N° d'ordre : 2010-XX-XXX

THÈSE

présentée

à l'Université de Cergy-Pontoise
École Nationale Supérieure de l'Électronique et de ses Applications

pour obtenir le grade de :

Docteur en Science de l'Université de Cergy-Pontoise
Spécialité : Sciences et Technologies de l'Information et de la Communication

Par

Bilal Shams

Équipes d'accueil :
Équipe Traitement des Images et du Signal (ETIS) - CNRS UMR 8051
et
Digital Solutions for Innovative IPs Team - STMicroelectronics Crolles

Titre de la thèse :

Les codes LDPC non-binaire du nouvelle generation

Soutenue le 08-12-2010 devant la commission d'examen composée de :

Didier Demigny	IUT (Lannion), Université Rennes-1	Examinateur
Valentin Savin	CETA-LETI, Grenoble	Examinateur
Emmanuel Boutillon	LAB-STICC, UBO, Lorient	Rapporteur
Laura Conde-Canencia	LAB-STICC, UBO, Lorient	Rapporteur
Dr. Jossy Sayir	University of Cambridge, UK	Rapporteur
Vincent HEINRICH	STMicroelectronics, Crolles	Encadrant
David DECLERCQ	ENSEA - Université de Cegy Pontoise	Directeur de thèse

Author's Publications

■ INTERNATIONAL CONFERENCES

$[C_1]$ B. Shams, D. Declercq, V. Y. Heinrich, "Non-binary split LDPC codes defined over finite groups", in Proc. of IEEE ISWCS'09, Siena-Tuscany, Italy, Sept. 2009

$[C_2]$ B. Shams, D. Declercq, V. Y. Heinrich, "Improved decoding architecture for non-binary split codes defined over finite groups", in Proc. of IEEE ICFCC'10, Wuhan, China, May 2010

Dedication

To Dr. Shams Rahman, my dad
also a Ph.D.
who is in the Heavens now

Acknowledgements

I carried out my thesis at the laboratory ETIS (Equipe Traitment d'Image et du Signal) at ENSEA (Ecole Nationale Supérieure de l'Electronique et ses Application and STMicroelectronics, crolles.

I would like to thank both teams at ETIS and STMicroelectronics, who welcomed me into their teams and gave me the oppertunity to develop myself into a better professional.

My foremost thanks to Prof. David Declercq who supervised me during my Ph.D. He has always been kind, patient and encouraging. I take great pride in having to work with him, who is a big name in the field of LDPC codes. I feel greatly indebted to him and I have learnt a lot from him.

My utmost gratitudes for Vincent Heinrich who was my supervisor at STMicroelectronics. He has always been cool, calm and dynamic. He has taught me a lot about the corporate sector.

I would also like to mention the support of my darling wife Zakia durig my Ph.D. She has always been encouraging and motivated me during my hard times.

Above all, I am thankful to the Almighty for His guidance and the blessings he has always been showering on us all.

Abstract

In this thesis we present our work in the domain of non-binary decoding algorithm for general classes of non-binary LDPC codes. Binary Low-Density Parity-Check (LDPC) codes were originally presented by Gallager in 1963, and after some fundamental theoretical advancement, they were considered in standards like DVB-S2, WI-MAX, DSL, W-LAN etc. Later on, Non-Binary LDPC (NB-LDPC) codes were proposed in literature, and showed better performance for small lengths or when used on non-binary channels. However, the advantages of using NB-LDPC codes come with the consequence of a heavily increased decoding complexity. For a code defined in $GF(q)$, the complexity is of the order $\mathcal{O}\left(q^2\right)$. Similarly, the memory required for storing messages is of order $\mathcal{O}\left(q\right)$. Consequently, the implementation of an LDPC-decoder defined over a field order $q > 64$ becomes practically impossible. The main objective of the thesis is to develop reduced complexity algorithms for non-binary LDPC codes that exhibit excellent performance and are practically implementable. For better decoding performance, not only the decoding algorithm is important, but also the structure of the code plays an important role. With this goal in mind, a new family of codes called cluster-NB-LDPC codes was developed and specific improvements of the NB decoder for cluster-NB-LDPC codes were proposed. Our principal result is that we were able to propose decoders for cluster-NB-LDPC codes with reduced complexity compared to usual decoders for NB-LDPC codes on fields, without any performance loss in error correction capability.

In the first part of the thesis, we modify the EMS algorithm for cluster codes. We see that the direct implementation of the EMS algorithm to NB cluster-LDPC codes is not a feasible option. There is a loss in performance and an increase in decoding complexity. Therefore, we propose some modification in the procedure, which not only significantly improves the decoding performance but also decreases the decoding complexity. It places the same limits on the number of operations at the check nodes as the EMS algorithm for $GF(q)$-codes i.e. $\mathcal{O}\ (n_m log n_m)$, with $n_m << q$. We then propose another method, based on the diversity of cluster codes, to improve the performance of the EMS algorithm for cluster codes. It also helps in reducing the overall complexity of the decoder. In the end we compare the decoding performance using this method and analyze the effect on the decoding complexity.

In the last part of the chapter, we propose a new direction for the decoding of LDPC codes. It is based on the creation of lists of codewords that are local to the parity check nodes. The list is constructed recursively in a tree structure, which makes it a good candidate for hardware implementation. It is a new method and requires further improvement. As an initial report, we have obtained good results with less number of computations.

Keywords: Non-binary LDPC codes, simplified decoding of non-binary LDPC codes, EMS decoding algorithm, Non-binary cluster LDPC codes, local list based decoding

Résumé

Dans cette thèse, nous présentons nos travaux dans le domaine des algorithmes de décodage des codes LDPC non-binaires généralisés. Les codes LDPC binaires ont été initialement proposés par Gallager en 1963, et après quelques avancées théoriques fondamentales, ils ont été proposés dans des standards tels que DVB-S2, WI-MAX, DSL, W-LAN etc. Plus tard, les codes LDPC non-binaires (NB-LDPC) ont été proposés dans la littérature, et ont montré une meilleure performance pour de petites tailles de code ou lorsqu'ils sont utilisés sur des canaux non-binaires. Cependant, les avantages de l'utilisation de codes NB-LDPC impliquent une augmentation importante de la complexité de décodage. Pour un code défini dans un corps de Galois $GF(q)$, la complexité est d'ordre $\mathcal{O}\ (q^2)$. De même, la mémoire requise pour le stockage des messages est d'ordre $\mathcal{O}\ (q)$. Ainsi, l'implémentation d'un décodeur LDPC défini sur un corps de Galois pour $q > 64$ devient impossible dans la pratique. L'objectif principal de cette thèse est de développer des algorithmes avec une bonne performance et complexité réduite de sorte qu'ils deviennent implémentables. Pour une performance de décodage optimisée, non seulement l'algorithme est important, mais également la structure du code joue un rôle clé. Avec cet objectif à l'esprit, une nouvelle famille de codes appelés « cluster-NB-LDPC codes » a été élaborée ainsi que des améliorations spécifiques du décodeur non-binaire pour ces codes. Le résultat principal est que nous avons pu proposer des décodeurs pour les codes cluster-NB-LDPC avec une complexité réduite par rapport aux décodeurs classiques pour les codes NB-LDPC définis sur les corps de Galois, sans aucune perte de performance dans la capacité de correction

d'erreur.

Dans la première partie de la thèse, nous avons modifié l'algorithme EMS pour les cluster-codes. La généralisation directe de l'algorithme EMS aux codes cluster-NB-LDPC n'est pas réaliste . Il y a une perte de performance et une augmentation de la complexité. Par conséquent, nous proposons quelques modifications dans la procédure, qui non seulement améliore considérablement les performances de décodage, mais diminue également la complexité. Au niveau des noeuds de parité, cet algorithme conserve les mêmes limites sur le nombre d'opérations que l'algorithme EMS pour $GF(q)$-codes, $\mathcal{O}\ (n_m log n_m)$ avec $n_m << q$. Nous proposons ensuite une autre méthode, basée sur la diversité des codes cluster, afin d'améliorer les performances de l'algorithme EMS pour les codes cluster-LDPC. Il contribue également à réduire la complexité globale du décodeur. Finalement, nous comparons les performances de décodage en utilisant cette méthode et analysons l'effet sur la complexité de décodage.

Dans la dernière partie du chapitre, nous proposons une nouvelle direction pour le décodage des codes LDPC. Elle est basée sur la création des listes des mots de code qui correspondent à des noeuds de parité. Les listes sont construite de manière récursive dans une structure en arbre, ce qui en fait un bon candidat pour l'implémentation matérielle. Il s'agit d'une méthode nouvelle et doit encore être améliorée mais à premiére vue nous avons obtenu de bons résultats avec un nombre réduit d'operations.

Mots de clé: Les codes LDPC non-binaires, décodage simplifié des codes LDPC non-binaires, L'algorithme EMS, cluster-LDPC codes non-binaires, décodage basée sur les listes

Contents

List of Figures

List of Tables

Abbreviations

APP	A-posteriori probability
AWGN	Added White Gaussian Noise
BER	Bit Error Rate
BIAWGN	Binary Input Additive White Gaussian Noise (channel)
BP	Belief Propagation
ECC	Error Correcting Codes
EMS	Extended Min-Sum
FER	Frame Error Rate
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Arrays
FT	Fourier Transform
GF	Galois Field
HDD	Hard Decision Decoder
ISI	Inter Symbol Interference
LDPC	Low Density Parity Check (code)
LUT	Look-up table
ML	Maximum Likelihood
MPA	Message Passing Algorithm
MSA	Min-Sum Algorithm
NB-LDPC	Non-binary Low Density Parity Check
PCM	Parity Check Matrix
PDF	Probability Density Function
SNR	Signal-to-Noise Ratio
SPA	Sum Product Algorithm

1

Introduction - Context and Background

1.1 Context and background

In this thesis we present our work in the domain of error correction codes (ECC), which makes possible the transmission of data over imperfect channels with the least possible errors. In this chapter we introduce the basic concepts explored in the thesis. We discuss the imporantance of LDPC codes in communication systems and the interest of using non-binary LDPC codes over their binary counterparts which is the main motivation behind this work.

1.1.1 Introduction

Figure 1.1 depicts the block diagram of a traditional communication system. The transmitter and receiver are the two entities communicating with each other whilst the channel adds imperfection to the transmission. The transmitter is composed of mainly three elements: source and channel encoders and a modulator. Similarly, at the receiver there is the de-modulator and the two decoders. The source encoder removes redundancy in the data received from the source, thus compressing the data. This helps in increasing the data rate of the transmitter. On the other hand, the channel encoder adds redundancy to the data received from the source encoder. This helps in making the data robust to the errors introduced by the channel. The ratio of the information data and the transmitted data forms the coding-rate **R** of the code, which is a parameter of the coding system. The modulator modulates the data as per the type and order of modulation installed. The channel adds degradation into the transmitted data. This includes ambiant noise generally modeled as additive white gaussian noise (AWGN), inter-symbol interference (ISI) due to multi-path fading and multi-user interference in multi-user systems.

The AWGN transmission channel is generally modeled as noise b with zero mean and variance σ^2, $(\mathcal{N}\ (0, \sigma^2))$. Consider a transmitted finite energy signal x. The received signal is then modelled as $y = x + b$. This transmission defines a gaussian channel characterized by the ratio of the signal energy and energy of noise E_b/N_0.

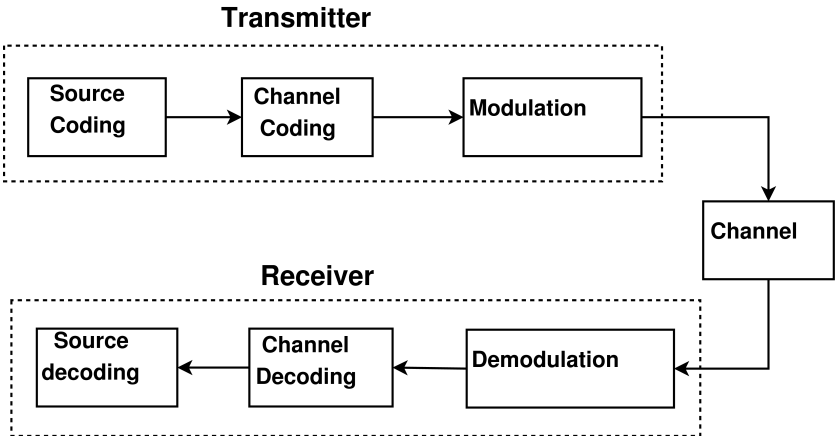

Figure 1.1: Block Diagram of a communication system

At the receiver, the demodulator converts the modulated signal into bit streams. The channel decoder applies an error correcting procedure to correct the errors introduced by the channel. The source encoder then decompresses the received information and reconstructs the transmitted data.

The principle of error correcting codes is to add redundancy to the data to be transmitted and then use this redundancy to re-create the data after it has been distorted by the channel. The channel encoder encodes a message of K bits to a codeword of N bits. The K bits are called as the information bits whereas the $M = N - K$ redundant bits are called as the parity bits. The ratio $R = K/N$ is called the rate of the code.

$$\{0,1\}^K \Rightarrow \mathrm{C} \subseteq \{0,1\}^N \tag{1.1}$$

Error correction methods consist in finding the codeword having the smallest distance with the received signal. The capacity of an error correcting code is measured as the minimum distance d_{min} of the code, which is the smallest distance between two elements of the codeword set $\mathcal{C}$. If the received signal has a distance greater than $d_{min}/2$ from the transmitted codeword, there is a possiblility that the codeword nearest to the received signal is not the codeword which was initially transmitted. Therefore d_{min} plays an important role in terms of the error correcting capability of a code. From a geometrical point of view, the codewords form hyper-spheres of radius $d_{min}/2$ in a N-dimensional space. If the received signal is outside the sphere, the probability of decoding a wrong codeword increases.

The decoding procedure is thus an algorithmic problem in which we search the nearest codeword to the received signal in a multidimensional space. The optimal solution is thus an extensive sequential search of the nearest codeword which can make the de-

coding procedure quite complex i.e. of the order $O(2^K)$. Thus various sub-optimal but faster solutions were proposed.

1.1.2 History

In 1948, Claude E. Shannon laid the mathematical foundation of modern information theory and gave the very first systematic framework for communication in his landmark paper [Sha48]. He introduced the concept of redundant channel coding as a method to achieve reliable communication on a noisy channel with known capacity. In particular, he proved that for sufficiently long codes arbitrarily reliable communication is possible at any coding rate below the capacity. Since then, the challenge of channel coding has been to design practical coding solutions that approach the channel capacity.

During the following decades, a lot of coding families were invented e.g. Hamming codes, Golay codes, Reed-Muller codes, convolutional codes, BCH codes, Reed-Solomon codes are just a few to mention. The goal was to construct codes with good properties and to find low complexity algorithms which are able to perform near optimum decoding for these codes. In 1993, Turbo codes [BGT93] were discovered which proved to be a major breakthrough for reliable communication through noisy channels. A practical coding scheme was presented that approaches the channel capacity within 1 dB at a bit error rate (BER) of 10^{-6}. More importantly it showed the potential of iterative decoding as a mean of approaching the channel capacity. Since then, there has been a tremendous amount of research on codes that are iteratively decodable and generally defined by graphs.

Low-Density Parity-Check (LDPC) codes were originally presented by Gallager [Gal62] in 1963, but received little attention at that time. The large computational needs for the decoding of long LDPC codes prevented their widespread use until major advances were made in computing, which eventually allowed cost-effective decoding implementation. With the breakthrough of turbo codes, they were rediscovered [Mac97] and regained more attention. However, one noticable and important exception is the work of Tanner [Tan81], in 1981, in which he generalized LDPC codes and introduced a graphical representation of LDPC codes, called as Tanner graph. Tanner graphs of LDPC codes are bipartite graphs that contains two types of nodes which are connected via edges. They are called the variable and check nodes representing the symbols and the parities respectively.

More recently, binary LDPC codes were considered in standards like DVB-S2, WIMAX, DSL, W-LAN etc. However, they start to show their weakness when the code size is small or moderate and when higher order modulation is used for transmission. Non-binary LDPC (NB-LDPC) codes were proposed [DM98] and they showed better

performance for codes of smaller length and defined in higher field orders. NB-LDPC codes then became candidates for future communication systems.

There are three aspects related to the design of a good channel decoder: its performance, the technology and its usage. The performance aspect plays an important role towards the quality of transmission. From a technological point of view, the decoding algorithm is generally composed of heavy computations and it must be capable of performing these computations with the least latency and highest data rate. This makes it necessary to envisage the decoding algorithm with a reduced complexity of implementation. Hence, in order to realize the hardware implementation of the decoder, we have to make a compromise between performance and complexity. On the other hand, digital communication systems are being increasingly used in the daily lives of the general public (mobile phones, digital TVs, PDAs, GPS etc.). More and more devices are being introduced and the number of communicating devices has increased. This results in the evolution of the use of technology and thus has two consequences. Firstly, there is a direct impact on the consumption characteristics and secondly the flexibility necessary to adapt the devices to different standards and technologies. Alongside, the prices of the circuits manufacturing must also be controlled.

1.2 Motivation and objective of the thesis

Binary-LDPC codes are based on the verification of parities of a matrix using for decoding, called the parity check matrix (PCM). The PCM has a very low density of ones, such that as the size of the matrix tends to infinite, the ratio of the non-zero elements to the null elements tends to zero [Rya03]. Non-binary-LDPC codes are a direct generalization of the binary case and the non-zero elements of the PCM are then defined in a Galois field, denoted as $GF(q)$. The NB-PCM is a low density matrix and it is represented by the bipartite Tanner graph.

It is widely accepted that the degradation in performance of an LDPC decoder comes due to dependance of the messages passed betweeen the nodes of the Tanner graph. This dependance is induced because of the topological structure of the code which includes cycles, stopping and trapping sets. This dependance is even more enhanced when the messages used to initialize the decoder are already correlated by the channel. NB-LDPC codes help in reducing the dependance of the messages for the following main reasons:

- For the same code-rate and binary-length, the graph corresponding to a NB-LDPC codes is typicaly less dense as compared to the graph corresponding to the a binary-LDPC code [MD99]. As a consequence, the graph corresponding

to the NB-LDPC code has better topological properties i.e. larger girth and less stopping and trapping sets.

- For a binary LDPC code transmitted with a high order modulation, the Maximum a-posteriori (MAP) demapper creates probabilities at the binary level for the decoder. This implies that decoder input messages are correlated even in the absence of cycles. However, if the LDPC code is defined in the same or higher order of Galois field as the order of modulation, the NB-LDPC decoder is initialized with non-correlated messages. Therefore, NB-LDPC codes perform better for high order modulation [SF02].

However, the advantages of using NB-LDPC codes come with the consequence of an heavily increased decoding complexity. For a code defined in $GF(q)$, the complexity is of the order $\mathrm{O}(q^2)$ [WSM04a]. Similarly, the memory required for storing messages is of order $\mathrm{O}(q)$. Consequently, the implementation of an LDPC-decoder defined over a field order $q > 64$ becomes practically impossible.

The main objective of the thesis is to develop a reduced complexity algorithm for non-binary LDPC codes that exhibit excellent performance together with an efficient parallel implementation. For better decoding performance, not only the decoding algorithm is important, but also the structure of the code plays an important role. With this goal, NB-LDPC codes defined over finite groups were proposed [CPD$^+$09]. A special case of group-LDPC codes is when the variable nodes are processed in a smaller group-order as compared to the check nodes. We refer to this family of codes as NB cluster-LDPC codes. They are designed to have a higher minimum distance and are foreseen to be good candidates for high rate applications. For this reason, we have concentrated our work on this family of codes and we propose decoding algorithms for NB cluster-LDPC codes.

1.3 Thesis organization

In chapter 2, we explain the basics of non-binary LDPC codes. In the first part of the chapter, we explain the graphical and matrix representation of non-binary LDPC defined over finite Galois fields and their generalization to finite groups. We then introduce the concept of clusters-LDPC codes defined over finite groups. In the second part of the chapter, we present a detailed and state of the art description of various non-binary LDPC decoders proposed in literature. In the third and last part of the chapter, we make a comparison between the complexities of various decoders.

In chapter 3, we present the generalization of the EMS algorithm to NB cluster-LDPC codes. We refer to it as the cluster-EMS algorithm. In the second part of the chapter,

we propose some improvements in the decoding procedure to adapt the EMS algorithm to cluster-codes in a better manner. We also present a hardware architecture for our proposed improved cluster-EMS algorithm and compare its complexity to other algorithms. In the third part of the chapter, we then propose a method to reduce the surface area of the decoder. The method is based on the implementation of several instances of a check update process in parallel with a very low complexity and then fusioning the output of each instance to compute the extrinsic output of the check node. We also analyse the effect on the area of the decoder.

In chapter 4, we explain our second proposed decoder which is based on the concept of list decoding. The decoding algorithm is based on the creation of lists based on local parities at the check nodes level and therefore, we call it local-list decoder. In the second part of the chapter, we propose an idea to reduce the complexity of the local-list decoder, but our proposed idea was not supported by Monte-Carlo simulations. However, for the sake of information, we explain the idea.

In chapter 5, we conclude the manuscript and present the summary and perspectives of the work.

2

Non-binary LDPC codes

THIS chapter presents the background and state of the art information about non-binary LDPC codes. In the first part of the chapter, we present the structure and representation of the various families of LDPC codes. We start by introducing NB-LDPC codes defined over finite Galois fields and explain the Belief-Propagation (BP) algorithm used for decoding. After that, we move towards NB-LDPC codes defined over groups and emphasize on a special case of NB-LDPC codes defined over finite groups which we term as cluster NB-LDPC codes. In the second part of the chapter, we introduce various low complexity decoding algorithms that have been proposed in the literature. As a conclusion to the chapter, we make a comparison between the decoding complexity and performance for some of these algorithms.

2.1 Classification of Non-binary LDPC codes

Low density Parity check codes (LDPC) are a class of linear block codes that uses a generator matrix for encoding and a parity check matrix (PCM) for decoding. The name low density parity check comes from the fact that the PCM has a very low density of non-zero elements as compared to the number of zero elements. By definition, the PCM ratio between the number of non-zero elements and the number of zero elements tends to zero as the size of the matrix tends to infinity. The PCM is represented in the form of a Tanner graph which is a bipartite graph with two types of nodes. This facilitates the use of an iterative decoding algorithm where messages are being passed between the two sets of nodes. The messages represent in some sense *reliabilities* of the symbol values. The decoder operates in an iterative fashion, and A-posteriori probabilities (APP) are calculated at the end of each iteration to make a decision on the received symbols.

We first explain the structure and decoding of codes defined over finite Galois fields and then detail the concept of LDPC codes defined over finite groups. We then explain an interesting family of codes which we call cluster-LDPC codes.

p	Primitive Polynomials
1	$1+x$
2	$1+x+x^2$
3	$1+x+x^3$, $1+x^2+x^3$
4	$1+x+x^4$,$1+x^3+x^4$
5	$1+x^2+x^5$,$1+x+x^2+x^3,x^5$,$1+x^3+x^5$,$1+x+x^3+x^4+x^5$, $1+x^2+x^3+x^4+x^5$,$1+x+x^2+x^4+x^5$

Table 2.1: Primitive Polynomials

2.1.1 NB-LDPC codes defined on finite Galois fields

Galois Fields

A Galois field is a finite field with a finite order, which is either a prime number or the power of a prime number. A field of order $n^p = q$ is represented as $GF(n^p)$ or $GF(q)$ and it contains q-elements which are denoted as $\{0, 1, \alpha, \alpha^2, ..., \alpha^{q-2}\}$ respectively. A specific type called as *characteristic-2* fields are the fields when $n = 2$. All the elements of a *characteristic-2* field can be represented in a polynomial format [PFD06a]. The primitive polynomial of the field is defined to be an irreducible polynomial that generates all the other polynomials. As a result, the symbol represented by the primitive polynomial is called as the primitive symbol α of the field, the powers of which construct all the other elements of the field. For any finite field $GF(2^p)$, there exists a primitive polynomial of degree p over $GF(q)$ [PFD06b]. Table 2.1 lists the primitive polynomials for p equals 1 through 5. We can be observe that for $p = 1$, the field is a binary field and for $p \geq 2$, it represents a non-binary field. Binary LDPC codes are defined over a Galois field $GF(2)$, with $\{0, 1\}$ being the field elements. Hence non-binary LDPC codes can be considered as a direct generalization of binary LDPC codes.

Each element of a *characteristic-2* field is represented by a polynomial with binary coefficients. Table 2.2 shows the example for $p = 3$ while considering the primitive polynomial $1 + x + x^3$. The field consists of 8 elements and each one has a binary representation composed of the binary coefficients of the associated polynomial. With this representation finite field addition and multiplication becomes polynomial addition and multiplication, where the addition is modulo-2. We will see later in the chapter that the PCM of a non-binary LDPC code can also be represented in a binary matrix form using the binary coefficients of the polynomial representation.

Element	Binary Rep.	Polynomial Sum
0	000	0
α^0	100	1
α^1	010	α
α^2	001	α^2
α^3	110	$\alpha+1$
α^4	011	$\alpha^2+\alpha$
α^5	111	$\alpha^2+\alpha+1$
α^6	101	α^2+1

Table 2.2: Binary and Polynomial representation of Finite Field $GF(8)$

The parity check matrix

An LDPC block code defined over a finite Galois field $GF(2^p)$ is described as a K-dimensional subspace $\mathcal{C}$ of the vector space $GF(2^p)^N$ of N-tuples. Hence the code $\mathcal{C}$ has a total of $(2^p)^K$ codewords. The code $\mathcal{C}$ is a subspace and is spanned by the basis:

$$\mathbb{G} = \{g_0, g_1, ..., g_{K-1}\} \tag{2.1}$$

Each codeword $c \in \mathcal{C}$ is then written as:

$$c = u_0 g_0 + u_1 g_1 + ... + u_{K-1} g_{K-1} \tag{2.2}$$

for any vector $\{u_i\}$. In matrix form, this equation can be written as $c = \mathbf{u}.G$ where $\mathbf{u} = [u_0, u_1, ..., u_{K-1}]$ and G is the $(K \times N)$ generator matrix whose rows are the vectors $\{g_i\}$ of the basis.

$$C = \{c = u.G, \forall u \in GF(2^p)^K\} \tag{2.3}$$

The $M = (N-K)$ dimensional dual space $\mathcal{C}^\perp$ of $\mathcal{C}$ comprises of vectors $x \in GF(2^p)^N$ for which $x\mathcal{C}^{\perp} = 0$. The basis of $\mathcal{C}^\perp$ is represented as:

$$\mathbb{H}^\perp = \{h_0, h_1, ..., h_{M-1}\} \tag{2.4}$$

Therefore, this implies that:

$$C^\perp = \{c \in GF(2^p)^N : H.c \overset{GF(2^p)}{=} 0\} \tag{2.5}$$

where H is a $(M \times N)$ matrix whose rows are the $\{h_i\}$ elements of the basis. It is the generator matrix of the dual space $\mathcal{C}^\perp$ and is called as the parity check matrix (PCM) of the code $\mathcal{C}$. The rows of H can also be seen as the coefficients of a linear equa-

tion system. Each equation is called as a parity check equation as it performs the test whether the received word verifies the parity. There are $M = N - K$ such tests and it gives the name parity check matrix (PCM). The PCM for a (10×5) LDPC code is shown in Fig. 2.1.

$$\mathbf{H} = \begin{bmatrix} h_{00} & h_{01} & h_{02} & h_{03} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & h_{10} & 0 & 0 & 0 & h_{11} & 0 & 0 & h_{12} & h_{13} \\ 0 & 0 & h_{20} & 0 & h_{21} & 0 & 0 & h & h_{22} & 0_{23} \\ h_{30} & 0 & 0 & h_{31} & h_{32} & 0 & h_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & h_{40} & h_{41} & h_{42} & 0 & h_{43} \end{bmatrix}$$

Figure 2.1: A non-binary LDPC parity check matrix

The original PCM proposed by Gallager had equal number of non-zero elements in all rows and columns, d_c and d_v respectively. Such type of a code is called as a regular LDPC code. However a code with a different number of non-zero elements in its rows and columns is termed irregular LDPC codes.

Using the generator matrix G, for a frame of length K, the encoders create a codeword of length N, where $N - K$ are the redundant symbols. The rate of the code can be defined as:

$$R = \frac{K}{N} \tag{2.6}$$

The rate of a code helps in determining the data-rate of the communication system as it defines the total number information sent versus the redundancy added to the transmission.

The Tanner graph representation

In 1981, Tanner presented a bipartite graphical representation of binary LDPC codes which is called as a Tanner graph after his name [Tan81]. A graph is called bipartite if it has two disjoint sets of nodes which are connected via edges and no two edges of the same set are connected to each other. The two sets of nodes represent the variables and the functions between them respectively. In the case of LDPC codes, the variables are the symbols of the codeword and are thus called as variable nodes. The functions between the variables correspond to the parity check equations and thus, these nodes are called as parity check nodes.

Consider a $(M \times N)$ regular-LDPC code, with d_v and d_c non-zero elements in the PCM at each column and row respectively. The Tanner graph is constructed by associating a variable node to each column and a parity node to each row of the PCM.

There are M parity check nodes and N variable nodes. For each non-zero element h_{ij}, an edge is created between the i^{th} parity node and the j^{th} variable node. This way each parity node is connected to d_c variable nodes and likewise each variable node is connected to d_v parity check nodes. d_c and d_v are termed as the degree of connection of the check and variable nodes respectively.

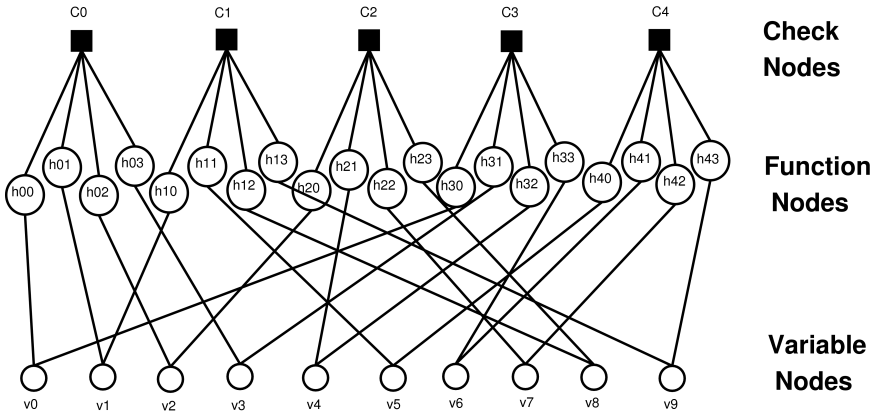

Figure 2.2: The Tanner Graph for a non-binary parity check matrix

When the non-zero elements h_{ij} are the elements of a non-binary finite field $GF(q)$, we add another class of node called as the function nodes [DF07]. These nodes correspond to multiplication of the codeword symbols with the non-zero elements $h_{ij} \in GF(q)$. After multiplication, the messages at the check nodes input become independent of the non-binary values of the PCM and they can be processed concurrently. Fig. 2.2 represents the Tanner graph for the PCM of Fig. 2.1 with $(M, N) = (5, 10)$ and $(d_c, d_v) = (4, 2)$.

The function node models the multiplication of the symbols c_j and the non-zero elements h_j by a cyclic permutation of the message values [DF07] and this is why they are also referred as permutation nodes. A cyclic permutation of a message U_{vp} is shown in Fig. 2.3. The LLR value of the symbol 0 remains at the same index and all the other values are shifted in a cyclic fashion by α^3.

A *cycle* in a Tanner graph is represented by a path that finishes at the same node from where it had emerged. The shortest cycle of a graph is called as its *girth*. The smallest possible girth of an LDPC code is 4. This is because the Tanner graph is a bipartite graph where two similar types of nodes do not interconnect directly. Cycles bring in dependance between the transmitted which as a consequence degrades the decoding performance of the code. Therefore, it is desirable to avoid cycles, specially short length cycles.

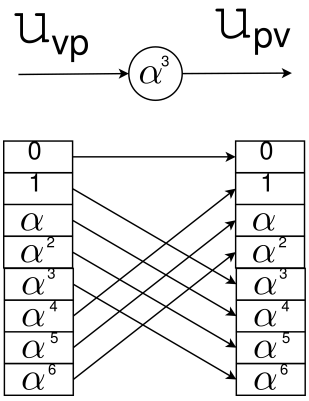

Figure 2.3: A cyclic permutation of a message at the permutation node

The binary representation of the non-binary PCM

When the field order of a NB-LDPC code is a power of 2, the non-binary elements of the field have a polynomial representation [PFD06b], which in turn provides a binary representation of the field elements.

Consider a $(M \times N)$ regular NB-LDPC code defined in a Galois field $GF(q)$, with $q = 2^p$ as the order of the field. The degrees of connection of the variable and check nodes are (d_v, d_c) respectively. The non-binary parity check matrix H associated to the code has its non-zero elements belonging to the Galois field $GF(2^p = q)$. The non-zero elements belong to the set $S = \{\alpha^i : i = 0, ..., q-2\}$, with α being the primitive element of the field. We define a primitive polynomial of degree p for the field.

$$p(x) = a_0 + a_1 x + a_2 x^2 ... + x^p \tag{2.7}$$

A companion matrix A of size $p \times p$ is also associated to the primitive polynomial [PFD06a].

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & ... & 0 \\ 0 & 0 & 1 & 0 & ... & 0 \\ 0 & 0 & 0 & 1 & ... & 0 \\ . & . & . & . & ... & . \\ 0 & . & . & . & ... & 1 \\ a_0 & a_1 & a_2 & a_3 & ... & a_{p-1} \end{bmatrix}$$

where $[a_0, ..., a_{p-1}]$ are the coefficients of the primitive polynomial $p(x)$.

Since $p(x)$ is the primitive polynomial of the field $GF(2^p)$, A is called as its primitive element under matrix representation. As a result, the powers of the matrix A generate the binary matrix representations of all the other elements of the field. Thus

the non-zero elements h_{ij} of the PCM can be written in the form of a $(p \times p)$ binary matrices H_{ij}, where H_{ij} is the result of the tranpose of a power of the primitive matrix of the Galois field. Subsequently, the $(M \times N)$ NB-PCM can be written in the form of a $M_b \times N_b$ binary PCM, where $M_b = pM$ and $N_b = pN$ as shown in Fig. 2.4. The zero elements of the PCM are represented with all-zero matrices of size $(p \times p)$.

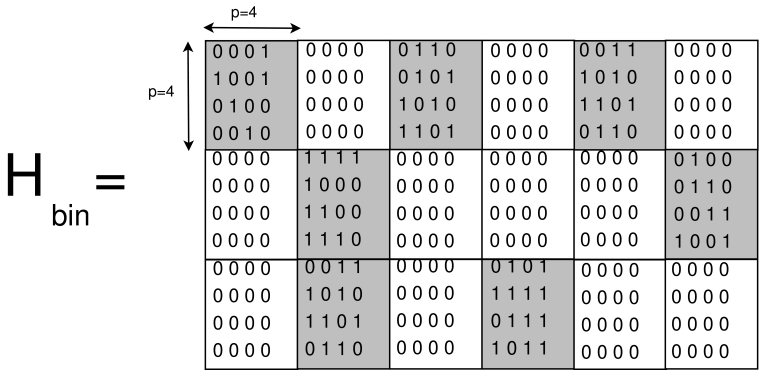

Figure 2.4: Binary representation of a GF(q)-LDPC code

The arithmetics on the $GF(q)$ elements are carried out using modulo-2 addition and multiplication of the polynomial representations. Similarly, the arithmetics on the matrix representation can be carried out using modulo-2 arithmetics over the matrix representations. The parity check equation can, thus, be written in the vectorial domain as:

$$\sum_{j:H_{ij}\neq 0} H_{ij} X_j^T = 0^T \tag{2.8}$$

where H_{ij} is the matrix representation of the Galois field element h_{ij}, X_j is the p-bits binary mapping of the symbol c_j and 0 is the all zero component vector.

As described earlier that the function nodes are added to a NB-Tanner graph which makes the check nodes independant of the non-binary elements of the PCM. The function nodes realize the multiplication of the messages with the non-binary elements by a cyclic permutation of the message values. For a function node bearing the value h_{ij}, the message values are shifted in a cyclic fashion by a value h_{ij}. The cyclic permutation of the message values can be obtained using the binary matrix representation H_{ij} of h_{ij} and the p-bits binary mapping $\{b_{\alpha^i}[k]\}_{k=0,\ldots,p-1}$ of the symbols $\alpha^i \in GF(q)$. The multiplication of the matrix H_{ij} with b_{α^i} results in the binary mapping of another

element α^j, where $(\alpha^i, \alpha^j) \in GF(q)$.

$$b_{\alpha^j} = H_{ij}.b_{\alpha^i} \tag{2.9}$$

This multiplication of subsequent elements are followed in a cyclic fashion and thus it results in a cyclic permutation of the message values as shown in Fig. 2.3.

The Belief propagation decoding algorithm

Gallager proposed a quasi-optimal decoding algorithm for binary LDPC codes which is termed Belief propagation (BP) [Gal62]. It is an iterative message passing algorithm which is also called as the Sum-Product algorithm (SPA). As the name belief propagation suggests, the algorithm is based on the propagation of messages composed of the probabilities of the symbols c_n. To each edge of the Tanner graph, two messages are associated for the two directions of propagation i.e. one from the variable nodes to the check nodes and the other in the reverse direction.

The algorithms used for the decoding of binary LDPC codes can also be generalised to non-binary LDPC codes defined over finite fields by employing the various operations in correspondance to finite fields. MacKay et al. generalised the belief propagation algorithm to non-binary LDPC codes defined over finite fields [DM98]. The messages flowing through the graph are of size q and they represent the probabilities of the symbols c_n, when the code is defined in a finite field $GF(q)$. For example, a message vector associated to a random variable z is written as:

$$P(z) = [\ P[0]\ \ P[1]\ \ P[\alpha]\ \ ...\ \ P[\alpha^{q-2}]\ ] \tag{2.10}$$

where $P[\alpha^i] = P(z = \alpha^i)$ represents the probability of the random variable z being equal to $\alpha^i \in GF(q)$.

The Tanner graph is processed in two steps:

1. First the variable nodes processes the inputs by a term by term multiplication of the $d_v - 1$ input messages to calculate the extrinsic outputs of the variable nodes.

2. The second steps involves the computation of the extrinsic outputs of the parity nodes while verifying the parity equations at each check node.

The i^{th}-parity equation is written as:

$$\sum_{j=0}^{d_c-1} h_{ij} \otimes c_j = 0 \tag{2.11}$$

where h_{ij} are the non-zero elements of the i^{th} row of the parity matrix H, c_j are the symbols of the codeword that participate in the parity and d_c is the degree of connection of the check node. The messages circulating between the nodes contain extrinsic information. This means that for any edge j on a node of degree d, the output on the edge j is calculated as a function of all the other inputs excluding the input message from edge j itself. This holds for both, the variable and check nodes.

The Belief propagation (BP) algorithm is based on the computation of the a-posteriori probability of the codeword symbols. For a transmitted codeword $c = [c_0 c_1 ... c_{N-1}]$, we are interested in computing the aposteriori probability (APP) of a given symbol α^j of a word c_i with in the transmitted codeword, given the received word $y = [y_0 y_1 ... y_{N-1}]$.

$$P(c_i = \alpha^j | y) \tag{2.12}$$

For the messages that traverse the Tanner graph, we use the representation as follows: U is a message flowing from the variable nodes towards the check node and V is a message in the reverse direction. Thus, $\{V_{p_i v}\}_{i=0...d_v-1}$ is the set of messages entering a variable node v of degree d_v and $\{U_{vp_i}\}_{i=0...d_v-1}$ is the set of output messages of the same variable node. The index $p_i v$ indicates the direction of propagation of the message i.e. from the permutation node p_i to the variable node v and vp_i in the reverse direction. Similarly, the sets $\{U_{p_i c}\}_{i=0...d_c-1}$ and $\{V_{cp_i}\}_{i=0...d_c-1}$ are the set of inputs and outputs of the check node c of degree d_c. Fig. 2.5 represents the Tanner graph of a single parity check and denotes the various messages traversing though the graph.

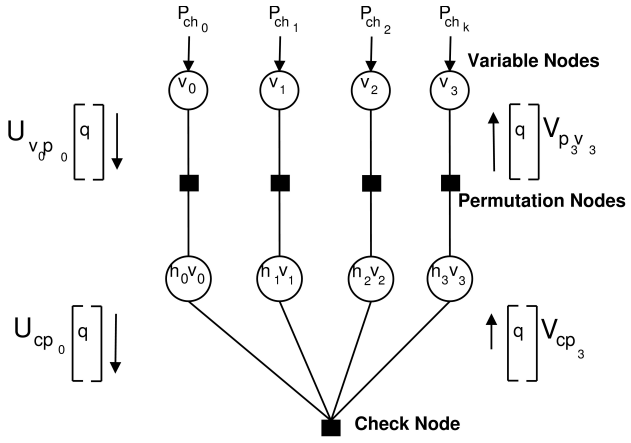

Figure 2.5: The messages flowing across a Tanner graph for the BP algorithm

The Belief Propagation (BP) algorithm is composed of six steps:

1 - Initialization:
During this step, all the messages U are initialized with the likelihood information

from the channel.

$$U_{vp_i}[\ 0\ ...\ \alpha^{q-2}\ ] = [\ P_{ch_v}[0]\ P_{ch_v}[1]\ ...\ P_{ch_v}[\alpha^{q-2}]\ ] \qquad , \quad v = 0, ..., N-1 \quad (2.13)$$

where $\{0, 1, ..., \alpha^{q-2}\} \in GF(q)$ are the elements of the field and:

$$P_{ch_v}[\alpha^i] = P(y_v|c_v = \alpha^i) \quad (2.14)$$

is the likelihood probability of the symbol α^i information calculated at the channel output.

2 - Variable nodes processing:
Consider a variable node v of degree d_v with the input messages $\{V_{p_k v}\}_{k=0...d_v-1}$ as shown in Fig.2.6. To calculate the output messages U_{vp_k} on the edge k, we consider all the input messages except for the input message on the edge k.

$$U_{vp_k}[\alpha^i] = \mu_{vp_k} P_{ch_v}[\alpha^i] \prod_{j=0, j\neq k}^{d_v-1} V_{p_j v}[\alpha^i] \quad (2.15)$$

where $\alpha^i \in GF(q)$, $k = 0, ..., d_v - 1$ and μ_{vp_k} is a normalization factor such that:

$$\sum_{i=0}^{q-1} U_{vp_k}[\alpha^i] = 1 \quad (2.16)$$

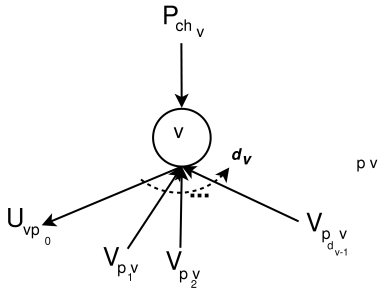

Figure 2.6: The degree d_v variable node update

3 - Cyclic-permutation nodes processing:
These nodes multiply the codeword elements with the non-binary elements h_p of the PCM. Since Finite Galois fields are cyclic and multiplication in GF(q) correspond to a cyclic permutation of the message values [DF07]. The message U_{vp} from the variable

node v is udpated by the permutation node p carrying a value h_p, to a message U_{pc} as:

$$U_{pc}[\ 0\ ...\ \alpha^{q-2}\ ] = \Psi_{h_p} U_{vp}[\ 0\ ...\ \alpha^{q-2}\ ] \tag{2.17}$$

where Ψ_{h_p} is the $(q \times q)$ permutation matrix that corresponds to the non-binary value h_p. All the values of the message are circularly shifted except for the value labelled by the symbol 0, which remains at its place. The permutation process is expressed as:

$$U_{pc}[h_p \otimes \alpha^i] = U_{vp}[\alpha^i] \tag{2.18}$$

where $\alpha^i \in GF(q)$ are the symbols of the Galois field. The permutation process is depicted in Fig. 2.3 in the previous section.

4 - Check nodes processing:
After the multiplication with the non-binary elements of the PCM, all the parity nodes have the same behaviour and they become independant of the non-binary values of the parity check matrix. Each check node then computes the probabilities of the symbols that verify the parity conditioned to the input messages of that check node. The check update process can be expressed as a convolution of the incoming messages [DF07].

$$V_{cp_k} = \circledast_{j=0, j\neq k}^{d_c-1} U_{p_j c} \tag{2.19}$$

The convolution can be realised as:

$$V_{cp_k}[\alpha^i] = \sum_{\delta:\ \sum_{m=0}^{d_c-1} \beta_{m(x)}=0} \delta \prod_{j=0, j\neq k}^{d_c-1} U_{p_j c} \tag{2.20}$$

with $v = \{0, 1, ..., d_c - 1\}$, $\alpha^i \in GF(q)$, β_m represent the symbols that take part in the parity and $\delta \in \{0, 1\}$ depending on whether the parity equation is verified.

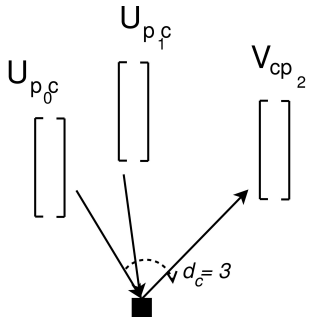

Figure 2.7: The degree $d_c = 3$ check node update

Fig. 2.7 depicts a check node of degree $d_c = 3$. The check node update eq. (2.20) can be explained as: The message value $V_{cp_2}[\alpha^i]$ is calculated as a sum of all the products, $U_{p_0c}[\alpha^j]U_{p_1c}[\alpha^k]$, that verify the condition $\alpha^j \otimes \alpha^k \otimes \alpha^i = 0$ with $\alpha^j, \alpha^k, \alpha^i \in GF(q)$ and $\otimes$ represents multiplication in the field.

5 - Reverse-permutation processing:
After the messages have been updated at the check nodes, they are permuted back in the reverse direction at the permutation nodes. The value h_p^{-1} is used for the reverse permutation. It is realised by using eq. 2.40 in the reverse direction.

$$U_{pv}[\alpha^i] = U_{cp}[h_p \otimes \alpha^i] \tag{2.21}$$

6 - APP Calculation and the codeword decision:
Finally, the a-posteriori probabilities (APP) of the symbols are calculated at the variables using the new probabilities. A decision is then made on each symbol based on the highest APP.

$$\hat{v}_n = \underset{\alpha^i \in GF(q)}{max} P_{ch}[\alpha^i] \prod_{j=0, j\neq k}^{d_v-1} V_{p_j v}[\alpha^i] \qquad , \quad \alpha^i \in GF(q) \tag{2.22}$$

The steps 2-6 are iteratively repeated until a valid codeword has been obtained or a fixed number of iterations has been completed. For a valid codeword to be obtained, the decoded codeword $\hat{v}$ must satisfy $H\hat{v} = 0$, where H is the parity check matrix. If the maximum number of iterations are completed without decoding a valid codeword, a decoding failure is declared.

The main obstacle in the path of the hardware implementation of the BP decoding algorithm is its computational complexity, the major factor being the check nodes processing, which is composed of a high number of additions and multiplications. The decoding complexity of a $GF(q)$ non-binary decoder is of the order $\mathcal{O}(q^2)$. Table 2.3 presents the no. of computations for a variable and check node of degrees d_v and d_c respectively. The nodes are processed with a recursive strategy called as the forward-backward (F/B) strategy [HDYW06], which is detailed in the next section.

Moreover, the normalization of the messages at the check nodes output, eq. 2.15, requires q divisions. As a consequence of the complexity, the hardware implementation for codes defined in order $q > 16$ can not be realised. Therfore, we directed our research towards proposing low complexity non-binary LDPC decoders and present our work in chapter-3 and chapter-4.

Process	**Additions** $(+)$	**Multiplications**$(\times)$	**Division** $(\div)$
BP variable update	$d_v q$	$(2d_v - 1)q$	-
BP check update	$3(d_c - 2)q^2$	$3(d_c - 2)q^2$	q

Table 2.3: Computational complexity of the BP algorithm

Binary LDPC codes vs. Non-binary LDPC codes

Binary LDPC codes have been extensively studied in literature and have been shown to approach the shannon limit performance for long length codes [RU01, CFRU01, LMSS01]. A lot of research has been carried out in finding ways to improve not only their decoding performance but also their structure which is better adapted to the transmission channel. In [RSU01], the authors proposed a method which allows us to predict the performance of binary LDPC codes in terms of probability of error. The proposed method is called as *density evolution* and it follows the evolution of the probability densities along with the propagated messages in the graph. However, it is difficult to analyse and predict the performance of NB-LDPC codes because the methods proposed for binary LDPC codes cannot be directly generalized to the non-binary case. Though, some authors have tried to generalize the methods based on density evolution for binary LDPC codes in order to optimise the construction of irregular NB-LDPC codes [LFK03, BB06, BT05, RU05]. However, the obtained profiles are not very accurate and could only be applied to very long codes. The difficulty is that the messages of a non-binary BP decoder are defined in high dimensional spaces, and it is in general difficult to keep track of their densities.

It is well known that the loss in performance of the BP algorithm comes from the inter-dependance of the messages flowing in the Tanner graph [YHB04]. The inter-dependance is created by specific topological structures that exist e.g. cycles, stopping and trapping sets. The reason is that the belief propagation algorithm is based on the application of Bayes rules locally at each check node, after which the a-posteriori probabilities are calculated for each variable node. In [KFL01], it was proved that for a cycle free graph, the local Bayesian function can be factorized to the a-posteriori probabilities of the symbols. In the cycle free case, all the messages are independant from each other, however, the presence of cycles bring along dependance between the messages which impairs the calculation of the exact a-posteriori probabilities. This determines that the BP algorithm is an optimal algorithm in the cycle free case and becomes sub-optimal with the introduction of cycles in the graph. The performance of the BP decoder is further decreased when the decoder is initialized with messages which have already been inter-correlated by the channel [Voi07].

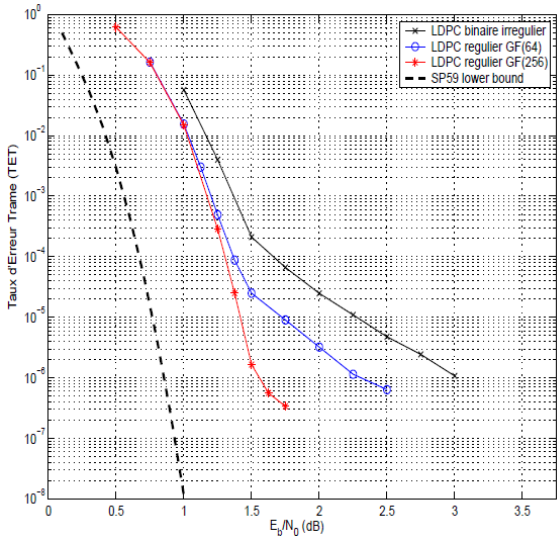

Figure 2.8: Binary vs non-binary LDPC codes, $N_b = 3008$ bits and $R = 1/2$

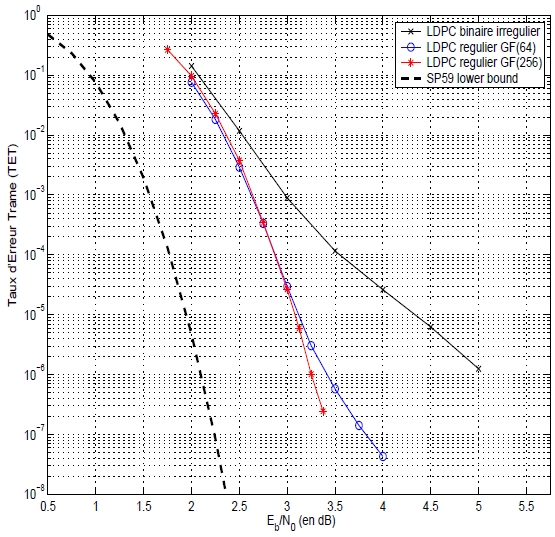

Figure 2.9: Binary vs non-binary LDPC codes, $N_b = 565$ bits and $R = 2/3$

NB-LDPC codes [DM98] have shown better performance as compared to their binary counterparts, specially for higher order Galois fields [SF02, DCG04, BB06, MWZ06] and for codes of smaller length [MD99, SZAG06, ZLT+08]. This is mainly because the graph of a NB-LDPC code is more sparse as compared to their homologous binary counterparts for the same rate and binary code length [HFE04, PFD06a]. As a

result, we have a graph associated to NB-LDPC codes with better properties in terms of the number of cycles and their minimum length. Hence, a NB-LDPC code helps in avoiding short length cycles and thus improves the performane of the BP algorithm by avoiding the correlation induced in the messages due to the topological structures. The shortest cycle in a graph is called its girth. The girth of an irregular binary code of length $N_b = 848$ bits and rate $R = 0.5$ has typically a girth equal to $g_b = 6$ whereas a NB-LDPC code of the same length and rate has a girth $g_{nb} = 14$, provided that we have well constructed codes [DCG04, VP08, BB06].

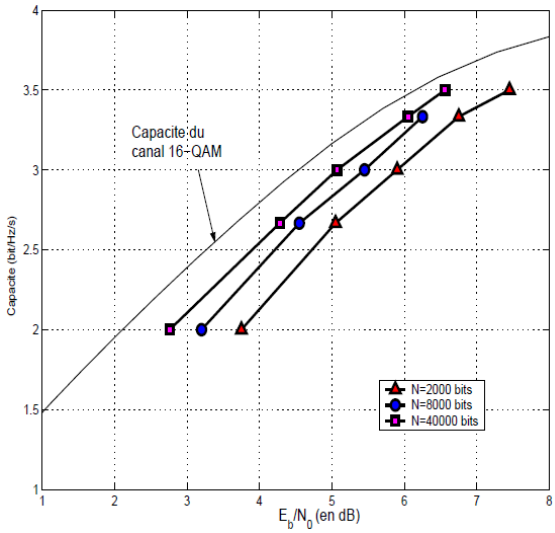

Figure 2.10: Performance of regular $GF(256)$ LDPC codes over a 16-QAM channel at a BER of 10^{-5}

It was also demonstrated that NB-LDPC codes have a better performance for channels with high spectral efficiency i.e. while using high order modulation [PML+09, BB06, DCG04, MD99]. For the case of a $M - QAM$ modulated transmission while using binary LDPC codes, the MAP demapper at the receiver side constructs symbol likelihoods which are inter-correlated at the binary level. This means that the decoder is initialized with already correlated messages even in the absence of cycles in the decoding graph. However, if the code is constructed in a non-binary Galois field with order greater than or equal to the modulation order, the decoder is initialized with uncorrelated messages which in turn improves the performance of the BP decoder. This was proved analytically and by means of simulations in [SF02].

Binary LDPC codes have shown good performance in the presence of burst errors [Tan81, WS98, YR01, PNF04]. However, NB-LDPC have shown even better performance in the presence of burst errors [MSV08, CWL05, LZL+07]. This is because

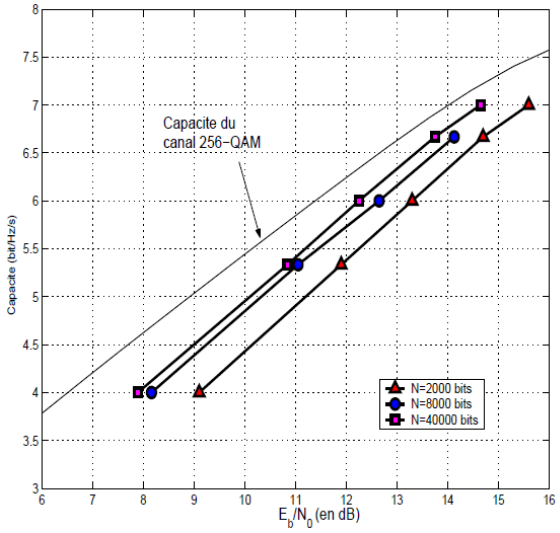

Figure 2.11: Performance of regular $GF(256)$ LDPC codes over a 256-QAM channel at a BER of 10^{-5}

consecutive bits are grouped together forming symbols in the non-binary field $GF(q)$. Moreover, NB-LDPC codes are being used in MIMO systems where they have shown better performance [JYgXL09, BF04] as compared to binary LDPC codes. It exhibits a better balance between performance and complexity when detection and decoding are performed jointly as compared to when done separately[PC06].

The main reason to use NB-LDPC codes in place of binary LDPC codes lies in the fact that for a finite length code, while using a sub-optimal low complexity algorithm, NB-LDPC decoders have a performance very close to the maximum-likelihood decoder (MLD) as compared to binary LDPC decoders [Voi07]. We now present the comparison of the error correcting performance of binary vs. non-binary LDPC codes.

Fig. 2.8 and 2.9 represent the performance comparison for short length codes transmitted over a binary-AWGN channel. The codes considered have the same rate and length. We see a gain in performance while using $GF(64)$ as compared to the binary codes. Similarly, while using $GF(256)$ codes, we also obtain a very low error floor.

We now present the performance comparison of NB-LDPC codes for high order modulations. The figures 2.10 and 2.11 represent the values of the E_b/N_o for which the frame error rate (FER) is 10^{-5} for different ultra-sparse $(d_v = 2)$ NB-LDPC codes defined over $GF(256)$. The codeword lengths considered were $N = 2000$, 8000 and 40000 bits. We can observe Fig. 2.10, that for $N = 40000$, there is a loss of 0.5dB as compared to the channel capacity for 16-QAM modulation. However, for 256-QAM

modulation in Fig. 2.11, the loss in performance from the channel capacity is increased to 1.2dB. The same behaviour is observed for $N = 8000$ and 2000 as well. This shows that using an order of field higher than the modulation order results in better performance.

Therefore, NB-LDPC codes are considered as the candidates for future communication systems.

2.1.2 NB-LDPC codes defined on general linear groups

In this section, we detail another class of low density parity check codes defined over finite linear groups $\mathbb{G}(q)$, which is a more general framework as compared to LDPC codes defined over finite Galois fields $GF(q)$. This allows us to consider a wider class of LDPC codes as compared to codes defined in a finite Galois field [SD06].

Parity check constraints defined over groups

Let $\mathbf{c} = [c_0...c_{N-1}]$ be a codeword of an LDPC code defined over a finite Galois field $GF(q)$, with $q \geq 2$. The i^{th}-parity equation is written as:

$$\sum_{j:h_{ij}\neq 0} h_{ij}c_j \overset{GF(q)}{=} 0 \tag{2.23}$$

where h_{ij} are the non-zero elements of the i^{th} row of the PCM. This definition can be generalized to the case where the elements of the PCM are defined over a general linear group $\mathbb{G}(q)$ [RU05]. With this definition, the i^{th}-parity equation becomes:

$$\sum_{j} f_{ij}(c_j) \overset{\mathbb{G}(q)}{=} 0 \tag{2.24}$$

where f_{ij} is a general linear function. This definition is very general and encompasses the classical NB-LDPC codes defined over finite fields as a sub-case.

A special case of interest is when the considered group has its cardinalty q as a power of 2, i.e. the group of type $\mathbb{G}(q) = (\frac{\mathbb{Z}}{2\mathbb{Z}})^p$ with $p = log_2(q)$. The variable nodes for such type of a code represent the elements of the finite linear group $\mathbb{G}(2^p)$. A symbol $c_j \in \mathbb{G}(2^p)$ can be represented by a p-bits binary map $X_j = \{b_j[k]\}_{k=0,...,p-1}$. Any choice of binary map can be made, however, in order to compliant with codes defined over finite Galois fields, we choose the binary map corresponding to a Galois field with the same number of elements, and defined with the minimal root polynomial of the field.

For a code defined over a linear group $\mathbb{G}(2^p)$, the functions $f_{ij}(.)$ are linear and can be represented by a $(p \times p)$ binary matrix [SD07]. We refer to the binary matrix as a *cluster* and denote the j^{th} cluster of the i^{th} parity equation as H_{ij}. A check node of degree d_c is then composed of d_c clusters and it forms a parity matrix H_{cc_i} of the local component code i.e. it can be interpreted as a local binary component code with p rows and $d_c p$ cols.

$$H_{cc_i} = [H_{i1}\ H_{i2}\ ...\ H_{id_c}] \tag{2.25}$$

Using the notations defined above, the i^{th} parity equation can now be written in the matrix form as:

$$\sum_j H_{ij} X_j^T = \mathbf{0} \tag{2.26}$$

For a $(M \times N)$ LDPC code defined over a finite group $\mathbb{G}(2^p)$, the parity check matrix H can now be seen as a binary matrix of size $(M_b \times N_b)$, $M_b = pM$ and $N_b = pN$. Each non-zero element of the PCM is represented by a $(p \times p)$ non-zero cluster whereas the zero elements of the PCM have an all zero $(p \times p)$ matrix representation. Fig. 2.12 shows the PCM of a code defined in $\mathbb{G}(2^4)$ with clusters of size (4×4).

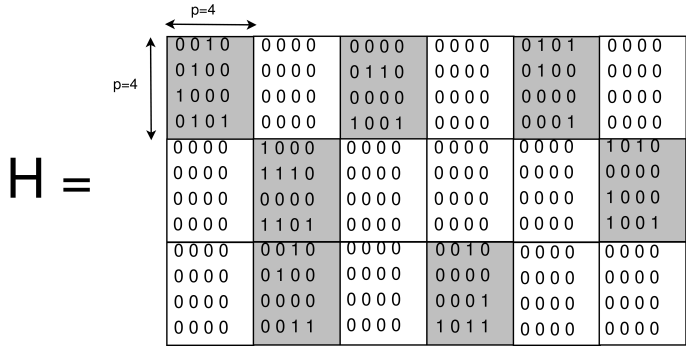

Figure 2.12: The square clusters in a binary PCM

This binary parity matrix can be decoded using a non-binary Tanner graph with N variable nodes and M check nodes. The functions $f_{ij}(.)$ are attached to the function nodes, which correspond to a linear mapping from $\mathbb{G}(2^p)$ to $\mathbb{G}(2^p)$, such that:

$$\alpha^n = f_{ij}(\alpha^m) \tag{2.27}$$

where α^n,$\alpha^m \in \mathbb{G}(2^p)$ are the symbols of the group. The linear mapping for a function $f_{ij}(.)$ is obtained using its cluster representation H_{ij} as:

$$b_{\alpha^n} = H_{ij}.b_{\alpha^m} \tag{2.28}$$

where b_{α^n} and b_{α^m} are the p-bits binary representation of the symbols α^n and α^m respectively.

If the cluster H_{ij} is a full rank matrix, the function then corresponds to a random permutation, which may not be cyclic. In the case, when it is a rank deficient matrix, it corresponds to a projection over some of the elements of the group. Fig. 2.13 depicts both cases with the two different non-zero clusters and their respective mapping functions. The binary cluster in Fig. 2.13(a) shows a full rank cluster with a full rank projection where all the elements are mapped from $\mathbb{G}(q)$ to $\mathbb{G}(q)$. The binary cluster in Fig. 2.13(b) represents a rank-deficient projection where some elements are projected more than once, leaving others un-projected. The un-projected symbols are called as un-authorized states and carry a value of $-\infty$.

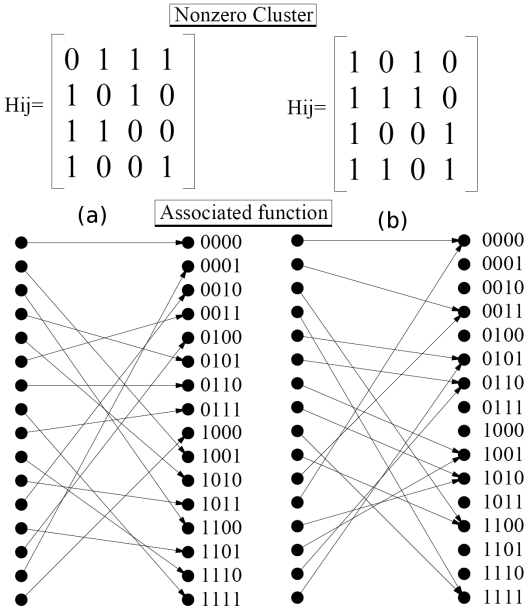

Figure 2.13: Binary clusters and their associated functions (a). Full-Rank (b) Rank-deficient

BP decoding of LDPC codes defined over groups

NB-LDPC codes defined over groups can also be decoded using the belief propagation (BP) algorithm. The BP decoder is very similar in nature to the classical BP decoder for codes defined over finite fields. The only difference is that the non-zero values of the PCM are replaced by more general linear functions from $\mathbb{G}(q)$ to $\mathbb{G}(q)$, which are

defind by the binary matrices H_{ij} forming the clusters. Following the same notation for the messages flowing in the Tanner Graph, $\{U_{vp}, U_{pc}, V_{cp}, V_{pv}\}$, the decoding procedure involves the following main steps:

- ***Variable nodes update*:** similar to the finite field decoder, for an edge connected to a variable node, the output extrinsic messages is calculated as the term by term product of the channel likelihood vector and all the other input mesages, excluding the input message on the edge itself. For a degree d_v variable node v, the reliability of the symbol $\alpha^i \in \mathbb{G}(q)$ in the output message U_{vf_k}, passed to the function node f_k is computed as:

$$U_{vf_k}[\alpha^i] = \mu_{vf_k} P_{ch_v}[\alpha^i] \prod_{j=0, j\neq k}^{d_v-1} V_{f_j v}[\alpha^i] \tag{2.29}$$

  where $V_{f_j v}$ are the input messages and μ_{vp_k} is a probability normalization factor.

- ***Function nodes update*:** the messages are updated using the function $f_{ij}(.)$ associated to the cluster of the edge of the Tanner graph. The update operation is defined as:

$$U_{pc}[\alpha^n] = U_{vp}[\alpha^m] \qquad m = 0...q-1 \tag{2.30}$$

  where $(\alpha^n, \alpha^m) \in \mathbb{G}(q)$ and their relationship is defined by eq. (2.28). The process is performed over all values α^m that have an image α^n through the linear function $f_{ij}(.)$.

- ***Check nodes update*:** this step is again identical to the BP decoding procedure for codes defined over finite fields as defined in eq. 2.20.

$$V_{cf_k}[\alpha^i] = \sum_{\delta:\, \sum_{m=0}^{d_c-1} \beta_{m(x)}=0} \delta \prod_{j=0, j\neq k}^{d_c-1} U_{p_j c} \tag{2.31}$$

  with $\alpha^i \in \mathbb{G}(q)$, β_m represent the symbols that take part in the parity and $\delta \in \{0, 1\}$ depending on whether the parity equation is verified.

- ***Inverse function nodes update*:** This update process uses the function $f_{ij}(.)$ in the reverse direction i.e. by identifying the values α_m which have the image β_n. The update equation is defined as:

$$V_{fv}[\alpha^m] = V_{cf}[\alpha^n] \qquad \forall \alpha^m, \alpha^n \in \mathbb{G}(q) \;:\; \alpha^n = f_{ij}(\alpha^m) \tag{2.32}$$

These steps constitute a decoding iteration of a general LDPC code defined over a finite group. The APP is calculated at the end of each iteration to make a decision on

the received codeword as explained for LDPC codes defined over Galois fields.

Advantages of generalized NB-LDPC Codes

Although binary LDPC codes are a powerful enough class of codes to achieve near capacity performance, they show their weakness while using short length codes or when a high order modulation is used. On the other hand, NB-LDPC codes have shown better performance in these two conditions [HFE04]. Thus, increasing the alphabet size was seen as another way of achieveing capacity, though with an increased decoding complexity. However, the relationship between alphabet size and performance is not a simple one and an increase in the underlying alphabet does not necessarily lead to increased performance [RU05]. For the same noise realization, a Belief propagation decoder on a specific Tanner graph can either: *(i)* converge to the right codeword *(ii)* converge to the wrong codeword *(iii)* diverge after a certain number of iteration [PDL08]. Hence, the structure of the underlying Tanner graph plays an important role in the decoding performance of an LDPC code.

The main interest in defining NB-LDPC codes over groups is that the family is larger than classical NB-LDPC codes defined over finite Galois fields. By considering codes in a wider ensemble, it is therefore possible to find better codes without changing the Tanner graph density or the order of the symbols finite set. This leads to a higher degree of freedom in terms of code construction without actually increasing the decoding complexity. In [PDL08], the authors showed that non-binary codes built on finite fields is actually a limitation, both from performance and implementation aspects. By considering NB-LDPC codes on a general linear group, they showed in particular that a resonable additional complexity, a slight performance gain can be obtained as compared to Galois field codes.

In [DM98], it was shown that for codes defined over $GF(q)$, selecting carefully the non-binary entries of the PCM can improve the overall performance of the code when compared to randomly chosen coefficients. The selection of the non-zero values have an impact on both, the waterfall region and error floor zone, which are also dependant on both the field order and the code rate. Choosing properly the edge labels has a direct influence on the local minimum distance of the code, and hence the global minimum distance as well. In [PFD06a], it was shown the error floor region can be lowered by avoiding low weight codewords induced by cycles and stopping sets in the unerlying Tanner graph. NB-LDPC codes defined over groups offer a higher degree of freedom in terms of code construction and hence powerful codes can be constructed that have a higher minimum distance than for the regular NB-LDPC codes defined over Galois fields. For example, The best d_c-tuples of coefficients for a $GF(64)$ code with $d_c = 4$ have a minimum distance of $d_{min} = 3$ in $GF(64)$, while it is possible to consider com-

ponent codes with $d_{min} = 4$ for codes defined over a general linear group [CPD+09]. This has a direct impact on the waterfall region performance of the LDPC codes.

The decoding algorithm used for decoding LDPC codes defined over finite groups helps in generalizing the BP algorithm to other types of codes also. In [PDL07], turbo codes were decoded using the generalized concept of the non-binary belief propagation for group-LDPC decoders.

2.1.3 Cluster NB-LDPC codes defined on Groups

In this section, we explain in detail a sub-family of codes defined over finite groups which we call as Cluster NB-LDPC codes. We explain this type of codes in a seperate section because the primary target of the work presented in the thesis is this family of codes. However, our work is applicable to other families of codes also. The main principle of these codes is that the binary clusters are not square but rectangular clusters which provides us with an even higher degree of freedom in terms of code construction. As a result of the presence of rectangular clusters, the variable and check nodes are processed in different order of messages.

Definition of cluster NB-LDPC codes

The motivation behind the definition of cluster codes comes from the work introduced in [VDV+08] where the authors proposed to divide the symbols at the variable nodes into sub-symbols, i.e. consider a sub-symbol of order $2^{p/n_s}$ instead of considering a $GF(2^p)$ symbol, where n_s is the splitting order of the symbols. For any symbol $c_k \in GF(2^p)$, let $\{b_k\}_{k=0...p-1}$ be the corresponding p-bits binary mapping. A sub-symbol of c_k is defined as a fraction of the bits from the binary mapping. For example, a symbol in $GF(64)$ can be split into two sub-symbols by forming two disjoint set of 3-bits each. Before the input at each check node, n_s sub-symbols are joined together to form a $GF(2^p)$ symbol and is then processed by the check node. Likewise, in the reverse direction, each symbol is divided into n_s symbols, before being processed by the variable nodes. This way the variable nodes are processed in a smaller order as compared to the check nodes.

This idea was generalised to NB-LDPC codes defined over groups in [SD06] and rectangular clusters were considered instead of square binary clusters. With the same representation as in the previous section, the clusters H_{ij} are of size $(p_2 \times p_1)$ instead of $(p \times p)$, where $p_1 \leq p_2$. However, unlike the idea of [VDV+08], sub-symbols are not joined together to form symbols, but the rectangular clusters form the functions $f_{ij}(.)$ which are used to transform a lower order message into a higer order message i.e. from $\mathbf{G}(2^{p_1}) \rightarrow \mathbf{G}(2^{p_2})$.

Structure of cluster codes

A sub-symbol is described as a symbol in $\mathbf{G}(2^{p_1})$, with $p_1 \leq p_2$. For each symbol $c_n \in \mathbf{G}(2^{p_1})$, there exists a p_1-bits binary mapping $X_j = \{b[k]\}_{k=0...p_1-1}$. The parity equation with binary clusters H_{ij} can be expressed as:

$$\sum_{j=0}^{d_c-1} H_{ij} X_j = 0 \tag{2.33}$$

The functions defined by the $(p_2 \times p_1)$ rectangluar clusters H_{ij} are linear mapping functions in $\mathbf{G}(2^{p_1}) \rightarrow \mathbf{G}(2^{p_2})$. This mapping is different from the mapping function discussed in the previous section. It is a mapping between two groups of different orders. A lower order message is mapped to a message defined in a higher order. Fig. 2.14 shows a rectangular binary cluster and its associated binary mapping. The mapping function is obtained using eq. 2.28. As observed, some of the symbols are left un-mapped. These symbols are considered to be un-authorized states and carry the value $-\infty$.

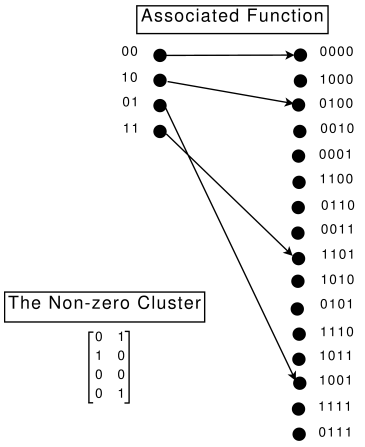

Figure 2.14: A rectangular cluster and its associated function node projections

In principle, the values of the column dimension p_1 can be different for different clusters for the same check node as explained in [SD06, SD07] i.e. for a check node, we have p_{1_j}, where $j = 0, ..., d_c$. This results in a very generalised family of codes with a large degree of freedom in terms of code construction. The code is no longer described in a single finite field or group. However, for sake of simplicity, we consider a single value of p_1 at all the clusters of the check nodes. For the case, when $p_1 = p_2$, the cluster is square and we have a regular NB-LDPC code defined over a finite group.

A parity check matrix for cluster NB-LDPD codes is shown in Fig. 2.15. The binary matrix is divided into clusters of size $(p_2 \times p_1)$. The non-zero values of the PCM are represented by non-zero binary clusters and all the null values are represented by the all zero matrix.

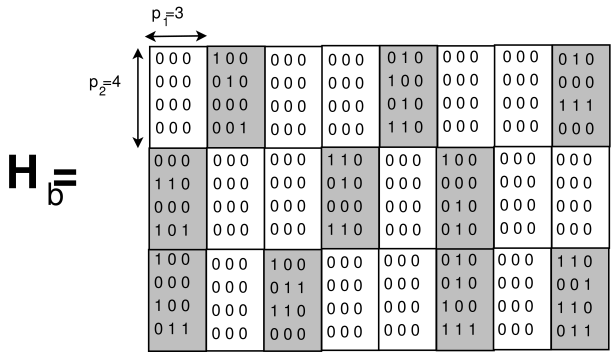

Figure 2.15: A binary parity check matrix with rectangular clusters

There are two main steps into the design of cluster NB-LDPC codes:

(i) building the Tanner graph, i.e. optimizing the edge connections.
(ii) selecting the non-zero values of H, i.e. carefully choosing the functions $f_{ij}(.)$ or equivalently H_{ij}.

The first step aims at maximizing the local girth and can be implemented by using the PEG algorithm [HEA05, VP08]. The non-zero values can then be selected using the method proposed in [PFD06a]. In any case, they must be carefully selected to ensure good performance in the waterfall region and obtain a low error floor.

NB-LDPC codes defined over groups can now be summarized as follows [Dec08]:

- If the elements of the PCM are defined in a group linear $\mathbb{G}(q)$, any general function can be used in eq. (2.24). The number of possible functions in this case are $q^q - 1$.

- A subset of codes defined over groups, which is of particular interest, is the linear case when the functions $f_{ij}(.)$ have binary matrix representations of size $(p \times p)$. This results in the functions $f_{ij}(.)$ to be either a permutation of the message values or a projection depending on whether the matrix representation H_{ij} is full rank or rank deficient. The total number of possible function in this case is $2^{p^2} - 1$.

- If the clusters H_{ij} are the binary companion matrices of the elements of a finite Galois field $GF(2^p)$, the functions $f_{ij}(.)$ are a cyclic permutations, which correspond to the multiplication of the non-zero element $h_{ij} \in GF(q)$. There are only $(q-1)$ possible choices for the functions.

- If the clusters H_{ij} are rectangular clusters of size $p_2 \times p_1$, such that $p_2 \neq p_1$, the functions $f_{ij}(.)$ acts as a mapping between messages of two different orders. There are $2^{p_1 p_2} - 1$ choices for the functions.

- If the columns of the rectangular cluster H_{ij} are the columns from the campanion matrices of the elements of a finite Galois field, we get the case defined in [VDV$^+$08]. The sub-symbols have to be combined to form symbols before the permutation nodes and symbols have to be marginalized to sub-symbols in the reverse direction.

Some properties and constraints of cluster NB-LDPC codes

We now discuss the various properties of cluster NB-LDPC codes:

i). Minimum distance of the code:
The fact that the binary parity check matrix H_{bin} is composed of rectangular clusters instead of square clusters bring more degree of freedom in terms of code construction. The density of 1's is less localized and as a consequence, more powerful codes with good minimum distance d_{min} can be constructed. As already discussed, a higher minimum distance has a direct impact on the error floor performance and on the probability of undetected errors i.e. when the decoder converges to a wrong codeword [PFD06a].

ii). Memory requirement for BP decoding:
The non-binary BP algorithm is based on the propagation of probabilities of the elements of the field/group through the Tanner graph.Each message vector is composed of q elements and storing all the messages leads to heavy storage requirements. Thus, memory storage is one of the several issues that stands in the way of the hardware implementation of non-binary LDPC decoders.

Cluster NB-LDPC codes provides a partial solution for this problem. The messages at the variable nodes represent the probabilities of sub-symbols instead of symbols. As a result, the message vector at the variable node is of length 2^{p_1} instead of the 2^{p_2} length message at the check node. For n_q bits quantization, the memory requirements of the BP decoding algorithm for the different families of codes are stated in Table 2.4. The table indicates the memory required for message storage in one direction of the Tanner graph only. Here $2^p = 2^{p_2}$ is the order of the group or field in which the code is defined. For a rate $R = 1/2$ code defined in $GF(64)$ or $\mathbb{G}(64)$, with length $N_b = 576$

Field	Group	Cluster
$2^p.d_c.M.n_q$	$2^p.d_c.M.n_q$	$2.d_c.2^{P_2/2}.M.n_q$

Table 2.4: Memory required to store messages in a NB-Tanner Graph

bits for example, the field and group decoders would require $12288.n_q$ bits to represent the messages. A similar specifications cluster code with $p_1 = 3$ would require $3072.n_q$ bits and for $p_1 = 4$, it would require $4608n_q$ bits.

iii). Decoding in $\mathbb{G}(2^{p_2})$***:***
The variable nodes are processed in $\mathbb{G}(2^{p_1})$ which implies to a significant reduction of memory requirement for storing the messages. However, the check nodes are processed in a higher order $\mathbb{G}(2^{p_2})$ which preserves the advantages of a higher order LDPC code.

iv). Degree of check nodes d_c***:***
Cluster codes generally have a higher degree of check nodes as compared to the regular LDPC codes defined over fields or groups. This is because the symbols are the variable nodes have been divided into sub-symbols, hence an increase in the number of edges. As compared to a code defined in $\mathbb{G}(64)$ or $GF(64)$ with the degree of check nodes $d_c = 4$, a cluster NB-LDPC code with $p_1 = 3$, the check nodes degree is $d_c = 8$. Similarly, for a cluster code with $p_1 = 4$, the degree of connection is $d_c = 6$.

2.2 Simplified implementation of NB-LDPC decoders

In this section, we present the various non-binary LDPC decoders that have been presented in literature. However, in this section, we only present their decoding procedures. The decoding complexities of the various algorithms will be discussed in section 2.3. We start by explaining the derivatives of the BP algorithm: the FFT-based BP, log-FFT-BP and the log-BP. Then we discuss the Extended-Minsum algorithm which is a generalization of the binary Min-Sum algorithm. It is an efficient and reduced complexity decoding algorithm which uses truncated messages. We also present the symbol flipping algorithm and stochastic decoders, which are the generalizations of the binary bit-flipping algorithm and the binary stochastic decoders respectively.

The BP algorithm for codes defined over a finite Galois field $GF(q)$, as explained in section 2.1.1, is composed of the following main steps:

- *Initialization:* The variable nodes are initialized with the belief of the corresponding variable, based solely on the channel information.

- *Variable nodes processing:* A message is passed from each variable node n to a check node m which expresses the belief of the n^{th} variable, given all the information from the channel and all the connected check nodes excluding the check node m itself.

- *Function nodes processing:* The function nodes process the messages by a cyclic permutation of the message values

- *Check nodes processing:* Each check node m sends a message to each of its adjacent variable node n, carrying the belief of the n^{th} variable, given the information from all the adjacent variable nodes excluding the variable node n itself.

- *Reverse Function nodes processing:* The message values are shifted in a cyclic fashion in the reverse manner.

- *Codeword Decision:* Each variable node n computes the most likely value $\hat{c}$ of n, based on the information from the channel and its adjacent check nodes. If all the checks are satisfied i.e. $H.\hat{c} = 0$, a codeword has been successfully decoded and it is returned. In the other case, the processes is iteratively repeated.

A decoding failure is generally declared if after a fixed number of iterations no valid codeword has been produced. In order to reduce the decoding complexity, log domain version of the algorithms have been proposed. In that case, the term "LLRs" is used instead of probabilites. We now present the various low complexity decoding algorithms which are sub-optimal forms of the belief propagation (BP) algorithm.

2.2.1 FFT-based BP algorithm

In the BP algorithm, the check nodes processing consumes the major portion of the computational complexity. In [MD99] the authors proposed to process the check nodes in the frequencey domain, and hence the convolution can be replaced by a product. This allows to reduce the computational complexity of the BP algorithm to $\mathcal{O}\ (p2^p)$. Consider U and V as two messages to be convolved:

$$\mathbb{F}(U \overset{GF}{\circledast} V) = \mathbb{F}(U).\mathbb{F}(V) \tag{2.34}$$

where $\mathbb{F}(.)$ represents the Fourier transform.

Tensorial representation of messages:

The authors of [BD03] proposed a tensoral representation of the messages in $GF(2^p)$ which simplifies the FFT based approach and results for field order of $2^p = 256$

were reported. When the Galois field is *characteristic-2*, i.e. it is of the order 2^p, the messages can be given a tensorial representation of p-dimensions and of size 2 in each dimension. Each element of the field $GF(2^p)$ can be represented by a degree-$(p-1)$ polynomial $i(x) = \sum_{l=0}^{p-1} a_l x^{l-1}$, where $a_l \in \{0,1\}$. Any set of p binary values $\{a_l\}_{l=0,...,p-1}$, thereby determines a unique field element $\alpha^i \in GF(q)$. Using this representation, a message on the edge connected to a variable node is a tensor $\{U[a_0...a_{p-1}]\}_{i_0...i_{p-1}}$ indexed by the binary coefficients of α^i [DF07]. For example, $U[0,1,1]$ corresponds to the probability $P(\alpha^i = x + x^2|.)$ in $GF(8)$, conditionally to the random variables depending on α^i in the factor graph.

The Fourier transform:

The tensorial representation makes the Fourier transform (FT) very simple as it corresponds to p second order Fourier transforms, one in each dimension of the tensor. The FT of a p-dimensional size-2 tensor $U_{a_0...a_{p-1}}$ is given by:

$$\mathbb{F}(U) = U \times_0 F \times_1 F... \times_{p-1} F \tag{2.35}$$

where '$\times_k$' represents the tensor product in the kth dimension of the tensor and F is the 2×2 matrix of the $2^n d$-order Fourier transform:

$$F = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The tensor product $Z = U \times_k F$ is defined [DF07], for $(i_1...i_k, i_{k+1}..i_p) \in (0,1)^{p-1}$:

$$Z[i_1...i_{k-1}, 0, i_{k+1}..i_p] = \frac{1}{\sqrt{2}}(U[i_1...i_{k-1}, 0, i_{k+1}..i_p] + U[i_1...i_{k-1}, 0, i_{k+1}..i_p])$$
$$Z[i_1...i_{k-1}, 1, i_{k+1}..i_p] = \frac{1}{\sqrt{2}}(U[i_1...i_{k-1}, 0, i_{k+1}..i_p] - U[i_1...i_{k-1}, 0, i_{k+1}..i_p])$$

The Fourier transform of the tensor can be calculated using the above equations for indexes k, i.e. in all dimensions of the tensor. This implementation of the tensoral product of the Fourier transform is similar to the butterfly implementation of Fast Fourier Transform (FFT). As can be observed, the Fourier transform calculation involves only addition.

The Walsh-Hadamard Transform:

The second order FT matrix can also be extended to the matrix form called the Walsh-Hadamard transorm. By repeative calculation of the 2^{nd} order tensoral product in all dimensions, the Walsh-Hadamard transform matrix can be constructed. For example,

the Walsh-Hadamard transform matrix for a vector of size 16×16 i.e. 4 dimensions, is:

$$FT = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

For a message of order q, the FT-matrix requires q^2 additions to compute the Fourier transform. However, the tensor product method requires $qlog(q)$ additions, which makes it a simpler process as compared to the FT-matrix.

The check update process:

With the FFT based BP Algorithm, the variable and permutation nodes processing remains the same. However, as shown in the Tanner graph of Fig. 2.16, the parity check nodes are converted into product nodes after the messages are transformed into the frequencey domain. Thus the check node update equation can be written as:

$$V_{tp} = F\Bigg(\prod_{c=1, c\neq t}^{d_c} F(U_{pc})\Bigg) \tag{2.36}$$

With this implementation, the computational complexity of the check node update process is reduced to $\mathcal{O}\ (d_c q log(q))$.

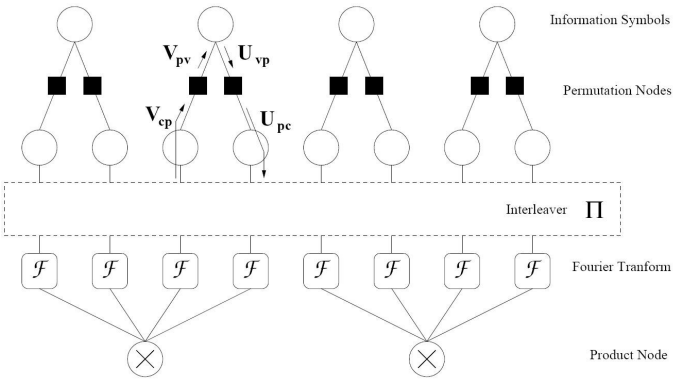

Figure 2.16: Tanner graph for FFT-based BP decoding

2.2.2 Log-FFT belief propagation algorithm

From a practical implementation point of view, the probability domain BP algorithm has several drawbacks as compared to its log-domain implementation: *(1)* Log-BP is more robust to quantization noise and usually requires less quantization levels as compared to the probability domain BP [PL00, WSM04a]. *(2)* Probability domain BP requires message multiplication whereas log-BP uses only additions. *(3)* The log-BP implementation has a lesser latency as compared to the probability domain implementation as the later requires more clock cycles for the multiplications. *(4)* By using the algorithm explained in [HEAD01a], the log-BP can be further simplified to the max-log-BP algorithm. *(5)* Unlike the the probability domain BP which requires an extra normalization step, the log-BP does not require the sum of the LLR values of a message to be unity.

Log-likelihood ratios (LLRs)

We now introduce the notion of an LLR of a variable v over a finite Galois field $GF(q)$. An LLR stands for log-likelihood ratio and is the ratio of the probabilites of a symbol α^i and the symbol 0, where $\{0, 1, \alpha, .., \alpha^{q-2}\} \in GF(q)$.

$$L(\alpha^i) = L(v = \alpha^i) = log\left(\frac{P(v = \alpha^i)}{P(v = 0)}\right) \tag{2.37}$$

with $P(v = \alpha^i)$ represents the probability that v takes the value α^i. An LLR vector stores the log-likelihood ratios of all the symbols of the field GF(q).

$$L(v) = [0 \;\; L(v = 1) \;\; L(v = \alpha) \;\; ... \;\; L(v = \alpha^{q-2})] \tag{2.38}$$

With this representation, the symbol 0 always carry an LLR value of 0. The most likely symbol has the highest LLR value and the least likely symbol has the least LLR value. We keep the same representation of messages as before and represent, by U, the LLR messages flowing from the variable nodes towards the check nodes and by V, we represent the LLR message flowing in the reverse direction. The indices vp, pc, cp and pv indicate the source and destination nodes of the respective LLR message.

The log-BP algorithm is composed of the following main steps:

1). Initialization:
The decoder is initialized with the channel output LLRs which are calculated as per eqs. 2.37 and 2.38. All the other incoming edges of the variable nodes are initialized to zero.

2). Variable update:
The variable update process becomes a simple addition of the input messages and the channel information. The LLR value of the symbol $\alpha^k \in GF(q)$ from the variable node v on the edge j connected to the function node f_j is given by:

$$U_{vf_j}[\alpha^k] = L_{ch_v}[\alpha^k] + \sum_{i=0, i\neq j}^{d_v-1} V_{f_i v}[\alpha^k] \tag{2.39}$$

Here, contrary to probability-domain BP, we do not require normalization of the symbol probabilities as they are log-likelihood ratios.

3). Permutation update:
The permutation nodes processing remain the same as the probability domain and it is just a cyclic permutation of message values.

$$U_{f_j c}[h_{f_j} \otimes \alpha^i] = U_{vp}[\alpha^i] \tag{2.40}$$

where h_{f_j} is the non-zero element of the PCM carried by the function node f_j and $\alpha^i \in GF(q)$ are the symbols of the Galois field.

4). Check update:
The log-FFT-BP Algorithm combines the advantages of FFT and the log domain implementation. The check nodes are processed in the log domain, with the advantage that we have only additions to perform. However, the FFT cannot be calculated in the log-domain as it is implemented in the real domain. Therefore, we require a transfor-

mation, between the real and log domains, of the type:

$$\begin{aligned} T &: \mathbb{R} \rightarrow \mathbb{R}/b = log(a) \\ T^{-1} &: \mathbb{R} \rightarrow \mathbb{R}/a = e^{b} \end{aligned} \tag{2.41}$$

where a and b are the real and log domain messages respectively. However, the resultant values of the FFT can be negative, therefore, the conversion from the real domain to log domain becomes complicated. This problem can be solved using the following representation [SC03]:

$$b = (b', b'') = (sign(a), log(|a|)) \tag{2.42}$$

and

$$a = b' exp(b'') \tag{2.43}$$

These equations make possible the use of FFT and the log-domain check update process together. The process of the transformation of the messages between the log and FFT domains is depicted in Fig. 2.17. With this implementation, we have a check nodes processing complexity of $\mathcal{O}\ (qlog_2(q))$.
A disadvantage associated with the implementation of log-FFT-BP is the exponential computations involved in the algorithm, which may not be practical. An attempt was also made to tackle this issue with the use of look-up tables (LUT). However, this approach was still not of a great interest because with increased field order, the LUT accesses grows in $qlog_2(q)$ for a single message [DF07]. In [SC03], the authors had shown with simulation results that the LUT approach is manageable for fields up to $GF(16)$.

2.2.3 The log domain non-binary BP algorithm

The log-domain BP algorithm consists of LLR vectors which are computed as per eqs. 2.37 and 2.38. The variable nodes processing of the log-BP algorithm remains the same as the log-FFT-BP algorithm. However, the check update mechanism is modified as we no longer need the messages in the frequency domain. The messages are processed in the log domain as log density rations (LDRs). Thus eliminating the importance of the field order to be power of 2, as we had discussed in the previous section that the FT is easy to calculate when the field order is a power of 2. Therefore, the log-BP algorithm can be applied to any field order q as long as q is a prime number or the power of a prime number [DF07].

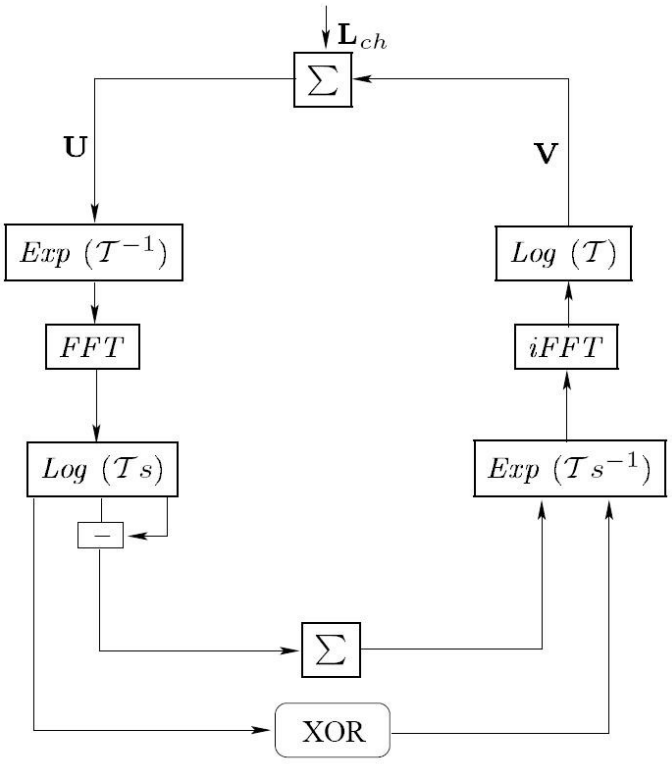

Figure 2.17: The log-FFT based BP algorithm

The box-plus operator (max*):

The box-plus operator was first introduced in [HooMM99] for $GF(2)$. It was extended to the non-binary case in [WSM04b] and it was proved that for two random variables v_1 and v_2 in $GF(q)$:

$$L(v_1 + v_2) = L(v_1) \boxplus L(v_2) \tag{2.44}$$

and similarly for two elements A_1 and A_2 in $GF(q)$:

$$L(A_1 v_1 + A_2 v_2) = \boxplus(L(v_1), L(v_2), A_1, A_2) \tag{2.45}$$

The box-operator can be expressed in terms of the Jacobi logarithm:

$$ln(e^{v_1} + e^{v_2}) = max^*(v_1, v_2) = max(v_1, v_2) + ln(1 + e^{-|v_1 - v_2|}) \tag{2.46}$$

The max^* operator is composed of a maximization and a correction factor by the term $ln(1 + .)$. The correction factor can be stored in small look-up tables (LUTs) as a function of $|v_1 - v_2|$ [HEAD01b]. Moreover, the max^* operator for multiple variables can be computed recursively i.e. $max^*(v_1, v_2, v_3) = max^*(max^*(v_1, v_2), v_3))$.

The check update process:

After introducing the concept of box-plus operator, we now present the log-domain check udpate process. Here note that the permutation nodes processing, i.e.the multiplication of the messages with the non-binary elements of the PCM, is a part of the check update process and is not executed as a seperate process. Consider a degree d_c check node with its adjacent variables $\{v_{c_k}\}_{k=0,\dots,d_c-1}$. Consider two new random variables over $GF(q)$:

$$\sigma_{c,v_{c_l}} = \sum_{j \leq l} h_{c,v_{c_j}} L(v_{c_j}) \tag{2.47}$$

$$\rho_{c,v_{c_l}} = \sum_{j \geq l} h_{c,v_{c_j}} L(v_{c_j}) \tag{2.48}$$

where $L(v_{c_j})$ represents the LLR vector associated with the variable node v_{c_j} connected on the edge j. Using these variables, the check node output message $L(c_k v)$ on the edge k is be computed as:

$$L(c_k v) = L(h^{-1}_{c,v_{c_k}} \sigma_{c,v_{c_{k-1}}} + h^{-1}_{c,v_{c_k}} \rho_{c,v_{c_{k+1}}}) \tag{2.49}$$

This equation is of the same format as eq. 2.45 and thus can be computed using the box-plus operator.

$$L(c_k v) = \boxplus(h^{-1}_{c,v_{c_k}} \sum_{j=0, j\neq k}^{d_c-1} h_{c,v_{c_j}} L(v_{c_j}))) \tag{2.50}$$

The box-plus operator is futher simplified to only a maximization process. In order to compensate for the further loss in information, a fixed offset is used which is added to all the output messages. Due to the loss in information there is a slight degradation in the decoding performance. However the box-plus operator significantly reduces the computational complexity and thus high field order decoders became possible.

2.2.4 The Extended Min-Sum algorithm

The Extended-Min-Sum (EMS) algorithm is a generalization of the binary Min-Sum algorithm. It was first proposed in [DF07] where the authors showed that the EMS algorithm in $GF(2)$ is equivalent to the Min-Sum algorithm for binary codes. The authors proposed to use truncated message vectors of size n_m at the check nodes in place of the size-q message vector, with $n_m << q$. This reduced the check nodes complexity to $\mathbf{O}(n_m q)$. This idea was further studies in [VDV$^+$10] and extended the concept to the whole of the Tanner graph i.e. variable nodes also. Thus, the EMS algorithm uses the n_m highest LLR values at both the check and variable nodes and the remaining LLR values of a vector are replaced by a single value γ. The authors also

proposed a compensation offset correction for the loss in the performance. In order to efficiently process the check nodes, the authors used a recursive implementation called as the Forward-Backward (F/B) strategy. The complexity of the check nodes was then reduced to $\mathbf{O}(n_m log(n_m))$.

Structure of the truncated messages:

We now present the structure of the truncated messages. We use the same representation for messages i.e. U_{vp} and U_{pc} are the messages flowing from the variable nodes towards the check nodes. V_{cp} and V_{pv} are the messages flowing in the direction of the variable nodes. However, these messages are sorted in decreasing order and are limited to size n_m. The remaining $(q - n_m)$ elements of the LLR vector are considered to carry the value γ. Thus each message is of size n_m and in order to keep track of the symbol corresponding to each LLR value, we need to associate an additional vector β, that indicate the set of symbols corresponding to the truncated LLR mesage. Thus, for a message vector U_{vp}, the highest LLR value is stored at $U_{vp}[0]$ and $\beta_{U_{vp}}[0]$ carries the corresponding symbol information. The channel likelihood information is however not truncated and stored in a q-size vector sorted in decreasing order. With this notation, a full representation of a message L, sorted in decreasing order, can be expressed as:

$$L = [L[0] \;\; L[1] \;\; ... \;\; L[n_m - 1] \;\; \gamma \;\; \gamma \;\; ... \;\; \gamma] \tag{2.51}$$

The corresponding symbol information is carried in the vector β_L of size n_m:

$$\beta_L = [\beta_{L[0]} \;\; \beta_{L[1]} \;\; ... \;\; \beta_{L[n_m-1]}] \tag{2.52}$$

Thus, we require only two vectors, each of size n_m, to store a complete message. Since the vectors are stored in decreasing order, it also means that the value γ verifies $\gamma < L[n_m - 1]$. The value of γ can be calculated using the normalization of the mesages in the probability domain. In [VDV$^+$10], this value was approximated to be the $(n_m + 1)^{th}$ value of the message vector. Moreover, a suitable offset can also be added to compensate for the loss of information due to the truncation of messages. Thus, the value of γ can be expressed as:

$$\gamma = L[n_m] + \text{offset} \tag{2.53}$$

The offset can be calculated in several ways. Generally, it is calculated using Monte-Carlo simulations for different values of offset, and selecting the offset with the best frame error rates (FER).

The forward-backward (F/B) strategy

The forward-backward (F/B) strategy is a recursive process and follows the prinicple of "divide and conquer". It is based on the division of a process into several elementary modules and the outputs of the various elementary modules are then joined together to form the whole process. For LDPC codes, the node update process is divided into elementary modules such that each module has only two inputs edges. Fig. 2.18 shows the F/B strategy for a degree 5 check node with $\{U_i\}_{i=0,..,4}$ as inputs and $\{V_i\}_{i=0,..,4}$ as the respective outputs.

As we can observe, the input messages are processed in the forward direction and in

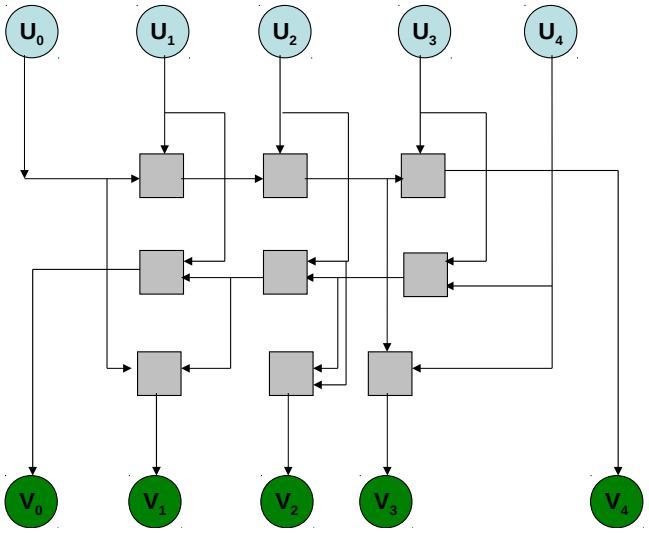

Figure 2.18: The forward-backward strategy for a check node of degree $d_c = 5$

the backward direction and the various inputs and intermediate messages are then combined together to form the various extrinsic output messages. Also it can be noticed in the figure, that the structure is linear and can be extended easily to any degree of check nodes. For a check node of degree d_c, the number of elementary steps required are:

$$n_{elem} = 3(d_c - 2) \tag{2.54}$$

The decoding algorithm

We now explain the various steps involved in the decoding algorithm. The variable update, permutation nodes update and the check update process. The F/B strategy is used to process the check nodes, therefore, for the check update process, we discuss only elementary processes i.e. with only two inputs. The various elementary processes

can be joined together in the forward-backward manner to obtain the extrinsic outputs of a check node.

i). Variable nodes update:
Consider the LLR-vectors $(V_0, .., V_{d_v-1})$ as the inputs from the check nodes to a variable node of degree d_v. The vectors are sorted in decreasing order, each of size n_m, with the vectors $(\beta_{V_i}, ..., \beta_{V_{d_v-1}})$ carrying the symbol information corresponding to each LLR-vector respectively. The compensation values for each LLR-vector is represented as $(\gamma_{V_0}, ..., \gamma_{V_{d_v-1}})$ respectively. L_{ch} is the channel information LLR-vector of size 2^p sorted in decreasing order with $\beta_{L_{ch}}$ carrying the symbol information. The output vector U is calculated as the highest n_m values from the symbol by symbol summation of all the inputs. The LLR-value of a symbol $\alpha^i \in GF(2^p)$ is calculated as:

$$U_k[\alpha^i] = L_{cv}[\alpha^i] + \sum_{j=0, j\neq k} Y_j \tag{2.55}$$

with

$$Y_j = \begin{cases} V_j[\alpha^i] & \text{if} \quad \alpha^i \in \beta_{V_j} \\ \gamma_{V_j} & \text{otherwise} \end{cases} \tag{2.56}$$

The compensation value is used when the LLR-value corresponding to the symbol is not in the respective input message. The n_m symbols with the highest LLR-values are kept in the output U. The symbols carrying smaller LLR-values are truncated.

ii). Permutation nodes update:
The permutation nodes permutes the message values according to the non-binary values h_i of the PCM. For this purpose, only the indices of the symbol vectors β are updated and the LLR-values remain unchanged.

$$\beta_{U_{p_i c}}[k] = h_i.\beta_{U_{vp_i}}[k] \tag{2.57}$$

where the multiplication in performed in $GF(q)$.

iii). Elementary check nodes update:
The check nodes are processed using the Forward-Backward strategy. The various elementary processes are then joined together to form the output of the check nodes as discussed above. Here we explain only the elementary check update process i.e. a process with two inputs and a single output LLR-vector only.

For the elementary check node update process, consider the LLR vectors U_1 and U_2 as inputs and the output V. All the vectors are of size n_m and sorted in decreasing order.

β_{U_1},β_{U_2} and β_V are the vectors carrying the symbol information corresponding to LLR values of each vector respectively. Let $S(\beta_V[k])$ be the set of all the possible symbol combinations that satisfy the parity equation, i.e.

$$\beta_{U_1}[i] \oplus \beta_{U_2}[j] \oplus \beta_V[k] = 0 \tag{2.58}$$

The output message V is then obtained as:

$$V[k] = \underset{S(\beta_V[k])}{max} (U_1[i] + U_2[j]) \tag{2.59}$$

This computational complexity is then dominated by $\mathbf{O}(n_m^2)$. However, the authors of [VDV+10] proposed a better method to search the n_m highest combination out of the possible n_m^2 combinations. This reduced the complexity to $\mathbf{O}(n_m^2/2)$. The method is based on scanning a 2-D matrix M of size $(n_m \times n_m)$ which is formed from the sum of the two inputs such that:

$$M[i,j] = U_1[i] + U_2[j] \quad i = 0, ..., n_m - 1 \tag{2.60}$$

For each value of $M[i,j]$, the associated symbol information $\beta_M[i,j]$ is calculated as:

$$\beta_M[i,j] = \beta_{U_1}[i] \overset{GF}{\oplus} \beta_{U_2}[j] \tag{2.61}$$

The matrix M is depicted in Fig. 2.19. Since the two inputs are sorted in decreasing order, so the highest values of the sum of the two inputs are present in the upper left triangle of the matrix M. Therefore, we need to only scan the highest $n_m^2/2$ elements present in upper left triangle to find the highest n_m combinations. Along with the matrix M, the procedure also requires a sorter of size n_m as depicted in Fig. 2.19. The check elementary processed can thus be implemented as:

1. Initialization: Insert the first column of the matrix- M into the sorter and initialize no. of elements copied $n = 0$
2. Output: Move the highest value of the sorter towards the outputs
3. Test: Does the associated $GF(q)$ symbol already exist in the output vector
   - No: Copy the highest value and symbol to the output vector $V[n]$ and $\beta_V[n]$ and update $n = n + 1$
   - Yes: Discard the value and take no action
4. Control: Replace the value in the sorter by its right neighbour with regard to matrix M
5. if $(n = n_m)$ exit - else go to step-2

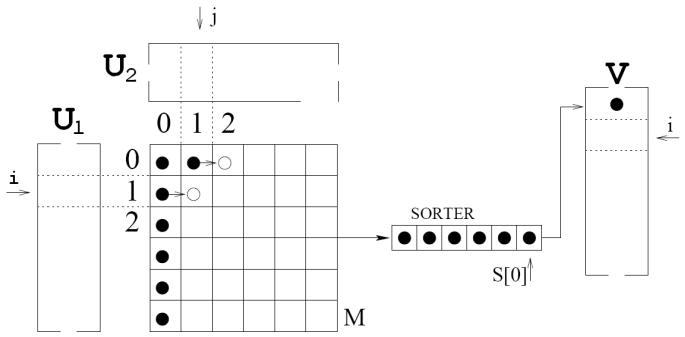

Figure 2.19: The elementary check update process

As can be deducted from the pseudocode that the algorithm cannot be stopped just after n_m values have been scanned as because it is possible that two or more values might correspond to the same $GF(q)$ symbol. However, it is certain to find the n_m symbols with the highest LLR values with in the upper left $n_m^2/2$ elements of M. Thus, the number of elements to be scanned is limited to the interval $n_c = [n_m, n_m^2/2]$.

2.2.5 Symbol Flipping based decoding

The symbol-flipping algorithm was proposed in [KH06] as an extension of the bit flipping algorithm [HH02, MF05]. However, this symbol based hard decision decoding algorithm works only for low order of Galois fields. For higher orders, the algorithm looses a lot of its efficiency.

The symbol-flipping algorithm was proposed for a generalization of LDPC codes that uses local component codes, i.e. each parity node is considered to be a local consitutent code. Furthermore, several check nodes are clumped together to form super codes. For a super-codes based generalized LDPC code with degree of variable nodes d_v, there are d_v super codes and each variable node is connected to each super code.

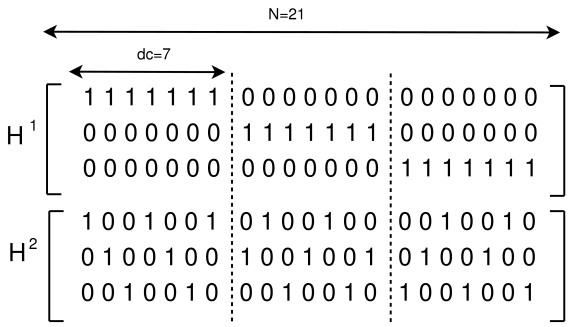

Figure 2.20: Parity check matrix H of a generalized LDPC code based on super-codes

A super-codes based $(M \times N - K)$ generalized LDPC code can be seen as the intersection of $C = \bigcap_{j=1}^{d_v} C^j$ of d_v super codes. Fig. 2.20 shows an example of a PCM of this type of a LDPC code of size (21×6). As we can observe, the PCM can be divided into two sub-PCMs H^1, H^2, hence 2 super-codes. The PCMs H^2 is obtained as a random permutation of the PCM H^1, i.e. $H^j = \pi_j(H^1)$, where $j = 2, .., d_v$ and π represents a permutation. Therefore the codewords of the super-codes can also be expresses as $C^j = \pi_j(C^1)$. The variable d_v implies that each variable node is connected to d_v local component codes (parity nodes) represented by d_v different clumps of super codes C^j,$j = 1, .., d_v$ respectively. Fig. 2.21 represents the corresponding Tanner graph for the PCM in Fig. 2.20.

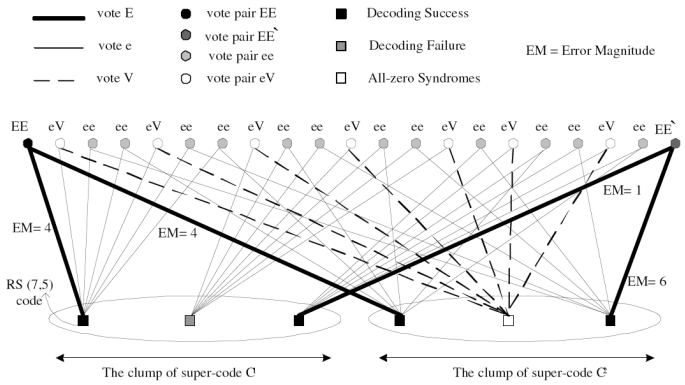

Figure 2.21: Tanner Graph and vote generation for the PCM for super-codes based generalized LDPC codes

The authors of [KH06] used the Reed-Solomon (7×5) code as its local component code. The constituent codes not only find out the error positions but also produce the error magnitudes. Based on these results, the component codes generate votes for a specific symbol. The votes are generated using using either the Peterson-Gorenstein-Zierler (PGZ) or the Berlekamp-Massey (BM) [HLY02] hard decision decoders (HDD), which are then sent towards the variable nodes. The votes are generated based on the following decoding scenarios at the constituent codes.

- *All-zero syndrome:* this implies the presence of a valid codeword, hence all the constituent d_c symbols are labelled by the character V.

- *Non-zero syndrome and decoding success:* This indicates the presence of an invalid but correctable received word. The error positions are labelled by E whereas the correct symbol are labelled by e.

Vote Pair	EE	EE'	Ee	EV,ee	eV	VV
Vote Value	4	3.5	3	2	1	0

Table 2.5: Vote values for the symbol flipping decoding for generalized LDPC based on 2-super codes

- *Non-zero syndrome and decoding failure:* This indicates that no error position was identified that caused the decoding failure, therefore no corrective action can be gleaned out from this constituent decoder. All the RS code symbols are labelled by e, similar to the correctly decoded symbols in scenario 2.

The various votes V,e and E are given numerical values in order to rank the votes according to their reliabilities. In [HH02], it was suggested that values $V = 0$, $e = 1$ and $E = 2$ produce the lowest BERs where a higher vote represents a lower reliability. Table 2.5 represent the values of the joint votes of the two super-codes as suggested in [HH02]. A vote pair of EE' indicates that different error magnitudes were suggested by the two constituent decoders. The decoding process can be summarized as:

1. Invoke the PGZ or BM HDDs for the non-binary constituent RS codes of the super codes C^1 and C^2.

2. Compute the vote values at the N variable nodes using Table 2.5 and rank the symbols according to their calculated reliabilities.

3. Correct the least reliable symbols, i.e. the maximum votes values, according to the error magnitudes. The vote pair ee and eV are not to be corrected as no valid error magnitude was calculated.

4. Repeat steps (1) to (3) until we have VV for all the N symbols or a maximum number of iterations have reached.

Simulation results were presented in [KH06] for LDPC codes defined over $GF(32)$ communicating over AWGN and uncorrelated Rayleigh fading channels while using BPSK modulation. The non-binary symbol-flipping algorithm showed improved performance as compared to their binary counter part using the bit-flipping algorithm.

2.2.6 Non-binary Stochastic decoders

Binary stochastic decoders use random bit-streams to represent message probabilities [TGM06] and have a very simple hardware architecture with reduced wiring complexity. Subsequently, stochastic decoders were implemented on field-programmable gate arrays (FPGAs) [TMG07, TMG08] which were area-efficient, fully parallel and had a

high-throughput and a performance close to the channel capacity. As the aftermath, a non-binary stochastic decoder was presented in [SMG09] in order to enable the practical implementation of non-binary LDPC decoders.

The non-binary stochastic decoding algorithm uses streams of symbols chosen from $GF(2^p)$ to represent the messages. The probability of a symbol is determined by dividing the no. of times the symbol occured in the stream over the total length of the stream. Such type of message representation makes very simple the underlying circuitry and its manipulation. Thus $\overline{U}_{vp}(t)$ denotes the symbol at position t of the stream at the output of the variable node v. The notation $\overline{U}$ is used to differentiate a stream from a message U carrying probabilities of the $GF(2^p)$ symbols. The output streams of the check nodes are represented as $\overline{V}_{cp}$.

In binary stochastic decoders, a phenomenen called as latch-up occurs which is caused by cycles in the graph. The switching activity between the symbols become very low and the stochastic streams become correlated and invalidates the independancy assumption considered while while constructing the stochastic equations. Two solutions were proposed;

1). Noise-dependant scaling: Increase the switching activity by scaling down the channel likelihood information. For example, for BPSK modulation over AWGN, the scaled likelihood $l'(b_i)$ of the bit b_i is computed as a function of the unscaled likelihood $l(b_i)$ as:

$$l'(b_i) = [l(i)]^{\frac{2\alpha\sigma_n^2}{Y}} \tag{2.62}$$

where σ_n^2 is the noise variance and $\frac{\alpha}{Y}$ is scaling factor and is calculated to determine the best performance the SNR range of interest.

2). Edge-memories (EM): Insert finite depth memory buffers called as edge memories between the variable and permutation nodes. The EMs randomly reorder symbols in the output streams of the variable nodes and thus breaks correlation between the streams without affecting the overall stream statistics. The EM contents are updated when the variable node output is changed. The output of the EM is then the output of its connected variable node. In case the variable nodes output is no changed, the output of the EM is a stream of randomly selected symbols from its contents.

We now present the stochatic processing of the various nodes of a Tanner graph of a non-binary LDPC code:

Variable nodes processing:
The variable node output value of the stream at time t is computed as: copy the input

symbol to the output symbol if the input symbols on all the other incoming edges are equal at time t, otherwise use the value copied in the last interval.

$$\overline{U}_{vp}(t) = \begin{cases} a & \text{if } \overline{V}_{iv} = a, \forall i : i \neq p \\ \overline{U}_{vp}(t-1) & \text{otherwise} \end{cases} \tag{2.63}$$

This process can be implemented with XNOR and AND gates as shown in Fig. 2.22 for a $GF(2^3)$ code with $d_v = 2$. The output of the circuit provides the enable signal to the EMs. The circuit can be extended to higher order fields by using more XNOR gates and a larger AND gate. Thus, increasing the number of bits which are required to represent the $GF(2^p)$ symbols in the stochastic streams. If the degree of connection d_v is increased, the number of inputs to each XNOR gate is increased accordingly.

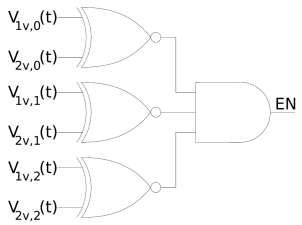

Figure 2.22: The Stochastic variable update process for a degree $d_v = 2$

Permutation nodes processing:
The permutation nodes multiply the non-zero elements h_j of the PCM with the input messages. In the BP algorithm, this is achieved by a cyclic shift. However, a stochastic permutation node carries out the process by multiplying the non-zero element h_j with the stochastic stream.

$$\overline{U}_{pc}[\alpha^i] = \overline{U}_{vp}[\alpha^i.h_j] \quad (\alpha^i, h_j) \in GF(2^p) \tag{2.64}$$

For the messages passing in the reverse direction i.e. from the check towards the variable nodes, the inverse node multiplication can be implemented by multiplication with h_j^{-1}.

At the hardware level, the permutation node can be implemented using $GF(2^p)$ multipliers. Since, for a particular edge of a code, the non-binary element is always h_j, the multiplier can be designed to multiply by a specific constant $GF(2^p)$ element instead of a generic $GF(2^p)$ multiplier. This significantly reduces the circuit complexity.

Check nodes processing:
For the check node of degree $d_c = 3$, an output is defined as the addition of the other

two inputs.

$$\overline{V}_{c0}(t) = \overline{U}_{1c}(t) \overset{GF}{\oplus} \overline{U}_{2c}(t) \tag{2.65}$$

This equation can be generalized to any degree of check nodes as:

$$\overline{V}_{cp}(t) = \sum_{i=0, i\neq p}^{d_c} \overline{U}_{ic}(t) \tag{2.66}$$

where the summation is a $GF(2^p)$ addition. For a $GF(2)$ code, this equation reduces to the equation defined in [TGM06].

Since the $GF(2^p)$ symbols can be represented in their polynomial formats, so $GF(2^p)$ addition can be realized using XOR gates. Fig. 2.23 presents a stochastic check node for a $GF(2^3)$ code with $d_c = 4$. The circuit can be extended to a higher degree of check nodes by increasing the number inputs to the XOR gates. Accordingly, for a higher order code, the number of XOR gates have to be increased.

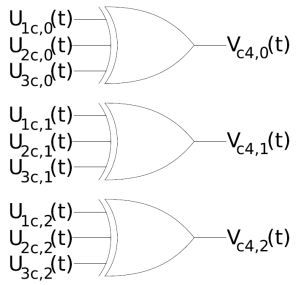

Figure 2.23: The stochastic check update process for degree $d_c = 4$

2.3 Complexity comparison of NB-LDPC decoders

Non-binary LDPC codes were first investigated by Davey and Mackay in 1998 where they generalised the binary Belief-Propagation (BP) or the Sum-Product algorithm (SPA) to decode non-binary LDPC codes defined over a Galois field $GF(q)$ [DM98]. However, the disadvantage associated with the NB-LDPC decoder was its complexity, which is of the order $\mathbf{O}(q^2)$. Moreover, it also increased the memory requirement for storing messages. As a result fields of order $q > 16$ were not considered initially. To reduce complexity, Mackay and Davey also devised a Fast Fourier Transform (FFT) based BP algorithm [MD99]. However, the FFT could only be implemented for field orders of the power of two i.e. $GF(2^p)$. This idea was extended into the log domain and the log-FFT-BP algorithm was proposed in [SC03]. However combining

Algo.	Multiplications	Divisions	Max*	Max	Additions
BP	q	-	-	-	$q-1$
FFT-BP	q	-	-	-	$q-1$
log-BP	-	-	$q-1$	-	q
EMS	-	-	-	$n_m(2n_m)$	n_m

Table 2.6: Number of operations required for an elementary process of a variable node

log values and the FFT require exponential and logarathmic computations, which may not practical. Therefore, the authors proposed the use of look-up tables (LUT) which again limited the use of higher order fields as the number of access to the LUTs increased with the increased message sizes. In [BD03], the authors proposed a tensoral representation for the FFT computation and as a result the authors reported results for field orders upto $2^p = 256$. The binary Min-Sum algorithm [CF02] was generalised to non-binary LDPC codes in [WSM04b]. Though the algorithm was composed of additions only, the complexity remained of order $\mathbf{O}(q^2)$ in addition. In [DF07], the authors proposed to use only a certain number n_m of highest message values at the check nodes. The algorithm was referred to as the Extended-Min-Sum (EMS) algorithm. This decreased the complexity of the check update process to $\mathbf{O}(n_m q)$. This idea was extended to the variables nodes as well in [VDV$^+$10] and the EMS algorithm was proposed to have truncated message vectors at both, the variable and check nodes. This decreased the memory requirements in the whole of the Tanner graph alongwith a heavy reduction in the decoding complexity at both nodes. Other very low complexity decoders have also been proposed where the authors generalized the sub-optimal binary decoding techniques to non-binary LDPC codes. The symbol flipping algorithm [KH06] and the NB-stochastic decoders [SMG09] performed very efficiently with very low complexities, however they are limited to small field orders only.

In the previous section, we had described that the processing for a high degree check node can be implemented using the forward-backward (F/B) strategy. The F/B strategy breaks the whole process into several elementary processes such that each elementary process has two input edges and an output edge. The various inputs and intermediate messages are then joined together to form the whole process as shown in 2.18. For a degree d_c check node, $3(d_c - 2)$ number of elementary steps are required.

In this section, we compare the decoding complexities of the elementary processes of the algorithms: BP, FFT-BP, log-BP, EMS. We present the complexity in terms of number of operations (additions, multiplication, divisions etc.). Table 2.3 depicts the complexities for the elementary process of a variable node. Table 2.3 presents the complexities of the elementary process of a check node.

Algo.	Multiplications	Divisions	Max*	Max	Additions
BP	q^2	q	-	-	$q(q-1)$
FFT-BP	$q(d_c p+1)$	q	-	-	$d_c qp$
log-BP	-	-	$q(q-1)$	-	q^2
EMS	-	-	-	$n_m n_{op}$	$n_m + n_{op}$ (real) $n_m + n_{op}$ (in $GF(q)$)

Table 2.7: Number of operations required for an elementary process of a check node

We can observe that the number of operations can be tremendously decreased while using the EMS algorithm as compared to the BP decoding algorithm. However, this decrease is interesting only for small values of the truncation parameter i.e. $n_m << q$. For larger values of n_m, the EMS algorithm does not give the same advantage in terms of decreased complexity.

2.4 Conclusion

Binary LDPC codes have shown to approach the Shannon limit for long codes. But they show their limitations when smaller length codes are employed or when a high order modulation is used. For that reason, non-binary LDPC codes defined over finite Galois fields were proposed. They show a gain in performance for short length codes and for a field order higher than the modulation order.

However, the main disadvantage associated with NB-LDPC codes is their decoding complexity. For a NB-LDPC code defined over a Galois field $GF(q)$, the complexity of the BP algorithm is of the order $\mathbf{O}(q^2)$. Hence, low complexity algorithms were proposed. The FFT-based BP, log-FFT-BP and log domain BP algorithms were the sub-optimal derivatives of the BP algorithm but their decoding complexity was still impractical. The symbol-flipping and the non-binary stochastic decoders offer a very low complexity which is implementable, however, they can be applied only to short length codes defined in low order Galois fields. There is no existing evidence that they can be applied to high order fields or larger length codes. However, the Extended-Min-Sum (EMS) algorithm emerged to be a promising candidate. It was a generalization of the binary Min-Sum algorithm and proposed the use of truncated messages in the Tanner graph which not only reduced the decoding complexity but also offered to decrease the memory required for storing messages. A truncated $GF(q)$ message means

to store only a certain number n_m of high LLR-values of the message and replacing the $q - n_m$ least reliable message values by a single value γ. Since then, research has also been directed towards improving the performance of the EMS algorithm in order to have a very low value of n_m e.g. $n_m \approx 12$ for a $GF(64)$ message.

Along with the decrease in the decoding complexity, the code structure itself has been a topic of interest for researchers. As has been discussed in the chapter that the code design and structure of the Tanner graph plays an important role in the decoding performance of a LDPC code. There are a lot of publications on the design and development of LDPC codes. In this regard, LDPC codes were proposed to be defined over finite groups such that the non-binary elements of the PCM had square binary matrix representations of size $(p \times p)$. This provided code designers with a higher degree of freedom as the binary clusters can be constructed to have a higher minimal distance and thus stronger codes can be constructed.

Later, the binary matrix representations were proposed to be rectangular clusters of size $(p_2 \times p_1)$, with $p_1 < p_2$. This provided an even higher degree of freedom in terms of code construction. We refer to them as "cluster codes". Cluster codes generally have a lower error floor as compared to the $GF(q)$ codes using the BP decoding algorithm. However, low complexity decoding algorithms have not yet been proposed for cluster codes. And this is the main emphasis of the work presented in this thesis.

3

Improved EMS algorithm for cluster codes

In this chapter, we present our work on the generalization of the EMS algorithm, proposed in [VDV$^+$10], to cluster codes which we refer to as cluster-EMS. We start by presenting a direct generalization of the EMS algorithm to cluster codes. We present an improved method to compute the n_m highest values as compared to the one presented in [VDV$^+$10]. In section II, we present an improved decoding algorithm where we identify and amend the problems associated with the direct generalization of the cluster-EMS algorithm. In section III, we then make use of the diversity of the Tanner graph associated to cluster codes and propose to implant parallel scrambled local processes to compute the check nodes output. It helps in reducing the surface area of the hardware architecture.

3.1 Generalization of EMS to cluster codes

In this section, we present the direct generalization of the EMS algorithm to cluster codes. We also propose a new method to search the highest n_m values at the elementary procedure. The proposed method requires less resources as compared to the method proposed in [VDV$^+$10] by efficiently scanning the 2-D matrix formed from the combination of the two inputs to the elementary process.

The main feature of cluster codes is that the variable and check nodes are processed in different orders of messages. The variable orders are processed in a smaller order as compared to the check nodes. This is due to the fact that binary clusters in the PCM are rectangular. Hence, the functions of the non-binary Tanner graph associated to the clusters act as a mapping between messages of two different orders. This process has been explained in the previous chapter and depicted in Fig. 2.14. In principle, the various variable and check nodes can be defined over different group orders for the same code. This is achieved by defining different size clusters across the parity check matrix as shown in Fig. 3.1. As we can observe the order of the clusters are defined over different values $2^{p_1}, 2^{p_2}, 2^{p_3}, ..., 2^{p_n}$,, where $p_1 \neq p_2 ... \neq p_n$.

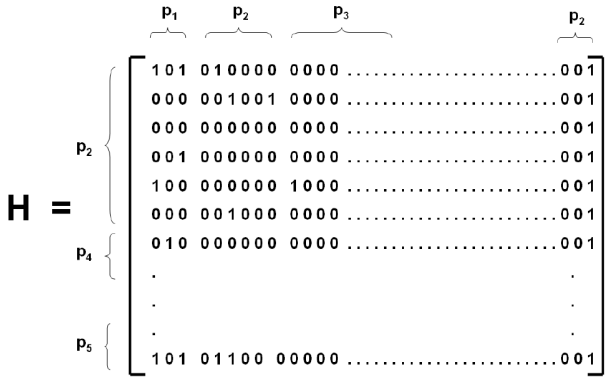

Figure 3.1: A binary PCM composed of clusters of different sizes

For the sake of simplicity, we will consider the cluster size to be fixed to $(p_2 \times p_1)$ throughout the manuscript, where $p_1 < p_2$. The variable nodes are thus processed in a smaller order $\mathbb{G}(2^{p_1})$ as compared to the check nodes $\mathbb{G}(2^{p_2})$. The function nodes act as a mapping between the messages of the two orders $\mathbb{G}(2^{p_1}) \leftrightarrow \mathbb{G}(2^{p_2})$. However, a messages at the input of check nodes has 2^{p_1} authorized states only, where the remaining $(2^{p_2} - 2^{p_1})$ states are declared unauthorized and carry the value $-\infty$. Thus, the messages at the check nodes input are stored in vectors of size 2^{p_1} each. Fig. 3.2 represents the Tanner graph for the cluster codes and the various messages flowing through it. The messages are non-truncated, and thus we have a reduction in memory without the loss of information. However, the check nodes are processed using the forward-backward strategy, and in order to have a reduced complexity decoder, the output messages of the various elementary processes are truncated to size n_m.

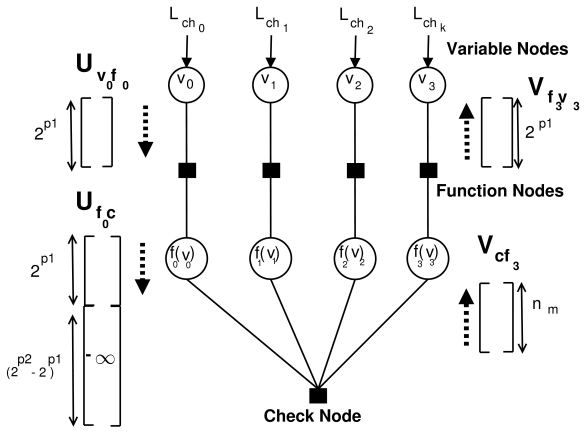

Figure 3.2: The messages flowing across a Tanner graph for cluster-LDPC codes

3.1.1 The decoding algorithm

We keep the same representation of messages flowing through the Tanner graph as mentioned in the previous chapter i.e. U_{vf} and U_{fc} are the LLR-messages flowing from the variable direction v to the check node c via the function node f. Similarly V_{cf} and V_{fv} represent the messages flowing in the reverse direction.

Strictly positive LLR-vectors

We use a representation of the LLRs with which we have strictly positive LLR-values. There is no notion of a negative LLR value and the least reliable symbol is associated to an LLR equal to zero. The LLR value of a symbol is thus computed as:

$$L(\alpha^i) = log\left(\frac{P(v=\alpha^i)}{P(v=\alpha^j_{min})}\right) \tag{3.1}$$

where α^j_{min} is the least reliable symbol in the vector. With this representation a sorted LLR vector has the form:

$$L(v) = [L(\alpha^i)\ \ L(\alpha^j)\ \ L(\alpha^k)\ \ ...\ \ 0] \tag{3.2}$$

where $L(\alpha^i)$ is the LLR value of the most reliable symbol and 0 belongs to the least reliable symbol. Moreover, with this representation of LLRs, the compensation value γ, can also be considered as zero. Hence, we dont require to carry an extra element γ.

With cluster codes, there are two symbol sets belonging to two different groups; $\mathbb{G}(2^{p_1})$ and $\mathbb{G}(2^{p_2})$. We represent by $\alpha^i_{p_1}$, a symbol belonging to the group $\mathbb{G}(2^{p_1})$ and by $\alpha^i_{p_2}$, a symbol belonging to the group $\mathbb{G}(2^{p_2})$. Since the variable nodes are processed in $\mathbb{G}(2^{p_1})$, the channel likelihood vector L_{ch} is computed in order 2^{p_1}. Moreover, it is not truncated as $2^{p_1} < 2^{p_2}$ and we dont require it to be sorted in ascending or descending order. The LLR-value of a symbol α^i_{p1} is computed as per eq. (3.1).

ii). Variable nodes update:

All the messages involved at the variable nodes are of the same order and represent all the possible 2^{p_1} symbols without any message truncation. Therefore, the variable nodes processing correspond to an extrinsic element by element sum of the input messages.

$$U_{vf_i}[\alpha^k_{p_1}] = L_{ch_v}[\alpha^k_{p_1}] + \sum_{j=0, j\neq f}^{d_v-1} V_{f_j v}[\alpha^k_{p_1}] \tag{3.3}$$

where $\alpha_{p_1}^k \in \mathbb{G}(2_1^p)$. As we can observe that this process is simple an element by element addition of the LLR-vectors and the channel likelihood. In section 3.2.4, we will see that it requires only a simple adder. Thus, it is a very simple process as compared to the classical EMS algorithm for $GF(q)$-LDPC codes.

iii). Function nodes update:

The function nodes update is a mapping process between messages of two different orders i.e. between orders 2^{p_1} and 2^{p_2} as depited in Fig. 3.2. The input message vector U_{vf} is an order-2^{p_1} message and the output message U_{fc} is defined in order 2^{p_2}. The mapping process can be depicted as:

$$U_{fc}[\alpha_{p_2}^i] = \begin{cases} U_{vf}[\alpha_{p_1}^j] & \text{if} \quad \alpha_{p_2}^i = f(\alpha_{p_1}^j) \\ -\infty & \text{otherwise} \end{cases} \tag{3.4}$$

where $\alpha_{p_2}^i \in \mathbb{G}(2^{p_2})$ and $\alpha_{p_1}^i \in \mathbb{G}(2^{p_1})$ and $f(.)$ is the mapping function. The output message thus contains 2^{p_1} LLR values and the remaining $(2^{p_2} - 2^{p_1})$ carry the value $-\infty$. Therefore, the LLR vector can also be represented by a vector U_{pc} is also a vector of size 2^{p_1}, with another vector $\beta_{U_{pc}}$ carrying the symbol information corresponding to each LLR-value.

The function nodes processing can also be explained as: consider the set $S(\beta_f)$, a set of $\mathbb{G}(2^{p_2})$ symbols mapped by the function node:

$$S(\beta_f) = \{\alpha_{p_2}^i : \alpha_{p_2}^i = f(\alpha_{p_1}^j)\} \tag{3.5}$$

Thus the message U_{vf} is mapped to a message U_{fc} such that:

$$\beta_{U_{fc}}[i] = S(\beta_f)[i] \tag{3.6}$$

iv). Check nodes update

Once the messages are mapped to order-2^{p_2}, they are sorted and the log domain check udpate process can be now be carried out using the forward-backward strategy explained in section 2.2.4. The input messages are of size 2^{p_1} and the intermediate messages are truncated to size n_m with $n_m << 2^{p_2}$, the check update process if founded on the EMS algorithm. For the elementary process, we used a recently introduced method [BCC10] which is more efficient as compared to the one presented the classical EMS algorithm in [VDV$^+$10]. This method computes the highest n_m combinations of the LLR-values while requiring reduced resources.

Consider two inputs U_{p_1c} and U_{p_2c}, with $\beta_{U_{p_1c}}$ and $\beta_{U_{p_2c}}$ carrying their corresponding

symbol information. Their combination form the 2-D matrix, denoted M, and depicted in the Fig. 2.19. As the input messages are sorted in decreasing order, the higher values are present in the upper-left triangle of the matrix M with highest value at index $(0, 0)$. The sorter is initialized only with the element $M[0][0]$. Then, two neighbours come into play, the right neighbor $M[i][j+1]$ and the bottom neighbor $M[i+1][j]$ respectively. The right neighbor is added if the sorter does not already contain an element belonging to the same column as the neighbor. Similarly, the bottom neighbor is added on the condition that the sorter does not possess an element belonging to the same row as the neighbor. There are four possible scenarios for the two neighbors.

- **Add the right neighbor:** The right neighbor $M[i][j+1]$ is added if the sorter does not contain an element $M[k][j+1]$, such that $k < i$ as shown in Fig. 3.3(a).
- **Add the bottom neighbor:** The bottom neighboring element $M[i+1][j]$ is added if the sorter does not contain an element $M[i+1][k]$, such that $k < j$. It is shown in Fig. 3.3(b).
- **Add both neighbors:** Both, right and bottom, neighbors are added if the sorter does not contain preceeding elements belonging to the same column and row of each neighbor respectively, as shown in Fig. 3.3(c).
- **Dont add any neighbors:** If both neighbors have their preceeding elements respectively, then there is no need to insert any of the neighbors to the sorter, as shown in Fig. 3.3(d).

In Fig. 3.3, the dark colored elements represent the LLR-combinatoins that have already been processed, gray represent the LLR-values currently residing in the sorter and the crossed element represent the element which is the current highest element in the sorter and copied to the output. The right and bottom neighbors are represented with dots. For a right neighbor, if there is a gray element in the same column, it is not inserted into the sorter. Similarly, for each bottom neighbor to be added, it has to be verified that there is no gray element belonging to the same row.

With this method, elements are added into the sorter in a more efficient manner. An element is added to the sorter, if it does not contain a higher valued element belonging to the same row or column. In order to compute the highest n_m values, a sorter of size $\sqrt{2n_{op}}$ is sufficient. n_{op} is the number of operations required to compute the highest n_m values and it is generally fixed to $n_{op} = 2n_m$. The authors of [VDV$^+$10] had proposed a sorter of size n_m.

Moreover, a compensation offset is also added to the output LLR values. In [VDV$^+$10], the offset is added to to the non-truncated LLR values. This keeps the compensation value $\gamma = 0$. Fig. 3.4 represents the elementary check update process in the form of a flow-chart.

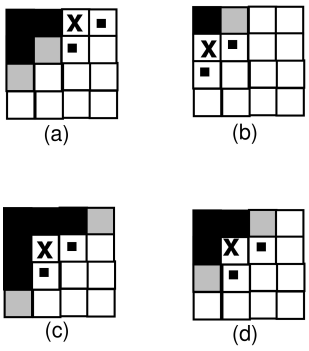

Figure 3.3: Elementary check update process, various scenarios for adding neighbors to the sorter

v). Reverse function nodes processing

At the check nodes output, a message V_{cp} is an order-2^{p_2} message of length n_m, with the vector $\beta_{V_{cp}}$ carrying the corresponding symbol information. The function nodes then maps the message V_{cp} into a message V_{pv} defined in order 2^{p_1} by using the function $f_{ij}(.)$ in the reverse direction. However, since the the message V_{cp} is a truncated message of length n_m, it may not necessarily contain the LLRs of the symbols belonging to the symbols-set $S(\beta_f)$ defined by the mapping function $f(.)$. In such a case, the LLR-value zero is used.

$$V_{pv}[\alpha_{p_1}^j] = \begin{cases} V_{cp}[\alpha_{p_1}^i] & \text{if} \quad \alpha_{p_2}^i \in S(\beta_f) \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

This results in one whole iteration to be completed. Just like in all iterative decoders, at the end of each iteration, the APP is calculated and tests are performed to verify whether a valid codeword had been received or not. In the case of an invalid codewode, the iterations are repeated until either a valid codeword is received or a fixed number of iterations have expired. If the iterations expires without decoding a valid codeword, a decoding failure is declared for the received word.

3.1.2 Monte-Carlo simulation results

In this section, we present the decoding performance, in terms of frames error rate (FER), of the EMS algorithm generalized to cluster codes . We also present the decoding performance of the EMS algorithm applied to classical LDPC defined over a Galois field $GF(q)$, with the same code length (N) and rate (R).

Fig. 3.5 presents the decoding performance for a code of length $N = 575$-bits and

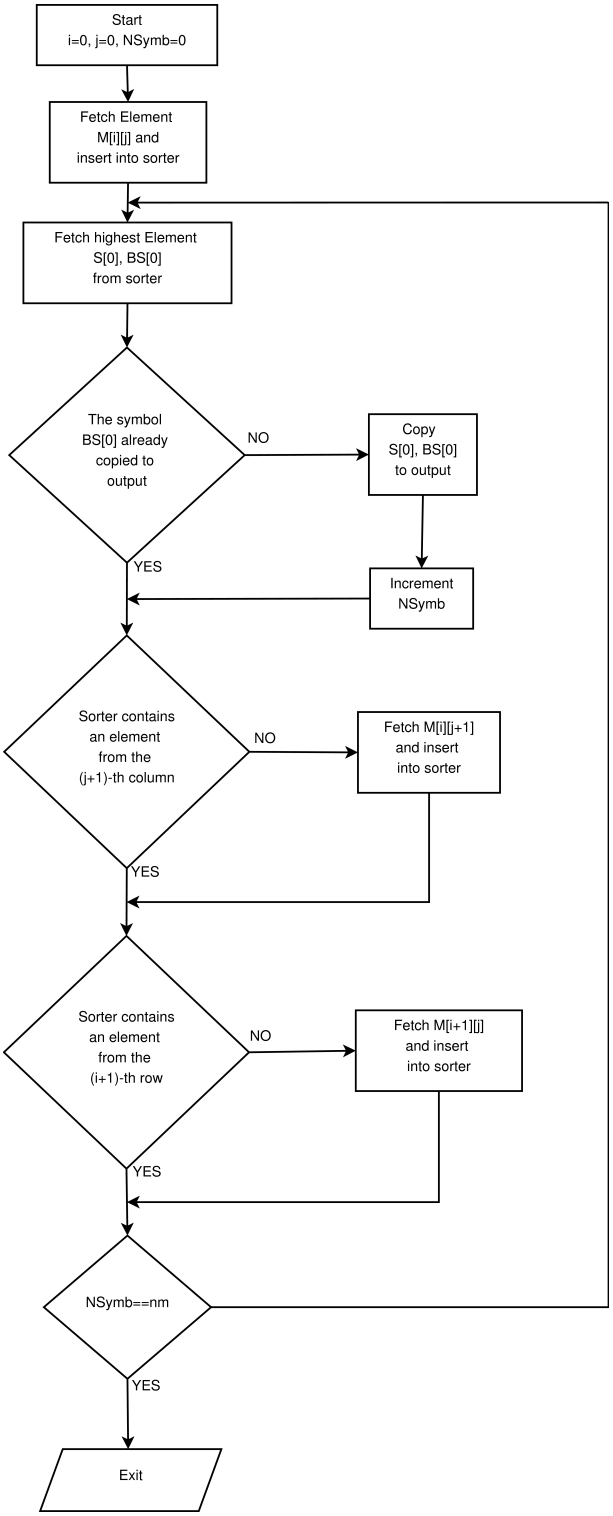

Figure 3.4: Flow chart for the check update algorithm

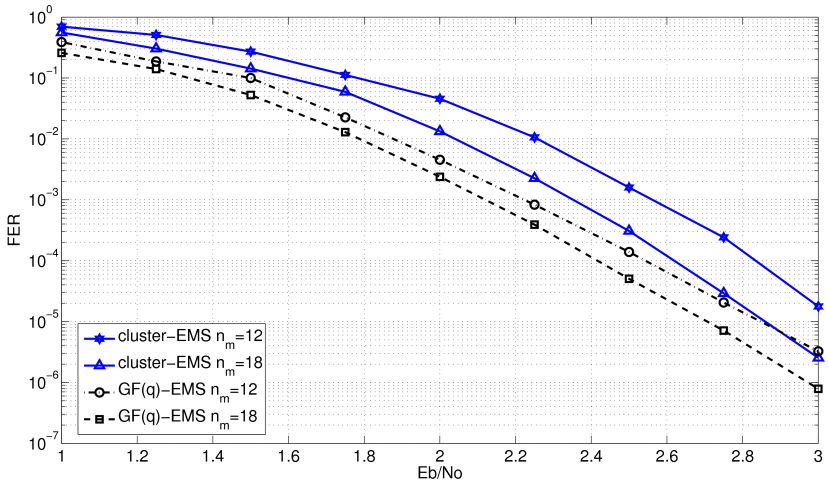

Figure 3.5: EMS for a $GF(64)$-code and cluster-code with $p_1 = 4$, $p_2 = 6$, $N = 576$-bits and $R = 1/2$

rate $R = 1/2$. The Galois field code is defined over $GF(64)$, whereas the cluster-code is constructed over a cluster size $(p_2 \times p_1) = (6 \times 4)$. Thus, the variable nodes are processed in order 2^4 and the check nodes in order 2^6. The messages flow across the Tanner graph using flooding scheduling. The maximum number of allowed iterations are fixed to 100. We compare the performances for the message truncation values $n_m = 12$ and $n_m = 18$. For lower Eb/No, as compared to the classical the $GF(q)$ code, the cluster-LDPC code observes a loss of around 0.3dB and 0.4dB for $n_m = 12$ and $n_m = 18$ respectively. However, the performance gap between the two codes decreases as Eb/No increases and it is expected that cluster-EMS might eventually output perform the $GF(q)$ EMS.

In Fig. 3.6, we verify the performances for a half-rate code with a larger length i.e. 2304-bits. The $GF(64)$-code again shows an improved performance in the water fall region as compared to the (6×4) cluster-code. Cluster-codes exhibit a loss of about 0.2dB for both $n_m = 12$ and $n_m = 18$. However, for higher Eb/No we observe a significant improvement in performane and cluster-EMS outperforms the $GF(64)$ code at about $Eb/No >$ 2dB for both, $n_m = 12$ and $n_m = 18$. Thus, we can deduct that cluster codes tend to have a lower error floor as compared to $GF(q)$ codes.

In Fig. 3.7, we study the effects of using different cluster orders for the same code i.e. (6×3) and (6×4). Thus we verify the performance while first processing the variable nodes in order $2^3 = 8$ and then $2^4 = 16$. The check nodes are processed in order $2^6 = 64$ for both cases. As compared to the $GF(64)$ code, we have the same

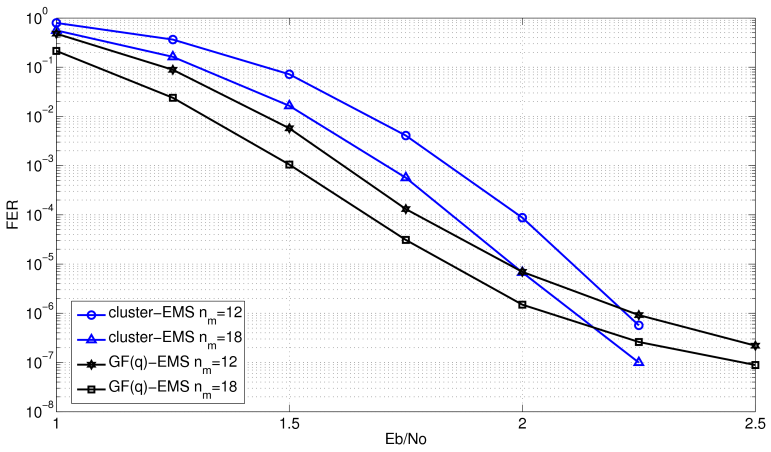

Figure 3.6: EMS for a $GF(64)$-code and cluster-code with $p_1 = 4$, $p_2 = 6$, $N = 2304$-bits and $R = 1/2$

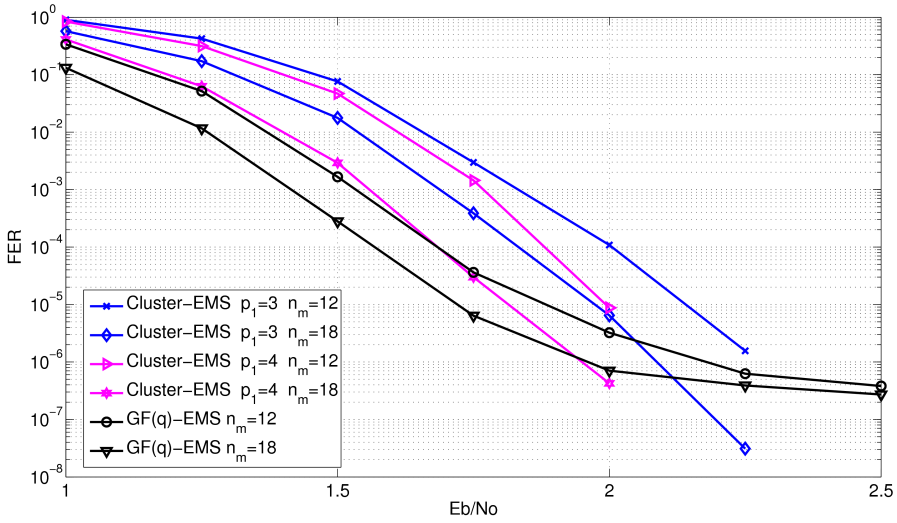

Figure 3.7: EMS for a $GF(64)$-code and two cluster-code with $p_1 = 3, 4$, $p_2 = 6$, $N = 3000$-bits and $R = 1/2$

kind of performance as before i.e. loss in the waterfall region and a gain for higher Eb/No. However, an important point to note here is that the code with cluster order (6×4) performs better than code with a cluster order (6×3). This is because the code with the $\mathbb{G}(16) \leftrightarrow \mathbb{G}(64)$ is more close to that of the $GF(64)$ code as compared to the one with $\mathbb{G}(8) \leftrightarrow \mathbb{G}(64)$. Thus, the variable nodes order also plays an important role

in defining the performance of a cluster code.

3.2 Improved EMS decoder for cluster codes

The EMS decoder generalized to cluster codes, presented in the previous section, is not completely adapted to cluster-LDPC codes. In this section, we present an improved version of the algorithm. We propose two modification at the check nodes level and prove the increase in decoding performance with Monte-Carlo simulations. We also present a hardware architecture for the proposed improved decoder and compare the decoding complexity with the $GF(q)$-EMS decoder.

3.2.1 A second elementary process

As a first modification, we propose to add another elementary process into the forward-backward mechanism that replaces uniquely the elementary processes used to compute the extrinsic outputs of the check nodes. The elementary processes used to compute the intermediate messages remain the same as discussed in section 3.1.

In the method proposed in section 3.1, at the output of the check nodes, a message V_{cf} is an LLR-vector of size n_m and defined in $\mathbb{G}(2^{p_2})$. The vector $\beta_{V_{cf}}$ carries the corresponding symbol information. The function node f then maps the message to order 2^{p_1}, using the set $S(\beta_f)$ defined in eq. 3.5. It is the set of symbols $\alpha^i_{p_2} \in \mathbb{G}(2^{p_2})$ formed by mapping the symbols $\alpha^j_{p_1} \in \mathbb{G}(2^{p_1})$ using the function $f_{ij}(.)$. If the symbol corresponding to the symbol $\alpha^j_{p_1} \in \mathbb{G}(2^{p_1})$ is not present in $S(\beta_f)$, a null LLR-value is mapped instead.

The vector V_{cf}, at the check nodes output, is a truncated message and contains n_m LLR-values corresponding to n_m symbols stored in $\beta_{V_{cf}}$. The probability that a symbol $\alpha^i_{p_2} \in S(\beta_f)$ is present in $\beta_{V_{cf}}$ is $\frac{n_m}{2^{p_2}}$, whereas $n_m << 2^{p_2}$. This means that there is a high probability that symbols belong to the set $S(\beta_f)$ may not be present in the check node output messages. Thus, most of the time null LLR-values will be mapped and the message V_{fv} at the output of the function node will contain mostly zeros. Therefore, we have a loss of information and consequently a loss in decoding performance.

As a remedy to the loss of information, we proposed to use a second type of elementary process. It is not an additional extra process as it replaces at certain instances the previousely explained elementary process. We refer to this new elementary process as **Elem2** and we call the previous elementary process as **Elem1**. The forward-back strategy depicted in Fig. 2.18 has all the elementary nodes implemented as Elem1. Now, the elementary nodes that compute uniquely the extrinsic outputs are replaced

by Elem2. Elem1 remains un-changed while computing the intermediate messages

The basic principle of Elem2 is same as Elem1 i.e. scanning the 2-D matrix M formed by the combination of the two sorted inputs. However, unlike Elem1, the output of Elem2 is not the highest n_m combination LLR-values and their corresponding symbol information. Rather it is an output vector of size 2^{p_1} and contains the LLR-values of only the symbols mapped by the corresponding function node. This means that for the edge connected to the function node f, with the associated symbols set $S(\beta_f)$, Elem2 computes the LLR-values of the symbols $\alpha^i_{p_2} \in S(\beta_f)$. With this, all the original LLR values are then mapped onto the 2^{p_1}-order message as the LLR-values of all the symbols to be mapped are present in V_{cf}.

The second elementary process, Elem2, uses following procedure:

1. Inputs: The two input messages (U_1, β_{U_1}) and (U_2, β_{U_2}), The function node symbol-set $S(\beta_f)$
2. Initialize: Initialize the sorter to all-zeros and insert $M[0][0]$, no. of elements copied$= n$, Initialize all output LLR-values to -1
3. Output: Highest value of the sorter $s[0]$ and the corresponding symbol information $\beta_s[0]$
4. Test: $\beta_s[0] \in S(\beta_f)$ and $V[\beta_s[0]] = -1$
   - Yes: Copy $s[0]$ and $\beta_s[0]$ to output $V[\beta_s[0]]$, increment $n = n + 1$
   - No: Discard the value and take no action
5. Control:
   - Add right neighbor of the element $s[0]$ with respect to M, if no element of the same column is present in sorter s
   - Add bottom neighbor of the element $s[0]$ with respect to M, if no element of the same row is present in sorter s
6. if $(n = 2^{p_1})$ exit - else go to step-3

This modified elementary process may introduce a slight increase in complexity as the matrix M is scanned to find the higher LLR combination of certain 2^{p_1} symbols. Thus, it may lead to scanning a higher number of elements of M as compared to Elem1. However, with the second modification that we proposed, this increase in complexity is annuled as it places a limitation of scanning the highest $n_m^2/2$ elements, which is similar to Elem1.

3.2.2 Improved estimation of output LLRs

This modification is applied to both elementary steps, Elem1 and Elem2. It not only improves the output of the check nodes, it also helps in cancelling the increase in complexity due to the second elementary step, Elem2.

The check node input message U_{fc} can be written in the full sized sorted LLR-vector as:

$$U_{fc} = [U_{fc}[0] \;\; U_{fc}[1] ... \;\; U_{fc}[2^{p_1} - 1] \;\; -\infty \;\; -\infty \;\; ... \;\; -\infty] \tag{3.8}$$

The symbols corresponding to the LLR-value of $-\infty$ are un-authorized states. However, a temporary message T_i, computed while using the forward-backward strategy may not necessarily contain un-authorized states. However, since we are using truncated messages of size n_m and the truncated symbols carry an LLR-value equal to zero, the intermediate message T_j in the full sized sorted LLR-vector format is represented as:

$$T_j = [T[0] \;\; T[1] \;\; ... \;\; T[n_m - 2] \;\; 0 \;\; 0 \;\; ... \;\; 0] \tag{3.9}$$

Consider an elementary process Elem1 with the two message vectors U_{fc} and T_j, as inputs. The matrix M formed from the combination of the two inputs vectors is a matrix of size $(2^{p_2} \times 2^{p_2})$ as shown in Fig. 3.8. The lower $((2^{p_2} - 2^{p_1}) \times 2^{p_2})$ elements carry the value $-\infty$ because of the un-authorized states in U_{fc}. Similarly, the each row from the right $(2^{p_1} \times 2^{p_1} - n_m + 1)$ elements carry the same LLR-value $U_{fc}[0]$. They are represented by a cross in the figure.

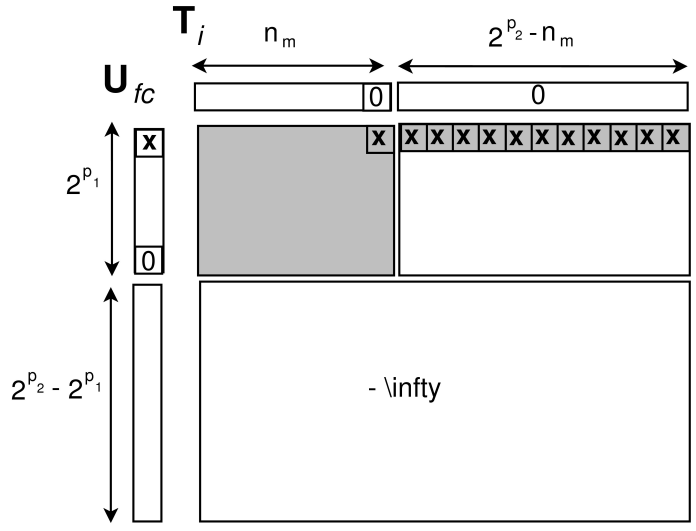

Figure 3.8: The matrix M formed with full-size input vectors U_{fc} and T_j

The elementary processes select the highest combination LLRs-values by starting with

the element $M[0][0]$. The right and bottom neighbours are then inserted into the sorter. After choosing any number of combinations, say n, if the element at index $M[0][n_m - 1]$ is selected i.e. the element depicted with a cross in Fig. 3.8, the remaining $(2^{p_2} - n)$ symbols will also be selected from the same row. Thus, all the remaining symbols will carry the same LLR-value $U_{fc}[0]$. The output V_i can thus be expressed as:

$$V_i = [M[0][0] \;\; V[1] \;\; ... \;\; V[n] \;\; U_{fc}[0] \;\; U_{fc}[0] \;\; ... \;\; U_{fc}[0]] \tag{3.10}$$

Now, applying the same procedure to truncated messages of length n_m, whenever the last element of the first row is selected, we stop scanning further elements and fill the remaining output indices with the LLR-value last selected. Thus, the output has now closer resemblance to the output of the full-size LLRs. Also, we place a limitation on the maximum number of elements to be scanned i.e. $n_m^2/2$. Therefore, Elem2 has the same limits as Elem1. This process cannot be applied to the side containing the values $-\infty$.

3.2.3 Monte-Carlo simulation results

In this section, we present the Monte-Carlo simulation results for the improved cluster-EMS decoder. We compare the decoding performance with the pervious version of the decoder and clearly observe an improved decoding performance with the proposed modifications. Let us also recall that cluster codes were proposed to attain better performance in the error floor while accepting a slight degradation in the waterfall region as comapred to $GF(q)$ codes.

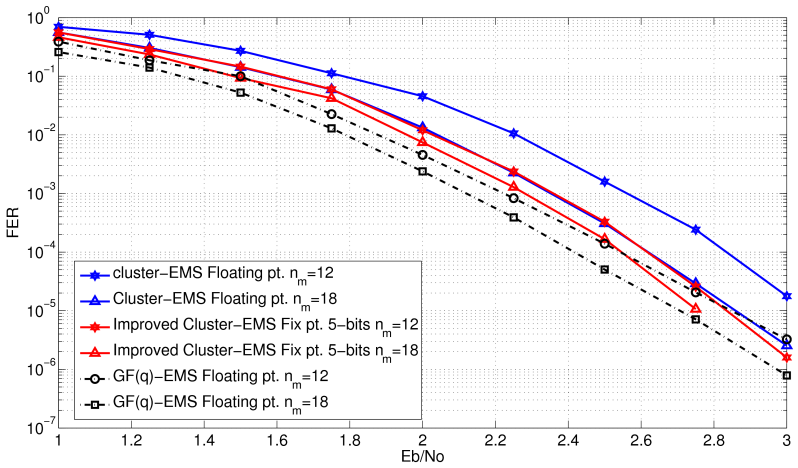

Figure 3.9: Improved EMS for a cluster-code with $p_1 = 4$, $p_2 = 6$, $N = 576$-bits and $R = 1/2$

Fig. 3.9 presents the simulation results for a half rate code of length N=576-bits. The regular field-LDPC codes is defined over $GF(64)$ whereas the cluster code has a cluster-size of (6×4). The variable nodes are thus processed in order-2^4 and the check nodes in order 2^6. As compared to EMS for the $GF(64)$ code, in the water-fall region, the direct generalization of cluster-EMS has a loss of 0.3dB and 0.4dB for $n_m = 12$ and $n_m = 18$ respectively. With the modified cluster-EMS algorithm, the loss is is reduced to 0.1dB and 0.15dB respectively. Note that this small loss comes mainly from the code itself rather than the decoder behaviour, so this small loss was expected. This is verified since the performance degradation is further reduced for lower error rates, until eventually the cluster codes outperform the $GF(q)$ code.

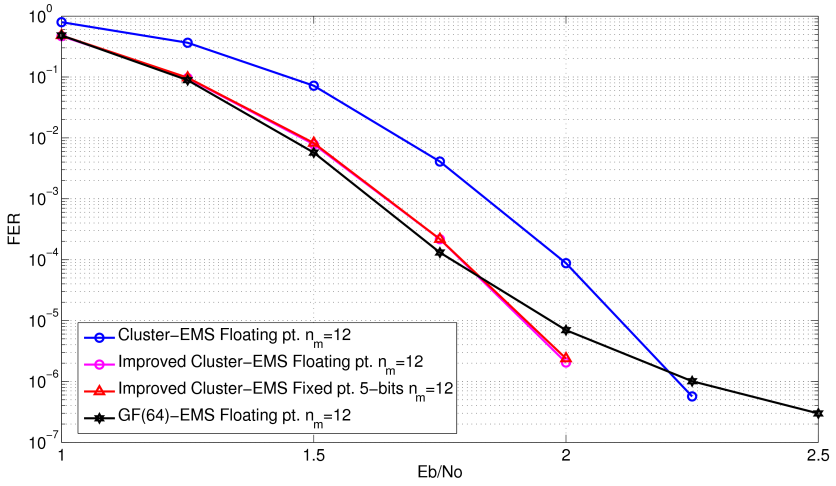

Figure 3.10: Improved EMS for a cluster-code with $p_1 = 4$, $p_2 = 6$, $N = 2304$-bits and $R = 1/2$

In Fig. 3.10, we consider a half rate code with $N = 2304$ bits and a cluster size (6×3). We present the performance of the improved cluster-EMS using both, floating point and fixed point implementations. For the fixed point implementation, we use a special LLR structure which is explained in the next section. we observe that as compared to the floating point implementation, there is a negligible loss in performance while using the fixed point implementation with 5-bits quatization levels. Moreover, like in the previous case, cluster-EMS exhibits a loss in performance in the waterfall region as compared to the code over $GF(64)$. However, the modified version of EMS for cluster codes outperforms the $GF(q)$-EMS for $Eb/No > 1.8$dB for $n_m = 12$ and there is a gain of almost $0.25dB$ as compared to the previous version of cluster-EMS. This demonsrates the superiority of the newly proposed decoding procedure.

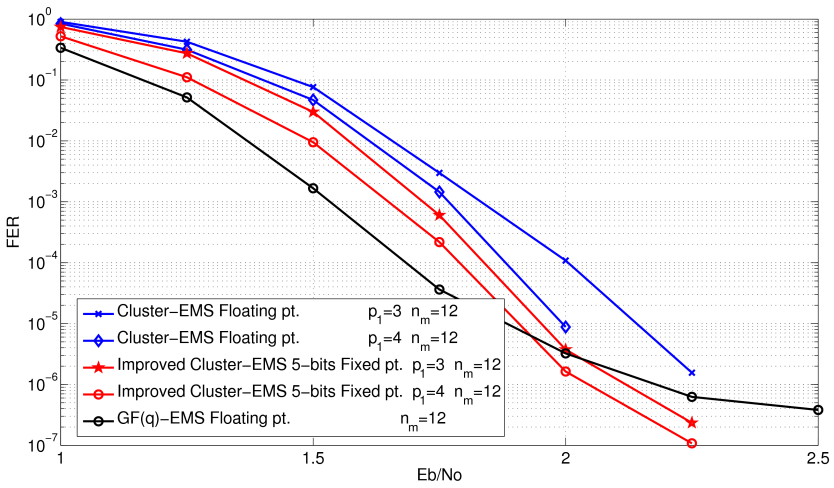

Figure 3.11: Improved EMS for two cluster-codes with $p_1 = 3, 4$, $p_2 = 6$, $N = 3000$-bits and $R = 1/2$

In Fig. 3.11, we present the performance of the improved EMS decoder using clusters of two different sizes of the same code i.e. cluster sizes (6×3) and (6×4) and $N = 3000$ bits. We have an improved performance as compared to the previous version of EMS algorithm for cluster codes. With $n_m = 12$, for $p_1 = 4$ and $p_1 = 3$ respectively, the loss with respect to the EMS for a $GF(64)$ code in the water-fall region is reduced to 0.1dB and 0.2dB with the new cluster-EMS as compared to the 0.25dB and 0.3dB loss with the older version. Moreover, the new version of cluster-EMS outperform the $GF(64)$ code at $Eb/No = 1.75$dB for $p_1 = 4$ and 2dB for $p_1 = 3$.

3.2.4 Hardware Architecture

In this section, we propose a serial hardware architecture for our proposed decoder. Due to the sequential nature of the decoder, the decoder requires less resources which is critical for a NB-LDPC decoder. Fig. 3.12 depicts the main components of the decoding process. The various components of our proposed decoder requires less resources as compared to the architecture proposed for the EMS algorithm for $GF(q)$ codes in [VVD$^+$07].

Quantization and LLR-vectors:

The channel information is stored in likelihood vectors $\{L_i\}_{i=0,\ldots,N-1}$, each of size 2^{p_1}, acting as inputs to the N variable nodes. However, we use a different format for LLRs as compared to the previousely used LLR-formats. The LLR of a symbol $\alpha^i \in \mathbb{G}(q)$ is

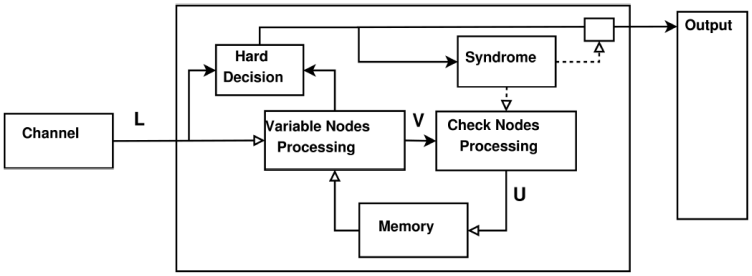

Figure 3.12: The main components of the decoder

structured as:

$$L(\alpha^i) = log\left(\frac{P(\alpha_{max})}{P(\alpha^i)}\right) \tag{3.11}$$

where α_{max} is the symbol in the message which is most reliable. With this representation, a higher LLR value represents a lower probability of likelihood. The most reliable symbol always carry an LLR-value of 0. Thus, an LLR-vector sorted on the basis of reliability will be sorted in ascending order.

This structure of LLRs results in a better quantization scheme as compared to the previously defined LLR-formats. It is more robust and reduces the saturation noise which in turn improves the decoding performance. For 5-bits quantization, the LLR-values varies in the range $\{0, ..., 31\}$, 0 being the LLR-value of the most reliable symbol whereas the least possible reliabity LLR-value is 31. Saturation thus occurs at the less reliable symbols and not at the most reliable ones. The LLR-value of the most reliable symbol always remain at 0.

With this representation of LLRs, the arithematic operations of the decoding procedure remain the same, however the logical operations may be changed. A $max()$ operation has to be replaced by a $min()$ and vice-vera.

Variable udpate process:

The variable nodes are processed in a smaller order $\mathbb{G}(2^{p_1})$ as compared to the check nodes $\mathbb{G}(2^{p_2})$ and thus smaller message vectors. This results in a very simple variable update process. For a variable node of degree $d_v = 2$, the output U_i is computed as a simple element by element addition of the input message v_i and the channel likelihood L_i. It requires 2^{p_1} clock cycles. For a higher degree node, the number of cycles are increased $(d_v - 1)$ times.

With a parallel architecture, using 2^{p_1} processing units in parallel will execute the

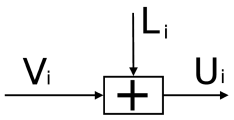

Figure 3.13: Variable update processor

process in a single clock cycle. This shows that cluster NB-LDPC codes with the proposed architecture could be a viable solution for very high rate applications.

Check update process:

The two types of elementary modules depicted in section 3.2 are joined together in a recursive fasion to process the check nodes.

Elementary process 1:
Fig. 3.14 shows the proposed architecture for the check update process. It consists of a sorter, an adder, a GF-adder and a symbol verification unit. The sorter input LLRs and their corresponding symbol index information are received in a serial fasion and stored in the input registers. The two input LLRs are fed to the adder. The symbol index information also flows along with the LLRs until the final output. To compute the n_m highest values, the architecture presented in [VVD$^+$07] requires a sorter of length n_m. However, with the bubble-check algorithm we require a sorter of length $\sqrt{2n_{op}}$, where n_{op} is the number of operations required to compute the highest n_m LLR combination values. It is fixed to $n_{op} = 2n_m$ as proposed in [CCAGB00].

Moreover, the serial architecture of [VVD$^+$07] requires the initialization of the sorter which takes n_m cycles, however our serial architecture does not require it. The process is executed just after the first values of both inputs are received. This leads to the reduction of the overall latency.

The highest LLR-value of the sorter and its corresponding symbol are then copied to the output provided that the same symbol has not already been copied to the output. This verification is processed by the symbol verification unit (SVU). In the next cycle, new LLR-values and their corresponding symbol information are fetched and processed as explained in section 3.1. When the two neighboring values have to be fetched, no value is copied to the output during the first cycle. During the second cycle, the highest value of the sorter is copied to the output after the second neighbour is fetched. However, when no neighbour is required to be inserted into the sorter, the right neighbour is fetched in order to avoid the wastage of the cycle. We have verified through various simulations that the number of cycles for this elementary step when fixed to $n_m + 5$ gives good performance.

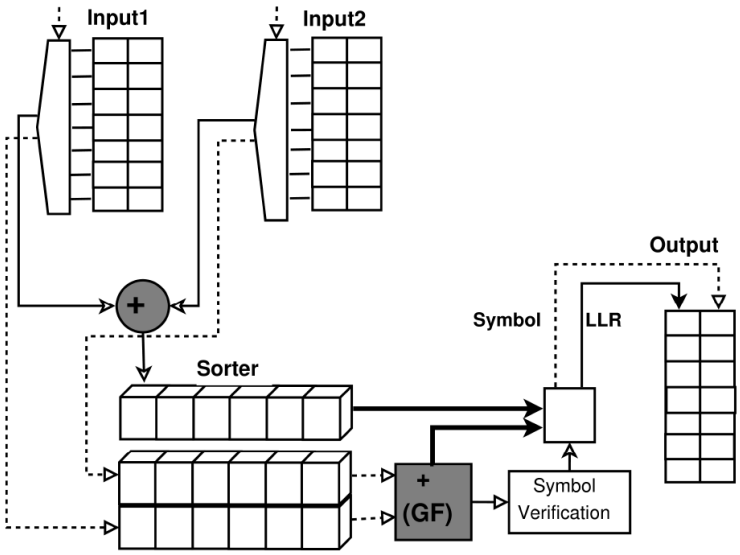

Figure 3.14: Improved EMS for two cluster-codes with $p_1 = 3, 4$, $p_2 = 6$, $N = 3000$-bits and $R = 1/2$

Elementary Process 2:
The second elementary process follows the same architecture as the first one except for the symbol verification unit. The symbol verification unit allows an LLR value to be delivered at the output only if the corresponding symbol belongs to the set of symbols mapped by the function node which is also given as an input. This architecture has the same overall complexity as Elem1, but we showed that the symbol verification unit has a strong impact on the error correcting performance.

3.2.5 Complexity comparison

We now analyze the computational complexity of our proposed decoder. We also compare it with the EMS algorithm proposed for $GF(q)$-codes in [VDV$^+$10].

The EMS based algorithm uses truncated message vectors in order to reduce the decoding complexity and the memory requirements. For a $GF(q)$ code, it uses vectors of size n_m instead of q, where $n_m << q$. For each LLR-vector, only the highest n_m values are considered. The remaining $(q - n_m)$ elements are represented by single value γ. However, we considered LLR-vectors with strictly positive LLR-values and considered $\gamma = 0$. The memory required to store the message vectors depend linearly on n_m and the complexity is dominated by $\mathcal{O}\left(n_m^2\right)$. Therefore, the value of n_m should be selected such that it assures both, decoding performance and an acceptable decod-

ing complexity.

An advantage of cluster-codes is the memory reduction. The variable nodes are gen-

Algo.	**Variable Nodes**	**Check Nodes** (Input/Output)	**Check Nodes** (Intermediate)
$GF(q)$-Minsum	q	q	q
$GF(q)$-EMS	n_m	n_m	n_m
Cluster-EMS	2^{p_1}	2^{p_1}	n_m

Table 3.1: Size of LLR-vectors

erally defined in a smaller order $\mathbb{G}(2^{p_1})$ as compared to the check nodes $\mathbb{G}(2^{p_2})$, where $p_1 \leq p_2$. With full complexity vector representation, we gain a significant amount of memory with cluster-LDPC codes as compared to $GF(q)$-LDPC codes. However, the gain may vary for the EMS-based algorithms depending on the value of n_m and p_1. For example, for a rate $R = 1/2$ code, the value $24 \leq n_m \leq 12$ is generally considered [CCAGB00]. For a cluster-LDPC code with $p_1 = 3$, a message is composed of $2^3 = 8$ LLR-values whereas for $p_1 = 4$, each message is composed of 16 values. However, the added advantage of cluster codes is that the messages are reserved in their full format without any truncation. Message trunacation only occurs for the intermediate message computed during the forward-backward procedure at the check nodes.

Table 3.2.5 presents the complexity of variable nodes update process in terms of number of operations performed. As we can observe the variable update process for cluster-EMS is a very simple process and is realized by the element by element addition of the input vectors. Table 3.2.5 presents the number of operations performed at the check nodes. There is no difference in terms of number of operations performed at the check nodes for the same value of d_c. However, However, since for cluster codes, the degree of check nodes is increased by a factor p_2/p_1 as compared to a $GF(2^p)$ code, where $p = p_2$. Consequently the check nodes processing complexity is also increased by a factor $2^{p_2}/2^{p_1}$.

Algo.	**Max**	**Real Addition**
$GF(q)$-EMS	$n_m(n_m + 2)$	n_m
Cluster-EMS	-	2^{p_1}

Table 3.2: Number of operations at a variable node with $d_v = 2$

Algo.	Max	$GF(q)$-Addition	Real Addition
$GF(q)$-EMS	$3(d_c-2)n_{op}\sqrt{2n_{op}}$	$3(d_c-2)n_{op}$	$3(d_c-2)(n_{op}+n_m)$
Cluster-EMS	$3(d_c-2)n_{op}\sqrt{2n_{op}}$	$3(d_c-2)n_{op}$	$3(d_c-2)(n_{op}+2^{p_1})$

Table 3.3: Number of operations at a check node

3.3 Diversity of group-LDPC codes

LDPC codes defined over groups offer diversity in terms that different non-binary Tanner graphs of the same code formed form the binary PCM can derive different decoding performances [Dec08]. We make use of this idea and apply it locally at the check nodes level. We propose to use several discrete processes of the same check node in parallel with very low values of n_m. The output of all the various processes are then fusioned together to form the extrinsic outputs that exhibits good decoding performance.

3.3.1 Parallel cluster-EMS check update processes

The parity equation for the i^{th} check node can be written as:

$$\sum_{j=0}^{d_c-1} H_{ij}x_j = 0 \tag{3.12}$$

where H_{ij} are the non-zero elements of the i^{th} row of the PCM. This parity equation can be seen as a local codeword composed of d_c words. Applying a random permutation on the sequence order of the words will keep the same code space but will result in a different local decoding graph. As a consequence, if the same inputs are provided to the two graphs, the output dynamics of the two codes will be different. This result in a different convergence behaviour for each decoder.

We, now, define a diversity as a random permutation applied to the order of the elements of the local parity check matrix of a check node.

$$\mathbb{D}_{p_k}^{(k)} = \pi^{(k)}(H_{ij}) \tag{3.13}$$

A diversity set is now defined as a set of such type of permutations.

$$\{H_{ij}, \mathbb{D}_{p_1}^{(1)}, \mathbb{D}_{p2}^{(2)}, ..., \mathbb{D}_{p_{N_d-1}}^{(N_d-1)}\} \tag{3.14}$$

where N_d is the number of diversities. The check node can now be processed with the different elements of the diversity sets. For the same input LLR-vectors, each instance

of the check node using the diversity set will result in different extrinsic output LLR-values. The outputs of the different instances are then fusioned to obtain the output LLRs which are then fed to the variable nodes. This process is shown in Fig. 3.15.

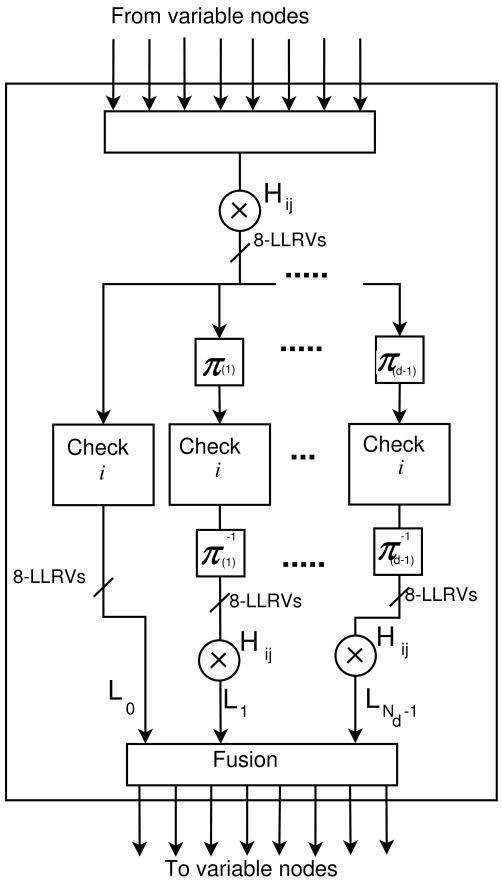

Figure 3.15: Multiple instances of a check update process scrambled in parallel using diversity of group-LDPC codes

To fusion the outputs of the various instances, there are various merging techniques that one can think of. The check update process is founded on the EMS algorithm, the goal of which is to compute the maximum LLR-value of the combination of inputs that verify the parity. Thus, we also use the $max()$ operator also to merge the LLR-vectors. For a certain symbol, we choose the highest LLR-value of that symbol amongst all the LLR-vectors. For example, using $N_d = 3$, for the symbol α^i, the output is chosen as:

$$V_0[\alpha^i] = max(L_0^{v_0}[\alpha^i], L_1^{v_0}[\alpha^i], L_2^{v_0}[\alpha^i])] \tag{3.15}$$

where $L_k^{V_j}$ is the j^{th} LLR-vector corresponding to the output LLR-vector V_j and associated with the check node instance k.

The output LLRs, while using the diversity set, exhibits better convergence behaviour than while using a single instance of the check node. It makes use of the diversity and thus better LLRs are constructed. We prove this with Monte-Carlo simulations as explained in the next section. This method can, thus, be helpful in proposing a low complexity decoder for group-LDPC codes i.e. instead of using a single instance of a check node with a certain value of n_m, employ N_d instances of the check node with a very low value of n_m. This results in an increase in the number of processes but it reduces the overall complexity of the decoder as the complexity at each process separately is significantly reduced. The effect on the over-all complexity is dependent on the number of instances N_d and the value of n_m. We discuss this in section 3.3.3.

Another important point to note is the permutations applied to the sequence order of the non-zero clusters H_{ij}. While constructing a PCM, the binary clusters H_{ij} are positioned such that the code attains the best performance i.e. it has a higher minimum distance and a larger girth. However, a random permutation of the clusters may effect this performance and thus the diversity of group-LDPC may not be efficiently utilised. Thus, different permutations may result in different decoding performances and the best permutation set can be chosen either at the time of construction of the PCM or by brute force method.

3.3.2 Monte-Carlo simulation results

In this section,we present the decoding performance of the proposed decoder that uses the diversity of group-LDPC codes. For various values of n_m, we compare its performance for various values of N_d.

Fig. 3.16 presents the decoding performance for a half-rate code of length $N = 3000$ bits and $(p_1, p_2) = (3, 6)$. Thus, the variable nodes are processed in $\mathbb{G}(8)$ and the check nodes in $\mathbb{G}(64)$. We compare the performance for $N_d = \{1, 2, 3\}$ and $n_m = \{6, 9, 12\}$. The decoder with $N_d = 1$ is the regular cluster-EMS decoder proposed in section 3.2. For $n_m = 12$, there is a gain of about 0.1dB with $N_d = 2$ as compared to the decoder with $N_d = 1$. Similarly for $n_m = 9$, with $N_d = 2$ we gain about 0.15dB. Moreover, the decoders with $N_d = 2$ tend to have a lower error floor. This can also be observed from the performance of the decoder with $N_d = 3$ and $n_m = [6, 6, 6]$. As compared to the regular cluster-EMS decoder with $n_m = 12$ decoder, there is a loss of about 0.2dB in the waterfall region. However, this loss reduces for higher Eb/No, and it is expected that with $N_d = 3$, we have better performance at low error rates of about 10^{-8}. For $N_d = 2$ and $n_m = [9, 6]$, there is also a gain of about 0.05dB as compared to the regular

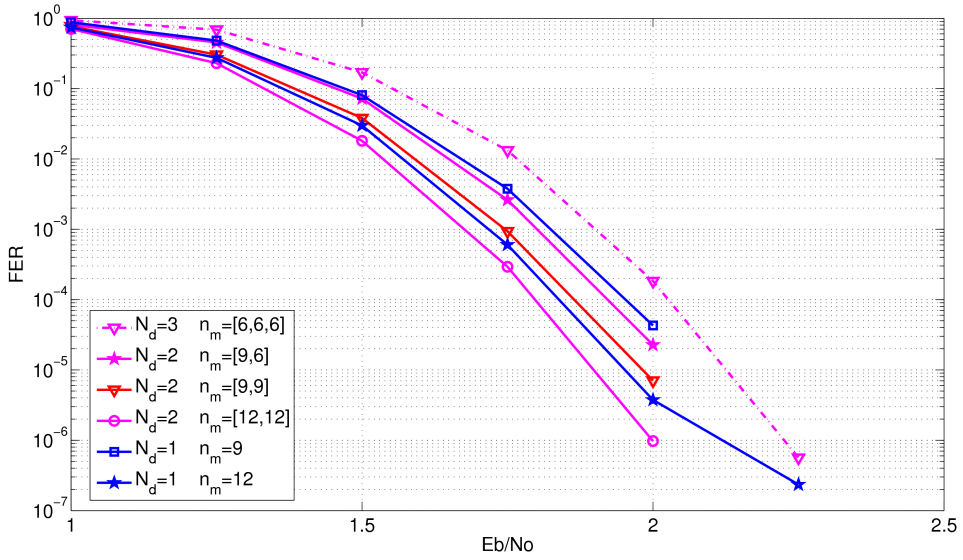

Figure 3.16: Decoder diversity $N = 3000$, $R = 0.5$, $(p_1, p_2) = (3, 6)$

cluster-EMS decoder with $n_m = 9$.

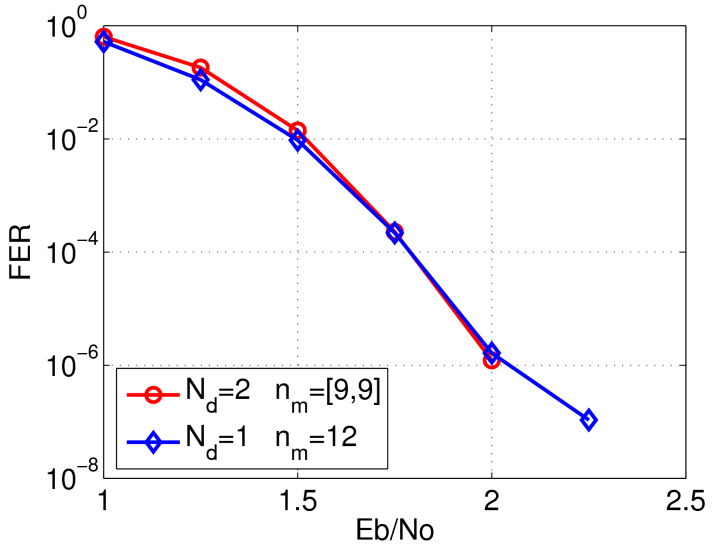

Figure 3.17: Decoder diversity $N = 3000$, $R = 0.5$, $(p_1, p_2) = (4, 6)$

Fig. 3.17 presents the decoding performance for the same code as but with the with a

different order of variable nodes. The clusters are of size (6×4), and thus the variable nodes are processed in order 2^4. With $N_d = 2$ and $n_m = 9$, we see the same performance as before i.e. better performance for higher Eb/No. A lower error floor is also expected.

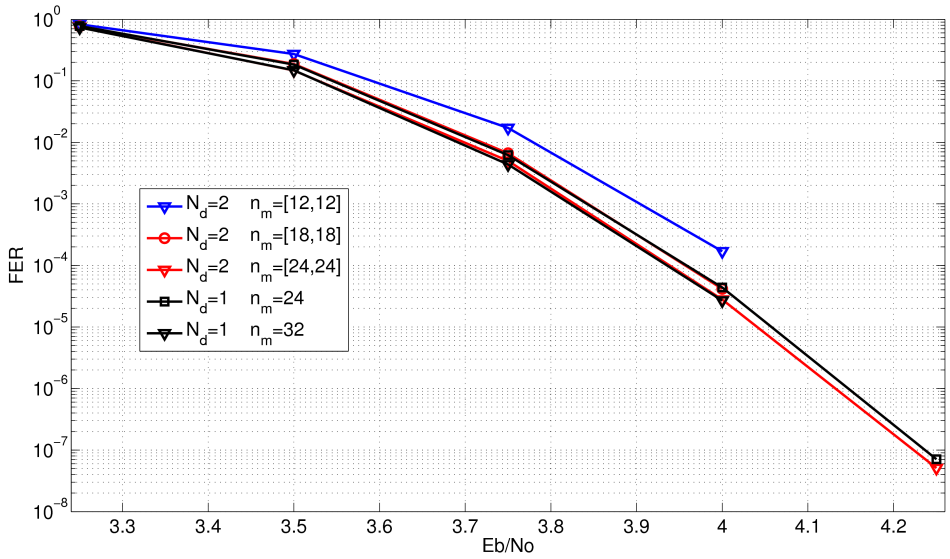

Figure 3.18: Decoder diversity $N = 4800$, $R = 0.88$, $(p_1, p_2) = (3, 7)$

Fig. 3.18 presents the performance for a code of rate $R = 0.88$ and length $N = 4800$ bits. We observe that with $N_d = 2$ and $n_m = 24$, we attain the same performance as the regular cluster-EMS with $N_m = 32$. Similarly, for $N_d = 2$ and $n_m = 18$, we attain the same performance as the regular cluster-EMS decoder with $n_m = 24$.

Thus, we can deduct from the results that, for the same value of n_m, we gain in performance for a higher value of N_d. Therefore, we can obtain a decoder with the same performance but with a lower value of n_m. Moreover, while using the diversity of cluster-LDPC codes, we attain a lower error floor. However, the disadvantage associated is that the number of processes are increased N_d times. We discuss the effect of the value of N_d below.

3.3.3 Complexity comparisons

The total silicon area of the decoder is increased when several instances of the check udpate process scrambled in parallel are added to the decoding procedure. It is in-

creased by a factor linearly proportional to N_d, the total number of processes scrambled in parallel. However, this increase is compensated for by using a smaller value of n_m, with which we require smaller size components and thus the overall surface area is reduced. To get a complete and accurate idea on the comparison of the surface area for various values of N_d and n_m, a physical implementation based on the RTL description is needed. However, we have not realized the hardware implementation and here we present only an analytical comparison of the decoders based on perception and experience.

We consider the architecture presented in section 3.2.4. Fig. 3.13 represents the variable node process which is a simple adder. Fig. 3.14 presents the check node process which compounds almost half of the decoder area. The remaining surface area is occupied by the memory elements, variable and function nodes update process, interleaver and other processes. In order to make a comparison between a classical cluster-EMS decoder and a cluster-EMS decoder with multiple check update processes scrambled in parallel, we need to examine the effects of two factors on the total surface area: *(i).* Effect of the value of n_m *(ii).* Effect of N_d, number of parallel check update processes

Since the check update process consitutes almost half of the decoder area, for $N_d = 2$ and keeping the same value of n_m, the overall area is increased by a factor of 1.5. Fig. 3.19 shows the graph for the effect of N_d on the normalized area of the decoder, for any fixed value of n_m.

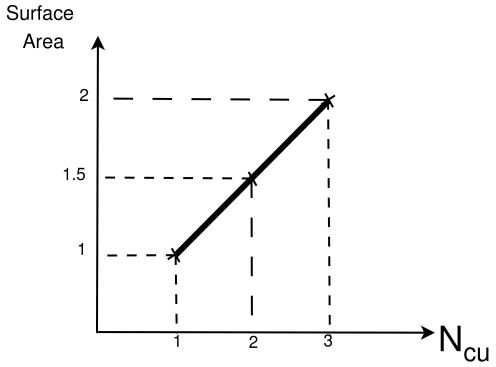

Figure 3.19: Normalized area vs. number of parallel check-update processes

The surface area is increased due to the increased number of check update processes. To reduce this effect, we consider a smaller value of n_m at the various processes. With reference to Fig. 3.14, the value of n_m effects the input registers, sorter, symbol verification unit and the output registers. The input and output registers and the symbol verification unit is increased or decreased directly proportional to the value of n_m,

whereas the sorter realizes a rather heavier increase in complexity with an increased value of n_m. Moreover, with an increased value of n_m, we require more memory elements as to store the messages produced during the forward-backward processing.

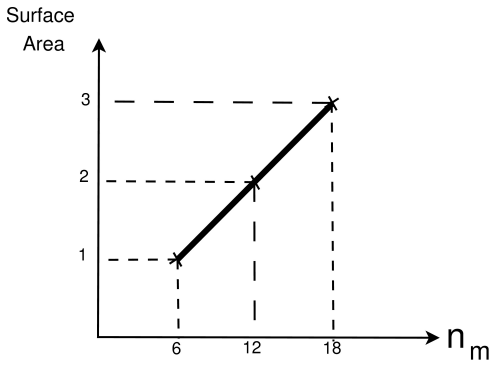

Figure 3.20: Normalized area vs. message truncation value n_m

Using the above mentioned analysis, it can be stated that two check-update processes in parallel and the value of n_m reduced by a factor of 1.5 will result in the same surface area as the regular cluster-EMS decoder. For example, cluster-EMS decoders with $\{N_d = 1, n_m = 12\}$ and $\{N_d = 2, n_m = 9\}$ will result in the same surface area for both decoders. Thus, overall, it can be said that the surface area is doubled when the value of n_m is increased by a factor of 2. Fig. 3.20 shows the graph for the effect of the value of n_m on the surface area for a single check update process.

3.4 Conclusion

In this chapter, we generalized the $GF(q)$-EMS decoder to NB cluster-LDPC codes defined over finite groups. We first present a direct generalization of the algorithm and show its performance with Monte-Carlo simulation results. In the second part of the chapter, we discuss the drawbacks associated with the direct generalization and propose two modifications in the procedure that improves the decoding performance. We prove with Monte-Carlo simulation results that there is a significant gain in performance with the proposed modifications. Moreover, the proposed modifications donot have a significant effect on the decoding complexity.

In the last part of the chapter, we proposed a method to make use of the diversity of cluster-LDPC codes. We propose to use several low complexity instance of a check node in parallel and fusion their outputs to compute the extrinsic LLR-values of the

check node. With this procedure, we can reduce the surface area of the decoder. Moreover, we also obtain a lower error floor which we prove with Monte-Carlo simulations results.

4

A new decoding algorithm for NB-LDPC codes using local lists

THE work presented in this chapter has been done in collaboration with Prof. Emmanuel Boutillon of Université de Bretagne-Sud (UBS).

Along with iterative decoding, another method which gained interest is list decoding. The term list decoding was first proposed by Elias in [Eli57] in which the decoder generated multiple messages rather than one message after reception and one of them is selected as the received symbol sequence. This increased the decoding capability of the decoder as the codeword search radius is increased. In [For96] and [Cha72], following the same idea, lists of codewords are created from the channel information using the most reliable and the least reliable bit positions respectively. The most likely codewords is then chosen based on certain tests and conditions. In [FBL00], a SISO algorithm called as augmented list decoding - Fang-Battail-Buda-Algorithm (ALD-FBBA) was proposed. It first sorts the symbol reliabilities received at the channel output and makes a hard decision over them. An encoding matrix is constructed based on the hard decision which is then used to construct a set of codewords. A marginalization procedure is applied on the set which results in the soft output of the algorithm. In [JHH07], a list decoder for binary LDPC codes was presented.

In this chapter, we combine the two decoding methods, iterative and list decoding and present an iterative list-based decoding algorithm for cluster codes defined over finite groups. Our proposed decoding algorithm called as local-list based decoding is based on creating lists of codewords not on the global code but rather on local parity nodes. The lists are then used to create extrinsic LLR-values for the symbols at the output of check nodes. The messages are then passed to the variable nodes to make a decision on the codeword. Thus, iterative decoding is combined with list decoding.

In the first section, we explain in detail our proposed local-list based decoding algorithm. We present its decoding performance with Monte-Carlo simulation results and then analyze its decoding complexity. In the last part of the chapter, we propose a modification in the decoder that helps in reducing the decoding complexity by reducing the size of the lists computed during the decoding procedure. H the proposed idea

also results in a performance loss.

4.1 List Decoding

A trellis is a graphical representation of a code (block or convolutional), in which any path represents a codeword. Fig 4.1 represents the trellis for a codeword set $\mathcal{C}$ composed of 6-words. There are $2^3 = 8$ states at each level of the trellis as each word is composed of 3-bits. The best known and most commonly used trellis based maximum likelihood decoding (MLD) algorithm is the Viterbi algorithm [Vit67]. It finds the globally most likely sequence of symbols in the given trellis which is the sequence closest in distance to the received noisy sequence [Lou95].

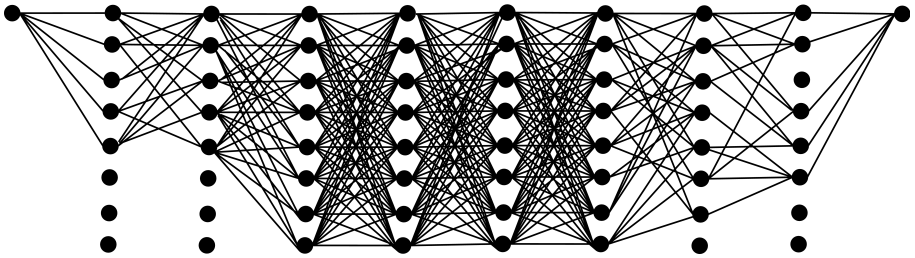

Figure 4.1: A trellis for an order-64 check node of degree $d_c = 6$

However, the main idea behind list decoding is that the decoding algorithm produces a list of possible codewords instead of creating a single possible codeword. All the codewords in the list have a distance less than a certain defined threshold e from the received noisy sequence. This allows for handling a greater number of errors. A list-based viterbi decoding algorithm was proposed in [SS94]. It produced an ordered list of the L globally best candidates after a trellis search, sorted in decreasing order of likelihoods. The list is then used to make a final decision on the received codeword.

Let $\mathcal{C}$ be a code of length d_c over an alphabet $\mathcal{A}$ of size q. The list decoding problem can be formulated as:

Input: Received word $x \in \mathcal{A}^{\mathcal{N}}$, and error bound e
Output: A list β_{LL} of all codewords $x_1, x_2, ..., x_{n_L} \in C$ whose distance from y is at most e

Fig. 4.2 shows a size n_L list of codewords β_{LL} and their likelihoods LL. Each codeword is composed of d_c words, where each word c_{ij} is an order-q symbol.

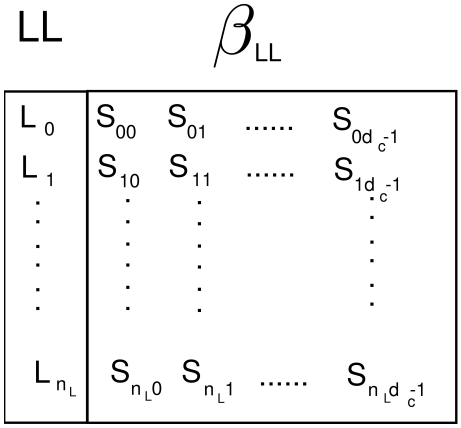

Figure 4.2: A list of codewords and their likelihoods

4.2 The local-list based decoder

For NB-LDPC codes defined over groups, the parity equation is defined as:

$$\sum_j f_{ij}(c_j) \overset{\mathbb{G}(q)}{=} 0 \tag{4.1}$$

where f_{ij} is any general function which could be linear or non-linear depending on the group over which the code is defined. When the considered group has a cardinality of 2 i.e. it is of the type $\mathbb{G}(q) = (\frac{\mathbb{Z}}{2\mathbb{Z}})^p$, the variables belong to the finite linear group $\mathbb{G}(2^p)$. Each symbol $c_j \in \mathbb{G}(2^p)$ can be represented by a p-bits binary map $X_j = \{b_j[k]\}_{k=0,\dots,p-1}$. The functions $f_{ij}()$ are represented by a $(p \times p)$ binary matrix. A special case of these codes is when the functions $f_{ij}()$ are represented by a rectangular binary matrix H_{ij} of size $(p_2 \times p_1)$. We refer to this matrix as a cluster and hence we give the name *non-binary cluster-LDPC codes* to the family. The main principle of this family of codes is that the variable nodes are now defined over a group $\mathbb{G}(2^{p_1})$, and consequently the variables have a p_1-bits binary representation.

For the i^{th} parity equation, we denote the j^{th} cluster as H_{ij}. Any degree-d_c check node is thus composed of d_c such clusters, and it forms a local parity check matrix H_{cc_i} of the local component code. Thus, it can be interpreted as a binary local component code with a parity check matrix of size $p_2 \times d_c p_1$.

$$H_{cc_i} = [H_{i1}\ H_{i2}\ ...\ H_{id_c}] \tag{4.2}$$

The i^{th} parity equation can now be expressed in the matrix form as:

$$\sum_{j=0}^{d_c-1} H_{ij} X_j^T = \mathbf{0} \tag{4.3}$$

With this representation, a binary word $[X_0 \ ... \ X_{d_c-1}]$ that satisfies eq. (4.3 forms a $d_c p_1$-bits binary codeword. Accordingly, since X_j is the p_1-bits binary mapping of the symbol $c_j \in \mathbb{G}(2^{p_1})$, the combination $[c_0 \ ... \ c_{d_c-1}]$ becomes a codeword defined the group $\mathbb{G}(2^{p_1})$. It is composed of d_c words and is local to the check node.

Now, the main idea of our local-list based decoder is to find a list of n_L such codewords at each parity check node. The list is then used to extract extrinsic LLR-values for the output at each edge of the check node. Thus the process is completed in two stages:

1). Creation of the local lists at each check nodes
2). Extraction of extrinsic LLR-values from the local lists

4.2.1 Creation of the list

Consider a degree d_c check node with input LLR-vectors $\{U_i\}_{i=0,...,d_c-1}$ sorted in decreasing order of LLR-values. The vectors $\{\beta_{U_i}^{bin}\}_{i=0,...,d_c-1}$ carry the p_1-bits binary mapping of the corresponding symbols. The inputs to the check nodes are processed a tree form as shown in Fig. 4.3. At each elementary node of the tree, two messages are joined together to form a new list of binary words sorted in decreasing order of LLRs which then acts as an input to the next level of the tree. At the output of the last stage of the tree, we have a list of codewords, each of $d_c 2^{p_1}$ bits.

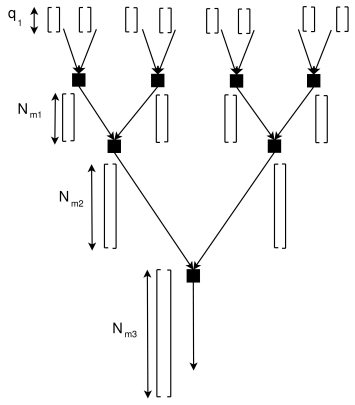

Figure 4.3: Tree for creation of list

Elementary tree-node

We now explain the procedure followed in an elementary node of the tree. Each elementary node receives two messages: $(I_A, \beta_{I_A}^{bin})$ of size N_{m_A} and (I_B, β_{I_B}) of size N_{m_B}, sorted in decreasing order of LLR-values. The vector β_{I_A} carries the p_A-bits binary mapping of the symbols corresponding to the LLR-values in I_A and similarly $\beta_{I_B}^{bin}$ carry the p_B-bits binary mapping of the symbols corresponding to the LLR-values in I_B. The output is a list (M, β_M^{bin}) with the N_{m_M} symbols carrying the highest combination of LLR-values.

$$\begin{aligned} M[i] &= I_A[j] + I_B[k] \\ \beta_M^{bin}[i] &= [\beta_{I_A}^{bin}[j] \quad \beta_{I_B}^{bin}[k]] \end{aligned} \tag{4.4}$$

with $0 \leq i \leq N_{m_M} - 1$, $0 \leq j \leq N_{m_A} - 1$ and $0 \leq k \leq N_{m_B} - 1$. The symbols in β_M^{bin} have a $p_M = p_A + p_B$ bits binary representation. The procedure is depicted in Fig. 4.4.

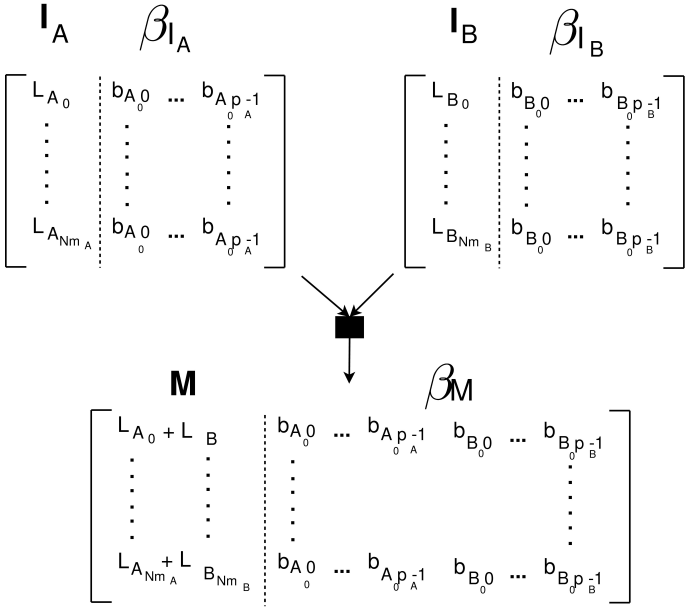

Figure 4.4: Elementary step of the tree

The output list of each node then acts as inputs to the lower nodes of the tree and this continues until the final list $\{LL, \beta_{LL}^{bin}\}$ is created. The symbols in β_{LL}^{bin} have a $d_c p_1$-bits binary representation and can be viewed as composed of d_c words, each of p_1-bits. In order to ensure that all the elements of $\{LL, \beta_{LL}^{bin}\}$ are codewords, each element of the list must satisfy the equation:

$$H_{cc_i} . \beta_{LL}^{bin}[j]^T = 0^T \tag{4.5}$$

where H_{cc_i} is the binary representation of the $(p_2 \times d_c p_1)$ PCM constructed from the i^{th}-parity. It is important to note here that the codewords are verified only at the very last node of the tree i.e. while computing $\{LL, \beta_{LL}^{bin}\}$.

It can be observed that the dimensions of the lists increases with each level. In order to have a reduced complexity process, the size of the list N_{m_i} at each stage N_{s_i} of the tree must be limited i.e. $N_{m_i} << (N_{m_A} * N_{m_B})$, with N_{m_A} and N_{m_B} being the sizes of the inputs lists. As obvious, this may result in a loss of information, however a compensation offset can be used to improve the decoding.

4.2.2 Extraction of LLRs from a local list

Once the list (LL, β_{LL}^{bin}) is created, the next step is to extract extrinsic LLR-values for each output edge of the check node. The list β_{LL}^{bin} is limited to size N_{m_L} and carries codewords composed of d_c words, each of p_1 bits. The list is sorted in decreasing order of LLR-values carried by LL. For the extraction of extrinsic LLRs, we also require the check nodes input messages $\{U_j\}_{j=0,..,d_c-1}$ and the binary mapping of their corresponding symbols $\{\beta_{U_j}^{bin}\}_{j=0,...,d_c-1}$. The extrinsic output LLR-values of the check node are copied to the vectors $\{V_i\}_{i=0,...,d_c-1}$, each of size 2^{p_1}.

As an initialization procedure, a boolean table $Flag$ of size $(d_c \times 2^{p_1})$ is initialized to all-zeros. When the output value $V[i][j]$ is updated, $Flag[i][j]$ is set to 1. The pseudo-code to create LLRs from the list is as follows:

Algorithm 1: List Processing:

```
for i = 0 to N_{m_L} − 1 do
  for j = 0 to d_c − 1 do
    symb ← j^{th} symbol of the i^{th} codeword
    newLLR ← LLRvalue of symb in U_j
    if Flag[j][symb] == 0 then
      V_j[symb] ← LL[i] - newLLR
      Flag[j][symb] ← 0
    end if
  end for
end for
```

The List processing steps are used to extract the output extrinsic LLR-values for the symbols corresponding to the output $\{V_j\}_{j=0,..,d_c-1}$. However, due to the fact that the lists are truncated to a limited size, it may occur that the LLR-value corresponding to a certain symbol is not present in the list of codewords. Thus, a post-processing step is

necessary to find out these missing values and insert a certain pre-defined value at their places. Since in our case the LLRs are strictly positive, so fix the predefined-value to 0. Moreover, an offset is also added in the post-processing step which compensates for the loss of information due to the reduced size lists. The value of offset can be empirically determined using Monte-Carlo simulations. The post-processing steps are depicted in the Algorithm 2.

Algorithm 2: Post Processing:

for $i = 0$ to $d_c - 1$ **do**
  $minval \leftarrow \min\{V_i[0], ..., V_i[2^{p_1} - 1]\}$
  for $j = 0$ to $2^{p_1} - 1$ **do**
    if $Flag[j][symb] == 0$ **then**
      $V_j[symb] \leftarrow offset$
    else
      $V_j[symb] \leftarrow minval + offset$
    end if
  end for
end for

The check node output vectors $\{V_j\}_{j=0,...,d_c-1}$ are then sent directly to the variable nodes.

4.2.3 The decoding algorithm

We now summarize the decoding procedure of the local-list based decoder for a $(M \times N)$ NB-cluster LDPC code, with cluster size $(p_2 \times p_1)$.

Initialization

The decoder is initialized with the channel information in the form of LLR-vectors $\{L_i\}_{i=0,..,N-1}$ of order-2^{p_1}. The binary mapping of the corresponding symbols are carried by the vectors $\{\beta_{L_i}^{bin}\}_{i=0,...,N-1}$.

$$\begin{aligned} L &= [L[0] \;\; L[1] \;\; ... \;\; L[2^{p_1} - 1]] \\ \beta_L^{bin} &= [\beta_{L[0]}^{bin} \;\; \beta_{L[1]}^{bin} \;\; ... \;\; \beta_{L[2^{p_1}-1]}^{bin}] \end{aligned} \tag{4.6}$$

where $L[i]$ represents the LLR-value of the i^{th}-symbol. Here, we would like to re-call that we have strictly positive LLR-values. For a symbol α^i, it is calculated as:

$$L(\alpha^i) = log\left(\frac{P(v = \alpha^i}{P(v = \alpha^j_{min})}\right) \tag{4.7}$$

where α^j_{min} is the least probable symbol of the variable v.

Variable nodes update

The variable nodes output is computed as an extrinsic element by element sum of the inputs messages and the channel likelihood vectors.

$$U_{vk}[i] = L_{ch_v}[i] + \sum_{j=0, j\neq k}^{d_v-1} V_{jv}[i] \tag{4.8}$$

Check nodes update

The output message U_{vk} of the variable node v is directly fed to the check node k. As already discussed that the check nodes update process is executed in two stages. The first step of the check update procedure is the creation of the list of codewords. At each check node k, a local list $\{LL_k, \beta^{bin}_{LL_k}\}$ is created as explained in section 4.2.1. This list is then used to extract the extrinsic output LLR-values as explained in section 4.2.2.

APP calculation

The extrinsic LLR-vectors at the output of the check nodes are sent then directly to the variable nodes where the a-posteriori probabilities (APPs) of the symbols are calculated and a decision is made on the global codeword $\hat{v}$. For a valid codeword to be obtained, it must satisfy the equation:

$$H.\hat{v}^T = 0 \tag{4.9}$$

where H is the parity check matrix of the code. If $\hat{v}$ does not satisfy the equation, the process is iteratively repeated until either successful decoding or a fixed number of iterations have been executed. If a valid codeword is not obtained after the iterations, a decoding failure is declared.

4.3 Decoding performance and complexity

In this section, we analyze the complexity and decoding performance of the proposed decoder. We study the complexity in terms of number of operations performed at the check nodes. For the decoding performance, we present Monte-Carlo simulation results of the proposed decoding method. We compare the decoding performance with the cluster-EMS decoder proposed in chapter 3.

Algo.	**Real Additions**	**GF(q)-Addition**	**Max**
Cluster-EMS	$3(d_c-2)(n_{op}+2^{p_1})$	$3(d_c-2)(n_{op})$	$3(d_c-2)n_{op}(\sqrt{2n_{op}})$
list-decoder	$\sum_{i=0}^{N_s-1} N_{n_i}N_{m_i} + d_c 2^{p_1}$	-	$\sum_{i=0}^{N_s-1} N_{n_i}N_{m_i}\sqrt{2N_{m_i}} + d_c n_L$

Table 4.1: Number of operations required to process the check nodes

Table 4.1 presents the number of operations performed while processing a single check node of degree d_c for the proposed algorithms. For the cluster-EMS algorithm, n_m is the message truncation value and n_{op} is the total number of necessary steps so that all the n_m values of the output vector are computed. This value is generally fixed to $2n_m$ [CCAGB00]. For the local-list based decoder N_s is the total number of stages in the tree and N_{n_i} is the number of nodes at each stage-i. N_{m_i} is the size of the list considered at each stage-i.

List-Name	$N_{m_1}, N_{m_2}, N_{m_L}$	**No. of max()**	**No. of additions**
List-I	$24, 64, 50$	1904	274

Table 4.2: List size and no. of operations for the code with $d_c = 8$ and $p_1 = 3$

Table 4.2 presents the size of the lists used at the various nodes of the tree. In order to compute the final list of 50 codewords, the decoder requires 1904 $max()$ operations and 274 additions. For a cluster-EMS decoder with $n_m = 12$, we see require 3024 $max()$ operations and 648 additions. When we compare to performances for these parameters, as shown in Fig. 4.5, we observe that we have the same decoding performance. Thus, a very less number of operations, we are able to attain the same decoding performance as the cluster-EMS decoder. This makes the local-list based decoder an interesting candidate for error correcting applications.

Table 4.3 presents the number of operations for the the list used for decoding the

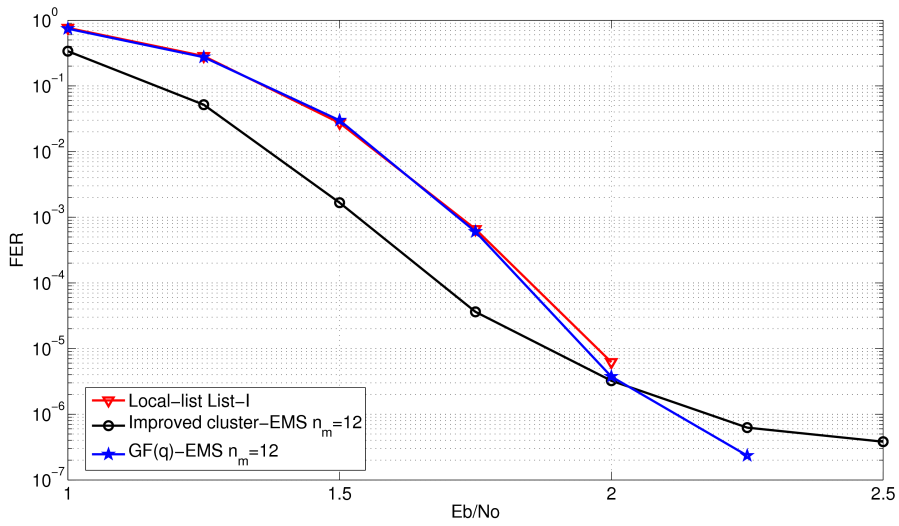

Figure 4.5: local-list decoder vs. cluster-EMS decoder $N = 3000$, $R = 0.5$, $(p_1, p_2) = (3, 6)$

same code as above but with $p_1 = 4$. Thus, the check node inputs are LLR-vectors of size 16. The cluster-EMS decoder with $n_m = 12$ requires 2016 $max()$ operations and 432 additions. When we compare the performances shown in Fig. 4.6, we observe that, contrary to the previous case, we have a loss in performance of about 0.2dB. This is due to the fact that we need to calculate 16 LLR-values for each output edge of the check node, whereas for the previous case, we required to compute only 8 LLR-values. Thus, we require lists with even larger dimensions.

List-Name	$N_{m_1}, N_{m_2}, N_{m_L}$	**No. of max()**	**No. of additions**
List-I	$38, 82, 80$	2208	280

Table 4.3: List size and no. of operations for the code with $d_c = 6$ and $p_1 = 4$

Thus, we can conclude that the local-list decoder is an interesting prospect for code with higher degree of check nodes and lower order of variable nodes. For codes with higher order of variable nodes, we require lists of large size, which may become impractical.

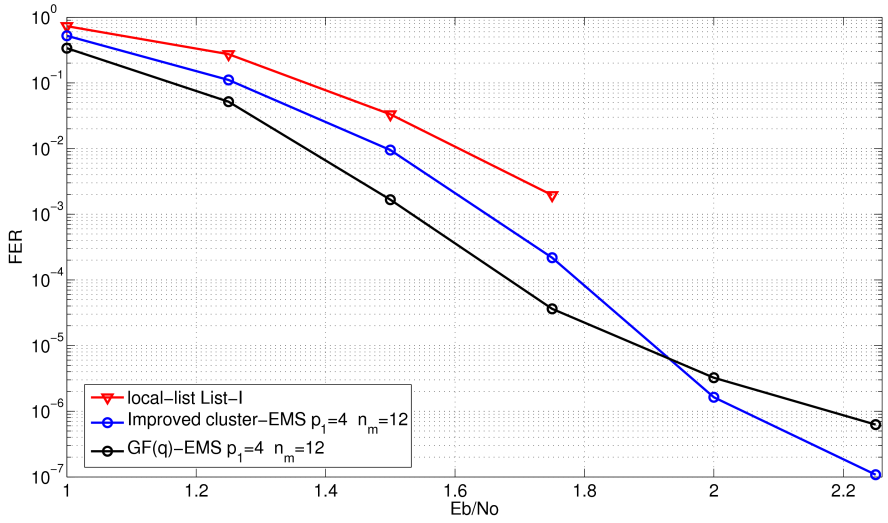

Figure 4.6: local-list decoder vs. cluster-EMS decoder $N = 3000$, $R = 0.5$, $(p_1, p_2) = (4, 6)$

4.4 Reduced list-size decoder

The problem with the local-list based decoder, proposed in the previous section, is that the sizes of the lists at the various stages of the tree are quite large. For codes with low order of variable nodes, we gain in terms of the number of operations performed as compared to the cluster-EMS algorithm. However, the problem in the implementation of the decoder remains in the size of the sorters which are required to compute the highest N_{m_i} combinations LLR-values at any stage-i of the tree. In order to compute a list of size N_{m_i}, we require a sorter of size $\sqrt{2N_{m_i}}$. With large values of N_{m_i}, the decoding algorithm remains counter productive as the implementation complexity remains cumbersome. In order to solve this problem, we tried to propose a solution. The idea seemed to be interesting, but we did not obtain an encouraging results. However, for the sake of information, we explain our proposed idea.

The list-decoding is based on the creating a list $\{LL, \beta_{LL}\}$ of codewords which is local to a parity check node. The list of codewords can be represented in the form of a trellis as shown in Fig. 4.1, where each path represent a codeword. Each path carries a certain weight which is represented as the LLR-value of the codeword. The local-list decoder is based on selecting N_L path with the highest corresponding weights. We observed that the path connections were saturated at certain nodes at the various stages of the trellis i.e. at a certain stage of the trellis, there were a lot many connections to a few symbols and other symbols were left unconnected as depicted in Fig. 4.7.

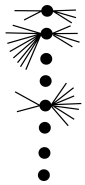

Figure 4.7: Path saturation at certain nodes of the trellis

The saturation effects the list (LL, β_{LL}^{bin}) of codewords at the output of the tree. It is composed of codewords with the connected symbols only. The symbols which are left un-connected are not present in the list, and thus the output LLR-value for those symbols cannot be computed when we extract LLRs from the list at each check node. This results in a loss of information and in order to compensate for the loss, an predefined value plus an offset is used instead. However, this is not sufficient and large size lists are required in order to attain good performance.

Thus, we tried to propose a solution by scattering the connections through out the nodes of the trellis. At any stage-i of the trellis, compute a certain number N_{max} of connections carrying the highest weights, following the same procedure as proposed in section 4.2. The remaining $N_{m_i} - N_{max}$ values are spread out throughout the various other nodes(symbols) of the inputs. This procedure can be explained in another way: from all the incoming n_L connections to a stage-i, remove the $N_{m_i} - N_{max}$ connections and assign them to symbols which are not connected to any node of stage-$(i-1)$.

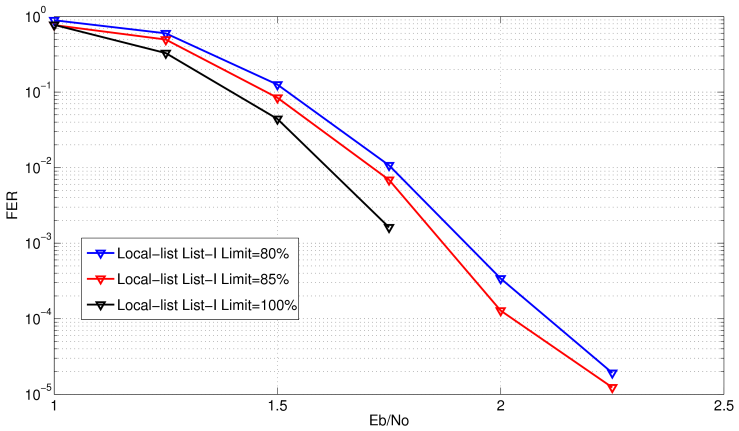

Figure 4.8: Local-list decoder $N = 3000$, $R = 0.5$, $(p_1, p_2) = (3, 6)$

Fig. 4.8 shows the Monte-Carlo simulation results for a LDPC code of rate $R = 1/2$ and length $N = 3000$ bits. $Limit$ defines the percentage of the symbols selected on the basis of highest combination LLR-values. For example, at any stage-i of the tree, we compute a list of size N_{m_i}. We compute $(Limit)\%$ of the symbols that carry the highest combination LLR-values. The remaining symbols are selected from outside the area that fall into the N_{m_i} highest combinations. From the remaining symbols, each symbol is selected only once. However, we can observe from Fig. 4.8 that we have a loss in performance of about 0.12dB and 0.15dB for $Limit = 85\%$ and $Limit = 80\%$ respectively. Thus, this method is not of interest and we need to develop another method to reduce the sizes of the lists at the various nodes of the tree.

4.5 Conclusion

In this chapter, we proposed a new decoding algorithm for non-binary cluster-LDPC codes defined over finite groups which joins iterative decoding with list-based decoding. At each check node, a list of codewords is constructed which is local to the parity node. The extrinsic output LLR-values are then extracted from the list of codewords, which are then sent towards the variable nodes. This process is iteratively repeated, until we attain successful decoding or a fixed number of iterations have been run. We proved with simulations results that for a higher degree of check node and a lower order of the variable nodes, the decoder attains better performance than the cluster-EMS decoder, proposed in chapter 3, with a reduced number of operations. However, we still need a mechanism to reduce the lists for codes with a lower degrees of check nodes or a higher order of the variable nodes.

The added advantage of the tree structure of the list-based decoder is that it can be implemented with a parallel architecture. Thus, it is suitable for high rate applications. However, the complexity of the parallel implementation of the sorter is increased exponentially with the size of the sorter, which remains a problem. Thus, implementation with large size lists is a difficult task and there is a need of a mechanism which aids in reducing the list sizes at the various stages of the tree.

We tried to propose a mechanism to reduce the list size at the various stages of the tree. However, our proposed idea was not supported by Monte-Carlo simulation results as we did not attain any interesting results. Further work can be done on decreasing the sizes of the lists and if we are able to do so, we can have an efficient and low complexity non-binary LDPC decoder that fulfills the requirements of new generation communication systems.

5

Conclusions and perspectives

5.1 Conclusions

We started the work with a detailed study of the state-of-the-art. We studies the various decoders proposed for non-binary LDPC codes defined over finite Galois fields. We then directed our research to a new family of LDPC codes defined over finite groups. We refer to this family of codes as cluster-LDPC codes due to the fact that the elements of the parity check matrix can be represented in the form of a non-binary cluster of size $(p_2 \times p_1)$.

The EMS algorithm is, so far, one of best proposed decoder for non-binary LDPC codes. It has a very efficient performance with a low complexity. Thus, as a first task, we generalized the EMS algorithm to cluster-LDPC codes. However, we saw that the direct generalization is not a viable solution and needs some improvements. So we notified the problems associated with the direct generalization and proposed two modifications in the decoding procedure. With simulations results, we proved that we have a significant gain in performance with the proposed modifications.

We then proposed another method that could prove useful in decreasing the surface area of the decoder. Cluster-LDPC codes introduce diversity into the decoder and we proposed to make use of this diversity. For each check, we proposed to implement in parallel, several instances of the check node update process. The output of the various instances are then fusion-ed together to form the extrinsic output of the check node. We proved with simulations results that we have better decoding performance when we used multiple instances of a check node. Moreover, we have a lower error floor.

We also proposed a new decoding algorithm for cluster-LDPC codes. It is based on the creation lists of codewords that are local to the parity nodes. The lists are then used to create the extrinsic output LLR-values of the check nodes. The decoder showed good performance for variable nodes defined in a lower group order. However, for high order variable nodes, the size of the lists gets high. Thus, we require a mechanism with which we can decrease the required list sizes. We also tried to propose a method to decrease the lists sizes, however, we did not attain encouraging results.

5.2 Perspectives

The following points constitute interesting perspectives to the work presented in this thesis.

- For the cluster-EMS algorithm, it would be interesting to see if the non-authorized states can be used in a better fashion, so that can reduce the over-all complexity of the decoder. This would also allow us to increase the value of n_m, and thus better performance.
- Regarding the list-based decoder, the size of the lists still remain a problem. Therefore, if we are able to find an interesting a mechanism to reduce the list size, we can obtain a very low complexity decoder.

Bibliography

[BB06] A. Bennatan and D. Burshtein. Design and analysis of non-binary ldpc codes for arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 52:549–583, Feb 2006.

[BCC10] E. Boutillon and L. Conde-Canencia. Bubble check: a simplified algorithm for elementary check node processing in extended min-sum non-binary ldpc decoders. *IEEE Electronic letters*, 46:633–634, April 2010.

[BD03] L. Barnault and D. Declercq. Fast decoding algorithm for ldpc codes over $gf(2^q)$. In *Proceedings of IEEE Information theory workshop*, pages 70–73, Paris, France, March 2003.

[BF04] G.J. Byers and F.Takawira. Non-binary and concatenated ldpc codes for multiple-antenna systems. In *Proceedings of AFRICON*, pages 83–88, Gaborone, Botswana, Sept 2004.

[BGT93] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting codes and decoding. *In the proc. of International Conference on Communication (ICC)*, pages 1064–1070, May 1993.

[BT05] G. Byers and F. Takawira. Exit charts for non-binary ldpc codes. In *Proceedings of ICC*, Seoul, Korea, May 2005.

[CCAGB00] L. Conde-Canencia, A. Al-Ghouwayel, and E. Boutillon. Complexity comparison of non-binary ldpc decoders. In *Proceedings of ICT-MobileSummit 09*, Santander, Spain, June 2000.

[CF02] J. Chen and M. Fossorier. Density evolution for two improved bp-based decoding algorithms of ldpc codes. *IEEE Communication letters*, 6:208–210, May 2002.

[CFRU01] S.Y. Chung, G.D. Forney, T.J. Richardson, and R.L. Urbanke. On the design of low density parity check codes within 0.0045db of the shannon limit. *IEEE Communication letter*, 5:58–60, February 2001.

[Cha72] D. Chase. A class of algorithms for decoding block codes with channel measurement information. *IEEE transcation of Information Theory*, 18:170–182, January 1972.

[CPD+09] W. Chen, C. Poulliat, D. Declercq, L. Conde-Canencia, A. Al-Ghouwayel, and E. Boutillon. Non-binary ldpc codes defined over the general linear group: Finite length design and practical implementation issues. In *Proceeding of VTC'09 (special session FP7-ICT-RAS-cluster),*, Barcelona, Spain, April 2009.

[CWL05] Junbin Chen, Lin Wang, and Yong Li. Performance comparison between non-binary ldpc codes and reed-solomon codes over noise burst channels. In *Proceedings of IEEE ICCS'05*, pages 1–4, China, May 2005.

[DCG04] D. Declercq, M. Colas, and G. Gelle. Regular $gf(2^q)$ ldpc coded modulations for higher order qam-awgn channels. In *Proceedings of IEEE ISITA*, Parma, Italy, Oct 2004.

[Dec08] D. Declercq. Non-binary group decoder diversity for decoding dense block codes. In *Proceeding of IEEE Turbo-coding symposium*, Lausanne, Switzerland, Sept. 2008.

[DF07] D. Declercq and M. Fossorier. Decoding algorithms for non-binary ldpc codes over $gf(q)$. *IEEE Transactions on communication*, 55:633–643, April 2007.

[DM98] M. Davey and D.J.C. Mackay. Low density parity check codes over gf(q). *IEEE communication letter*, 2:165–167, June 1998.

[Eli57] P. Elias. List decoding for noisy channels. *MIT, Cambridge, MA, Tech*, 1957.

[FBL00] J. Fang, F. Buda, and E. Lemois. Turbo product code: A well suitable solution to wireless packet transmission for very low error rates. *in 2nd international symposium on Turbo codes and related topics*, pages 101–111, 2000.

[For96] G.D. Forney. Generalised minimum distance decoding. *IEEE transcation of Information Theory*, 12:125–131, April 1996.

[Gal62] R. G. Gallager. Low density parity check codes. *IEEE transaction on Information theory*, IT-8:21–28, Jan 1962.

[HDYW06] Xiaofei Huang, Suquan Ding, Zhixing Yang, and Youshou Wu. Fast min-sum algorithms for decoding of ldpc codes over $gf(q)$. In *Proceedings of IEEE ITW'06*, pages 96–99, Beijing, China, Oct. 2006.

[HEA05] X-Y. Hu, E. Eleftheriou, and D.M. Arnold. Regular and irregular progressive edge growth tanner graphs. *IEEE trans. on Inf. Theory*, 51:386–398, Jan. 2005.

[HEAD01a] X. Hu, E. Eleftheriou, D.M. Arnold, and A. Dholakia. Efficient implementation of the sum-product algorithm for decoding ldpc codes. In *Proceedings of IEEE Globecom*, 2001.

[HEAD01b] X. Hu, E. Eleftheriou, D.M. Arnold, and A. Dholakia. Efficient implementation of the sum-product algorithm for decoding ldpc codes. In *Proceedings of IEEE Globecom*, 2001.

[HFE04] X.Y. Hu, M. Fossorier, and E. Eleftheriou. Binary representation of cycle tanner graph $gf(2^q)$ codes. In *Proceedings of IEEE ICC*, pages 528–532, Paris,France, June 2004.

[HH02] S. Hirst and B. Honary. Decoding of generalized low density parity check codes using weighted bit-flip voting. *IEEE proceedings on communications*, 149:1–5, Feb 2002.

[HLY02] L. Hanzo, T. Liew, and B.L. Yeap. Turbo coding, turbo equalization and space-time coding. John Wiley and sons ltd., 2002.

[HooMM99] J. Hagenauer, E. offer offer, C. Méasson, and M. Mörz. Decoding and equalization with analog non-linear networks. *European transcations communication*, 10, Nov. 1999.

[JHH07] J. Justesen, T. Hoholdt, and J. Hjaltason. Iterative list decoding of some ldpc codes. *IEEE transcation of Information Theory*, 53:4276–4284, Nov. 2007.

[JYgXL09] Xueqin Jiand, Yier Yan, Xiang gen Xia, and Moon Ho Lee. Application of non-binary ldpc codes based on euclidean geometries to mimo systems. In *Proceedings of Intern. conference on wireless comm. and Signal processing*, pages 1–5, Nanjing, China, Nov 2009.

[KFL01] F. Kschischang, B. Frey, and H.A. Loeliger. Factor graphs and the sum product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, Feb 2001.

[KH06] Fang-Chun Kuo and L. Hanzo. Symbol flipping based decoding of generalized low density parity check codes over gf(q). In *Proceedings of IEEE WCNC'06*, pages 1207–1211, Las-Vegas, NV, April 2006.

[LFK03] G. Li, I. Fair, and W. Krzymien. Analysis of nonbinary ldpc codes using gaussian approximation. In *Proceedings of ISIT*, Kanagawa, Japan, July 2003.

[LMSS01] 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*, 47:619–637, Feb 2001.

[Lou95] Hui-Ling Lou. Implementing the viterbi algorithm. *IEEE signal processing magazine*, 12:42–52, Sep. 1995.

[LZL+07] Lan Lan, Lingqi Zeng, Y.Y. Lei, Shu Lin, and K. Abdel-Ghaffar. Construction of quasi-cyclic ldpc codes for awgn and binary erasure channels: A finite field approach. *IEEE Transactions on Information Theory*, 53:2429–2458, July 2007.

[Mac97] D.J.C. Mackay. Good error correcting codes based on very sparse matrices. *IEEE International symposium on Information theory*, 2:113, July 1997.

[MD99] D.J.C. Mackay and M. Davey. Evaluation of gallager codes for short block length and high rate applications. *proc. of IMA workshop on codes, systems and graphical models*, pages 113–130, 1999.

[MF05] N. Miladinovic and M.P.C. Fossorier. Improved bit-flipping decoding of low density parity check codes. *IEEE Transactions on Inf. Theory*, 51:1594–1606, April 2005.

[MSV08] A. Morinoni, P. Savazzi, and S. Valle. Efficient design of non-binary ldpc codes for magnetic recording channels, robust to error bursts. In *Proceedings of IEEE Int. Symp. on Turbo codes and related topics*, pages 288–293, Lausanne, Switzerland, Sept 2008.

[MWZ06] Long Ma, Lin Wang, and Jianwen Zhang. Performance advantage of non-binary ldpc codes at high code rate under awgn channel. In *Proceedings of Intern. conf. on Communication technologies*, Guilin, China, Nov 2006.

[PC06] Rong-Hui Peng and Rong-Rong Chen. Design of non-binary ldpc codes over $gf(q)$ for multiple-antenna transmission. In *Proceedings of MILCOM*, Washington DC, USA, Oct 2006.

[PDL07] C. Poulliat, D. Declercq, and T. Lestable. Decoding turbo codes with a non-binary belief propagation decoder. In *in the proc. of the 19th WWRF meeting wg4*, Nov 2007.

[PDL08] C. Poulliat, D. Declercq, and T. Lestable. Efficient decoding of turbo codes with non-binary belief propagation. *EURASIP JWCN, special issue on 'advances in Error control coding techniques'*, 2008.

[PFD06a] C. Poulliat, M. Fossorier, and D. Declercq. Design of non binary ldpc codes using their binary image: algebraic properties. In *Proceedings of IEEE ISIT*, Seattle, USA, July 2006.

[PFD06b] C. Poulliat, M. Fossorier, and D. Declercq. Opitmization of non-binary ldpc codes using their binary images. In *Proceedings of IEEE Int. Symp. on Turbo codes*, Munich, Germnay, April 2006.

[PL00] L. Ping and W.K. Leung. Decoding low density parity check codes with finite quantization bits. *IEEE communication letters*, 4:62–64, Feb 2000.

[PML+09] S. Pfletschinger, A. Mourad, E. Lopez, D. Declercq, and G. Bacci. Performance evaluation of non-binary ldpc coes on wireless channels. In *Proceedings of ICT Mobile summit*, Santandar, Spain, June 2009.

[PNF04] H. Pishro-Nik and F. Fekri. On decoding of low density parity check codes over binary erasure channel. *IEEE Transactions on Information Theory*, 50:439–454, March 2004.

[RSU01] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke. Design of capacity approaching irregular low density parity check codes. *IEEE Transactions on Information Theory*, 47:619–637, Feb 2001.

[RU01] T.J. Richardson and R.L. Urbanke. The capacity of low density parity check codes under message passing decoding. *IEEE Transactions on Information Theory*, 47:599–618, February 2001.

[RU05] V. Rathi and R.L. Urbanke. Density evolution thresholds and the stability conditions for non-binary ldpc codes. *IEEE Transactions on communication*, 152(6):1069–1074, Dec 2005.

[Rya03] W.E. Ryan. An introduction to ldpc codes. http://www.ece.arizona.edu/%7Eryan/New%20Folder/ryan-crc-ldpc-chap.pdf, Aug 2003.

[SC03] H. Song and J.R. Cruz. Reduced complexity decoding of q-ary ldpc codes for magnetic recording. *IEEE trans on magnetics*, 39, March 2003.

[SD06] L. Sassatelli and D. Declercq. Non-binary hybrid ldpc codes - structure, decoding and optimization. In *Proceedings of IEEE ITW'06*, Chengdu, China, Oct. 2006.

[SD07] L. Sassatelli and D. Declercq. Analysis of non-binary hybrid ldpc codes. In *Proceedings of IEEE ISIT'07*, Nice, France, June 2007.

[SF02] D. Sridhara and T.E. Fuja. Low density parity check codes defined over groups and rings. *In the proc. of Information Theory Workshop (ITW)*, Oct 2002.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.

[SMG09] G. Sarkis, S. Mannor, and W.J. Gross. Stochastic decoding of ldpc codes over gf(q). In *Proceedings of IEEE ICC'09*, pages 1–5, Dresden, Germany, June 2009.

[SS94] Nambirajan Seshadri and Carl-Erik W. Sundberg. List viterbi decoding algorithms with applications. *IEEE transcation on communications*, 42:313–323, Feb. 1994.

[SZAG06] S. Song, L. Zeng, and K. Abdel-Ghaffar. Algebric construction of non-binary quasi-cyclic ldpc codes. In *Proceedings of ISIT*, Seattle, USA, July 2006.

[Tan81] R.M. Tanner. A recursive approach to low complexity codes. *IEEE transaction on Information theory*, pages 533–547, Sept 1981.

[TGM06] S. Sharifi Tehrani, W. Gross, and S. Mannor. Stochastic decoding of ldpc codes. *IEEE communication letters*, 10:716–718, 2006.

[TMG07] S. Sharifi Tehrani, S. Mannor, and W.J. Gross. An area efficient fpga based architecture for fully parallel stochastic ldpc decoding. In *Proceedings of IEEE workshop on signal processing*, pages 255–260, Shanghai, China, Oct. 2007.

[TMG08] S. Sharifi Tehrani, S. Mannor, and W.J. Gross. Fully parallel stochastic ldpc decoders. *IEEE trans. on Signal processing*, 56:5692–5703, Nov 2008.

[VDV+08] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard. "Split non-binary ldpc codes". July 2008.

[VDV+10] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard. Low complexity decoding algorithm for non-binary ldpc codes in high order fields. *IEEE Transactions on communication*, 58(5):1–11, May 2010.

[Vit67] A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transcation of Information Theory*, 13:260–269, April. 1967.

[Voi07] A. Voicila. *Décodage simplifié des codes LDPC non-binaire*. thèse, Université de Cergy-Pontoise, Sept 2007.

[VP08] A. Venkiah and C. Poulliat. Design of cages with a randomized progressive edge growth algorithm. *IEEE Communication letters*, 12:301–303, April 2008.

[VVD+07] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, and P. Urard. "Architecture of a low-complexity non-binary ldpc decoder for high order fields". October 2007.

[WS98] A.P. Worthen and W.E. Stark. Low density parity check codes for fading channels with memory. In *Proceedings of* 36^{th} *allerton conf. on communication, control and computing*, pages 117–125, Sept 1998.

[WSM04a] H. Wymeersch, H. Steendam, and M. Moeneclaey. Computational complexity and quantization effects of decodign algorithms of ldpc codes over gf(q). Montreal, Canada, May 2004.

[WSM04b] H. Wymeersch, H. Steendam, and M. Moeneclaey. Log domain decoding of ldpc codes over $gf(q)$. In *Proceedings of IEEE Intern. Conf. on Communications*, pages 772–776, Paris, France, June 2004.

[YHB04] M.R. Yazdani, S. Hemati, and A.H. Banihashemi. Improving belief propagation on graphs with cycles. *IEEE Communication letters*, 8:57–59, January 2004.

[YR01] M. Yang and W.E. Ryan. Performance of (quasi-)cyclic ldpc codes in noise bursts on the epr4 channel. In *Proceedings of IEEE GLOBECOM*, pages 2961–2965, Texas, USA, 2001.

[ZLT+08] L. Zeng, L. Lan, Y. Yu Tai, B. Zhou, Shu Lin, A. Khaled, and A.S. Abdel Ghaffar. Construction of nonbinary cyclic, quasi-cyclic and regular ldpc codes: A finite geometry approach. *IEEE transactions on Communications*, 56, March 2008.