c vector column to get

$$c = \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

- Sort the elements of r in ascending order and keep the positions they held before sorting

Sorted r	Positions of elements before sorting
$r = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 4 \\ 4 \end{matrix}$	$ix = \begin{matrix} 1 \\ 2 \\ 5 \\ 8 \\ 6 \\ 7 \\ 3 \\ 4 \end{matrix}$

- Create a column vector of n*w_c =8 *1 of each element of the column vector c given above to get $ci = c(ix)$

$$ci = \begin{matrix} 1 \\ 2 \\ 5 \\ 8 \\ 6 \\ 7 \\ 3 \\ 4 \end{matrix}$$

- Create a row index by repeating a row vector of 1 to m into a block of w_r rows by 1 column and then reshape it into an n*w_c column vector

Row vector	Reshaped row indices
1 2 3 4 1 2 3 4	$\begin{matrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 3 \\ 4 \\ 4 \end{matrix}$

- Sparse the row indices and column indices into an $m \times n$ matrix (reshaped row index, column index ci) to get:

$$\begin{gathered}(1,1)\\(1,2)\\(2,5)\\(2,8)\\(3,6)\\(3,7)\\(4,3)\\(4,4)\end{gathered}$$

- Fill the matrix with ones on the indices that are mentioned above and zeros for the rest of the matrix elements

$$H_p = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For this example, H_p is the final matrix. However, in a general case the following steps should be considered.

- Add 1s to the parity check matrix to avoid rows with no 1s, if there are any. The places within a row in which these 1s are added are randomly selected.
- Avoid redundancy by adding a 1 to a column that has an even number of 1s. By now we have a preliminary full matrix, H_p.
- Eliminate situations where a pair of columns have a 1 in a particular pair of rows (4-cycle). This is done by randomly flipping any 1 among the four ones.

The first three steps may yield a regular matrix as evidenced below:

$$H_p = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Passing this preliminary regular matrix through the fourth step will yield an irregular matrix

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

The larger the matrix, the more it becomes more regular.

2.3 Encoding

This is a stage in which the parity check matrix created above is used to create redundancy bits (check bits) and the added to the source information. There is a canonical way of doing it by creating a generator matrix, then multiplying the source information with the generator matrix created in GF2 (binary Galois Field). This is the method used by the in-built MATLAB encoder. If you are to use your own parity check matrix, it has to be in a systematic form. It will be called canonical encoding in this section.

Another way of doing it is by LU decomposition which is less complex than the canonical way and very much effective[14] .

2.3.1 Canonical Encoding

This method makes use of the H matrix to create the generator matrix. First the H matrix has to be converted to its systematic form[5]:

$$H_{sys} = [A^T | I_m] \tag{2.2}$$

The positions of the identity matrix I_m and A^T can interchange. This operation is performed by Gaussian elimination and column re-ordering. Once there is a systematic parity check matrix, a valid canonical form of the generator matrix can be easily deduced

$$G_c = [I_k | A] \tag{2.3}$$

2.3.2 Encoding using LU decomposition

The whole encoding process is considered as solving a system of equations to get parity check bits. The parity check matrix H is partitioned into two parts A and B such that A is an $m \times m$ sub-matrix and B an $m \times (n - m)$ sub matrix:

$$H = [A|B] \tag{2.4}$$

The sub-matrix A has to be non-singular.

The system of equations to be solved is $Hx = 0$, where x is a vector column codeword. So, it becomes:

$$[A|B]\left[\frac{c}{s}\right] = 0 \tag{2.5}$$

Where c represents the parity check bits and s the source information bits. It follows that:

$$Ac + Bs = 0 \tag{2.6}$$

Since we're working in binary, this becomes

$$c = A^{-1}Bs \tag{2.7}$$

These parity check bits are created with computational time proportional to $m(n - m)$. It can be reduced however to approximately m^2 if the process is broken down into 2 steps:

- Compute $z = Bs$ which is of order m
- Compute $c = A^{-1}z$, of order m again, to make the total m^2

Since A is sparse, it can be efficiently reduced to an Upper or lower triangular matrix using the Gaussian-Jordan method.

$Ac = z$ which is reduced to an upper triangular matrix yields $Uc = y$. A can be reduced into a lower triangular matrix to get $Ly = z$.

$$A = L\,U \tag{2.8}$$

Taking A to be a 4×4 matrix[15],

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{43} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ L_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So, B, L, U are used to create parity check bits in a following manner[16]:

- z is computed first
- A is reduced to a lower triangular matrix by Gaussian elimination method and y is obtained through forward substitution. The equation used is $Ly = z$.
- A is reduced to an upper triangular matrix by Gaussian-Jordan elimination method and our parity check bits c are then obtained using back substitution. $Uc = y$ is the equation being used.

Hence the name LU decomposition.

To have A:

Set **L** and **U** to all zeros
Set **F** to **H**
For $i = 1\ to\ m$
Find a non-zero element in (row, column) i, i or in the later row/column.

Rearrange **F**'s and **H**'s rows and columns to put this element in row and column i, i.

Copy the column i of **F** up to row i of the column i of **U.**

Copy the column i of **F** from row i of the column i of **L**.

Add row i of **F** to later rows with a 1 in column i.

End

Set **B** to the last $n - m$ columns of the rearranged **H**.

Note that the rearranged **H** is the one saved for decoding.
The way in which the non-zero element is found is in such a way that the product the number of non-zeros in its rows minus one and the number of non-zeros in its columns minus one is minimized. It is called the minimal product rule. It minimizes the number of modifications to the other rows which often produce more non-zeros, hence maintaining the sparseness.

Example of LU decomposition:
Consider the following system of equations that need to be solved:

$$v + 2w - 3x = 1 \tag{2.9}$$
$$2v - w + 2x = 3 \tag{2.10}$$
$$3v + 3w - 4x = 5 \tag{2.11}$$

Presenting them in a matrix form $Ac = z$ where $c = (v, w, x)$:

$$\begin{bmatrix} 1 & 2 & -3 \\ 2 & -1 & 2 \\ 3 & 3 & -4 \end{bmatrix} \begin{bmatrix} v \\ w \\ x \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

Reducing A to a lower triangular matrix L to solve for $Ly = z$ by converting it to row echelon using Gaussian elimination method:

$$R_2 \to R_2 - 2R_1$$

$$R_3 \to R_3 - 3R_1$$

The transformations result in:

$$\begin{bmatrix} 1 & 2 & -3 \\ 0 & -5 & 8 \\ 0 & -3 & 5 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

$$R_3 \to 5R_3 - 3R_2$$

The final L matrix of A

$$\begin{bmatrix} 1 & 2 & -3 \\ 0 & -5 & 8 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 7 \end{bmatrix}$$

From the matrix, we have

$y_3 = 7$, $-5y_2 + 8y_3 = 1$ and $y_1 + 2y_2 - 3y_3 = 1$. Using back substitution

$$y_3 = 7$$

$$y_2 = \frac{8y_3 - 1}{5} = 11$$

$$y_1 = 1 - 2y_2 + 3y_3 = 0$$

So, $y = (0,11,7)$

For the upper triangular matrix to solve $Uc = y$

$$\begin{bmatrix} 1 & 2 & -3 \\ 2 & -1 & 2 \\ 3 & 3 & -4 \end{bmatrix} \begin{bmatrix} v \\ w \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ 11 \\ 7 \end{bmatrix}$$

$$R_1 \to 4R_1 - 3R_3$$

$$R_2 \to 2R_2 + R_3$$

$$\begin{bmatrix} -5 & -1 & 0 \\ 7 & 1 & 0 \\ 3 & 3 & -4 \end{bmatrix}\begin{bmatrix} v \\ w \\ x \end{bmatrix} = \begin{bmatrix} 21 \\ 29 \\ 7 \end{bmatrix}$$

$$R_1 \to R_1 + R_2$$

The final U matrix of A

$$\begin{bmatrix} 2 & 0 & 0 \\ 7 & 1 & 0 \\ 3 & 3 & -4 \end{bmatrix}\begin{bmatrix} v \\ w \\ x \end{bmatrix} = \begin{bmatrix} 50 \\ 29 \\ 7 \end{bmatrix}$$

The equations become

$$2v = 50,\ 7v + w = 29 \text{ and } 3v + 3w - 4x = 7$$

Using forward substitution

$$v = \frac{50}{2} = 25$$

$$w = 29 - 7v = -146$$

$$x = \frac{3v + 3w - 7}{4} = -90.75$$

Therefore,

$$c = (v, w, x) = (25, -146, -90.75)$$

Example of encoding using LU decomposition and using the MATLAB in-built encoder:

Using the matrix created above for an 8-bit information,s, at a code rate of $r = 1/2$, the resultant codeword is given in the Table 2.1 below. The first 8 bits are the check bits, the rest are information bits.

The rearranged H is the one shown below

$$nH = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Table 2.1: LU Decomposition Encoded Messages

Source information	Codeword
0 0 0 1 0 0 0 0	0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0	1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0
0 0 1 1 0 0 0 0	1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0
0 1 0 0 0 0 0 0	0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0
0 1 0 1 0 0 0 0	0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0
0 1 1 0 0 0 0 0	1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0
0 1 1 1 0 0 0 0	1 0 0 0 1 1 0 1 0 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0	0 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0	0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0
1 0 1 0 0 0 0 0	1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0
1 0 1 1 0 0 0 0	1 1 1 1 1 1 0 0 1 0 1 1 0 0 0 0
1 1 0 0 0 0 0 0	0 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0
1 1 0 1 0 0 0 0	0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0
1 1 1 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0	1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 0
1 1 1 1 1 0 0 0	1 1 1 1 0 1 0 1 1 1 1 1 1 0 0 0

To use the in-built MATLAB encoder for the same information, the rearranged H is used because it resembles more the systematic form of the parity check matrix. Also, the in-built encoder doesn't support multi-channel, so the vector columns of the source information are coded one by one. The table below shows the results using the MATLAB encoder. The parity check bits are at the end of the codeword in this format, but as earlier stated, their position is arbitrary.

Table 2.2: MATLAB Encoded Messages

Source information	codeword
0 0 1 1 1 1 1 0	0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 1
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0
1 1 1 1 1 0 0 0	1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1

The codeword is then passed over to the channel where it encounters either Gaussian noise or Rician Noise or the Rayleigh Noise. In this project, Gaussian and Rayleigh channels are going to be used.

Chapitre 3 : LDPC Decoding

3.1 Introduction

Decoding is a process of recovering the information sent at the receiver. The information received contains some errors due to the noise and interference in the channel that need to be corrected. However, not all errors can be corrected correctly as shall be explored in the errors section below. The best methods that best detect and correct these errors are shown in the figure below [17]:

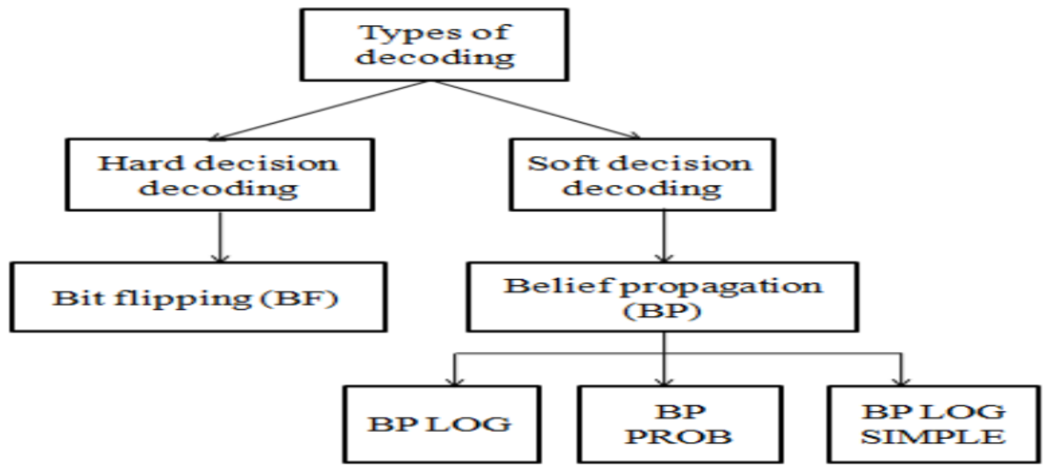

Figure 3. 1 : LDPC Decoders

All these methods are based on iterative Message passing[18], using the Sum-Product algorithm to be specific. We are going to have a look on Hard decision decoding, Soft decoding and errors and limitations of LDPC decoding.

3.2 Hard Decision decoding

This is basically the bit flipping method whereby the error in the codeword is detected and corrected as individual bits[17]. The following H matrix will be used to demonstrate how this works:

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

Its Tanner graph has six check nodes noted **c_1** to **c_6** and 12 variable nodes **v_1** to **v_{12}** as shown below:

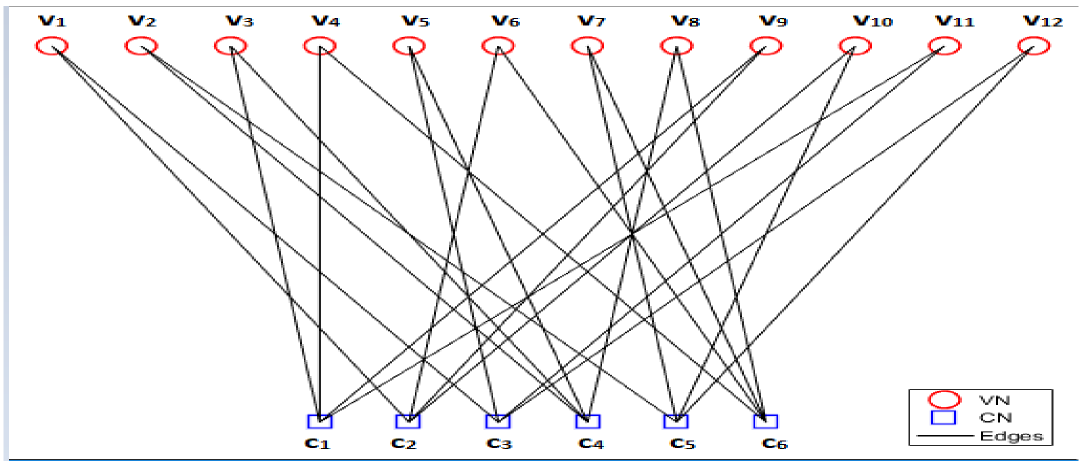

Figure 3. 2 : Tanner graph before decoding

The codeword $y = (1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0)$ is sent and the signal received correspond to the codeword:

$$y' = (1\ 0\ 0\ 1\ 0\ 1\ 0\ \mathbf{1}\ 0\ 0\ 1\ 0)$$

The first step is the one in which the message received from the channel is passed from the variable nodes to the check nodes through the edges. In our case **v_1**sends a 1 to **c_2 and c_3**and all other variable nodes will do so to their connected check nodes.

The second step is for the check nodes to calculate their parity check equations and sent back a bit they believe to be correct if the parity check equations are not satisfied (i.e. are equal to logical 1). In the Tanner graph, if the parity check equation for a particular check node is not satisfied, there is an odd number of connections to it as indicated at the check nodes c_4 and c_6.

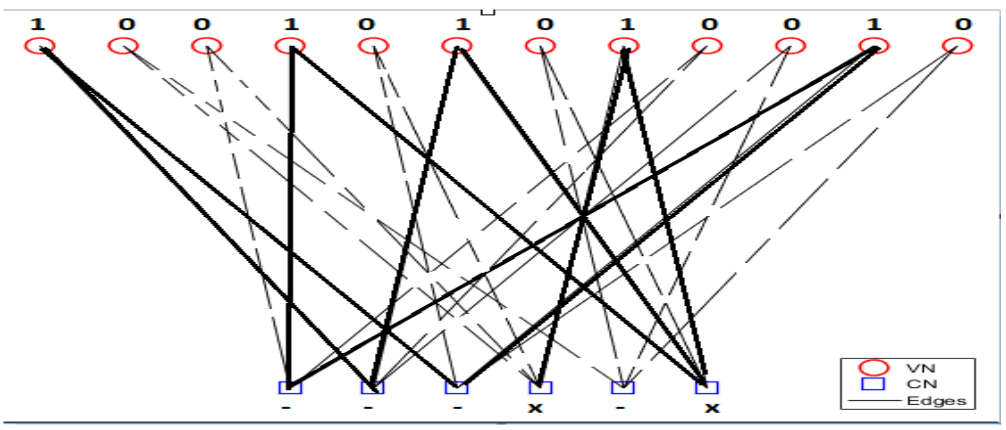

Figure 3. 3 : After one iteration

It is now clear that the parity check equations are binary additions of the messages from variable nodes.

$$c_1 = v_3 \oplus v_4 \oplus v_9 \oplus v_{11} = 0 \quad (3.1)$$
$$c_2 = v_1 \oplus v_6 \oplus v_9 \oplus v_{10} = 0 \quad (3.2)$$
$$c_3 = v_1 \oplus v_5 \oplus v_{11} \oplus v_{12} = 0 \quad (3.3)$$
$$c_4 = v_2 \oplus v_3 \oplus v_5 \oplus v_8 = 1 \quad (3.4)$$
$$c_5 = v_2 \oplus v_7 \oplus v_{10} \oplus v_{12} = 0 \quad (3.5)$$
$$c_6 = v_4 \oplus v_6 \oplus v_7 \oplus v_8 = 1 \quad (3.6)$$

If the result of the parity check equations is a 1 (parity check equation not satisfied), the check node sends back flipped results of the ones it received. Otherwise, it returns the same messages that it received. Below is a table showing what a check node received and what is sent back. The variable nodes that send (in received message column) and receive (in sent messages column) zero messages are in brackets

Check node	Received message	Sent message
c_1	v_4; v_{11};(v_3;v_9)	v_4; v_{11};(v_3;v_9)
c_2	v_1; v_6; (v_9; v_{10})	v_1; v_6; (v_9; v_{10})
c_3	v_1; v_{11}; (v_5; v_{12})	v_1; v_{11}; (v_5; v_{12})
c_4	v_8; (v_2; v_3; v_5)	v_2; v_3; v_5; (v_8)
c_5	(v_2; v_7; v_{10}; v_{12})	(v_2; v_7; v_{10}; v_{12})
c_6	v_4; v_6; v_8; (v_7)	v_7; (v_4; v_6; v_8)

Table 3.1 : Updating table

Step 3 is for the variable node to decide its value considering the response from the check node and send its decision back to the check nodes for parity check again, given that the maximum number of iterations allowed has not been reached. The decision is simply based on a majority vote (majority of ones or zeros at each variable node as shown in the Table 3.2).

According to the decisions shown in Table 3.2, the parity check equations are all satisfied since:

$$c_1 = v_3 \oplus v_4 \oplus v_9 \oplus v_{11} = 0$$
$$c_2 = v_1 \oplus v_6 \oplus v_9 \oplus v_{10} = 0$$
$$c_3 = v_1 \oplus v_5 \oplus v_{11} \oplus v_{12} = 0$$
$$c_4 = v_2 \oplus v_3 \oplus v_5 \oplus v_8 = 0$$
$$c_5 = v_2 \oplus v_7 \oplus v_{10} \oplus v_{12} = 0$$
$$c_6 = v_4 \oplus v_6 \oplus v_7 \oplus v_8 = 0$$

Variable node	Check nodes contributing	Original, check node value 1, check node value 2	Decision
v_1	c_2; c_3	1; 1; 1	1
v_2	c_4; c_5	0; 1; 0	0
v_3	c_1; c_4;	0; 0; 1	0
v_4	c_1; c_6	1; 1; 0	1
v_5	c_3; c_4;	0; 0; 1	0
v_6	c_2; c_6	1; 1; 0	1
v_7	c_5; c_6	0; 0; 1	0
v_8	c_4; c_6	1; 0; 0	0
v_9	c_1; c_2;	0; 0; 0	0
v_{10}	c_2; c_5	0; 0; 0	0
v_{11}	c_1; c_3;	1 ; 1; 1	1
v_{12}	c_3; c_5	0 ;0 ; 0	0

Table 3.2 Decision table

Since all parity check equations are satisfied, then the values of the variable nodes constitute the decoded codeword i.e.

$$y = (1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0)\ .$$

If the maximum number of iterations allowed is reached and the parity check equations are not satisfied, then the decoder is in the situation of a decoding failure.

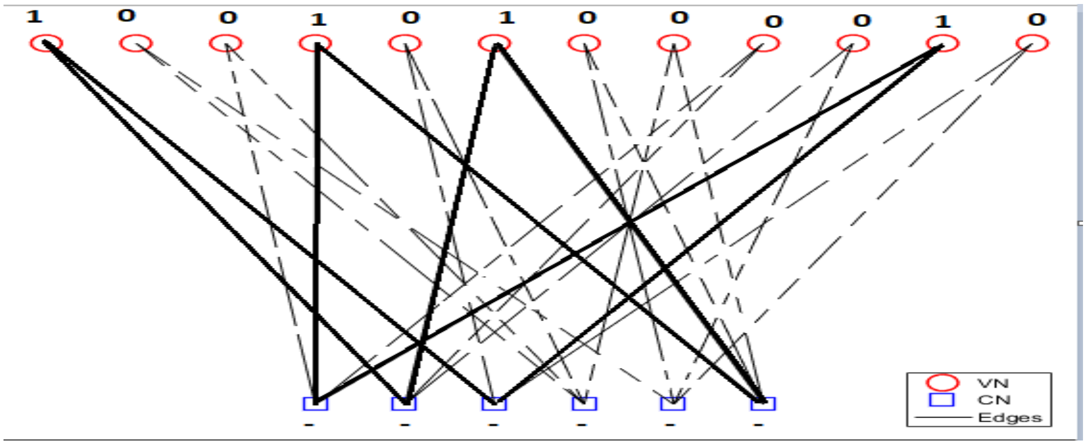

Figure 3. 4 : Tanner Graph after the second Iteration

3.3 Soft Decision Decoding

This is a decoding method based on the concept of Belief Propagation[19]. The core idea of this method is the same as in Hard decision except that whereas the input and output messages in the latter are bits, the probabilities are used in the Belief Propagation. Input bit probabilities are called the $apriori$ probabilities, for they were known before being processed by the decoder. The output bit probabilities are the $aposteriori$ probabilities.

The best way to start exploring the belief propagation is in its probability domain since the others are its derivatives.

3.3.1 Belief propagation Probability Domain

The optimal soft decoder seeks to maximize the probability of one of the codewords given the received information at the decoder and that all parity check equations are satisfied[9][20][21]. It seeks to maximize the probability $P(c|y, Hc^T = 0)$ where c is the sent codeword and y is the received codeword

The principal probability used here is $P_i = P(c_i = 1\ |y_i)$, the probability that a bit is 1 in the codeword, given the received bit. The other main probabilities used in the algorithm are the information probability, from a variable node c_i to a check node also known as pseudo-posterior probability $q_{i,j}$ and the response probability, calculated at the check node (factor node f_j) and then sent to the variable node $r_{j,i}$

The belief propagation algorithm runs in two main steps, $the\ Horizontal\ step$ and $the\ vertical\ step$[20].

Analogous to the bit flipping algorithm, the initialization in belief propagation is when the information probabilities from the channel are sent to the check node from the variable nodes

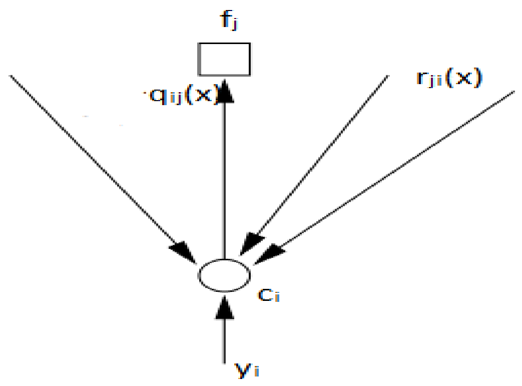

Figure 3. 5 : Vertical Step

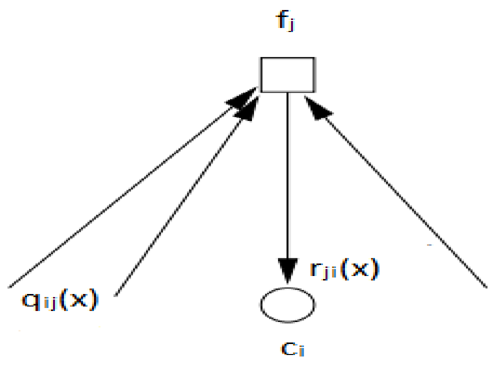

Figure 3. 6 : Horizontal Step

For initialization, the variable nodes send:

$$q_{i,j}(1) = P_i \tag{3.7}$$
$$q_{i,j}(0) = 1 - P_i \tag{3.8}$$

Horizontal step

As shown in Figure 3.6, this is the updating of $r_{j,i}$. It depends on all the variable nodes connected to the check node. The formula used is :

$$r_{j,i}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \epsilon V_j \backslash i} [1 - 2q_{i'j}(1)] \tag{3.9}$$

$$r_{j,i}(1) = 1 - r_{j,i}(0) \tag{3.10}$$

The formula calculates the probability that there is an even number of 1s among the variable nodes connected except for c_i (the meaning of $V_j \backslash i$). The probability $r_{i,j}(0)$ that c_i is a 0.

Vertical Step

This as shown in Figure 3.5, is the step in which the variable nodes update their messages according to the responses they get from the check nodes. The formulae for the updating are given below:

$$q_{i,j}(0) = K_{i,j}(1 - P_i) \prod_{j' \in c_i \backslash j} r_{j',i}(0) \tag{3.11}$$

$$q_{i,j}(1) = K_{i,j} P_i \prod_{j' \in c_i \backslash j} r_{j',i}(1) \tag{3.12}$$

Where $K_{i,j}$ are normalizing constants to ensure that $q_{i,j}(1) + q_{i,j}(0) = 1$ always and $c_i \backslash j$ means that all the check nodes except for f_j.

The variable node updates its current message $\hat{c}_i$by voting for the greater probability from the formulae below:

$$Q_i(0) = K_i(1 - P_i) \prod_{j \in c_i} r_{j,i}(0) \tag{3.13}$$

$$Q_i(1) = K_i P_i \prod_{j \in c_i} r_{j,i}(0) \tag{3.14}$$

The calculation includes all the check nodes connected to the variable node

Each bit of the resultant codeword is therefore obtained using

$$\hat{c}_i = \begin{cases} 1, & if\ Q_i(1) > Q_i(0) \\ 0, & else \end{cases}$$

The algorithm terminates when all the parity check equations are satisfied or when the maximum number of iterations allowed has been reached. Otherwise, it goes back to the horizontal step. The whole algorithm is summarized as follows:

Probability Domain Decoding Algorithm
Input: H; received vector Y; maximum number of iterations L, noise variance $N0/2$. **Initialization:** Set $q_{i,j}(x) = P_i(x)$ for all $H(i,j) = 1$. $$P_i = P(c_i = -1\|y_i) = {}^{1}\!/_{(1 + \exp(2y_i/N0/2))}$$ While number of iterations $l < L$ **Horizontal step :** $$r_{j,i}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \epsilon V_j \backslash i} [1 - 2q_{i'j}(1)]$$ $$r_{j,i}(1) = 1 - r_{j,i}(0)$$ **Vertical step :** $$q_{i,j}(0) = K_{i,j}(1 - P_i) \prod_{j' \epsilon c_i \backslash j} r_{j',i}(0)$$ $$q_{i,j}(1) = K_{i,j} P_i \prod_{j' \epsilon c_i \backslash j} r_{j',i}(1)$$ Also, $$Q_i(0) = K_i(1 - P_i) \prod_{j \epsilon c_i} r_{j,i}(0)$$ $$Q_i(1) = K_i P_i \prod_{j \epsilon c_i} r_{j,i}(0)$$ **Decision:** Set $\hat{c}_i = 1\ \ if Q_i(1) > Q_i(0)$ else Set $\hat{c}_i = 0$. If $H\hat{c} = 0$, then **Stop.** Otherwise, loop back to horizontal step.

Otherwise, let the last decoded iteration as the codeword.

Example from the codewords in section 2.3.2,

$$x = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

was sent and signal corresponding to:

$$y = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

was received. After decoding using this method in only 2 iterations, x was wholly recovered.

$$x' = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0]$$

was obtained when a second error was introduced into the sent codeword

$$y = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0].$$

This however is not the sent information but it is another possible sent information.

The disadvantage of this decoder is that it has got computational complexity greater than the others due to many terms multiplying each other[21].To counter that disadvantage, we introduce the logarithms to change the multiplications to the additions which are less of computational burden.

3.3.2 Belief Propagation Log Domain

In addition to lessening computational burden[21], introducing logarithms also helps in stabilizing the normalizing constants[9]. Everything is expressed as logarithms of ratios of probabilities. First there needs to be a mapping from $c_i \epsilon \{0,1\}$ to $X_i \epsilon \{+1, -1\}$. The formula $X_i = 1 - 2c_i$ is used to achieve the mapping. Now expressing the important probabilities that were used in the probability domain:

$$L(X_i) = ln \frac{P(X_i = +1 \, | Y_i = y_i)}{P(X_i = -1 \, | Y_i = y_i)} \qquad (3.15)$$

$$L(r_{j,i}) = ln\frac{r_{j,i}(+1)}{r_{j,i}(-1)} \tag{3.16}$$

$$L(q_{i,j}) = ln\frac{q_{i,j}(+1)}{q_{i,j}(-1)} \tag{3.17}$$

$$L(Q_i) = ln\frac{Q_i(+1)}{Q_i(-1)} \tag{3.18}$$

The check node response as illustrated in the probability domain is given by

$$r_{j,i}(-1) = 1 - r_{j,i}(+1) = \frac{1}{2} - \frac{1}{2}\prod_{i' \epsilon V_j \backslash i}[1 - 2q_{i'j}(1)] \tag{3.19}$$

Rearranging it gives us

$$1 - 2r_{j,i}(-1) = \prod_{i' \epsilon V_j \backslash i}[1 - 2q_{i'j}(1)] \tag{3.20}$$

Using the equivalence, $\tanh(\frac{1}{2}\log\frac{s}{t}) = 1 - 2t$, With $s = r_{j,i}(+1)$ and $t = r_{j,i}(-1)$, we will remain with

$$\tanh\frac{1}{2}L(r_{j,i}) = \prod_{i' \epsilon V_j \backslash i}\tanh\frac{1}{2}L(q_{i,j}) \tag{3.21}$$

Simplifying it to become

$$L(r_{j,i}) = 2\tanh^{-1}\prod_{i' \epsilon V_j \backslash i}\tanh\frac{1}{2}L(q_{i,j}) \tag{3.22}$$

Let $L(q_{i,j}) = \alpha_{i,j}\beta_{i,j}$ where $\alpha_{i,j} = sgn(L(q_{i,j}))$ and $\beta_{i,j} = |L(q_{i,j})|$

Therefore

$$L(r_{j,i}) = \left(\prod_{i' \epsilon V_j \backslash i}\alpha_{i',j}\right).2\tanh^{-1}\prod_{i' \epsilon V_j \backslash i}\tanh\frac{1}{2}\beta_{i\prime,j} \tag{3.23}$$

$$L(r_{j,i}) = \prod_{i' \in V_j \backslash i} \alpha_{i',j} \,.\, \emptyset \left[\sum_{i' \in V_j \backslash i} \emptyset(\beta_{i,j}) \right] \tag{3.24}$$

With

$$\emptyset(x) = -\ln \tanh \frac{x}{2} = \ln \frac{e^{\frac{x}{2}} + e^{-\frac{x}{2}}}{e^{\frac{x}{2}} - e^{\frac{x}{2}}} = \ln \frac{e^{x}+1}{e^{x}-1},$$

Also, the function $\emptyset(\emptyset(x)) = x$ (function is its own inverse).

The updates at the bit nodes are given as follows:

$$L(q_{i,j}) = L(X_i) + \sum_{j' \in c_i \backslash j} L(r_{j',i}) \tag{3.25}$$

$$L(Q_i) = L(X_i) + \sum_{j \in c_i} L(r_{j,i}) \tag{3.26}$$

The decision is made according to the condition:

$$\hat{c}_i = \begin{cases} 0, & if\ L(Q_i) > 0 \\ 1, & otherwise \end{cases}$$

Here is the summary of the belief propagation log domain decoding

Log Domain Decoder Algorithm
Input: H, Noise variance $N0/2$, maximum number of iterations K, channel output probabilities **Initialization**: for all (i,j) such that $H(i,j) = 1$ **Set** $L(q_{i,j}) = L(X_i) = \frac{2y_i}{N0/2}$ If the number of iterations <K, proceed. Otherwise stop and give the codeword from the last iteration as the final codeword.

Horizontal step

$$L(r_{j,i}) = \prod_{i' \in V_j \backslash i} \alpha_{i',j} \cdot \emptyset\left[\sum_{i' \in V_j \backslash i} \emptyset(\beta_{i,j})\right]$$

Where $\alpha_{i,j} = sgn(L(q_{i,j}))$ and $\beta_{i,j} = |L(q_{i,j})|$ and $\emptyset(x) = \ln\frac{e^x + 1}{e^x - 1}$,

Vertical step:

$$L(q_{i,j}) = L(X_i) + \sum_{j' \in c_i \backslash j} L(r_{j',i})$$

For each bit:

$$L(Q_i) = L(X_i) + \sum_{j \in c_i} L(r_{j,i})$$

Decision:

$$\hat{c}_i = \begin{cases} 0, & if\ L(Q_i) > 0 \\ 1, & otherwise \end{cases}$$

Example from the codewords in section 2.3.2,

$$x = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

was sent and signal corresponding to:

$$y = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

was received. After decoding using this method in only 2 iterations, x was wholly recovered.

$$x' = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0]$$

was obtained when a second error was introduced into the sent codeword

$$y = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0].$$

This however is not the sent information but it is another possible sent information.

3.3.3 Belief Propagation Simplified Log Domain

It is a variation of the BP Log domain with the difference being the inputs only. Unlike all other described soft decoders, this one doesn't make consider the noise variance δ^2. The probability $P_i(x)$ is replaced by minimum(x)[9]. The log likelihood function is replaced by the received vector waveform to simplify it further , hence the name BP simplified log domain.[21]

Example from the codewords in chapter 2.3.2,

$$x = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

was sent and signal corresponding to:

$$y = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

was received. After decoding using this method in only 2 iterations, x was wholly recovered.

$$x' = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0]$$

was obtained when a second error was introduced into the sent codeword

$$y = [0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0].$$

This however is not the sent information but it is another possible sent information.

Simplified Log Domain Algorithm
Input: H, maximum number of iterations K, channel output vector y **Initialization**: for all (i,j) such that $H(i,j) = 1$ **Set** $L(q_{i,j}) = L(X_i) = -y_i$ If the number of iterations <K, proceed. Otherwise stop and give the codeword from the last iteration as the final codeword. **Horizontal step** $$L(r_{j,i}) = \prod_{i' \epsilon V_j \backslash i} \alpha_{i',j} \,.\, \emptyset \left[\sum_{i' \epsilon V_j \backslash i} \emptyset(\beta_{i,j}) \right]$$

Where $\alpha_{i,j} = sgn(L(q_{i,j}))$ and $\beta_{i,j} = |L(q_{i,j})|$ and $\emptyset(x) = \ln\frac{e^x + 1}{e^x - 1}$,

Vertical step:

$$L(q_{i,j}) = L(X_i) + \sum_{j\prime \,\epsilon\, c_i \setminus j} L(r_{j\prime,i})$$

For each bit:

$$L(Q_i) = L(X_i) + \sum_{j \,\epsilon\, c_i} L(r_{j,i})$$

Decision:

$$\hat{c}_i = \begin{cases} 0, & if\ L(Q_i) > 0 \\ 1, & otherwise \end{cases}$$

3.4 Errors and Limitations of LDPC decoding

A decoding is successful when all bits are eventually correct [22]. For any given decoder input y, a failure set is defined as $T(y)$ such that for every decoding success, $T(y) = \emptyset$. Consequently, for a decoding failure, $T(y) \neq \emptyset$. $T(y)$ is called a trapping set and it is found by fixing a large number of iterations for example 150. If the decoder has not converged earlier, add a further smaller number of iterations, for example 20. The union of the bits which do not correctly decode at all in those additional 20 iterations is identified as the trapping set. These trapping sets depends on decoding algorithms and the decoder input. Generally, the Bit flipping algorithm exhibits larger trapping sets.

Chapitre 4 : Decoding signals in Noisy channels

4.1 Introduction

In this chapter, different performances of the four decoding algorithms (Bit-Flipping, Log Domain, Simplified Log Domain and Probability domain) presented in chapter3 are studied by simulation using MATLAB as follows:

- Effects of Signal to noise ratio on Bit Error Rate
- Effects of number of iterations on BER
- Effects of codelength on BER
- Effects of the channel Type on BER

The source messages used are created randomly using a MATLAB function and then modulated using BPSK. These are coded using LDPC coder studied in chapter 2 and passed on to the noisy channel (AWGN and Rayleigh) and then decoded using the 4 algorithms above mentioned.

4.2 Effects of EbN0 on BER

In this section, an investigation is made for the effects of increasing the signal to noise ratio for a codeword of length N=2000 and number of iterations k=3. This is done for 20 frames of the codeword through the Rayleigh channel.

As can be seen from figure 4.1, as the signal to noise ratio increases, the BER decreases. The soft decoders converge for SNR greater than 3dB and the hard decoder for SNR greater than 7dB. The Bit flipping decoder is less efficient than the other decoders. One can notice that the Log Domain and the Probability Domain give the same results and they are almost the same as the Simplified Log Domain.

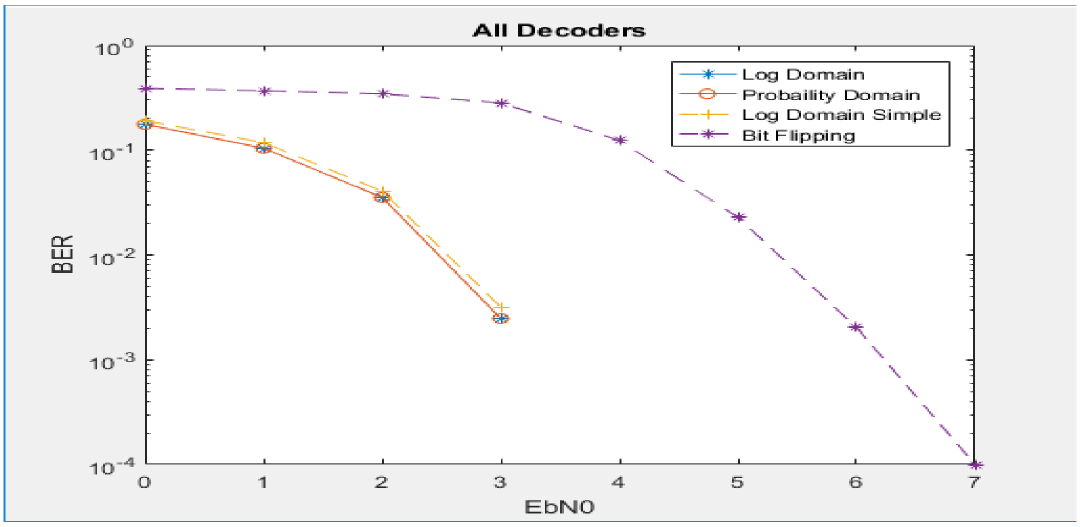

Figure 4. 1: effects of EbN0 on BER

4.3 Effects of number of iterations on BER

This section investigates the effects of number of iterations on BER by fixing the code length N=2000 and number of frames, 20, for all the decoders. This is done for the Rayleigh channel.

4.3.1 Bit-Flipping Decoder

The results of Figure 4.2 show the effects of the number of iterations on BER values. One can notice that the Bit Flipping decoder converges for lower SNR values when more iterations are used.

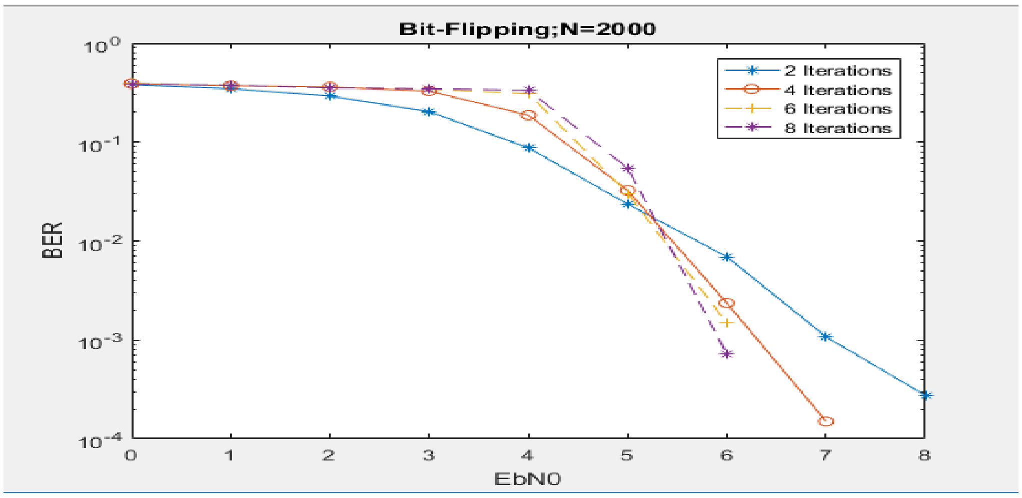

Figure 4. 2 : effects of number of iterations of a Bit flipping decoder

4.3.2 Log Domain Decoder

Figure 4.3 shows the performance of the log domain decoder for different number of iterations. One can notice that the decoder converges for SNR values greater 2dB in 6 and 8 iterations

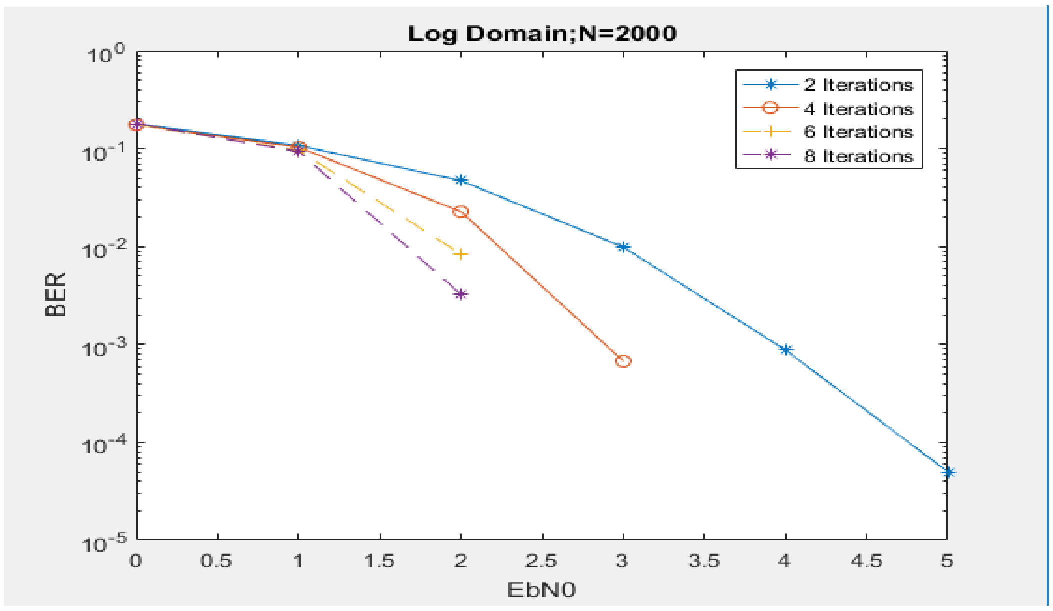

Figure 4. 3 : effects of number of iterations on a log domain decoder

4.3.3 Simplified Log Domain Decoder

Figure 4.4 shows the performance of the simplified log domain decoder for different number of iterations. One can notice that the decoder converges for SNR values greater 2dB in 6 and 8 iterations

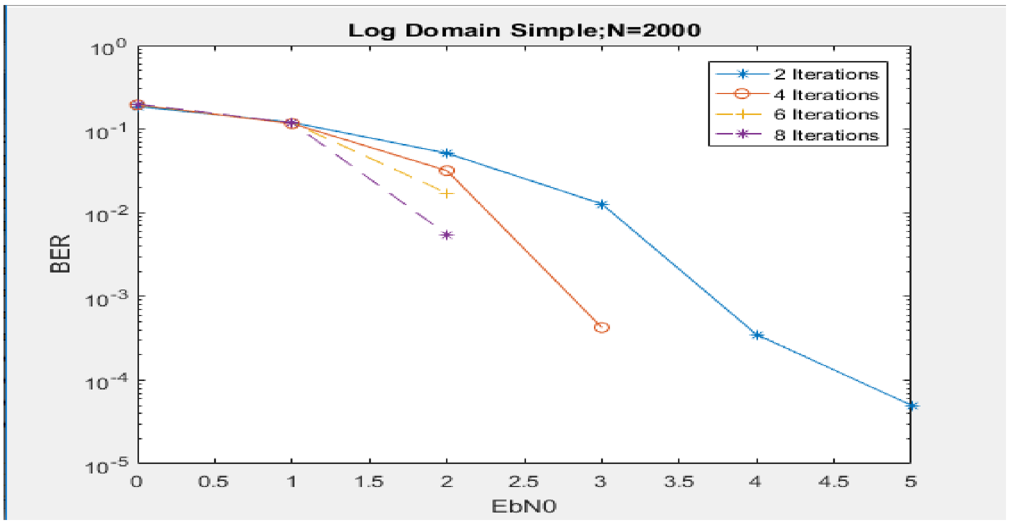

Figure 4. 4 : effects of number of iterations on a simplified Log domain decoder

4.3.4 Probability domain Decoder

Figure 4.5 shows the performance of the probability domain decoder for different number of iterations. One can notice that the decoder converges for SNR values greater 2dB in 6 and 8 iterations.

The probability domain decoder and the simplified log domain decoder give almost the same results.

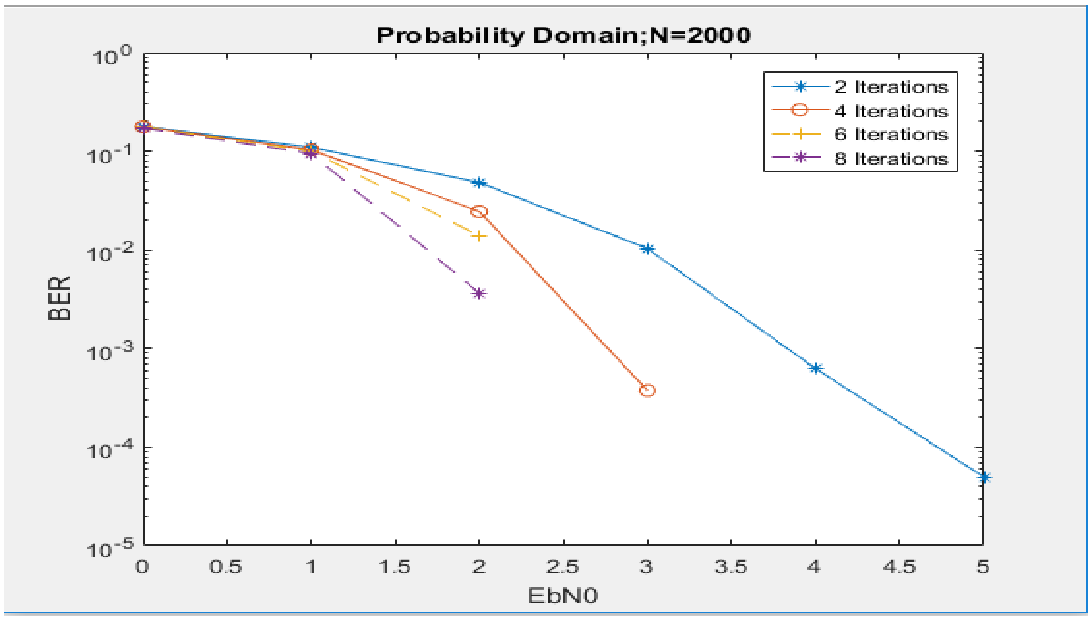

Figure 4. 5 : effects of number of iterations on a probability domain decode

4.4 Effects of codelength on BER

In this section, an investigation is done for the performances of different decoders for different code lengths using 20 frames in only 3 iterations for different SNR values of $EbN0 = [0 : 10]$. This is done for the Rayleigh channel.

4.4.1 Bit flipping decoder

Figure 4.6 shows the effects of code length on a bit flipping decoder.

Code lengths of N=1500 converge for SNR values greater than 6dB. For code lengths of N=500, N=1000 and N=2000, the convergence is reached for SNR values greater than 7dB. The one of N=500 exhibit a highest BER at 7dB and N=2000 a lowest.

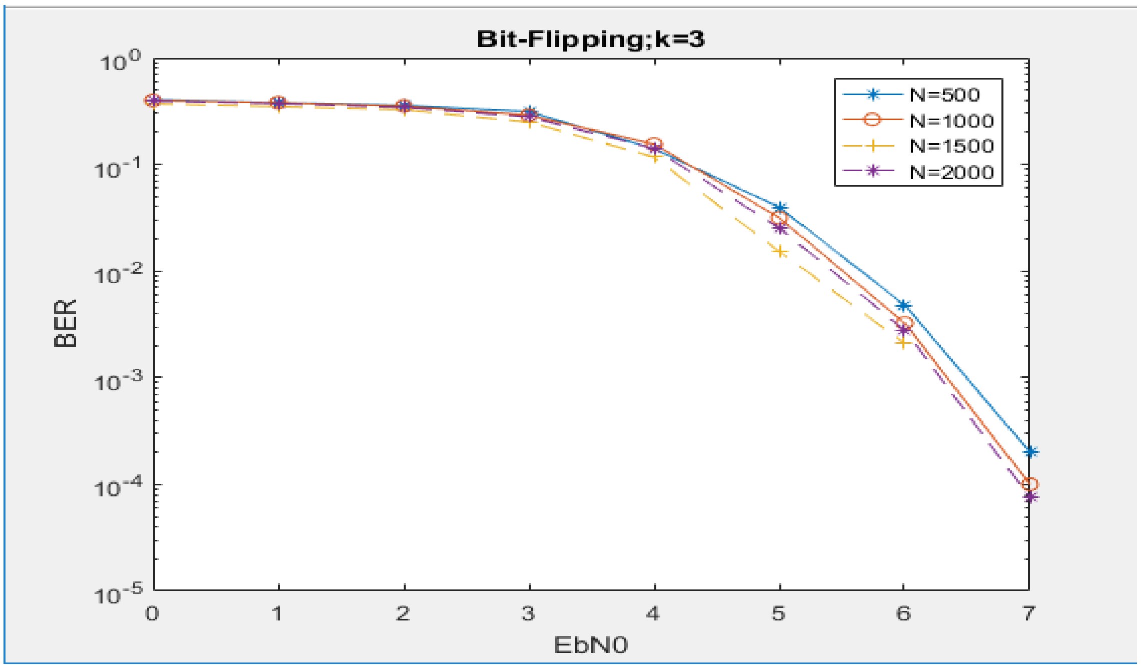

Figure 4. 6 : effects of code length on a bit flipping decoder

4.4.2 Log Domain decoder

Figure 4.7 shows the effects of code length on a log domain decoder

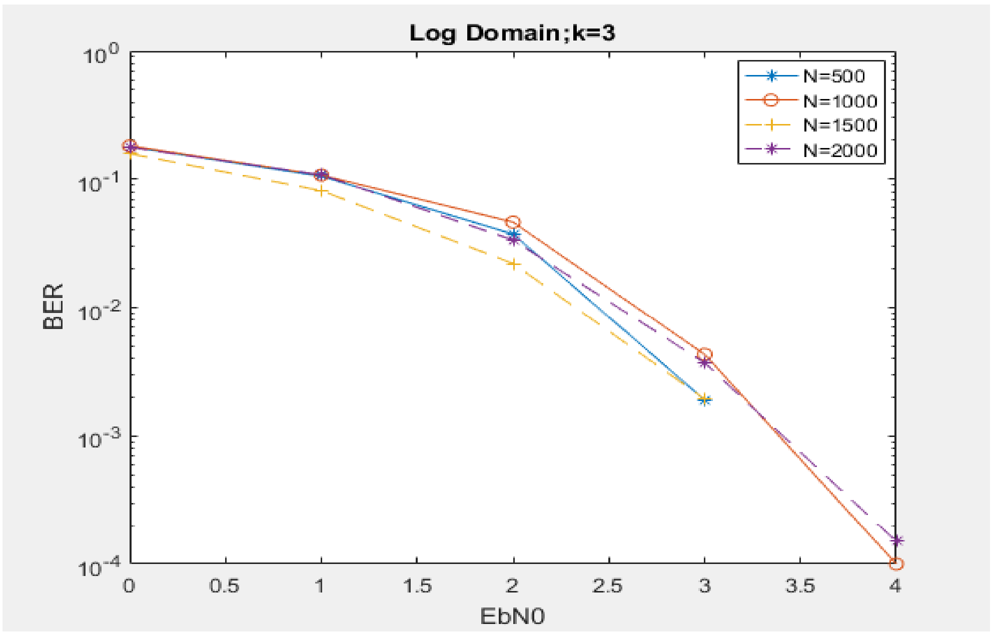

Figure 4. 7 : effects of code length on a log domain decoder

The code lengths of N=500, N=1500 are converging for SNR values greater than 3 dB. N=1000 and N=2000 are converging for SNR greater than 4dB with N=2000 having a higher BER at 4dB.

4.4.3 Simplified Log Domain Decoder

Figure 4.8 shows effects of code length on a simplified log domain decoder

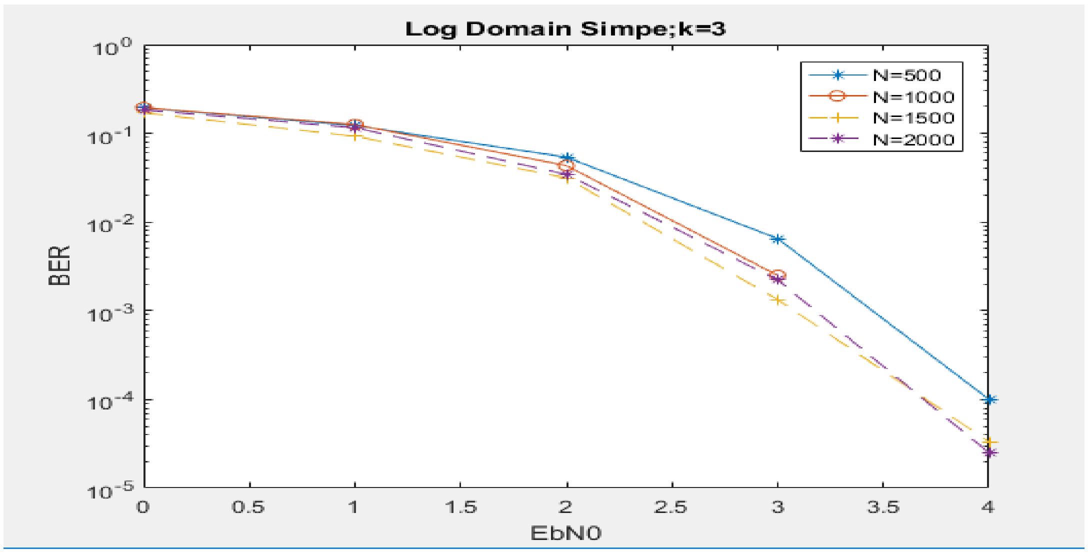

Figure 4. 8 : effects of code length on a simplified log domain decoder

N=500, N=1500 and N=2000 converge for SNR greater than 4dB with N=2000 having the lowest BER value and N=500 the highest. N=1000 converge for SNR greater than 3dB.

4.4.4 Probability Domain Decoder

Figure 4.9 shows the effects of code length on a probability domain decoder.

The decoder converges for SNR values greater than 3dB for the code lengths N=1500 and N=2000 which has a higher BER at 3dB. For N=1000 and N=500, convergence is reached for SNR values greater than 4dB and N=500 has a higher BER value at 4dB.

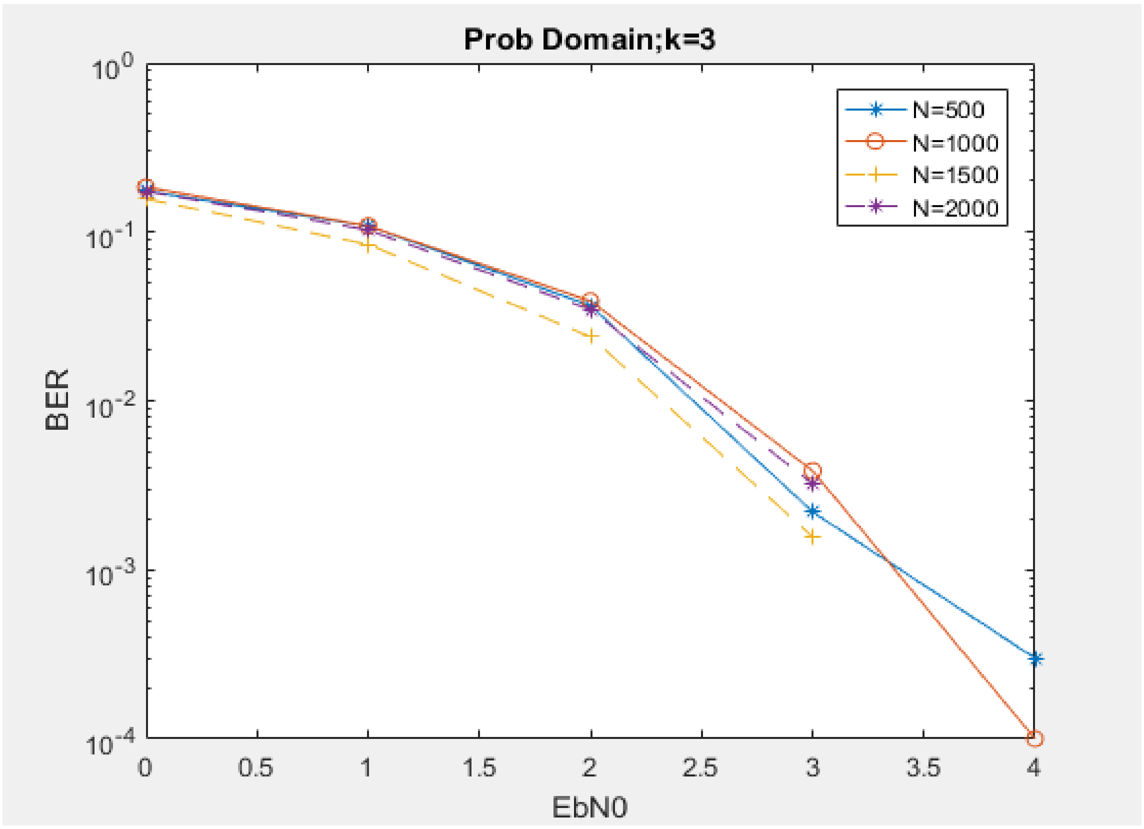

Figure 4. 9 : effects of code length on a probability domain decoder

4.5 Effects of the channel Type on BER

This section shows the performances of the decoders in decoding a codeword of length N=2000 in 20 frames using only 3 iterations and SNR values $EbN0 = [0:10]$.

4.5.1 Bit Flipping decoder

Figure 4. 10 shows the performance of a bit flipping decoder for different channel types.

The decoder converges at a greater SNR value for Rayleigh channel (7dB) than for AWGN (5dB). And for all values before convergence, the AWGN has smaller BER values.

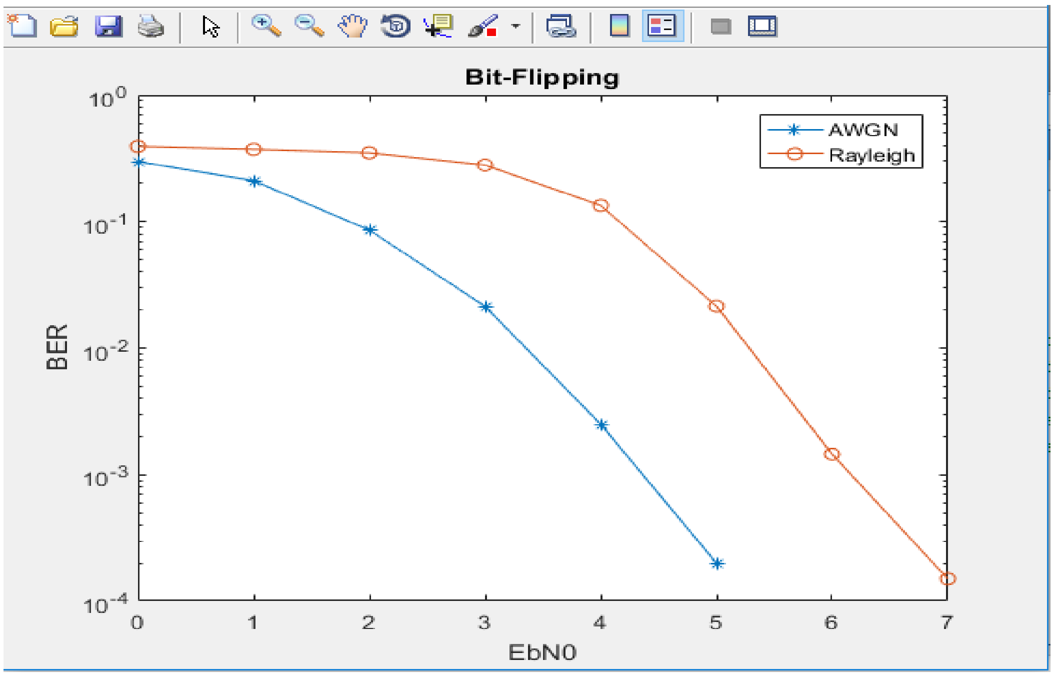

Figure 4.10: Performance of a bit flipping decoder for different channel types

4.5.2 Log Domain Decoder

Figure 4. 11 shows the performance of a Log domain decoder for different channel types.

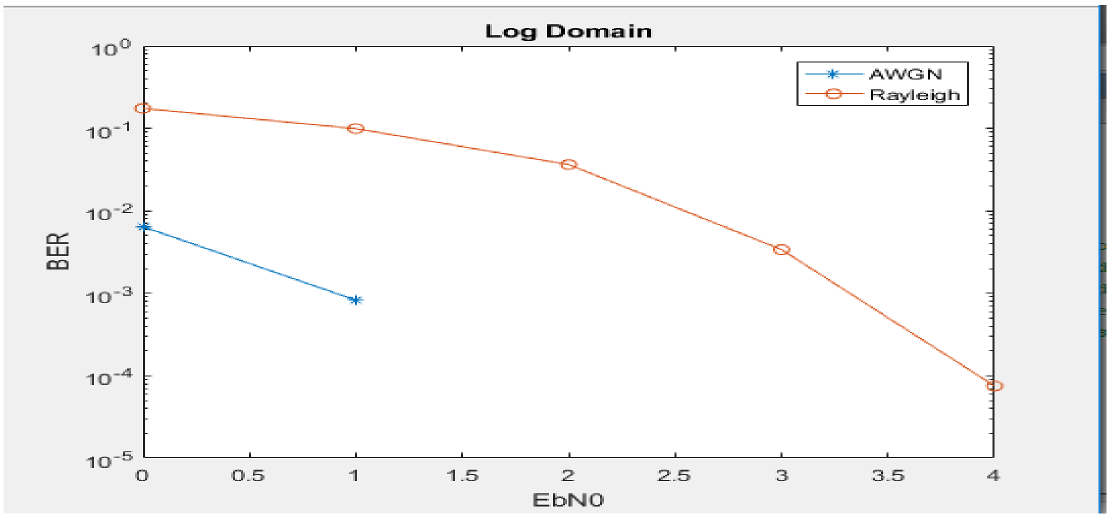

Figure 4.11: Performance of a Log domain decoder for different channel types

The decoder converges for SNR values greater than 1dB for an AWGN channel and 4dB for a Rayleigh channel.

4.5.3 Simplified Log Domain decoder

Figure 4. 12 shows the performance of a simplified log domain decoder for different channel types

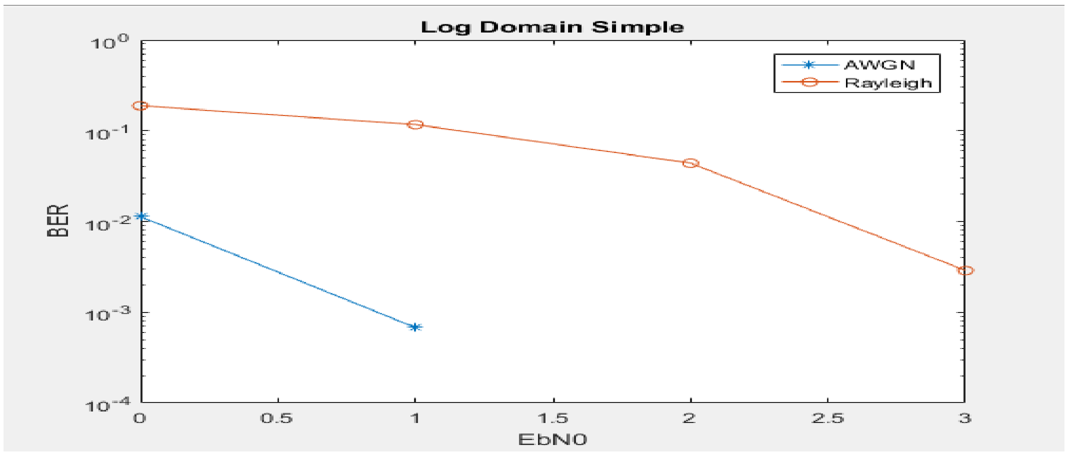

Figure 4. 12: Performance of a simplified log domain decoder for different channel types

The decoder converges for SNR values greater than 1dB in an AWGN channel. For the Rayleigh channel, it converges for values greater than 3dB.

4.5.4 Probability Domain decoder

Figure 4. 13 shows the performance of a bit flipping decoder for different channel types.

The decoder converges for SNR values greater than 2dB in an AWGN and for values greater than 4dB in a Rayleigh channel

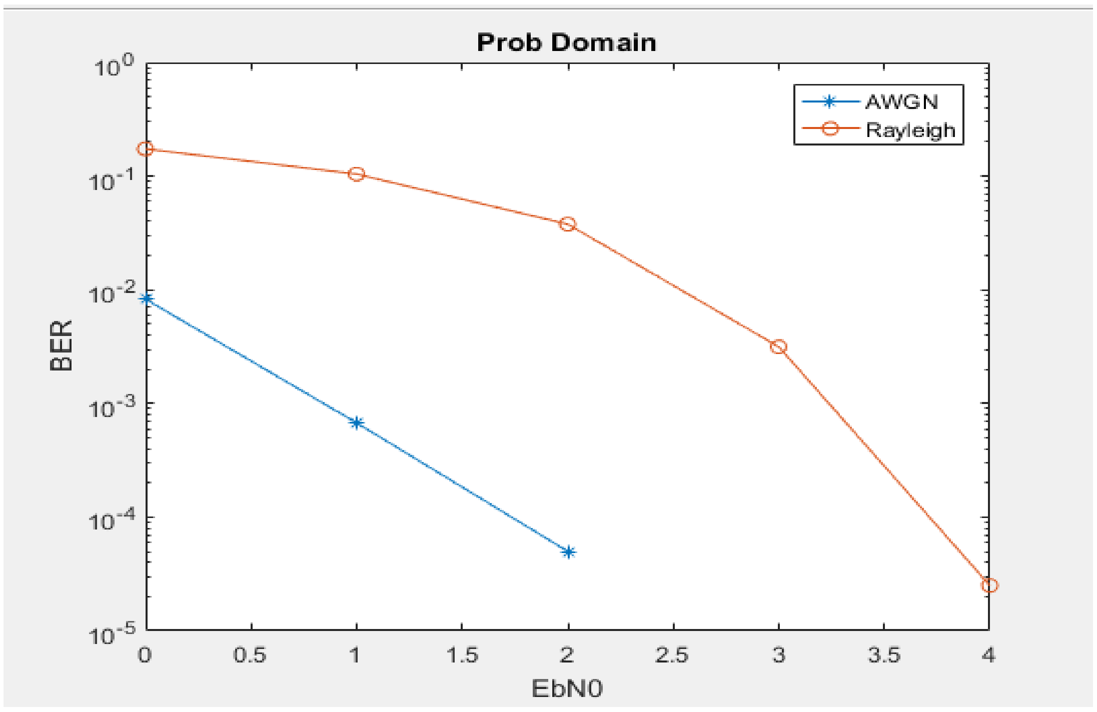

Figure 4. 13: Performance of a bit flipping decoder for different channel types

4.6 Conclusion

In this chapter, are studied the performances of the four decoding algorithms presented in chapter3 namely the effects of signal to noise ratio on BER, the effects of number of iterations on the BER, the effects of code length on BER and the effects of the channel type.

The first case studied as the effect of the signal to noise ratio on the results of the 4 decoders with fixed N=2000 and fixed k=3. It was found that the soft decoders converge after 3dB and the hard decoder after 6dB. The Bit flipping decoder is less efficient than the other decoders. It was noticed also that the Log Domain and the Probability Domain give the same results and they are almost the same as the Simplified Log Domain.

The second case was the investigation of the effect of the number of iterations with fixed code length (N=2000). It was found that the probability domain decoder and the simplified log domain decoder give almost the same results and they are more efficient than the bit flipping decoder.

The third case which is studied is the effect of code length with fixed number of iterations k=3. It was found that the convergence is best for N=2000

The last case which is studied is the effect of the type of channel on the results of decoders. All the decoders give better results with AWGN channel

General Conclusion

In chapter 1, a brief study of a digital communication system was done. It is noted that due to noise from the channels, modelled as Gaussian Channel, Rician Channel and Rayleigh channel, there needs to be a channel code in place to detect and correct the errors induced into the sent codewords. It is noted also that the block codes are better as a channel coding technic due to a memory overhead which adds encoding complexity to convolutional codes. Consequently, a brief study of the linear block codes is done. Decoding channels are also studied and the BSC's ability to give a result is the reason why it is chosen most of the times ad in this project also.

In chapter 2, LDPC codes are introduced, their matrix and graphical presentations are explored, particularly the Tanner graph. In a bid to construct an LDPC encoder, the creation of the parity check matrix is explored using two methods, the Gallagher construction and the Neal and MacKay construction. For their better irregular codes, the Neal and MacKay construction method is the used to encode a random source of binary stream of information using LU decomposition due to its lower complexity as compared to the systematic encoding.

In chapter 3, the iterative message passing decoding method called the sum-product algorithm is studied. The hard decision version of it called the bit flipping algorithm is studied in detail together with the soft decision. To aid the study of the soft decision decoding, the probability domain, the log domain and the simplified log domain decoding methods are used. A study also of the limitations of the detectable and correctable errors is also done and the all depend on the minimum distance of the code.

In chapter4, are studied the performances of the four decoding algorithms presented in chapter3 namely the effects of signal to noise ratio on BER, the effects of number of iterations on the BER, the effects of code length on BER and the effects of the channel type.

The first case studied as the effect of the signal to noise ratio on the results of the 4 decoders with fixed N=2000 and fixed k=3. It was found that the soft decoders converge after 3dB and the hard decoder after 6dB. The Bit flipping decoder is less efficient than the other decoders. It was noticed also that the Log Domain and the Probability Domain give the same results and they are almost the same as the Simplified Log Domain.

The second case was the investigation of the effect of the number of iterations with fixed code length (N=2000). It was found that the probability domain decoder and the simplified log domain decoder give almost the same results and they are more efficient than the bit flipping decoder.

The third case which is studied is the effect of code length with fixed number of iterations k=3. It was found that the convergence is best for N=2000

The last case which is studied is the effect of the type of channel on the results of decoders. All the decoders give better results with AWGN channel.

We can suggest for future work, to use the LDPC decoder with different modulation schemes QAM, M-PSK, and using real data like images, voice etc.

Bibliography

1. Gallagher RG. Low-density parity-check codes. Cambridge, Mass.: MIT-Press; 1963
2. Richardson T, Urbanke R. Modern Coding Theory. Cambridge: Cambridge University Press; 2008
3. Shannon CE." A Mathematical Theory of Communication", The Bell System Technical Journal, 1948, vol.27 pp 379-423,623-656 :55.
4. Proakis JG, Salehi M. Digital communications. 5th ed. Boston: McGraw-Hill; 2008. 1150 p.
5. Additive White Gaussian Noise Channels [Internet]. [cited 2018 Jan 14]. Available from: http://www.wirelesscommunication.nl/reference/chaptr05/digimod/awgn.htm
6. Neubauer A, Freudenberger J, Kuehn V. "Coding Theory: Algorithms, Architectures, and Applications", John Wiley and Sons, 2007, p 355:27-30.
7. Shu Lin DJCJ. "Error Control Coding Fundamentals and Applications", Prentice Hall, 1983, Electrical Engineering Series, p 624 :3-4
8. Simon Haykin. "Communication Systems", John Wiley and Sons, 4th ed, 2001, p 816:581-590,632-634,685-687
9. Costello DJ. Jr "An Introduction to Low-Density Parity Check Codes", Dept of Electrical Engineering Notre Damme University, 2009 :78.
10. Sasmita. Interview Questions and Answers on Information Theory [Internet]. Electronics Post. 2017 [cited 2018 Apr 10]. Available from: https://electronicspost.com/interview-questions-and-answers-on-information-theory/
11. Tanner R. "A recursive approach to low complexity codes", IEEE Trans Inf Theory. 1981 September; vol.27(5):533–547.
12. Richardson TJ, Shokrollahi MA. "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes", IEEE Trans Inf THEORY. 2001;47(2):619-635.
13. MacKay DJC. "Good Error-Correcting Codes Based on Very Sparse Matrices", IEEE Trans Inf Theory, 1999; vol.45(2):399-432.

14. Qi H, Goertz N. "Low-Complexity Encoding of LDPC Codes: A New Algorithm and its Performance", Institute for Digital Communications Joint Research Institute for Signal & Image Processing School of Engineering and Electronics University of Edinburg, Scotland UK, :6.

15. Patrick O. Wheately,Curtis F. Gerard. "Applied Numerical Analysis", Addison-Wesley Publishing Company, 3rd ed, 1984, 579 p.

16. Software for Low Density Parity Check Codes [Internet]. [cited 2018 Jul 26]. Available from: http://www.cs.utoronto.ca/~radford/ftp/LDPC-2006-02-08/index.html

17. Sonia E, Gupta ES. "HARD DECISION AND SOFT DECISION DECODING ALGORITHMS FOR LDPC AND QC-LDPC CODES", International Journal of Computer Science and Mobile Computing, vol.4(9), September 2015; p 182-191

18. Kschischang F, J. Frey B, Loeliger HA. "Factor Graphs and the Sum-Product Algorithm", IEEE Transactions on Information Theory 47(2),2001, p498 - 519.

19. Neufeld E. Judea Pearl. "Probabilistic reasoning in intelligent systems: networks of plausible inference. Series in representation and reasoning", Morgan Kaufmann, San Mateo, 552 pp. Journal of Symbolic Logic, 1993 June; vol. 58(2):721–721.

20. Moon TK. "Error Correction Coding, Mathematical Methods and Algorithms", John Wiley and Sons, 2005, p 756: 83-88,634-654..

21. W. E. Ryan, *An introduction to LDPC codes*, in Coding and Signal Processing for Magnetic Recoding Systems (Bane Vasic, ed.), CRC Press, 2004

22. T. J. Richardson, “Error floors of LDPC codes,” in 41st Annual Allerton Conference on communications, Control and Computing, Oct. 2003, pp.;1426–1435