MONASH University

Low Complexity Decoding of Non-binary Low-density Parity-check Codes

V. B. Wijekoon

A thesis submitted for the degree of

Doctor of Philosophy

at the Department of Electrical and Computer Systems Engineering

Monash University

December 2020

Copyright Notice

© V. B. Wijekoon (2020)

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

To the two pillars that form the foundation,

To the resolute beam that lends support,

And,

To the gem that lights the world

Abstract

Recent technological advancements across multiple frontiers have resulted in an exponential increase of data produced annually, and this has created an unprecedented demand for efficient, reliable, and affordable data transmission and storage systems. Error control coding plays a vital role in any such system, and thus, more and more focus is now being placed on related research.

Binary low-density parity-check (LDPC) codes are well-known, in both academia and industry, for exceptional error-correcting performances that can be obtained with relatively low-complexity, iterative decoding algorithms. While they are the codes of choice in many practical standards and applications, their non-binary counterparts can offer even better performances, across many different types of channels. But the wide-spread usage of non-binary LDPC (NB-LDPC) codes has so far been restricted due to the high complexity of decoding algorithms.

In this dissertation, we propose a novel method to represent an NB-LDPC code over any subfield of the finite field used in construction. This allows a code to be depicted in a number of different ways, and we present a generic approach to adapt any decoding algorithm of NB-LDPC codes for using on any such representation. Due to the smaller size of subfields, these schemes offer significant gains in decoding complexity compared to decoding over the original field, while performance losses are minimal. We focus particularly on the binary representation, which offers the largest reduction of complexity, and also devise a binary majority-logic decoding algorithm based on that for use with resource-constrained systems.

Furthermore, we propose using iterative, non-binary message-passing algorithms for decoding Reed-Solomon codes, which is a class of algebraic codes. We also present the alternative parity-check matrices to facilitate this. These matrices are constructed in such a way that the number of sub-structures undesirable for iterative decoding is minimized, and the proposed decoding scheme on them offers significant gains compared to algebraic methods, while keeping decoding complexity at a reasonable level.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

V. B. Wijekoon
December, 2020

Publications during Enrolment

- **V. B. Wijekoon**, Emanuele Viterbo, and Yi Hong, "Decoding of NB-LDPC codes over Subfields", *IEEE Transactions on Communications*, Early Access Article, Nov. 2020
- **V. B. Wijekoon**, Emanuele Viterbo, and Yi Hong, "A Low Complexity Decoding Algorithm for NB-LDPC Codes over Quadratic Extension Fields", *Proceedings of IEEE International Symposium on Information Theory*, Virtual Conference, June 2020
- **V. B. Wijekoon**, Emanuele Viterbo, and Yi Hong, "LDPC-Staircase Codes for Soft Decision Decoding", *Proceedings of IEEE Wireless Communications and Networking Conference*, Virtual Conference, May 2020
- **V. B. Wijekoon**, Emanuele Viterbo, Yi Hong, Rino Micheloni, and Alessia Marelli, "A novel graph expansion and a decoding algorithm for NB-LDPC codes", *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1358-1369, Mar. 2020
- **V. B. Wijekoon**, Shuiyin Liu, Emanuele Viterbo, Yi Hong, Rino Micheloni, and Alessia Marelli, "Coset probability based majority-logic decoding for non-binary LDPC codes", *Proceedings of IEEE Information Theory Workshop*, Visby, Sweden, Aug. 2019
- Emanuele Viterbo, and **V. B. Wijekoon**, "Coset Probability Based Majority-logic Decoding for Non-binary LDPC Codes", US Patent, No. 10790854
- **V. B. Wijekoon**, Hoang Dau, and Emanuele Viterbo, "Iterative decoding of Reed-Solomon codes based on non-binary matrices," *Proceedings of IEEE International Symposium on Information Theory*, Paris, France, July 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1948, Shannon's seminal paper, *'A mathematical theory of communication'* [1], established the foundations of the fields of information theory and channel coding. There, Shannon defines the *capacity* of a channel as the maximum rate at which information can be transmitted virtually free of errors over that channel. One of the more intriguing theorems in the paper then proves the existence of coding schemes that enable nearly error-free transmission over any given channel at any rate less than the capacity. However, the proof of this theorem is based on randomly constructed codes with block lengths tending to infinity, which are not practically feasible due to high encoding and decoding complexities. Thus, since the publication of Shannon's paper, much effort has been devoted to construction of coding schemes that achieve channel capacity at relatively low encoding and decoding complexities.

The discovery of turbo codes in 1993 [2] marked a breakthrough in the search for the elusive capacity achieving codes. Although they do not actually achieve capacity, turbo codes *approach* capacity with increasing block length. Around the same time, low-density parity-check (LDPC) codes, introduced by Gallager in 1962 [3], were re-discovered [4]. Binary LDPC codes were also observed to approach capacity, with performance often surpassing that of turbo codes. Both these classes of codes employ simple, iterative decoding algorithms, and thus, in a practical sense, they marked the end of the search for capacity approaching codes.

Currently, binary LDPC codes have become the error-correcting codes of choice for many practical applications, such as Ethernet, Wi-Fi, and digital television, due to their exceptional performance with low-complexity decoding algorithms. Davey and Mackay introduced the non-binary (NB) counterparts of these codes in 1998 [5], and it was soon realized that NB-LDPC codes outperform the binary LDPC codes of comparable length, especially for short-to-moderate codeword

lengths. But these performance gains are yet to be realized in practice due to the high complexity of decoding algorithms. In the research community, a lot of focus has since been devoted to reducing the decoding complexity of NB-LDPC codes so that practical applications may also benefit from the significant performance gains.

The primary focus of this dissertation is also on reducing the decoding complexity of NB-LDPC codes. In this regard, we employ novel graphical representations of the codes to design low-complexity decoding schemes. Then we use the insights on performance of NB-LDPC codes to invent new strategies to decode another well-known class of non-binary codes, Reed-Solomon codes [6]. In the following section, we provide brief overviews of each chapter of the dissertation, which also summarise our contributions.

1.1 Overview of Chapters and Contributions

Chapter 2 : Preliminaries

In this chapter, we present the preliminary definitions and concepts necessary for the remaining chapters. We begin with some definitions and theorems from the study of finite groups and fields which are instrumental in construction and decoding of NB-LDPC codes, and are made use of in later chapters. Next we present an overview of the structure of NB-LDPC codes and their constructions. Then we review the major decoding algorithms proposed in the literature for these codes, all of which can be considered as extensions and simplifications of belief propagation, or sum-product algorithm [4], to the non-binary domain. At the end of the chapter, we review an alternative decoding approach, majority-logic decoding, proposed for resource-constrained systems that use NB-LDPC codes.

Chapter 3 : Expansions of Non-binary Factor Graphs

This chapter contains our major contribution, a novel method to expand a non-binary factor graph over any subfield of the original field. The method allows representing the same code in a number of different ways, which are used later for designing low-complexity decoding schemes. It could also be useful for constructing codes that are better suited for iterative decoding, which would be discussed in another chapter. More focus is given to two special forms of the expansion, those that result in the smallest and largest expanded graphs. We also provide brief overviews of the two most well-known expansions proposed in the literature for NB-LDPC codes, which can only produce binary graphs.

Chapter 4 : Decoding Based on Expanded Graphs

Here we design novel decoding strategies for NB-LDPC codes using the expansions presented in the previous chapter. Since iterative decoding is impacted by certain graph structures, we first present a brief literature review of those structures. Then we investigate how these are transformed through the graph expansions discussed in the previous chapter, and show that special conditions have to be met for a short cycle to exist in a graph produced by our expansion. Using the insights on harmful graph structures, we present a strategy to adapt any algorithm proposed for decoding NB-LDPC codes for any graph produced by that expansion. This makes many decoding options available for any application that uses these codes, where each offers a unique performance-complexity trade-off. Due to its unique features, we consider the binary case separately. This dissertation provides simulation results and complexity comparisons that gives evidence of the attractive complexity gains offered by the proposed strategies, obtained with minimal performance losses. We also propose a novel binary majority-logic decoding algorithm for NB-LDPC codes, which is based on our binary expansion. This algorithm outperforms similar existing algorithms with only a marginal increase in complexity, as evidenced by the simulation results and complexity analysis provided.

Chapter 5 : Iterative Soft Decoding of Reed-Solomon Codes

In this chapter we focus on a different class of non-binary codes, Reed-Solomon (RS) codes. Main contribution here is a novel iterative soft-decoding scheme for these codes. We propose constructing alternative non-binary parity-check matrices for RS codes that better facilitate iterative decoding. Insights on graphical structures undesirable for iterative decoding gained in the previous chapter is again made of use here. Our soft-decoding strategy for the new matrices is able to offer significant gains over hard-decoding, and is of much lower complexity than any soft-decoding algorithms proposed for RS codes so far in the literature.

Chapter 2

Preliminaries

This dissertation primarily focuses on decoding non-binary low-density parity-check (NB-LDPC) codes, and therefore, we use this chapter to present some basic background knowledge on them, primarily the related mathematical concepts and the decoding algorithms.

2.1 Groups and Finite Fields

Most of the time, NB-LDPC codes are constructed over finite fields, particularly those of characteristic 2 [5]. Next few chapters of this thesis primarily deal with reducing decoding complexity of such codes via expansions of the corresponding factor graphs. Expansions proposed are based on certain concepts related to mathematical groups and finite fields, which are briefly reviewed in this section.

2.1.1 Groups

In the following, we define a ***group***, a fundamental algebraic structure instrumental in the study of finite fields.

Definition 2.1. *A **group** $\mathbb{G}$ is a set associated with a binary operation $\otimes$ that satisfies the following four axioms.*

1. *Group is **closed** under the binary operation. That is;*
   $\forall g_i, g_j \in \mathbb{G}, \quad g_i \otimes g_j \in \mathbb{G}$
2. *Binary operation is **associative**. That is;*
   $\forall g_i, g_j, g_k \in \mathbb{G}, \quad (g_i \otimes g_j) \otimes g_k = g_i \otimes (g_j \otimes g_k)$

3. *There exists an* ***identity element*** ϵ*. That is;*
   $\exists \epsilon \in \mathbb{G}$ *such that* $\forall g \in \mathbb{G}, \quad \epsilon \otimes g = g \otimes \epsilon = g$

4. *There exists an* ***inverse*** *for any group element. That is;*
   $\forall g \in \mathbb{G}$, $\exists g^{-1}$ *such that* $\quad g \otimes g^{-1} = \epsilon$

In addition to the four group axioms, in certain groups the binary operation is also *commutative*. These are referred to as *abelian groups*, defined as follows.

Definition 2.2. *An* ***abelian group*** *is a group* $\mathbb{G}$ *with a* ***commutative*** *binary operation* $\otimes$*. That is;*

$$\forall g_i, g_j \in \mathbb{G}, \quad g_i \otimes g_j = g_j \otimes g_i$$

The number of elements in a group $\mathbb{G}$ is referred to as the *order of* $\mathbb{G}$, represented with $|\mathbb{G}|$. Many groups contain special subsets that exhibit properties of a group by themselves. These are known as *subgroups*, which are defined as follows.

Definition 2.3. *A* ***subgroup*** $\mathbb{H}$ *of a group* $\mathbb{G}$ *is a subset of* $\mathbb{G}$ *that forms a group under the same binary operation* $\otimes$ *of* $\mathbb{G}$*.*

When $\mathbb{H}$ is a subgroup of $\mathbb{G}$, this will be denoted as $\mathbb{H} \leq \mathbb{G}$. For any group $\mathbb{G}$, the subset containing only the identity element, $\{\epsilon\}$, would satisfy the definition of a subgroup. In the literature, this is known as the *trivial subgroup* [7]. Also, $\mathbb{G}$ itself can be considered a subgroup. A *proper subgroup* of $\mathbb{G}$ is a subgroup $\mathbb{H} \neq \mathbb{G}$, usually denoted as $\mathbb{H} < \mathbb{G}$.

An important concept related to subgroups is that of a *normal subgroup*, which may be defined as follows. In the following definition, and onwards, we use $\otimes$ to represent the group binary operation, and g^{-1} as the inverse of g.

Definition 2.4. *A* ***normal subgroup*** $\mathbb{N}$ *of a group* $\mathbb{G}$ *is a subgroup that is invariant under conjugation. That is;*

$$\forall n \in \mathbb{N}, \forall g \in \mathbb{G}; \quad g \otimes n \otimes g^{-1} \in \mathbb{N}$$

When $\mathbb{N}$ is a normal subgroup of $\mathbb{G}$, this will be denoted as $\mathbb{N} \triangleleft \mathbb{G}$.

A special type of subsets of group $\mathbb{G}$ can be generated using any subgroup $\mathbb{H}$ of $\mathbb{G}$. Known as *cosets*, these subsets have certain special properties that make them quite useful in many different fields, including coding theory. Concept of a coset is presented in definition 2.5.

Definition 2.5. *Let $\mathbb{H}$ be a subgroup of a group $\mathbb{G}$. Then **left cosets** of $\mathbb{H}$ are the sets;*

$$g \otimes \mathbb{H} = \{g \otimes h; h \in \mathbb{H}\}; \quad \forall g \in \mathbb{G}$$

***Right cosets** are similarly defined as;*

$$\mathbb{H} \otimes g = \{h \otimes g; h \in \mathbb{H}\}; \quad \forall g \in \mathbb{G}$$

If $\mathbb{G}$ is an abelian group, there would not be any distinction between the left cosets and the right cosets of $\mathbb{H}$. This is a condition equivalent to that in definition 2.4, which makes any subgroup of an abelian group a normal subgroup [7]. As a majority of groups encountered in this thesis are abelian, from here onwards, focus would only be on them. Any group should be considered as such, unless explicitly stated.

Since a subgroup $\mathbb{H}$ matches definition 2.5, with $g = \epsilon$, the identity element of $\mathbb{G}$, it can also be considered a coset of itself. But it is not a *proper coset*, which is a coset $\mathcal{C}_{\mathbb{H}} \neq \mathbb{H}$. Also, any coset $\mathcal{C}_{\mathbb{H}}$ should be of the same cardinality as $\mathbb{H}$. From the definition, it also seems that the number of cosets should be equal to $|\mathbb{G}|$. But all of these cosets would not be distinct. In fact, any two non-disjoint cosets will be identical [7]. Thus, distinct cosets of a subgroup $\mathbb{H}$ will always be disjoint, and the union of these will form the group $\mathbb{G}$. These observations provide the basis for the well-known *Lagrange's theorem*. In the following, we state this theorem for the sake of completeness, and direct the reader to [7] for the proof.

Theorem 2.1 (Lagrange's Theorem)**.** *Let $\mathbb{G}$ be a finite group, and $\mathbb{H}$ any subgroup of $\mathbb{G}$. Then the order of $\mathbb{H}$, $|\mathbb{H}|$, will divide the order of $\mathbb{G}$, $|\mathbb{G}|$.*

If the union of all distinct cosets of subgroup $\mathbb{H}$ forms the group $\mathbb{G}$, the set of distinct cosets can be considered a *partioning* of $\mathbb{G}$. This is referred to as the *coset decomposition* of $\mathbb{G}$, with respect to $\mathbb{H}$. If $\mathbb{H}$ is a normal subgroup, then the distinct cosets form a group themselves, referred to as a *quotient group*, which can be defined as follows.

Definition 2.6. *Let $\mathbb{N}$ be a normal subgroup of group $\mathbb{G}$. Then the **quotient group** of $\mathbb{N}$ in $\mathbb{G}$, ${}^{\mathbb{G}}/_{\mathbb{N}}$, is the set of cosets of $\mathbb{N}$. Elements of ${}^{\mathbb{G}}/_{\mathbb{N}}$ form a group under the same binary operation as $\mathbb{G}$.*

2.1.2 Homomorphisms

Often in the study of groups, one may encounter several of the same type. It might be possible to find a *mapping* from one such group to another such that the structure of the groups are

preserved. A *structure preserving map* of the sort between any two algebraic structures is called a *homomorphism*, which may be defined in the context of groups as follows.

Definition 2.7. *Let $\mathbb{G}$ and $\mathbb{H}$ be two groups, respectively associated with the binary operations $\otimes$ and $\ominus$. A **group homomorphism** from $\mathbb{G}$ to $\mathbb{H}$ is a mapping $\psi : \mathbb{G} \to \mathbb{H}$ that preserves the structure of the groups. That is;*

$$\forall g_i, g_j \in \mathbb{G}, \;\; \psi(g_i \otimes g_j) = \psi(g_i) \ominus \psi(g_j)$$

Two important concepts related to homomorphisms are those of *kernel* and *image*, which are defined in the following.

Definition 2.8. *Let $\mathbb{G}$ and $\mathbb{H}$ be two groups, and ψ a group homomorphism from $\mathbb{G}$ to $\mathbb{H}$. Then;*

- *The **kernel** of ψ is the subset of all elements of $\mathbb{G}$ that are mapped to the identity element ϵ of $\mathbb{H}$. That is*

$$\ker \psi = \{g \in \mathbb{G}; \;\; \psi(g) = \epsilon\}$$

- *The **image** of ψ is the subset of all elements of $\mathbb{H}$ to which at least one element of $\mathbb{G}$ is mapped by ψ. That is*

$$\operatorname{img} \psi = \{\psi(g); \;\; g \in \mathbb{G}\}$$

Note that $\ker \psi$ is a subgroup of $\mathbb{G}$, and $\operatorname{img} \psi$ is a subgroup of $\mathbb{H}$ [7]. Also, for some homomorphisms, $\operatorname{img} \psi = \mathbb{H}$, in which case they are called *surjective* homomorphisms. A special class of homomorphisms are *isomorphisms*, defined, in the context of groups, as follows.

Definition 2.9. *An **isomorphism** between two groups $\mathbb{G}$ and $\mathbb{H}$ is a homomorphism where each element of $\mathbb{G}$ is mapped to only one element of $\mathbb{H}$, and vice-versa.*

In mathematical terminology, a one-to-one mapping between the elements of two sets, such as the one described in definition 2.9, is referred to as a *bijection*, which makes an isomorphism a *bijective* homomorphism. In the following, we present an interesting theorem related to isomorphisms, which is made of use in later chapters. This theorem is often referred to as the *first isomorphism theorem*. Once more, we direct the reader to [7] for the proof.

Theorem 2.2 (First Isomorphism Theorem)**.** *Let $\mathbb{G}$ and $\mathbb{H}$ be groups, and ψ a homorphism from $\mathbb{G}$ to $\mathbb{H}$. Then:*

- *$\ker \psi$ is a normal subgroup of $\mathbb{G}$.*

- $\mathrm{img}\,\psi$ *is a subgroup of* $\mathbb{H}$.
- $\mathrm{img}\,\psi$ *is isomorphic to the quotient group* $\mathbb{G}\,/_{\ker\psi}$.

It is clear from the theorem that in the case where ψ is surjective, $\mathbb{H}$ is isomorphic to $\mathbb{G}\,/_{\ker\psi}$.

2.1.3 Finite Fields

As discussed previously, *finite fields* are the most commonly used algebraic structure to construct NB-LDPC codes. These are widely studied mathematical objects, and the literature on them is vast. Here we focus only on the properties of finite fields that are relevant for NB-LDPC codes. A finite field may be defined as follows.

Definition 2.10. *A* ***finite field*** $\mathbb{F}$ *is a set associated with two binary operations, commonly referred to as addition* ($\oplus$) *and multiplication* ($\otimes$)*, that satisfy the following conditions.*

1. $\mathbb{F}$ *is an abelian group under addition* $\oplus$.
2. $\mathbb{F}^* = \mathbb{F} - \{0\}$ *is an abelian group under multiplication* $\otimes$*, where* 0 *is the identity element of the additive group.*
3. *Multiplication* $\otimes$ *is distributive over addition* $\oplus$*. That is;*
   $\forall \beta_1, \beta_2, \beta_3 \in \mathbb{F},\ \ \beta_1 \otimes (\beta_2 \oplus \beta_3) = (\beta_1 \otimes \beta_2) \oplus (\beta_1 \otimes \beta_3)$

There only exists finite fields of order p^k, where p is a prime, and k is a positive integer. Furthermore, for any value of p and k, a finite field would exist [7]. In the field of order p^k, $\mathbb{F}_{p^k}$, adding p copies of the same element together would always result in the additive identity 0. This prime value p is called the *characteristic* of the field $\mathbb{F}_{p^k}$. This means that, in characteristic 2 fields, which are the ones mostly used for NB-LDPC construction, adding an element to itself would result in 0. Therefore, in a characteristic 2 field, additive inverse of any element is itself.

Finite fields are unique algebraic structures; there can only exist a single unique field of any given order [7]. Thus, the only difference in two fields of the same order would be in the *labeling of elements*. In such a case, an isomorphism would exist between the two, and one would be called *isomorphic* to the other.

A finite field may contain one or many elements which can *generate* its multiplicative group. Such an element is referred to as a *primitive* element of the field, which is formally defined in the following. Primitive element is only unique for fields of order 2 and 3 [7].

Definition 2.11. *Let $\mathbb{F}$ be a finite field. Then a **primitive element** α is an element of $\mathbb{F}$ that can generate its multiplicative group. That is;*

$$\forall \beta \in \mathbb{F}, \exists i \in \mathbb{Z} \text{ such that } \beta = \alpha^i$$

Elements of a finite field may often be represented as powers of a primitive element, similar to Definition 2.11. In such a representation, additive identity will be denoted with either 0 or $\alpha^{-\infty}$.

An alternative representation of fields which is also quite common is the *polynomial representation*. This representation uses the notion of *irreducible polynomials*, which can be defined as follows.

Definition 2.12. *An **irreducible polynomial** $\Pi(x)$ is a polynomial of degree > 0 that cannot be expressed as the product of two or more polynomials of degree > 0.*

Elements of the finite field $\mathbb{F}_{p^k}$ can be represented with polynomials of degree at most $(k-1)$ with coefficients from $\mathbb{F}_p$. Then the operations of addition and multiplication will all be performed *modulo* some irreducible polynomial, which is usually referred to as the *primitive polynomial* of the field. This irreducible polynomial, also over $\mathbb{F}_p$, is of degree k, and the coefficient of its highest degree term is 1 (multiplicative identity of $\mathbb{F}_p$). Therefore, in mathematical terminology, primitive polynomial is a *monic* irreducible polynomial.

At times, each finite field element may be represented with the vector of coefficients of the respective polynomial, which is known as the *vector representation*. With such a representation, addition operation becomes a vector addition in $\mathbb{F}_p$.

Any monic polynomial over a field is associated with a matrix over the same field. This matrix, referred to as the *companion matrix*, is quite useful in certain cases, such as when the polynomial is irreducible [7]. Following definition presents the structure of the companion matrix, for a general monic polynomial.

Definition 2.13. *Let $p_q(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \ldots + c_{n-1} \cdot x^{n-1} + x^n$ be a monic polynomial of degree n over $\mathbb{F}_q$. Then the **companion matrix** of $p_q(x)$ is the $n \times n$ square matrix $C(p_q)$ defined*

as follows, where $-c_i$ represents the additive inverse of c_i.

$$C(p_q) = \begin{bmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{bmatrix}$$

A less frequently used representation of a finite field uses the companion matrix of the field's primitive polynomial. For $\mathbb{F}_q$, where $q = p^r$ for some prime p, primitive polynomial is a monic irreducible polynomial of degree r, over $\mathbb{F}_p$. Distinct sums of powers of the companion matrix of such a polynomial has been observed to be isomorphic to $\mathbb{F}_q$ [8], and thus, the respective set of matrices can be considered a representation of $\mathbb{F}_q$, known as the *matrix representation*. It should be noted that this set would contain the $r \times r$ all-zero matrix, which maps to the additive identity. Also, any primitive element of the field would map to a matrix capable of generating all other non-all-zero matrices [8].

Finite fields can also contain special subsets that act as fields themselves, similar to the case with subgroups. These *subfields* are defined as follows.

Definition 2.14. *A **subfield** $\mathbb{K}$ of a field $\mathbb{F}$ is a subset of $\mathbb{F}$ which is itself a field with the same two binary operations as $\mathbb{F}$.*

In the literature, if $\mathbb{K}$ is a subfield of $\mathbb{F}$, then $\mathbb{F}$ is sometimes called an *extension field* of $\mathbb{K}$. This is because, in such a case, it is possible to represent elements of $\mathbb{F}$ as polynomials, or vectors, over $\mathbb{K}$.

In the following we state a theorem on the existence of subfields, which will be used in the remaining chapters of the thesis.

Theorem 2.3 (Subfield Criterion)**.** *Let $\mathbb{F}_{p^k}$ denote a finite field. Any subfield of $\mathbb{F}_{p^k}$ should be of order p^d, where $d \mid k$.*

Proof of the theorem is available from [9].

2.2 Non-binary LDPC Codes

A non-binary LDPC code, also called a Q-ary LDPC code, can be thought of as the vector space orthogonal to the row space of some *sparse* Q-ary matrix. In coding theory terminology,

this matrix is referred to as the *parity-check matrix* (PCM) of the code. Sparsity of the PCM is advantageous in terms of decoding, which will be discussed later in this chapter.

In the literature, factor graphs are often used in the analysis of NB-LDPC codes. A factor graph is a bi-partite graph, where one type of nodes represents a set of variables, and the other type represents the factors of some objective function in those variables. In the case of decoding LDPC codes, the objective is finding the most likely codeword, factors are the parity-check equations, and variables are the codeword symbols. Thus, in the general form of a PCM, rows are mapped to one type of nodes, *parity-check nodes*, and columns to the second type, *variable nodes*. For the sake of brevity, parity-check nodes are often referred to as simply *check nodes*. An *edge* exists between check node i and variable node j only if (i, j)th position of the PCM is non-zero. Non-zero value at that position will be used as the *label*, or the *weight*, of the edge. This graphical representation was first introduced by Tanner in 1981 [10], and hence, the graphs are sometimes referred to as *Tanner graphs*.

Figure 2.1 shows the factor graph representation of the 4×7 PCM H_x, given in (2.1). Parity-check equations and the corresponding nodes are labeled with C_is, while codeword symbols and the respective nodes are labeled with V_is. Non-zero values X_i are from a some suitable algebraic structure.

$$H_x = \begin{array}{c} \begin{matrix} V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 \end{matrix} \\ \begin{pmatrix} \mathrm{X}_1 & \mathrm{X}_3 & 0 & \mathrm{X}_6 & 0 & \mathrm{X}_{10} & 0 \\ 0 & 0 & \mathrm{X}_4 & 0 & \mathrm{X}_8 & 0 & \mathrm{X}_{12} \\ \mathrm{X}_2 & 0 & \mathrm{X}_5 & \mathrm{X}_7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathrm{X}_9 & \mathrm{X}_{11} & 0 \end{pmatrix} \end{array} \begin{matrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{matrix} \tag{2.1}$$

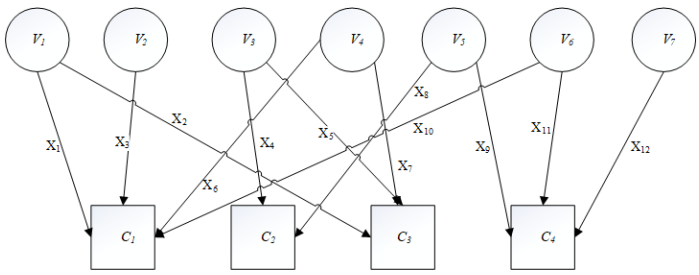

Figure 2.1: Tanner graph for PCM H_x (in eq. 2.1)

A variable node connected to some check node is considered to be in its *neighborhood*, and vice versa. Number of nodes in the neighborhood of a node is called the *degree* of that particular node. For example, in Figure 2.1, variable nodes V_5, V_6 and V_7 are in the neighborhood of check node C_4, and thus, C_4 is of degree 3. In the literature, degree of a check node and a variable node are referred to as *check degree* (d_c) and *variable degree* (d_v) respectively. These degrees are sometimes viewed from the perspective of the PCM, in which case they will be called the *row weight* and the *column weight*.

NB-LDPC codes can be divided in to two categories based on node degrees, similar to their binary counterparts. Codes where nodes of the same type are of the same degree are called *regular* NB-LDPC codes, and those with different degrees are called *irregular* NB-LDPC codes. For example, in code represented by Figure 2.1, check nodes are of degrees 2, 3 and 4, and hence, it is an irregular code. With irregular codes, the *average* node degrees are often used in performance analysis.

2.2.1 Construction of NB-LDPC Codes

An NB-LDPC code is constructed by labeling a factor graph (or a PCM) with suitable properties for iterative, message passing decoding using elements of some algebraic structure. Any structure that satisfies the following set of properties can be used in this regard [11].

- Structure should be finite.
- Structure should be closed under two operations, usually called addition and multiplication.
- An additive inverse should exist for the elements used in codewords.
- A multiplicative inverse should exist for the elements used as edge labels.

Many algebraic structures satisfy these requirements, and any of them can be used to define an NB-LDPC code. The following secondary set of requirements are generally expected for efficient implementation of encoding and decoding [11].

- Identity elements should exist for addition and multiplication.
- Addition should be commutative and associative.
- Multiplication should be commutative, associative, and distributive over addition.

A finite field is an algebraic structure that satisfies all primary and secondary requirements above. In addition to that, all of its elements have additive and multiplicative inverses, and thus, any element can be used as an edge label or a code symbol. Also, they are well understood mathematical structures, and are used in a number of other applications. Due to these reasons, finite fields are the most widely used algebraic structure to construct NB-LDPC codes [5]. In the literature, structures such as integer modulo rings have also been used for this purpose [11], [12].

A factor graph used to construct a NB-LDPC code is expected to have many of the properties expected of a graph used in binary LDPC construction. In binary LDPC codes, short cycles are known to create *stopping sets* and *trapping sets*, substructures of a factor graph that negatively impact the decoding performance [13], especially in the high signal-to-noise ratio (SNR) region, leading to *error floor*. These graph substructures are also not desirable in the non-binary domain [14]. Therefore, any graph used for binary LDPC codes, where the *girth*, or length of the shortest cycle, has been maximized, can be used for constructing NB-LDPC codes. A more detailed review on graph substructures that impact decoding of these codes will be given in Chapter 4.

Graph labeling step in NB-LDPC construction is often done in a random fashion, although recently more focus has been given to deterministic labeling [15],[16],[17],[18]. These labels are chosen in such a way that the minimum distance of the binary equivalent code is increased, and the impact of the undesirable graph structures is minimized. Sets of optimized edge labels for some code parameters are available in the literature [15],[17]. Simulation results show that such labeling can offer significant performance improvements over random labeling.

In the binary domain, irregular codes are known to perform better than regular codes in general, with some getting as close as 0.06dB to channel capacity [19]. But the structure of regular codes makes encoding and decoding more efficient. With NB-LDPC codes, a significant difference between performance of regular and irregular codes has not been reported in the literature. Best performance, especially with codes over larger alphabets, is with using *ultra sparse* graphs, where variable degree is set to the minimum possible value of 2 [15]. Codes with very good performance has been constructed using protographs as well [16].

Rather than using a factor graph of a binary LDPC code or constructing one in a random fashion, and then labeling it, one of the *structured* construction methods available in the literature could be used [20],[21],[22],[23]. These methods are usually algebraic in nature, and often result in cyclic or quasi-cyclic (QC), regular LDPC codes with a high variable degree (column weight). The codes perform almost as well as randomly constructed ones. Moreover, due to their cyclic or quasi-cyclic structure, these codes are much more efficient to encode. The high variable degree makes the

codes better suited for *majority-logic decoding* [24], the hard-decision decoding equivalent of NB-LDPC codes. Majority-logic decoding is the decoding approach of choice for resource-constrained systems that use NB-LDPC codes, and we focus more on this in Chapter 4.

2.3 Q-ary Sum Product Algorithm

LDPC codes are most often used with the iterative message passing decoding algorithms. They can be viewed as passing messages back-and-forth along the edges of the factor graph, the reason for the term *'message passing'*. The Sum-Product algorithm (SPA) [4], also known as *belief propagation* (BP), is the best performing message passing decoder for binary LDPC codes. Capacity approaching performance with long LDPC codes can be realized with the SPA [19], and its complexity is not practically infeasible. The SPA can be easily extended for the non-binary, or Q-ary, case, and this version is known as *Q-ary sum product algorithm* (QSPA) [5].

BP decoding, which includes both SPA and QSPA, attempts to compute the *maximum aposteriori probability* (MAP) of each codeword symbol. In the case of SPA, MAP estimate can be a single value (such as $\frac{Pr(0)}{Pr(1)}$), and in QSPA, it would take the form of a probability mass function (PMF). Computed MAP estimate will be exact only in the case where the corresponding factor graph is completely free of cycles. But with codes of finite lengths this is not possible, and hence, MAP estimate in BP decoding would only be suboptimal, which can be expected to get better with the girth of the code.

In the following, we briefly go over the primary steps of QSPA, with a code defined over a finite field $\mathbb{F}_q$.

1. Initialization

Every variable node will be initialized with a PMF of length q, computed with the received channel information. We denote this initial PMF of variable node n with $\underline{I}_n$.

After initialization, variable node n would send the PMF $\underline{I}_n$ to each check node m in its neighborhood. This operation is represented with the following equation, where $r_{n,m}^{(0)}$ denotes the message from variable node n to check node m during initialization (or the 0'th iteration).

$$\underline{r}_{n,m}^{(0)} = \underline{I}_n \tag{2.2}$$

Maximum number of decoder iterations, k_{max}, is also set.

2. Check Node Operations

A check node of a NB-LDPC code over $\mathbb{F}_q$ represents a parity-check equation over the field. In each iteration, this node would receive PMFs from every variable node in its neighborhood. Computations are then carried out to estimate a PMF for each such variable node using the PMFs received from all other nodes, such that the parity-check constraint represented is satisfied. These PMF estimates would be the messages sent back to the variable nodes.

Consider a check node m, and a variable node n in its neighborhood $N(m)$ during the k'th iteration of QSPA. For node n to take the value $\beta \in \mathbb{F}_q$, and the parity-check equation to be satisfied, there are several different configurations for the values of other variable nodes in $N(m)$. We represent this set of configurations as $C^m_{n,\beta}$.

$$C^m_{n,\beta} = \{\underline{c} = (x_0, x_1, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{d^m_c-1}); \sum_{i \in N(m), i \neq n} h_i \cdot x_i = -h_n \cdot \beta\} \tag{2.3}$$

Please note that $\underline{c}$ is of length $(d^m_c - 1)$, where d^m_c is the degree of check node m, and $h_i, x_i \in \mathbb{F}_q$. Each x_i represents the value taken by variable node i $(i \neq n)$ such that node n can take the value of β and parity-check equation is satisfied. h_is are the labels of the edges connected to node m, and $-(h_n \cdot \beta)$ is the additive inverse of $h_n \cdot \beta$. Probability of node n's value being β can then be computed as;

$$\underline{p}^{(k)}_{m,n}(\beta) = \sum_{\underline{c} \in C^m_{n,a}} \prod_{x_i \in \underline{c}} \underline{r}^{(k-1)}_{i,m}(x_i) \tag{2.4}$$

Message from check node m to variable node n is the PMF $\underline{p}^{(k)}_{m,n}$, the vector of values computed as in 2.4.

3. Variable Node Operations

Compared with the check node operations, those at a variable node are relatively simple. Main objective here is to update the respective PMF using estimates received from check nodes in the neighborhood. If the updated PMFs at any of the variable nodes are inaccurate, which would stop the decoder from converging to a codeword, each variable node would compute new PMF estimates to be sent to all connected check nodes.

Updating the PMF of variable node n, $\underline{V}(n)$, during decoding iteration k, is given by the following equation. Product is taken *element-wise*, and $M(n)$ denotes the neighborhood of variable node n.

$$\underline{V}^{(k)}_n = \underline{I}_n \times \prod_{i \in M(n)} \underline{p}^{(k)}_{i,n} \tag{2.5}$$

Symbol with the maximum probability in $\underline{V}_n^{(k)}$ is chosen as the *tentative decision* on node n. The vector of tentative decisions of all variable nodes is the decoder's estimate for the codeword at the k'th iteration. This vector is then evaluated by every parity-check node, and if all the constraints are satisfied, which means the decoder has converged to a codeword, it would terminate and output the corresponding codeword. If $k = k_{max}$, decoder would terminate with failure. Otherwise, new PMF estimates for connected check nodes are computed at each variable node. In keeping with the extrinsic principle of BP, estimate for check node m is computed using messages from other check nodes only. With the decoder operating in the probability domain, a normalization operation is also required here. The normalization $\mathbb{N}()$, for a vector $\underline{v}$ of length l, can be defined as follows.

$$\mathbb{N}(\underline{v}) = \underline{v} \Big/ \sum_{i=0}^{l-1} v_i$$

The new PMF estimates for connected check nodes are then computed as

$$\underline{r}_{n,m}^{(k)} = \mathbb{N}\Big(\underline{I}_n \times \prod_{i \in M(n), i \neq m} \underline{p}_{i,n}^{(k)}\Big) \tag{2.6}$$

Of the three steps of QSPA, check node computations seem the most complex. In the literature, these are usually viewed as a combination of two distinct sub-steps; permutation of PMFs and convolution of PMFs [25].

If all edge labels were 1, (2.3) would take the following form.

$$C_{n,\beta}^m = \{\underline{c} = (x_0, x_1, \ldots, x_{n-1}, x_{n+1}, \ldots, x_{d_c^m - 1}); \sum_{i \in N(m), i \neq n} x_i = -\beta\} \tag{2.7}$$

With an edge label $h_i \neq 1$, as is often the case with NB-LDPC codes, probability of some symbol $\beta \in \mathbb{F}_q$ in the PMF estimate sent by the variable node would turn in to probability of $h_i \cdot \beta \in \mathbb{F}_q$ at the check node. This means the PMF estimate is undergoing a *permutation* due to the edge label h_i. Traditional approach is to first carry out this permutation, for all incoming PMF estimates. Then all edge labels can be taken as 1 in any subsequent operation at check nodes. After permutation sub-step (2.3) can be written in a form similar to (2.7) as follows, where $x_i' = h_i \cdot x_i$ and $\beta' = h_n \cdot \beta$.

$$C_{n,\beta}^m = \{\underline{c} = (x_0', x_1', \ldots, x_{n-1}', x_{n+1}', \ldots, x_{d_c^m - 1}'); \sum_{i \in N(m), i \neq n} x_i' = -\beta'\} \tag{2.8}$$

When the PMFs are permuted according to the edge labels, operation in (2.4) becomes a circular convolution of them. This operation is shown in (2.9), where $\circledast$ represents circular convo-

lution, and $\underline{r}'^{(k-1)}_{i,m}$ represents the permuted versions of PMFs received from variable nodes.

$$\underline{p}^{(k)}_{m,n} = \circledast_{i \in N(m); i \neq n} \underline{r}'^{(k-1)}_{i,m} \tag{2.9}$$

Reverse of the permutation applied to the estimate received from variable node n, $\underline{r}^{(k-1)}_{m,n}$, has to be applied to $\underline{p}^{(k)}_{m,n}$ before sending it as a message to that node.

A direct implementation of QSPA, using the equations presented above, is not suitable for practical applications due to two reasons.

1. A probability domain implementation is not well suited for hardware.

2. A direct implementation repeats the same operation a number of times, particularly in computing (2.6) and (2.9), which increases the decoding complexity.

When it comes to implementing QSPA in hardware, probability domain has several drawbacks [26],[27]. Practical implementations of decoders almost always use fixed point arithmetic, and calculations in probability domain can easily lead to underflow issues, especially in steps such as (2.5). Also, channel information will have to be quantized, and effects of this would be felt more severely in probability domain [26]. Furthermore, a probability domain implementation would require a large number of multiplications, which are more costly at hardware level [26], [27]. Because of these reasons, practical implementations of QSPA would often be in either the log domain or the log-likelihood-ratio (LLR) domain. This allows multiplications to be replaced with additions, a more hardware friendly operation. Also, these implementations would not be as sensitive to quantization effects, and are better suited for fixed point arithmetic.

Carrying out a calculation such as (2.9) or (2.6) for each variable/check node separately would see many operations getting repeated. For example, with a check node of degree five, many pairs of received PMFs ($\underline{r}^{(k)}_{n,m}$) would undergo a convolution at least twice when completing (2.9). This is an unnecessary complexity cost, which could be easily avoided by carrying out any computation only once, and saving the result for future use. Different implementations of QSPA use different strategies in this regard.

In the following sub-sections, we briefly go over the two most common implementations of QSPA, LLR-QSPA [26] and fast Fourier transform based QSPA (FFT-QSPA) [25]. Each of these deals with the aforementioned shortcomings of a direct implementation in different ways, and presents unique advantages.

2.3.1 LLR-QSPA

LLR domain implementation of QSPA [26] is a generalization of the approach taken in binary LDPC decoders, where operations are almost always in LLR domain, with individual probabilities replaced with $\ln(\frac{Pr(0)}{Pr(1)})$. In LLR-QSPA, the ratio used is $\ln(\frac{Pr(\beta)}{Pr(\theta)})$, where θ is a fixed value, often 0, the additive identity. With this, conversion of a probability vector $\underline{v}$ to an LLR vector $\underline{V}$, which we denote with $\mathrm{llr}(\underline{v})$, can be represented as follows.

$$\underline{V}(\beta) = \mathrm{llr}(\underline{v}) = \ln\{\frac{\underline{v}(\beta)}{\underline{v}(0)}\} \tag{2.10}$$

Please note that due to the properties of logarithms, the operation $\mathrm{llr}(\underline{v}_1 \times \underline{v}_2)$ can be computed as $\mathrm{llr}(\underline{v}_1) + \mathrm{llr}(\underline{v}_2)$.

The decoder will be operating in the LLR domain from initialization, and therefore, initial PMFs $\underline{I}_n$, in (2.2), (2.5) and (2.6), have to be converted to LLR vectors as in (2.10). In the following, these initial LLR vectors will be represented with $\underline{L}_n^{(0)}$. We chose to represent the LLR domain equivalent of $\underline{V}_n^{(k)}$ in (2.5), the updated PMF of variable node n in iteration k, as $\underline{L}_n^{(k)}$. When the message vectors, or PMF estimates $\underline{r}_{n,m}^{(k)}$ and $\underline{p}_{m,n}^{(k)}$, in (2.2) to (2.9), are converted to the LLR domain, they will be represented with $\underline{R}_{n,m}^{(k)}$ and $\underline{P}_{m,n}^{(k)}$ respectively.

In LLR-QSPA, an additional operation, that of converting initial PMFs $\underline{I}_n$ to LLR domain has to be carried out during initialization. But any complexity increase therein is dwarfed by the complexity gains in variable node operations. This is one of the biggest advantages of LLR-QSPA, and therefore, we will first consider how variable node operations, represented by (2.5) and (2.6), are changed in the algorithm.

As was seen previously, many real number multiplications are required to complete (2.5), where the variable node PMFs are updated. But now, with message vectors in LLR domain, these multiplications can be replaced with simple additions. (2.11) in the following represents this operation.

$$\underline{L}_n^{(k)} = \mathrm{llr}(\underline{V}_n^{(k)}) = \mathrm{llr}\{\underline{I}_n \times \prod_{i \in M(n)} \underline{p}_{i,n}^{(k)}\} = \underline{L}_n^{(0)} + \sum_{i \in M(n)} \underline{P}_{i,n}^{(k)} \tag{2.11}$$

It is possible to change (2.6) by straightaway applying the $\mathrm{llr}()$ operation as in (2.11). But an alternative approach is taken to compute the new PMF estimates in LLR-QSPA. Rather than carrying out an expensive operation for each new estimate, the LLR vector received from check node m is simply *subtracted* from the updated PMF $\underline{L}_n^{(k)}$. This results in (2.6) being changed to the

following simpler form. Note that unlike in probability domain, a normalization operation is not required here, which results in another complexity gain.

$$\underline{R}_{n,m}^{(k)} = \underline{L}_{n}^{(k)} - \underline{P}_{m,n}^{(k)} \tag{2.12}$$

In check node operations though, there are no clear advantages in using LLRs, as (2.9) uses a number of additions in probability domain, for which there is no straight-forward counterpart in LLR domain. In the literature, a probability domain addition carried out in log/LLR domain can be referred to as a $\max^*$ operation [26]. (2.13) in the following represents this operation, which can be broken down to a comparison and a secondary computation. l_1 and l_2 represent LLRs of two probability values.

$$\max^*(l_1, l_2) = \ln(e^{l_1} + e^{l_2}) = \max(l_1, l_2) + \ln(1 + e^{-|l_1 - l_2|}) \tag{2.13}$$

Second term above, which is often called the *correction term*, is inversely proportional to the difference of the two LLR values. It can be easily implemented with the help of a look-up table. That way, a convolution between two PMFs in LLR domain can be completed using only additions, comparisons, and table look-ups.

It is evident that converting (2.9) to LLR domain will result in a very similar computation; convolution of LLR vectors, which can be completed with $\max^*$ operations defined in (2.13) and additions. But here also, in order to avoid repeating the same calculations, LLR-QSPA uses an alternative approach, the *forward-backward* matrix [28], a computation strategy used in a wide array of applications. In the following, we discuss how this strategy is applied.

Consider an arbitrary check node m of degree d_c. In any iteration k, to estimate PMFs (in LLR form) for each connected variable node, node m would first construct two matrices, forward matrix $\mathcal{F}_m^{(k)}$ and backward matrix $\mathcal{B}_m^{(k)}$. Each matrix would be of dimension $q \times d_c$, where q is the alphabet size, and they would be constructed using vectors $\underline{R}_{i,m}^{(k-1)}$ received from neighboring variable nodes.

The i'th column of $\mathcal{F}_m^{(k)}$ will hold the result of circular convolution between its $(i-1)$'th column and the LLR vector received from the i'th variable node, $\underline{R}_{i,m}^{(k-1)}$. First column will be equal to $\underline{R}_{1,m}^{(k-1)}$. Similarly, the i'th column of $\mathcal{B}_m^{(k)}$ will be the result of convolution between its $(i+1)$'th column and $\underline{R}_{i,m}^{(k-1)}$. Here, the last, or the d_c'th column, will be $\underline{R}_{d_c,m}^{(k-1)}$. Construction of the two matrices are represented with following equations.

$$\mathcal{F}_m^{(k)}(:,\, i) = \mathcal{F}_m^{(k)}(:,\, i-1) \circledast \underline{R}_{i,m}^{(k-1)}; \qquad i = 2, 3, \ldots, d_c \tag{2.14}$$

$$\mathcal{B}_m^{(k)}(:,\, i) = \mathcal{B}_m^{(k)}(:,\, i+1) \circledast \underline{R}_{i,m}^{(k-1)}; \qquad i = 1, 2, \ldots, d_c - 1 \tag{2.15}$$

Now instead of computing (2.9), PMF estimate for an arbitrary variable node $i \in N(m)$ can be found using $\mathcal{F}_m^{(k)}$ and $\mathcal{B}_m^{(k)}$ as follows.

$$\underline{P}_{m,i}^{(k)} = \mathcal{F}_m^{(k)}(:,\, i-1) \circledast \mathcal{B}_m^{(k)}(:,\, i+1); \qquad i = 2, 3, \ldots, d_c - 1 \tag{2.16}$$

Estimate for the first node will be $\mathcal{B}_m^{(k)}(:,\, 2)$, and for the d_c'th, it will be $\mathcal{F}_m^{(k)}(:,\, d_c - 1)$.

(2.14), (2.15), and (2.16) together complete the computation represented by (2.9) for all neighboring variable nodes n of check node m. As stated previously as well, convolution of two LLR vectors only requires additions, and $\max^*$ operations, which can be completed with a comparison, an addition, and a table look-up.

Forward-backward matrix approach makes sure that the same set of LLR vectors never undergo a convolution operation more than once. This significantly reduces the complexity of (2.9), when compared with a direct implementation. Convolution of two length q vectors is an operation of $\mathcal{O}(q^2)$ complexity, and this makes the overall complexity of forward-backward matrix approach $\mathcal{O}(3d_c \cdot q^2)$, since d_c LLR vectors of length q undergo convolution three times, to create the forward matrix, the backward matrix, and the final estimate for each variable node. For asymptotic complexity, the factor of three may be safely disregarded, resulting in a complexity of $\mathcal{O}(d_c \cdot q^2)$. Otherwise a direct implementation of (2.9) would be of $\mathcal{O}(d_c^2 \cdot q^2)$ complexity.

Thus, LLR domain implementation of QSPA manages to conduct decoding at a complexity order of $\mathcal{O}(q^2)$, and only requires additions, comparisons and table look-ups, all more efficient at hardware level than operations such as multiplications [29]. But the $\mathcal{O}(q^2)$ complexity order is still quite high, especially with larger values of q, for which NB-LDPC codes offer highest performance gains [15].

2.3.2 FFT-QSPA

It is a well-known fact that a convolution operation becomes an element-wise multiplication when the operands are transformed to an orthogonal domain, commonly referred to as frequency, or Fourier, domain. FFT-QSPA uses this observation to reduce the $\mathcal{O}(q^2)$ complexity of check node operations in QSPA. This algorithm was first proposed for codes over finite fields of characteristic 2 [25], the most commonly used algebraic structure for NB-LDPC codes. For these codes Walsh-Hadammard domain would be used as the orthogonal domain, while for codes such as those constructed over rings [11], Fourier domain could be used.

This dissertation primarily focuses on codes defined over finite fields of characteristic 2. Thus, for the sake of being more self-contained, we define the Walsh-Hadamard transform (WHT) and its inverse (IWHT) in the following.

Definition 2.15. *The **Walsh-Hadamard transform** (WHT) of a length 2^m vector $\underline{v}$ is the vector $\underline{\mathcal{V}}$ computed as;*

$$\underline{\mathcal{V}} = \mathcal{WHT}(\underline{v}) = \underline{v} \cdot H_m$$

H_m is the Hadamard matrix of dimension $2^m \times 2^m$, which is defined for $m > 0$ as

$$H_m = \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix}$$

and $H_0 = 1$.
*The **inverse Walsh-Hadamard transform** (IWHT) is used to compute $\underline{v}$ from $\underline{\mathcal{V}}$, as;*

$$\underline{v} = \mathcal{IWHT}(\underline{\mathcal{V}}) = \underline{\mathcal{V}} \cdot H_m^{-1}$$

H_m^{-1} is the inverse of H_m.
$\mathcal{WHT}()$ and $\mathcal{IWHT}()$ represent the Walsh-Hadammard transform and its inverse, respectively.

More in-depth investigations of the WHT and Hadamard matrices are available in [30].

Relationship between convolution and multiplication in an orthogonal domain is valid when vectors that undergo convolution are from the probability domain. Since the WHT of a PMF can contain negative values, for which logarithms are undefined, this approach may not suit log/LLR domain implementations well [27]. But a decoder operating entirely in probability domain is also undesirable, due to hardware implementation issues discussed earlier. Therefore, FFT-QSPA usually operates in log domain except at check nodes, where log values are first converted to probabilities so that WHT becomes feasible, and later converted back to log domain, for subsequent operations. Thus, initialization and variable node operations of this algorithm are very similar to those of LLR-QSPA. In fact, the sole difference here is that instead of LLR vectors, the logarithms of probability vectors are used, which also ensure that no multiplications are necessary for those steps. Therefore, we will only focus on check node operations, which take a significantly different form in FFT-QSPA.

Due to the special relationship between convolution and multiplication in orthogonal domains, (2.9) takes the following form in the Walsh-Hadamard domain, where $\underline{\mathcal{P}}_{m,n}^{(k)}$ and $\underline{\mathcal{R}}_{i,m}^{\prime(k-1)}$

denote the WHTs of $\underline{p}_{m,n}^{(k)}$ and $\underline{r}_{i,m}^{\prime(k-1)}$, respectively.

$$\mathcal{WHT}(\underline{p}_{m,n}^{(k)}) = \underline{\mathcal{P}}_{m,n}^{(k)} = \mathcal{WHT}\Big\{ \circledast_{i \in N(m); i \neq n} \underline{r}_{i,m}^{\prime(k-1)} \Big\} = \prod_{i \in N(m); i \neq n} \underline{\mathcal{R}}_{i,m}^{\prime(k-1)}$$

It is evident that PMF estimates for variable nodes, $\underline{p}_{m,n}^{(k)}$, can easily be computed with the IWHT as

$$\underline{p}_{m,n}^{(k)} = \mathcal{IWHT}\Big\{ \prod_{i \in N(m); i \neq n} \underline{\mathcal{P}}_{i,m}^{(k-1)} \Big\}$$

These operations can be more succinctly represented as (2.17) in the following, which replaces (2.9) in FFT-QSPA.

$$\underline{p}_{m,n}^{(k)} = \mathcal{IWHT}\Big\{ \prod_{i \in N(m); i \neq n} \mathcal{WHT}\{\underline{r}_{i,m}^{\prime(k-1)}\} \Big\} \tag{2.17}$$

To facilitate the computation of (2.17), the incoming PMFs, which will be in $\log$ domain, have to be first converted to probability domain. This operation is ususally conducted with look-up tables.

Computing (2.17) for each $n \in N(m)$ involves carrying out the same set of multiplications repeatedly. To avoid these repetitions, FFT-QSPA takes an approach similar to the forward-backward matrix computation of LLR-QSPA. WHTs of each incoming PMF, which are computed after permuting the received message vectors and transforming them to probability domain, are first stored in memory, to be re-used when required. Then, instead of computing (2.17) for each n, full product of WHTs, $\mathbb{T}_m^{(k)}$, is computed and stored. This operation is represented with the following equation. $\underline{r}_{i,m}^{\prime(k-1)}$ represents one of the probability vectors computed after permuting the received message vector.

$$\mathbb{T}_m^{(k)} = \prod_{i \in N(m)} \mathcal{WHT}\{\underline{r}_{i,m}^{\prime(k-1)}\} \tag{2.18}$$

Check node m's PMF estimate for a neighboring variable node n, $\underline{p}_{m,n}^{(k)}$, is then computed as follows, where $\mathcal{WHT}\{\underline{r}_{n,m}^{\prime(k-1)}\}$ is now available in memory, and division is carried out element-wise.

$$\underline{p}_{m,n}^{(k)} = \mathcal{IWHT}\Big\{ \mathbb{T}_m^{(k)} \,/\, \mathcal{WHT}\{\underline{r}_{n,m}^{\prime(k-1)}\} \Big\} \tag{2.19}$$

$\underline{p}_{m,n}^{(k)}$ above has to be converted to log domain, and re-permuted before sending to variable node n.

With this strategy, WHTs have to be computed only $2d_c$ times, and only $2d_c \cdot q$ multiplications and divisions are required. A direct implementation of (2.17) requires d_c^2 WHTs, and $\mathcal{O}(d_c^2 \cdot q)$

multiplications. Most complex computation in this approach is that of the WHT, which is of order $\mathcal{O}(q \cdot \log q)$. Therefore, the FFT-QSPA is $\mathcal{O}(q \cdot \log q)$ complexity, which is a significant reduction from $\mathcal{O}(q^2)$ of the LLR-QSPA, especially for larger q.

Although FFT-QSPA is more advantageous in terms of complexity order, it does have some issues in hardware implementation. Unlike LLR-QSPA, it requires many real number multiplications and divisions, more complex operations at hardware level, and a part of its operations are in probability domain, which are more difficult to handle with fixed point arithmetic. Therefore, LLR-QSPA is the version of QSPA considered most suitable for hardware implementations [27], [29], particularly for small to moderate values of q, for which complexity gains of FFT-QSPA are not very significant. In the interest of making FFT-QSPA more hardware friendly, a LLR domain implementation of it is considered in [27], which uses additional look-up tables for dealing with logarithms of negative values.

2.3.3 Simplifications

NB-LDPC codes offer very good performance with any implementation of QSPA discussed in the previous section, often outperforming their binary counterparts [15] which are the codes of choice in many practical applications. Still, NB-LDPC codes are yet to reach that levels of practical usage. This is mainly due to the high complexity of QSPA; $\mathcal{O}(q^2)$ or $\mathcal{O}(q \cdot \log q)$ are too high for many applications. With large alphabets, complexity of QSPA becomes especially high, which is unfortunate since performance of NB-LDPC codes improve with alphabet size [15].

Complexity bottleneck of any QSPA implementation is the check node calculations step. $\mathcal{O}(q^2)$ and $\mathcal{O}(q \cdot \log q)$ complexity orders of LLR-QSPA and FFT-QSPA are both due to that. Therefore, most approaches to reduce decoding complexity of NB-LDPC codes try to simplify check node computations.

As (2.3) shows, when computing the PMF estimate for some variable node, each symbol probability is calculated by summing up the probabilities of several different configurations. Some of these configurations will be highly probable ones, with relatively large probability values, while some others would have probabilities very close to zero. Impact of such highly improbable configurations on the final PMF estimate will be almost negligible, and therefore, one can afford to not consider these if it is advantageous in terms of complexity. This observation provides the basis to a number of simplifications of QSPA's check node operations.

One such simplification is the *max-log-SP algorithm* [26], a modification of LLR-QSPA.

In check node computations of LLR-QSPA, many different pairs of LLR vectors undergo convolution, for example in (2.14), (2.15), and (2.16). In such a convolution, each element of the resultant vector is the combination of many different terms which are combined using the $\max^*$ operation (see (2.13)). Max-log-SPA considers only the largest term out of these, and thus, comparisons are used instead of $\max^*$ operations. So the computation which required one comparison, two additions, and one table look-up is now approximated with a form that only requires a single comparison. This is the complexity advantage offered by max-log-SPA, but since it still needs to check all different terms to find the largest, its complexity order remains $\mathcal{O}(q^2)$. This algorithm can be considered the straight-forward generalization of the binary min-sum algorithm [31] for NB-LDPC codes [26],[28]. Results in the literature show that, although there is a complexity gain, performance loss in using max-log-SPA is significant [26].

In the literature, [32] considers a similar approach with FFT-QSPA, where, at check nodes, the n lowest values of a PMF received from a variable node are approximated with some uniform probability value, before transforming the PMFs to Hadammard domain. With PMFs of this form, a reduction in the number of operations required for WHT has been observed. But this complexity reduction directly depends on n, and also on positions of the values approximated [32], which can change for each PMF received by a check node. A larger value of n can be expected to significantly reduce decoding complexity, but the corresponding performance loss would also be significant. On the other hand, a smaller n would make performance losses minimal, but any gain in complexity will also be negligible.

One of the more well-known simplifications of QSPA, the *extended min-sum* algorithm (EMSA) [33], is also based on similar ideas. EMSA defines a configuration in a slightly different way to (2.3). Rather than considering them per variable node and symbol value, as (2.3) does, a configuration in EMSA is a set of d_c values, one for each variable node in the neighborhood, that together satisfy the parity-check constraint [33]. These can be defined as follows.

$$C^m = \{\underline{c} = (x_0, x_1, \ldots, x_{d_c-1}); \sum_{i \in N(m)} h_i \cdot x_i = 0\} \tag{2.20}$$

EMSA attempts to reduce the number of configurations in C^m by enforcing two conditions on them. First, a symbol value x_i in any configuration should correspond to one of the n_m largest values in the PMF estimate received from variable node i. Here, n_m is a user-defined parameter. In order to further reduce the number of configurations, a second restriction is applied; a configuration is only allowed to differ from the most likely one in at most n_c positions, where n_c

is also user-defined. These two restrictions make sure that only a fraction of the configurations in (2.20) is used in decoding.

In the interest of efficient hardware implementations, EMSA generally operates in the LLR domain. Similar to max-log-SPA [26], it approximates $\max^*$ operation by a comparison, thereby reducing the complexity of computing the LLR of a summation of two probabilities. But since EMSA only considers the n_m largest values in each received PMF estimate, the estimates have to be *sorted*, which is the complexity bottleneck of the algorithm. Sorting ensures that the algorithm is at least of $\mathcal{O}(q \cdot \log q)$ complexity, the complexity class of FFT-QSPA.

Other than sorting, check node operations of EMSA can be carried out using the forward-backward matrix approach [33] presented in sub-section 2.2.1. With this approach, complexity of the EMSA is $\mathcal{O}(n_m \cdot q)$. Results in the literature show that for $n_m < \log q$, performance loss relative to QSPA is too significant. Therefore, $n_m \geq \log q$ in most cases, which makes the complexity order of the EMSA higher than that of the FFT-QSPA. But when considering the individual operations required, it can be seen that complexity of EMSA is lower, since it mostly uses simple, hardware friendly operations such as comparisons and additions.

Due to the number of approximations used, PMF estimates computed at check nodes of EMSA can be quite different from those of QSPA, and this can lead to significant performance losses. In the literature, [33] suggests using a correction term, either additive or multiplicative, to make these estimates more accurate. This correction term have to be optimized through Monte-Carlo simulations for each code used with the algorithm. With an optimized correction term, EMSA manages to perform very close to QSPA [33]. An alternative, reduced complexity approach to forward-backward matrix computation for check node operations of EMSA has been suggested in the literature [34] along with a hardware architecture.

A somewhat different approach to reducing complexity of QSPA, 'min-max decoding', is presented in [28]. An algorithm that operates in LLR domain, it uses a slightly different form of LLRs to (2.10). To better understand the motive behind this different form of LLRs, it would first be beneficial to consider how data is transmitted in practice.

It is well known that in wireless communication, *modulation* is employed to transmit data [35], [36]. Simply put, digital modulation is changing different characteristics of a signal according to the data that has to be transmitted. This signal is usually referred to as the *carrier waveform*, and the characteristics that are changed could be any of amplitude, phase and frequency. For a more in-depth discussion on modulation, please refer [36].

The number of different ways in which the characteristics of the carrier waveform may

be changed is known as the *modulation order*, and each change is said to create a new *modulation symbol*. In most modulation schemes, these different symbols could be represented as points in a two dimensional plane, which is referred to as the *modulation constellation* [36]. The distance from the origin of the plane to such a point represents the amplitude for that symbol, and the angle, measured with respect to the horizontal axis, represents the phase. For any decoding scheme, the computation of initial channel estimates is directly dependent on the modulation scheme.

Figure 2.2 shows the 8-PSK modulation constellation, where the eight symbols are shown as red crosses. While the transmitter will only transmit one of these symbols, it may get corrupted due to channel imperfections, and what is received may not be a valid symbol. For example, in Figure 2.2, the black cross represents the received signal. The probability of a symbol being the one that was actually transmitted is inversely proportional to the distance between that symbol and this received signal. In min-max decoding, the traditional form of LLRs, given in (2.10), is changed in such a way that this relationship between probabilities and distances is somewhat preserved [28]. It should be noted though that this analogy is only valid when the size of the field used for constructing the NB-LDPC code is equal to the order of the modulation scheme used.

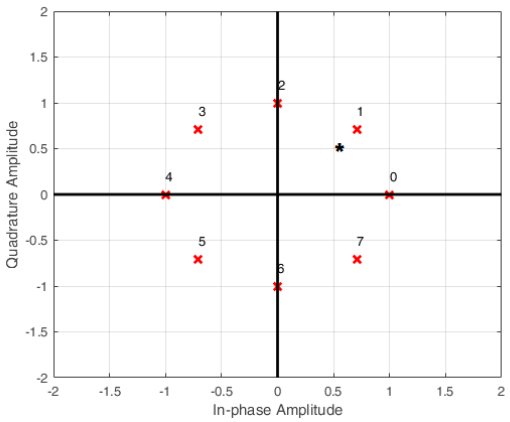

Figure 2.2: Modulation Constellation for 8-PSK

Instead of using a *fixed* symbol for computing the ratio of probabilities, the most likely symbol, which can change for each PMF and each iteration, is used in min-max decoding. With this form of LLRs, given that the modulation scheme used in transmission and the finite field used for code construction are of same size, each value in the message vector becomes analogous to the distance between that particular symbol and the most likely one [28]. New form is given in (2.21), where β^* and γ^* represent the most likely symbols. $\underline{\mathbb{R}}_{n,m}^{(k)}$ is the message from variable node n to

check node m, and $\underline{\mathbb{P}}_{m,n}^{(k)}$ from m to n, both in k'th decoder iteration.

$$\underline{\mathbb{R}}_{n,m}^{(k)}(\gamma) = \ln\{\frac{Pr(\gamma^*)}{Pr(\gamma)}\} \qquad \underline{\mathbb{P}}_{m,n}^{(k)}(\beta) = \ln\{\frac{Pr(\beta^*)}{Pr(\beta)}\} \tag{2.21}$$

Computation of message vector $\underline{\mathbb{P}}_{m,n}^{(k)}$ then becomes an evaluation of distances between different symbols and the most probable symbol for node n. This most probable symbol is the symbol β^* for which the following configuration $\underline{c}^*$ is in the set C_{n,β^*}^m (see (2.3)). γ_i^* represents the most likely symbol for variable node i according to the PMF estimate received from that node, and h_i represents the label of the edge between check node m and variable node i.

$$\underline{c}^* = (\gamma_0^*, \gamma_1^*, \ldots, \gamma_{n-1}^*, \gamma_{n+1}^*, \ldots, \gamma_{d_c-1}^*); \sum_{i \in N(m), i \neq n} h_i \cdot \gamma_i^* = -(h_n \cdot \beta_n^*) \tag{2.22}$$

In min-max algorithm, distance between β^* and any symbol β is evaluated as the distance between $\underline{c}^*$ and the set of all possible configurations for symbol β, from the set $C_{n,\beta}^m$ (see (2.3)) [28]. The distance between a specific point and another set of points can be considered as the minimum distance between that point and any point in the set. Distance may be evaluated using any of the p-norms [28]. For some configuration $\underline{c} = (\gamma_0, \gamma_1, \ldots, \gamma_{n-1}, \gamma_{n+1}, \ldots, \gamma_{d_c-1}) \in C_{n,\beta}^m$, distance to $\underline{c}^*$ is computed as;

$$dist(\underline{c}^*, \underline{c}) = \|dist(\gamma_0^*, \gamma_0), \ldots, dist(\gamma_{n-1}^*, \gamma_{n-1}), dist(\gamma_{n+1}^*, \gamma_{n+1}), \ldots, dist(\gamma_{d_c-1}^*, \gamma_{d_c-1})\|_p \tag{2.23}$$

But with the new form of LLRs, $dist(\gamma_i^*, \gamma_i)$ is simply γ_i, which simplifies (2.23) to the following form.

$$dist(\underline{c}^*, \underline{c}) = \|\gamma_0, \ldots, \gamma_{n-1}, \gamma_{n+1}, \ldots, \gamma_{d_c-1}\|_p \tag{2.24}$$

Min-max decoding uses *infinity norm* to compute (2.24), which is advantageous in terms of complexity, since only comparisons are required. Also, it has been noted in [28] that using infinity norm can help reduce the over-estimation error of check node estimates in the approximated BP-based decoders, such as in binary min-sum decoding [31]. With infinity norm, (2.24) changes to;

$$dist(\underline{c}^*, \underline{c}) = \max(\underline{\mathbb{R}}_{0,m}^{(k-1)}(\gamma_0), \ldots, \underline{\mathbb{R}}_{n-1,m}^{(k-1)}(\gamma_{n-1}), \underline{\mathbb{R}}_{n+1,m}^{(k-1)}(\gamma_{n+1}), \ldots, \underline{\mathbb{R}}_{d_c-1,m}^{(k-1)}(\gamma_{d_c-1})) \tag{2.25}$$

Check node estimate for variable node n, $\underline{\mathbb{P}}_{m,n}^{(k)}$, then takes the following form.

$$\underline{\mathbb{P}}_{m,n}^{(k)}(\beta) = \min_{\underline{c} \in C_{n,\beta}^m} (\max_{\gamma_i \in \underline{c}} \underline{\mathbb{R}}_{i,m}^{(k-1)}(\gamma_i)) \tag{2.26}$$

Forward-backward matrix approach has been suggested in [28] for completing (2.26) for all variable nodes in the neighborhood. Since the size of configuration sets $C^m_{n,\beta}$ (in (2.3)) are not reduced with the approximations used in min-max algorithm, its complexity order stays at $\mathcal{O}(q^2)$, same as LLR-QSPA. But it is evident from (2.26) that only comparisons are required for check node computations, which makes the min-max algorithm significantly more efficient at hardware level. The new form of the LLRs of (2.21) requires some additional computations to be carried out at variable nodes [28], but these can be completed with only comparisons and subtractions, and thus their impact on complexity is minimal.

2.3.4 Performance and Decoding Complexity

Any decoding algorithm for NB-LDPC codes is evaluated on two main criteria; decoding performance and complexity. Performance of the two major versions of QSPA, and two of its simplifications, max-log-SPA [26] and min-max decoding [28], with two NB-LDPC codes, are given in Figure 2.3 and Figure 2.4. Max-log-SPA is a special case of EMSA, where $n_m = q$ and $n_c = d_c - 1$, the highest possible values for the two parameters [33], making it the most performance-friendly configuration. Therefore, no other configuration of EMSA is considered in the figures.

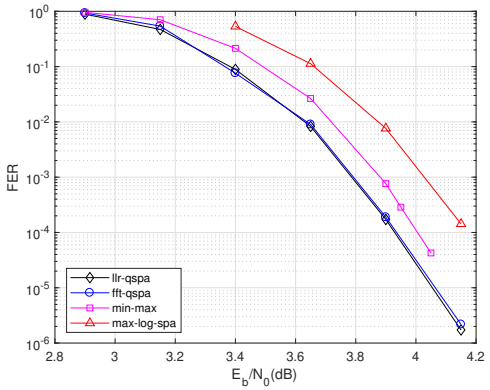

Figure 2.3: Perf. with a (1000, 861) code over $\mathbb{F}_{2^3}$

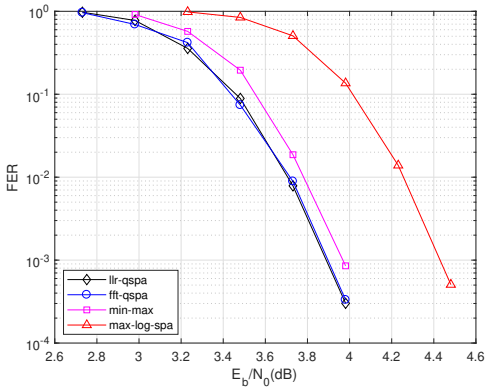

Figure 2.4: Perf. with a (361, 288) code over $\mathbb{F}_{2^6}$

The decoding performances are compared in terms of their frame error rates (FER). A rate 0.861, codeword length 1000 NB-LDPC code over $\mathbb{F}_{2^3}$ is used in Figure 2.3, and a rate 0.8, codeword length 361 code over $\mathbb{F}_{2^6}$ is used in Figure 2.4. Maximum number of decoding iterations is set to 50 for all algorithms. Performance of the two versions of QSPA, LLR-QSPA and FFT-

QSPA, are almost identical in both cases. Also, min-max algorithm performs very close to QSPA, with a gap less than 0.1dB. Max-log-SPA seems to suffer from a significant performance degradation though. Its gap with QSPA is 0.3dB for the code over $\mathbb{F}_{2^3}$, and close to 0.5dB for the code over $\mathbb{F}_{2^6}$. A correction factor, as suggested in [33], can help in reducing these gaps.

As discussed at the beginning of Section 2.2, decoding NB-LDPC codes consists of three major steps; initialization, check node operations, and variable node operations. Initialization is where the symbol probability values are computed for each variable node, based on channel observations. Therefore, this step is almost the same for every decoding algorithm. Decoding complexity is dominated by check node operations, while variable node operations also have a significant impact. Table 2.1 and Table 2.2 list complexities of these two steps for the two variants of QSPA, EMSA, and the min-max algorithm. Since the type of operation is as relevant for decoding complexity as the number of operations, four types are considered in the tables; comparisons ($\mathcal{C}$), additions/subtractions ($\mathcal{A}/\mathcal{S}$), multiplications/divisions ($\mathcal{M}/\mathcal{D}$), and table look-ups ($\mathcal{T}$). Complexities listed are for decoding a code over $\mathbb{F}_q$, and are for a single node, in a single iteration. d_v and d_c represent, respectively, the check degree and the variable degree of the code.

Algorithm	$\mathcal{C}$	$\mathcal{A}/\mathcal{S}$	$\mathcal{M}/\mathcal{D}$	$\mathcal{T}$
LLR-QSPA	$3d_c \cdot q^2$	$6d_c \cdot q^2$	-	$3d_c \cdot q^2$
FFT-QSPA	-	$2d_c \cdot q \cdot \log q$	$2d_c \cdot q$	$2d_c \cdot q$
EMSA	$3d_c \cdot n_m \cdot q$	$3d_c \cdot n_m \cdot q$	-	-
Min-max	$6d_c \cdot q^2$	-	-	-

Table 2.1: Check Node Complexity of QSPA Variants

Algorithm	$\mathcal{C}$	$\mathcal{A}/\mathcal{S}$	$\mathcal{M}/\mathcal{D}$	$\mathcal{T}$
LLR-QSPA	q	$2d_v \cdot q$	-	-
FFT-QSPA	q	$2d_v \cdot q$	-	-
EMSA	q	$2d_v \cdot q$	-	-
Min-max	$(d_v + 1) \cdot q$	$3d_v \cdot q$	-	-

Table 2.2: Variable Node Complexity of QSPA Variants

The two simplifications of QSPA, EMSA and min-max algorithm, only require comparisons and additions for decoding. Multiplications are necessary only for check node operations of FFT-QSPA, while table look-ups are required for $\max^*$ operation of LLR-QSPA, and converting between log and probability domains in FFT-QSPA. When considering the number and the type of operations, EMSA seems the least complex of the four, but when used in practice, a correction term

is necessary to ensure that the performance loss relative to QSPA does not become very significant [33]. This increases the number of additions required. Min-max algorithm has the highest complexity at variable nodes, due to the special form of LLRs it uses (see (2.21)), but as the increase is mostly in the number of comparisons, impact on overall complexity is minimal. Also taking in to account the performance of these algorithms (see Figure 2.3 and Figure 2.4), it can be determined that the min-max algorithm offers the best performance-complexity trade-off for decoding NB-LDPC codes, since, in comparison to any version of QSPA, its performance loss is minimal and the complexity gain is significant. The algorithm is also well suited for hardware implementations.

2.4 Majority Logic Decoding

Majority logic decoding (MLgD) is an approach that has successfully been used to decode different classes of channel codes such as Reed-Muller codes and repetition codes. More recently, a number of different decoding algorithms based on this approach has been suggested in the literature for NB-LDPC codes [24], [37], [38], [39], [23]. These are iterative algorithms, similar to QSPA and its simplifications, but require only a fraction of the operations necessary in those. Also, the operations required are either finite field computations, or integer additions. Therefore, MLgD algorithms are several times faster than QSPA variants. But this complexity gain comes at the cost of severe performance degradation, especially for codes constructed in a random fashion (see subsection 2.1.1), which generally have low variable degrees [39]. These algorithms are intended to be used with high variable degree codes, generated with structured construction methods similar to [20], [21], [22],[23]. Even with those codes, performance of the MLgD algorithms is significantly worse than that of QSPA variants, but their extremely low complexity allows NB-LDPC codes to be used in resource-constrained settings. In such environments, NB-LDPC codes with MLgD often outperform the alternatives such as Reed-Solomon codes (with Berlekamp-Massey decoding) [24], [23]. In the following, we give a brief overview of MLgD of a code defined over $\mathbb{F}_q$.

A variable node in an MLgD algorithm is initialized with an integer vector of length q, where each value represents the *reliability* of the corresponding symbol being the correct estimate for that node [24]. If some soft information is available, reliability of each symbol will be some integer. Size of these integers, in number of bits, depends on the capabilities of the hardware. Since MLgD is almost exclusively used in resource-constrained systems, integers used would often be only a few bits wide. In certain cases, no soft information would be available. Then, in the reliability vector, value for the hard decision symbol would be set to one, and others to zero. In many

algorithms, this initial reliability vector $\underline{\mathfrak{C}}_n$ is scaled by some factor δ for improved performance [24], [23]. Optimum value of δ has to be found through Monte-Carlo simulations for each code. The following equation represents initializing variable node n, where $\underline{\mathfrak{V}}_n^{(0)}$ represents the symbol reliability vector of node n after initialization.

$$\underline{\mathfrak{V}}_n^{(0)} = \delta \cdot \underline{\mathfrak{C}}_n \tag{2.27}$$

Unlike in QSPA variants, where a message is an estimate of the associated PMF, in MLgD, a message between two neighboring nodes is the most likely symbol for the corresponding variable node. These simplified messages reduce check node operations to a set of finite field computations, which can be carried out efficiently with look-up tables [24]. Using the messages $\mathfrak{r}_{i,m}^{(k)}$ received from neighboring variable nodes, a check node m computes the most likely symbol for node n, $\mathfrak{p}_{m,n}^{(k)}$, as in (2.28), where h_i represents the label of the edge between nodes m and i, and h_i^{-1} is the multiplicative inverse of h_i.

$$\mathfrak{p}_{m,n}^{(k)} = h_n^{-1} \cdot \Big(\sum_{i \in N(m); i \neq n} h_i \cdot \mathfrak{r}_{i,m}^{(k-1)} \Big) \tag{2.28}$$

When a check node sends a symbol to a neighboring variable node, it is often referred to as a *vote* for that particular symbol being the correct estimate for that node. Variable nodes *accumulate* these votes throughout iterations, combined with channel estimates, in symbol reliability vectors $\underline{\mathfrak{V}}_i^{(k)}$ [24]. Symbol with the highest reliability in $\underline{\mathfrak{V}}_i^{(k)}$ is chosen as both the tentative decision, and the message to all neighboring check nodes in the next iteration. This goes against the extrinsic principle of message passing, but is advantageous in terms of decoding complexity. Reliability vector update at a variable node n is represented by the following equation, where $\mathfrak{u}_\mathfrak{i}$ is the reliability update for the vote from neighboring check node i.

$$\underline{\mathfrak{V}}_n^{(k)}(\beta) = \underline{\mathfrak{V}}_n^{(k-1)}(\beta) + \sum_{i \in M(n), \mathfrak{p}_{i,n}^{(k)} = \beta} \mathfrak{u}_\mathfrak{i} \tag{2.29}$$

MLgD algorithms can be divided into two main categories, based on the availability of soft channel information during initialization. Those that use soft information for decoding are referred to as *soft reliability based* algorithms, and ones that have to rely on hard decisions from the channel are called *hard reliability based* algorithms. Majority of algorithms in both categories are modifications of either *iterative soft reliability based* (ISRB) MLgD or *iterative hard reliability based* (IHRB) MLgD, both proposed in [24].

Apart from information available for initialization, a main difference between soft reliability based and hard reliability based algorithms is in the way reliability vectors at variable nodes are updated using check node estimates. In hard reliability based algorithms, each *vote* increases the corresponding symbol's reliability value by one ($\mathfrak{u}_i$ in (2.29) is 1 for all i). But in soft reliability based algorithms, a vote increases the reliability by a value ≥ 1, which depends on the edge the vote was sent through. This value is referred to as *edge reliability* [24] $\mathfrak{e}_{m,n}$ ($\mathfrak{u}_i = \mathfrak{e}_{i,n}$ in (2.29)), and is computed for an edge between check node m and variable node n as follows, during initialization of the decoder.

$$\mathfrak{e}_{m,n} = \min_{i \in N(m), i \neq n} \left(\max_{\beta \in \mathbb{F}_q} \underline{\mathfrak{C}}_i(\beta) \right) \tag{2.30}$$

Simulation results in the literature show a significant gap between performance of soft reliability based and hard reliability based MLgD algorithms. But out of the two, hard reliability based ones are much better suited for resource-constrained systems, since additions carried out in decoding are always increments of one. [37] considers using soft information only during initialization of IHRB-MLgD. This version is popularly known as *enhanced* IHRB-MLgD (EIHRB), and it introduces another modification. During variable node operations, algorithm computes a different estimate for each check node, by removing its most current vote from the vector of reliability values. Initial soft information, and *pseudo-extrinsic* variable node messages, manage to reduce the gap with ISRB-MLgD significantly [37].

[23] proposes two algorithms similar to IHRB-MLgD and EIHRB-MLgD, *generalized bit flipping* (GBF), and *modified generalized bit flipping* (MGBF). These algorithms use truly extrinsic messages, but, at each variable node, they need vectors of accumulated reliability values to be maintained *for each neighboring check node*, which significantly increases memory requirements.

In all MLgD algorithms discussed so far, symbol reliability values are accumulated, which can cause issues such as integer overflow, particularly since the algorithms are intended for systems with limited resources. This is especially a problem in soft reliability based algorithms, where reliability updates are generally in values > 1. In ISRB-MLgD, any value in risk of integer overflow is *clipped*, and the reliability vector is normalized to reflect the clipping [24]. [38] proposes two variants of ISRB-MLgD and EIHRB-MLgD, *improved* ISRB-MLgD (IISRB) and *improved* EIHRB-MLGD (IEIHRB), that do not accumulate reliability values. Instead, the tentative decision and the estimate in each iteration is only based on the initial reliability vector, and votes received in that iteration.

With each neighboring check node voting for a single symbol, all symbol reliability values

of a variable node would not be updated in every iteration. Even with high variable degree codes, this will be possible only if the code is over a small alphabet. In the literature, this is considered the major reason for the high error-floor observed with MLgD [39]. For fixing this issue, [39] proposes *bit reliability based* (BRB) MLgD, where reliability values of the constituent bits of a symbol are used in place of symbol reliability values. Results show that while this improves the error floor, it can lead to a performance loss, in comparison to ISRB-MLgD, in the waterfall region of decoding [39].

FER performance of five different MLgD algorithms, ISRB-MLgD, IHRB-MLgD, EIHRB-MLgD, IISRB-MLgD, and IEIHRB-MLgD, and LLR-QSPA, with two different NB-LDPC codes, are given in following figures. Figure 2.5 shows performance of a (63, 37) code over $\mathbb{F}_{2^3}$, constructed using the structured method proposed in [23]. This code has a relatively large variable degree of 8. Figure 2.6 shows performance of a general NB-LDPC code defined over $\mathbb{F}_{2^4}$, of rate $\frac{2}{3}$, symbol length 384, and average variable degree 3.375. In both cases, the maximum number of decoding iterations is set to 50 for all algorithms, including LLR-QSPA. For soft reliability based MLgD algorithms (ISRB-, and IISRB-MLgD), 12-bit quantization is used, and for ones that use soft information only in initialization (EIHRB-, and IEIHRB-MLgD), 3-bit quantization is used. Scaling factors used in all MLgD algorithms have been optimized for each code through Monte-Carlo simulations.

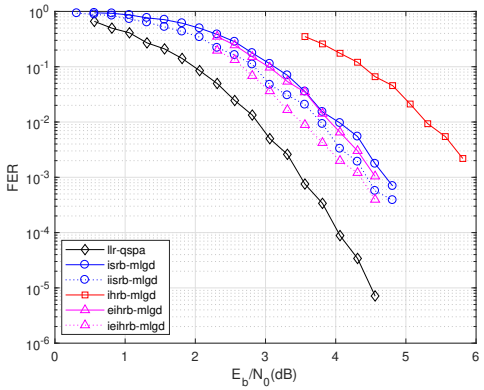

Figure 2.5: Perf. of a (63, 37) code over $\mathbb{F}_{2^3}$

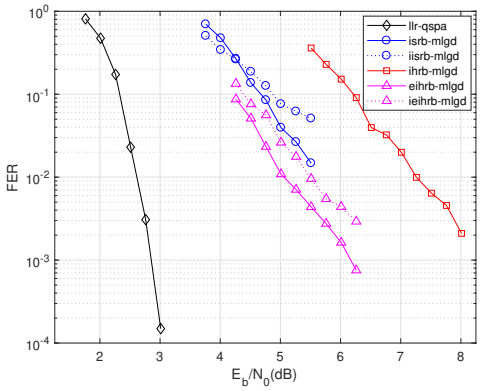

Figure 2.6: Perf. of a (384, 256) code over $\mathbb{F}_{2^4}$

Performance of MLgD in the two cases are significantly different from each other. Figure 2.5 shows that with a high variable degree code, most MLgD algorithms manage to perform within 1dB of QSPA, with the gap being the least for IEIHRB-MLgD, around 0.6dB. Considering

the quite significant gains of MLgD in decoding complexity and resource requirements, this level of performance loss should be acceptable in most cases. But, as can be seen in Figure 2.6, results are quite different with the low variable degree code, where a gap $>$ 3dB with QSPA is observable for all MLgD algorithms. Additionally, FER curves for these algorithms show early signs of error-floor. These observations suggest that MLgD is not a viable alternative to QSPA if the code being used is not from a construction that guarantees high variable degrees. As noted earlier, with each connected check node *voting* for just one symbol, only a few symbol reliability values are updated in each iteration when the variable degree is low. This can cause the decoder to be *stuck* in one state, especially if there are erroneous channel estimates with high reliability values.

Out of all MLgD algorithms, IEIHRB-MLgD is the best performing one in Figure 2.5, closely followed by IISRB-MLgD. These improved algorithms outperform the original versions, EIHRB-MLgD and ISRB-MLgD, by around 0.1-0.2dB. But in Figure 2.6, improved versions perform worse than the original ones, a phenomenon also observed in [39] for low variable degree codes. In both cases, enhanced hard reliability based algorithms manage to outperform soft reliability based ones, even though higher levels of quantization were used for the second type. This suggests that the significant performance gap observed between IHRB-MLgD and remaining MLgD algorithms is mostly due to the loss of soft information during initialization. Results suggest that if NB-LDPC codes are to be used in a resource-constrained setting, then IEIHRB-MLgD with a high variable degree code is the best choice. It marginally outperforms other MLgD algorithms, and is slightly better suited for hardware implementations than those, as reliability values are not accumulated.

Chapter 3

Expansions of Non-binary Factor Graphs

As was established in previous chapters, non-binary LDPC (NB-LDPC) codes offer exceptional error-correcting performances with Q-ary sum-product algorithm (QSPA), but decoding complexity is at a level that does not encourage practical usage. For a code over $\mathbb{F}_q$, QSPA is of $\mathcal{O}(q^2)$ complexity. This is particularly unfortunate since best performing NB-LDPC codes are over large fields, for which decoding complexity would be at very high levels. Furthermore, QSPA requires a lot of hardware resources, especially memory, and therefore, it can only be practically implemented in systems where that kind of resources are available. Due to these reasons, performance gains of NB-LDPC codes are yet to be widely achieved in practice.

Many researchers in the field have attempted to simplify QSPA to reduce its complexity, while ensuring that performance losses are minimal. A few approaches of the kind were discussed in some detail in the previous chapter, such as the FFT-based implementation of QSPA [25], extended min-sum algorithm [33], and min-max decoding [28]. Efficient hardware implementations have also been suggested for some of these [34]. But the performance-complexity trade-offs offered by sub-optimal decoding algorithms do not suit all applications that may benefit from using NB-LDPC codes. Also, for significant complexity gains, performance loss in comparison to QSPA can be significant.

An alternative strategy to reduce decoding complexity of NB-LDPC codes is to use smaller fields in decoding than the ones used in code construction. For that to be possible, a code should first be suitably represented over a smaller field. Since an NB-LDPC code is usually defined by its

parity-check matrix, and the associated factor graph, a suitable representation over a smaller field would be an *expansion* of the parity-check matrix and the factor graph. Any variant of QSPA can theoretically be used with the expanded graph. This is particularly attractive, as decoding complexity for a code over $\mathbb{F}_q$ could then potentially be reduced to $\mathcal{O}(r^2)$, where $r << q$.

In the literature, most of the focus in decoding based on expanded graphs has been devoted to using binary expansions [40], [41], [42], which is understandable since that leads to the highest complexity reduction. In the following sections, we first provide brief overviews of some of these binary expansions. Then, based on a novel approach, we propose a more general expansion, which allows a factor graph over $\mathbb{F}_q$ to be expanded into a graph over any subfield of $\mathbb{F}_q$. If the field is of characteristic 2, then the set of possible expansions would also include the binary one. This chapter focuses only on the expansions, while decoding based on expanded graphs is planned for the next.

3.1 Existing Binary Expansions

3.1.1 Binary Image

A straight-forward approach to convert a parity-check matrix over $\mathbb{F}_{2^r}$ to a binary one is to use the matrix representation of $\mathbb{F}_{2^r}$, introduced in subsection 2.4.3. This representation, which is based on the companion matrix of the primitive polynomial of $\mathbb{F}_{2^r}$, uses an $r \times r$ binary matrix for each element in the field. Thus, if the original matrix was of dimensions $m \times n$, its binary expansion would be a $rm \times rn$ matrix. This is referred to as the *binary image* of the original $\mathbb{F}_{2^r}$ matrix.

In the following, as an example, we present the binary image of a matrix over $\mathbb{F}_{2^3}$.

Example 3.1. *Consider the following* 3×4 *matrix* H*, over* $\mathbb{F}_{2^3}$*, where* α *denotes a primitive element.*

$$H = \begin{bmatrix} 0 & \alpha & \alpha^5 & 0 \\ 1 & 0 & \alpha^3 & \alpha^6 \\ \alpha^2 & \alpha^4 & 0 & 1 \end{bmatrix} \tag{3.1}$$

A primitive polynomial for $\mathbb{F}_{2^3}$ *is* $\Pi(x) = x^3 + x + 1$*, over* $\mathbb{F}_2$*. Based on definition 2.13, the corresponding companion matrix is;*

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.2}$$

Matrix A has 7 distinct powers, with A^7 being the 3×3 identity matrix. These matrices, together with the all-zero 3×3 matrix, form a field isomorphic to $\mathbb{F}_{2^3}$. Following table presents this representation, together with the power and vector representations of $\mathbb{F}_{2^3}$.

0	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}$	α^3	$\begin{bmatrix}1\\1\\0\end{bmatrix}$	$\begin{bmatrix}1&0&1\\1&1&1\\0&1&1\end{bmatrix}$
1	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$	α^4	$\begin{bmatrix}0\\1\\1\end{bmatrix}$	$\begin{bmatrix}0&1&1\\1&1&0\\1&1&1\end{bmatrix}$
α	$\begin{bmatrix}0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0&0&1\\1&0&1\\0&1&0\end{bmatrix}$	α^5	$\begin{bmatrix}1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1&1&1\\1&0&0\\1&1&0\end{bmatrix}$
α^2	$\begin{bmatrix}0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0&1&0\\0&1&1\\1&0&1\end{bmatrix}$	α^6	$\begin{bmatrix}1\\0\\1\end{bmatrix}$	$\begin{bmatrix}1&1&0\\0&0&1\\1&0&0\end{bmatrix}$

Table 3.1: Representations of $\mathbb{F}_{2^3}$

In the binary image of H, H_b, each symbol is replaced with the corresponding 3×3 matrix above. This results in

$$H_b = \left[\begin{array}{ccc:ccc:ccc:ccc} 0&0&0&0&0&1&1&1&1&0&0&0\\ 0&0&0&1&0&1&1&0&0&0&0&0\\ 0&0&0&0&1&0&1&1&0&0&0&0\\ \hdashline 1&0&0&0&0&0&1&0&1&1&1&0\\ 0&1&0&0&0&0&1&1&1&0&0&1\\ 0&0&1&0&0&0&0&1&1&1&0&0\\ \hdashline 0&1&0&0&1&1&0&0&0&1&0&0\\ 0&1&1&1&1&0&0&0&0&0&1&0\\ 1&0&1&1&1&1&0&0&0&0&0&1 \end{array}\right] \tag{3.3}$$

Binary image of a parity-check matrix (PCM), constructed as in Example 3.1, fully represents all the constraints in the original matrix. When it is used for decoding, codewords also have to be considered in their binary forms. This conversion is through the vector representation of the field, the concept of which was introduced in the previous Chapter. In that representation, a symbol in $\mathbb{F}_{2^r}$ is represented with a length r binary vector, or r bits, the number of bits equivalent to the information contained in one such symbol. When the channel in use has binary inputs, r bits will be transmitted per non-binary symbol.

From H and H_b in example 3.1, it becomes clear that the factor graph representation of the binary image uses r binary nodes for each non-binary node of the original graph. Connections between them are determined by the binary image, and the degree of any binary node will be equal to or larger than the degree of the corresponding non-binary node. Binary variable nodes that replace a non-binary variable node represent the constituent bits of the associated $\mathbb{F}_{2^r}$ symbol, whereas binary check nodes compute their probability estimates. Each bit transmitted when using a binary input channel is thus represented with a variable node in the factor graph of the binary image.

Although the binary image fully represents a PCM, its structure would not suit iterative decoding in most cases, and decoding schemes that use this approach, for example [43], [41], and [42], do not directly use it. Main reason for this is the large number of short cycles present in the binary image in comparison with the original matrix. These short cycles can create stopping sets that negatively impact iterative decoding [42]. This increase in short cycles is partly due to the matrix representation of some symbols themselves containing such cycles. Further, the binary expansion of a *regular* non-binary matrix may not have the same property. Example 3.1 above provides evidence of these. There, matrix H, which is over $\mathbb{F}_{2^3}$, has a single four-cycle, but its binary image, H_b, has several. Also, while H is of regular column weight two, H_b has column weights ranging from two to five.

Different decoding schemes based on the binary image use different strategies to limit the impact of short cycles and associated stopping sets. A brief overview of these strategies are presented in the next chapter.

3.1.2 Extended Binary Representation

Although it is fairly simple, binary image representation of the PCM of an NB-LDPC code is *too dense* to be used with iterative decoding algorithms without significant modifications. *Extended binary representation*, first proposed in [40], is an alternative approach that generates a *sparse* binary matrix better suited for iterative decoding.

Extended binary representation of a $m \times n$ matrix over $\mathbb{F}_{2^r}$ is a binary matrix of dimensions $((2^r-1)m + (2^r-1-r)n) \times (2^r-1)n$, significantly larger than the binary image, which is $rm \times rn$. This expansion is achieved through alternative matrix and vector representations of the field. As discussed in subsection 2.4.3, traditionally, $\beta \in \mathbb{F}_{2^r}$ is represented with a $r \times r$ binary matrix, or a binary vector of length r. But in the alternative representations, proposed in [40], same element will be represented with a $(2^r-1) \times (2^r-1)$ matrix and a vector of length (2^r-1). In the

following, we present these representations, which we refer to as the *extended matrix representation* and the *extended vector representation*. h_β and v_β denote, respectively, the traditional matrix and vector representations of $\beta \in \mathbb{F}_{2^r}$, while $\widetilde{h}_\beta$ and $\widetilde{v}_\beta$ denote the new ones.

Definition 3.1. ***Extended matrix representation*** *of a symbol $\beta \in \mathbb{F}_{2^r}$ is the $(2^r-1)\times(2^r-1)$ matrix $\widetilde{h}_\beta$ defined based on the traditional matrix representation of β, h_β, as follows.*

$$\widetilde{h}_\beta(i,j) = \begin{cases} 1; & if \;\; i_b \cdot h_\beta = j_b \\ 0; & otherwise \end{cases}, \qquad 1 \le i,j \le 2^r-1 \tag{3.4}$$

The binary representations of integers i and j, with the left most bit the least significant bit, are denoted with i_b and j_b respectively.

For any $\beta \neq 0$, h_β is an invertible matrix, and for such a matrix, $\widetilde{h}_\beta$ would be a permutation of the $(2^r-1)\times(2^r-1)$ identity matrix, as proven in [44]. $\widetilde{h}_0$ is the $(2^r-1)\times(2^r-1)$ all-zero matrix. Therefore, no cycle can exist in the extended representation matrix $\widetilde{h}_\beta$ for any β, a significant advantage over the traditional representation.

Definition 3.2. *Let $\mathbb{X}_{2^r}$ be the $r \times (2^r-1)$ matrix, where columns are the binary representations of integers $i = 1, 2, \ldots, (2^r-1)$, or $\mathbb{X}_{2^r} = [1_b, 2_b, \ldots, (2^r-1)_b]$. Then the* ***extended vector representation*** *of a symbol $\beta \in \mathbb{F}_{2^r}$ is the vector $\widetilde{v}_\beta$ defined as follows.*

$$\widetilde{v}_\beta = v_\beta \cdot \mathbb{X}_{2^r} \tag{3.5}$$

$\mathbb{X}_{2^r}$ in the above definition is actually the PCM of the $(2^r-1, 2^r-1-r)$ binary Hamming code [44], [45]. So the extended vector representations become codewords of its dual code, which is the $(2^r-1, r)$ binary simplex code [45]. In other words, bits of the extended vector representation of any symbol should satisfy the constraints of the simplex code generated by $\mathbb{X}_{2^r}$. These additional constraints should be a part of the extended binary representation, and for one codeword symbol, (2^r-1-r) such constraints have to be added.

In the following, extended binary representation of matrix H in example 3.1 is constructed.

Example 3.2. *Matrix $\mathbb{X}_{2^3}$ takes the following form.*

$$\mathbb{X}_{2^3} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.6}$$

Traditional representations of $\mathbb{F}_{2^3}$ presented in Table 3.1, and $\mathbb{X}_{2^3}$ above, are used to derive the extended matrix and vector representations given in Table 3.2. It can be seen that all extended vector representations are codewords of the $(7, 3)$ binary simplex code, and that extended matrix representations are permutations of the 7×7 identity matrix.

0	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix}$
1	$\begin{bmatrix}1\\0\\1\\0\\1\\0\\1\end{bmatrix}$	$\begin{bmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}$
α	$\begin{bmatrix}0\\1\\1\\0\\0\\1\\1\end{bmatrix}$	$\begin{bmatrix}0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\end{bmatrix}$

α^2	$\begin{bmatrix}0\\0\\0\\1\\1\\1\\1\end{bmatrix}$	$\begin{bmatrix}0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\1&0&0&0&0&0&0\end{bmatrix}$
α^3	$\begin{bmatrix}1\\1\\0\\0\\1\\1\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&1&0&0\\0&0&0&0&0&0&1\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&1&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\end{bmatrix}$
α^4	$\begin{bmatrix}0\\1\\1\\1\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&0&1&0\\0&0&1&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&0&1\\1&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&0&0\end{bmatrix}$

α^5	$\begin{bmatrix}1\\1\\0\\1\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&0&0&1\\1&0&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&1&0&0\end{bmatrix}$
α^6	$\begin{bmatrix}1\\0\\1\\1\\0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&0&0&1\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\end{bmatrix}$

Table 3.2: Extended Representations of $\mathbb{F}_{2^3}$

In the extended binary representation of matrix H in (3.1)*, denoted $\widetilde{H}_b$, each symbol is first replaced by its extended matrix representation, from Table 3.2. This operation creates the top half of $\widetilde{H}_b$, in* (3.8)*. As discussed earlier, the constraints that only involve the $(2^3 - 1)$ bits of the extended vector representation of a $\mathbb{F}_{2^3}$ symbol also have to be added to $\widetilde{H}_b$. These constraints are represented with $\mathbb{Y}_{2^3}$ below, which is the PCM of the simplex code represented by $\mathbb{X}_{2^3}$.*

$$\mathbb{Y}_{2^3} = \begin{bmatrix}1&1&1&0&0&0&0\\1&0&0&1&1&0&0\\0&1&0&1&0&1&0\\1&1&0&1&0&0&1\end{bmatrix} \tag{3.7}$$

Since H (in (3.1)*) represents a code of length 4, 4 sets of additional constraints have to be added to $\widetilde{H}_b$. This adds $(2^3 - 1 - 3) \cdot 4 = 16$ new rows to the extended binary representation. These rows form the bottom half of $\widetilde{H}_b$, in* (3.8).

$$\widetilde{H}_b = \left[\begin{array}{cccc}
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0 & 1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0 & 0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0 & 0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1 & 0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
\hline
1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0 & 0\,0\,1\,0\,0\,0\,0 \\
0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1 & 0\,0\,0\,1\,0\,0\,0 \\
0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1 \\
0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0 & 1\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,1\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0 \\
0\,0\,0\,0\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0 \\
\hline
0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,1\,0 & 0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,1\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1 & 1\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,1\,0\,0 \\
0\,0\,1\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,1\,0 \\
1\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,1 \\
\hline
1\,1\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
1\,0\,0\,1\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,1\,0\,1\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
1\,1\,0\,1\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
\hline
0\,0\,0\,0\,0\,0\,0 & 1\,1\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,1\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,1\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 1\,1\,0\,1\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
\hline
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,1\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,1\,1\,0\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,1\,0\,1\,0 & 0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,1\,0\,1\,0\,0\,1 & 0\,0\,0\,0\,0\,0\,0 \\
\hline
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,1\,1\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,0\,0\,1\,1\,0\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,1\,0\,1\,0\,1\,0 \\
0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0\,0 & 1\,1\,0\,1\,0\,0\,1
\end{array}\right] \tag{3.8}$$

Similar to the binary image, the extended binary representation of a PCM can also capture all of its constraints. In decoding, extended vector representation will be used to convert codewords to their binary forms, rather than the traditional vector representation used with the binary image. This results in a binary codeword $(2^r - 1)$ times longer than the non-binary one. Factor graph of the extended binary representation uses $(2^r - 1)$ binary nodes for each non-binary node of the original

graph. It also includes a number of new binary check nodes, $(2^r - 1 - r)$ per non-binary variable node, which are due to the constraints imposed by the extended vector representation.

From example 3.2, it can be observed that although both are binary representations of the same non-binary matrix H, $\widetilde{H}_b$ is a much larger matrix than H_b in example 3.1. This larger size allows $\widetilde{H}_b$ to be sparser than H_b; it is free of 4-cycles, even though one exists in H. In fact, 4-cycles can only exist in the extended binary representation if the 4-cycles in the original matrix satisfy a special condition. This is explored in chapter 4, which discusses graph sub-structures that negatively impact iterative decoding of non-binary codes.

Even though its larger size makes the extended binary representation of a PCM better suited to iterative decoding than the binary image, that becomes a disadvantage when it comes to decoding complexity. A $((2^r - 1)m + (2^r - 1 - r)n) \times (2^r - 1)n$ matrix creates a significantly larger number of variable and check nodes than a $rm \times rn$ matrix, even though they are of lower degrees.

But this slight increase of complexity is not the major disadvantage of the extended binary representation. It uses $(2^r - 1)$ bits for each symbol in $\mathbb{F}_{2^r}$, although one such symbol contains information equivalent to only r bits. As mentioned earlier, when the channel in use is a binary input channel, r bits will be transmitted per non-binary symbol. Thus, data received from the channel would only have information of that many bits. Initialization (see section 2.2) of the rest is the biggest obstacle to overcome when using the extended binary representation for decoding. A few strategies are proposed in the literature for this [44], [45], which will be discussed in the next chapter.

3.2 Subfield Expansion

According to theorem 2.3, a finite field $\mathbb{F}_{p^r}$ contains a subfield $\mathbb{F}_{p^m}$ for each $m \mid r$. In this section, we propose a method to expand a matrix over $\mathbb{F}_{p^r}$ to a matrix over any of those subfields. Any such expansion, could be used with a suitable algorithm for decoding NB-LDPC codes. This presents a range of decoding options, each offering a different performance-complexity trade-off.

In the following, we first present the mathematical concepts and relations that make the expansion possible, and then present the expansion, both from the perspective of the PCM, and the associated factor graph. We then focus on two cases that are of special interest. First is the binary expansion, possible when the fields are of characterisitic 2. Second is for codes constructed over finite fields of the type $\mathbb{F}_{p^{2m}}$, which can be viewed as *quadratic extensions* of $\mathbb{F}_{p^m}$. A PCM

of such a code can be expanded to a matrix over $\mathbb{F}_{p^m}$, which may sometimes be referred to as the *square-root expansion*. While binary expansion offers the highest complexity reduction, square-root expansion offers the best performance. Also, binary expansion our approach produces is the same as the extended binary representation, although they are arrived at in completely different ways.

3.2.1 α-connected Subgroups

Consider a finite field of characteristic p, $\mathbb{F}_{p^r}$, where a primitive element is denoted with α. Let $\mathbb{G}$ be a subgroup of the additive group of the field, $\mathbb{H} = \{\mathbb{F}_{p^r}, \oplus\}$. More subgroups of $\mathbb{H}$, of the same order as $\mathbb{G}$, can be generated using $\mathbb{G}$, according to the following lemma.

Lemma 3.1. *Let $\mathbb{G}$ be a subgroup of $\mathbb{H} = \{\mathbb{F}_{p^r}, \oplus\}$. Then, the set created by multiplying all elements of $\mathbb{G}$ with some $\alpha^i \in \mathbb{F}_{p^r}$ is also a subgroup of the same order.*

Proof. We denote addition and multiplication in $\mathbb{F}_{p^r}$ with, respectively, $\oplus$ and $\otimes$. Let $\mathbb{G}' = \alpha^i \cdot\otimes \mathbb{G}$, where $\cdot\otimes$ represents element-wise multiplication of a set. We will proceed to show that $\mathbb{G}'$ also satisfies the group axioms, presented in definition 2.1.

1. $\mathbb{G}$ is a subgroup under $\oplus$, and therefore, $\forall g_j, g_k \in \mathbb{G}$, $g_j \oplus g_k = g_l \in \mathbb{G}$. From the definition of $\mathbb{G}'$, $(\alpha^i \otimes g_j), (\alpha^i \otimes g_k), (\alpha^i \otimes g_l) \in \mathbb{G}'$. Since $\otimes$ is distributive over $\oplus$ in $\mathbb{F}_{p^r}$;

$$(\alpha^i \otimes g_j) \oplus (\alpha^i \otimes g_k) = \alpha^i \otimes (g_j \oplus g_k) = (\alpha^i \otimes g_l) \in \mathbb{G}' \tag{3.9}$$

Thus, $\mathbb{G}'$ is **closed** under addition $\oplus$.

2. $\oplus$ is the binary operation of $\mathbb{H}$, and therefore, its **associative**.

3. Let ϵ be additive identity in $\mathbb{H}$. Since $\mathbb{G}$ is a subgroup of $\mathbb{H}$, $\epsilon \in \mathbb{G}$. But $\forall \alpha^i \in \mathbb{F}_{p^r},\ \alpha^i \otimes \epsilon = \epsilon$. Therefore, $\epsilon \in \mathbb{G}'$, and $\mathbb{G}'$ contains an **identity element**.

4. Since $\mathbb{G}$ is a subgroup under $\oplus$, $\forall g_j \in \mathbb{G}$, the additive inverse $(-g_j)$ also exists in $\mathbb{G}$. By definition, $(\alpha^i \otimes g_i), (\alpha^i \otimes -g_j) \in \mathbb{G}'$. As $\otimes$ is distributive over $\oplus$;

$$(\alpha^i \otimes g_j) \oplus (\alpha^i \otimes -g_j) = \alpha^i \otimes (g_j \oplus -g_j) = \alpha^i \otimes \epsilon = \epsilon \tag{3.10}$$

Thus, additive inverse of $(\alpha^i \otimes g_j)$ is $(\alpha^i \otimes -g_j)$, and both exist in $\mathbb{G}'$. So **inverse** of any element of $\mathbb{G}'$ exists in $\mathbb{G}'$.

So $\mathbb{G}'$ satisfies all group axioms under $\oplus$. And from the definition of $\mathbb{G}'$, it is clear that it contains the same number of elements as $\mathbb{G}$. Therefore, $\mathbb{G}'$ is an subgroup of $\mathbb{H} = \{\mathbb{F}_{p^r}, \oplus\}$ of the same order as $\mathbb{G}$. □

Since $\mathbb{G}'$ can be obtained from $\mathbb{G}$ by multiplying all its elements with some power of α, and vice versa, we call $\mathbb{G}$ and $\mathbb{G}'$ *α-connected subgroups*. This term is formally defined as follows.

Definition 3.3. *Subgroups $\mathbb{G}$ and $\mathbb{G}'$ of $\mathbb{H} = \{\mathbb{F}_{p^r}, \oplus\}$, where one can be obtained from the other by multiplying all elements by some power of α, are called α-**connected subgroups**.*

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_3$ be subgroups of $\mathbb{H}$. If $\mathbb{G}_1$ and $\mathbb{G}_2$ are α-connected, and so are $\mathbb{G}_2$ and $\mathbb{G}_3$, then clearly $\mathbb{G}_1$ and $\mathbb{G}_3$ are also α-connected. This yields the following definition.

Definition 3.4. *Let $\mathcal{S} = \{\mathbb{G}_1, \ldots, \mathbb{G}_n\}$ be a set of subgroups of $\mathbb{H} = \{\mathbb{F}_{p^r}, \oplus\}$, of the same order. $\mathcal{S}$ is an α-**connected set** if;*

1. *Each $\mathbb{G}_i \in \mathcal{S}$ is α-connected with all other $\mathbb{G}_j \in \mathcal{S}$.*
2. *If $\mathbb{G}_i \in \mathcal{S}$ is α-connected with some $\mathbb{G}'$, then $\mathbb{G}' \in \mathcal{S}$.*

An α-connected set $\mathcal{S}$ can be generated using any $\mathbb{G}_j \in \mathcal{S}$, simply by multiplying $\mathbb{G}_j$ with different powers of α. Following lemma, which is on the cardinality of an α-connected set, also outlines a structured approach to construct such a set using one of the subgroups.

Lemma 3.2. *Let $\mathcal{S}$ be an α-connected set, in $\mathbb{F}_{p^r}$. Cardinality of $\mathcal{S}$ is i^*, where i^* is the smallest positive integer such that $\alpha^{i^*} \cdot \otimes \mathbb{G} = \mathbb{G}$, for some $\mathbb{G} \in \mathcal{S}$.*

Proof. We consider generating $\mathcal{S}$ using one subgroup $\mathbb{G}$. Other subgroups of $\mathcal{S}$ can be found using the following procedure.

1. Start with $i = 1$, and $\mathcal{S} = \{\mathbb{G}\}$
2. Compute $\mathbb{G}' = \alpha^i . \otimes \mathbb{G}$
3. If $\mathbb{G}' = \mathbb{G}$, terminate
4. Otherwise, add $\mathbb{G}'$ to $\mathcal{S}$, increment i by 1, and repeat from step 2

Let i^* be the smallest value of i when the above procedure is terminated, in step 3. Assume that subgroups $\mathbb{G}'$ generated until that point are not all distinct. This means that for some $i_1, i_2 < i^*$;

$$\alpha^{i_1} \cdot \otimes \mathbb{G} = \alpha^{i_2} \cdot \otimes \mathbb{G} \tag{3.11}$$

Without loss of generality, we assume $i_1 < i_2$. Then, multiplying both sides of (3.11) with α^{-i_1}, the multiplicative inverse of α^{i_1};

$$\mathbb{G} = \alpha^{i_2 - i_1} \cdot \otimes \mathbb{G} \tag{3.12}$$

According to (3.12), procedure of generating $\mathcal{S}$ should have been terminated when $i = i_2 - i_1 < i^*$, which is a contradiction. Thus, all subgroups $\mathbb{G}'$ generated for $0 < i < i^*$ should be distinct. Then, also counting $\mathbb{G}$, $\mathcal{S}$ should contain i^* subgroups.

Now we consider the case when $i \geq i^*$. Then i may be written as $i = ki^* + i_\delta$, where k, i_δ are positive integers, with $k > 0$ and $i_\delta < i^*$. Subgroup generated for such a value of i is

$$\mathbb{G}' = \alpha^i \cdot \otimes \mathbb{G} = \alpha^{ki^*+i_\delta} \cdot \otimes \mathbb{G} \tag{3.13}$$

Since $\alpha^{i^*} \cdot \otimes \mathbb{G} = \mathbb{G}$;

$$\mathbb{G}' = \alpha^{i_\delta} \cdot \otimes \mathbb{G} \tag{3.14}$$

$\mathbb{G}'$ is the subgroup generated for $i = i_\delta < i^*$, which is already in $\mathcal{S}$. Thus, any $\mathbb{G}'$ generated for $i \geq i^*$ would already have been generated for some other $i < i^*$.

Above shows that the cardinality of an α-connected set $\mathcal{S}$ is the minimum value of $i > 0$ for which $\mathbb{G} = \alpha^i \cdot \otimes \mathbb{G}$, where $\mathbb{G}$ is an arbitrary member of $\mathcal{S}$. □

A field $\mathbb{F}_{p^r}$ will contain several α-connected sets, of subgroups of different orders. Our focus is on sets where the subgroups are of order p^{r-m}, where $m \mid r$. Out of those, the set of *minimum cardinality* is of special interest, since such sets are instrumental in deriving the subfield expansion. This minimum cardinality, which depends on values r and m, is discussed in the following lemma. In the interest of simplicity, here onwards, we will be using $+$ and $\cdot$ in place of $\oplus$ and $\otimes$ to denote addition and multiplication in a field. Also, 0 and 1 would be used as the additive and multiplicative identities, respectively.

Lemma 3.3. *Consider subgroups of $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, of order p^{r-m}, where $m \mid r$. Minimum cardinality of an α-connected set of such subgroups is $\frac{p^r-1}{p^m-1}$.*

Proof. Let $\mathbb{G}$ be an arbitrary subgroup of $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, of order p^{r-m}. As proved in lemma 3.2, cardinality of the α-connected set containing $\mathbb{G}$ is the minimum value of $i > 0$, denoted i^*, for which $\alpha^i \cdot \mathbb{G} = \mathbb{G}$.

We denote with $\mathbb{S}_{i^*}$ the set of elements in $\mathbb{F}_{p^r}$ generated by α^{i^*}. Since $\alpha^{i^*} \cdot \mathbb{G} = \mathbb{G}$, for any $\alpha^{ni^*} \in \mathbb{S}_{i^*}$ where n is an integer, $\alpha^{ni^*} \cdot \mathbb{G} = \mathbb{G}$. Then, from the definition above, i^* is the smallest non-zero power of an element in $\mathbb{S}_{i^*}$.

$\mathbb{S}_{i^*}$ is a subgroup of the multiplicative group of $\mathbb{F}_{p^r}$, $\mathbb{K} = \{\mathbb{F}_{p^r}, \cdot\}$, which is a cyclic group. Therefore, $\mathbb{S}_{i^*}$ should contain the identity element of $\mathbb{K}$, *i.e.* 1. As $\mathbb{K}$ is a cyclic group,

$1 = \alpha^0 = \alpha^{p^r-1}$. Since i^* is the smallest non-zero power in $\mathbb{S}_{i^*}$, the following relationship should hold for some positive integer n' [7].

$$\alpha^{n'i^*} = \alpha^{p^r-1} \Rightarrow n'i^* = p^r - 1 \tag{3.15}$$

n' is actually the order of $\mathbb{S}_{i^*}$, and therefore

$$i^*|\mathbb{S}_{i^*}| = (p^r - 1) \tag{3.16}$$

Since $\alpha^{ni^*} \cdot \mathbb{G} = \mathbb{G}$ for any positive integer n, if $g \in \mathbb{G}$, then $g \cdot \mathbb{S}_{i^*} \subset \mathbb{G}$. We consider two such subsets, for $g_j, g_k \in \mathbb{G}$, where $g_j \neq g_k$ and $g_j, g_k \neq 0$. $g_j \cdot \mathbb{S}_{i^*}$ and $g_k \cdot \mathbb{S}_{i^*}$ are of the same cardinality. If the intersection of these two sets is non-empty, then, for some n_1 and n_2, where $n_1 < n_2$;

$$\alpha^{n_1 i^*} \cdot g_j = \alpha^{n_2 i^*} \cdot g_k \Rightarrow g_j = \alpha^{(n_2-n_1)i^*} \cdot g_k \tag{3.17}$$

Also, since $\alpha^{(n_2-n_1)i^*} \in \mathbb{S}_{i^*}$, $\alpha^{(n_2-n_1)i^*} \cdot \mathbb{S}_{i^*} = \mathbb{S}_{i^*}$. Then;

$$g_j \cdot \mathbb{S}_{i^*} = \alpha^{(n_2-n_1)i^*} \cdot g_k \cdot \mathbb{S}_{i^*} = g_k \cdot \mathbb{S}_{i^*} \tag{3.18}$$

(3.18) shows that when the intersection of $g_j \cdot \mathbb{S}_{i^*}$ and $g_k \cdot \mathbb{S}_{i^*}$ is non-empty, then they are the same set. So $\forall g_j, g_k \in \mathbb{G}$, sets $g_j \cdot \mathbb{S}_{i^*}$ and $g_k \cdot \mathbb{S}_{i^*}$ are either disjoint or the same set. Also, as $\mathbb{G}$ is a subgroup of $\mathbb{H}$, it must contain the additive identity of $\mathbb{F}_{p^r}$, *i.e.* 0, and $0 \cdot \mathbb{S}_{i^*} = \{0\}$. Then, considering the $(p^{r-m} - 1)$ elements of $\mathbb{G}$ that are not equal to 0, the following should hold for some integer n.

$$n|\mathbb{S}_{i^*}| = (p^{r-m} - 1) \tag{3.19}$$

(3.16) and (3.19) show that $|\mathbb{S}_{i^*}|$ is a factor of both $(p^r - 1)$ and $(p^{r-m} - 1)$. (3.16) also shows that i^* and $|\mathbb{S}_{i^*}|$ are inversely proportional. Thus, for $\{i^*\}_{min}$, $|\mathbb{S}_{i^*}|$ should take the largest possible value, which is the *greatest common divisor* of $(p^r - 1)$ and $(p^{r-m} - 1)$.

We observe the following relationships.

$$\begin{aligned}
(p^r - 1) &= (p^m - 1) \sum_{i=0}^{\frac{r}{m}-1} (p^m)^i \\
(p^{r-m} - 1) &= (p^m - 1) \sum_{i=0}^{\frac{r}{m}-2} (p^m)^i \\
\sum_{i=0}^{\frac{r}{m}-1} (p^m)^i &= p^m \sum_{i=0}^{\frac{r}{m}-2} (p^m)^i + 1
\end{aligned} \tag{3.20}$$

Third relationship above shows that there are no common factors of $\sum_{i=0}^{\frac{r}{m}-1}(p^m)^i$ and $\sum_{i=0}^{\frac{r}{m}-2}(p^m)^i$. Then, from the first two relationships, it is possible to conclude that $\gcd(p^r-1, p^{r-m}-1) = (p^m-1)$. For $\{i^*\}_{min}$, we substitute this value in (3.16).

$$\{i^*\}_{min}(p^m-1) = (p^r-1)$$
$$\{i^*\}_{min} = \frac{p^r-1}{p^m-1} \tag{3.21}$$

Thus, the smallest α-connected set containing subgroups of order p^{r-m} of $\mathbb{H}$ has a cardinality of $\frac{p^r-1}{p^m-1}$. □

Subfield expansion is based on the smallest α-connected sets, and while lemma 3.3 presents the cardinality of such a set, it does not reveal a method to generate one, for given values of p, r and m. As an initial step in this direction, we present the following lemma, which outlines a method to construct an α-connected set.

Lemma 3.4. *Let $\mathbb{G}$ be a subgroup of order p^m in $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, and ψ some surjective homomorphism $\psi : \mathbb{H} \to \mathbb{G}$. The kernels of the set of homomorphisms $\psi_i(h) = \psi(\alpha^{-i} \cdot h)$, for $i = \{0, 1, ..., p^r-2\}$, form an α-connected set containing subgroups of order p^{r-m} of $\mathbb{H}$.*

Proof. As discussed in the last chapter, kernel of a homomorphism between two groups is a subgroup of the first. Thus, $\ker(\psi)$ is a subgroup of $\mathbb{H}$. ψ is surjective, and therefore, as a consequence of the first isomorphism theorem (theorem 2.2 in previous chapter) [7];

$$\frac{|\mathbb{H}|}{|\ker(\psi)|} = |\mathbb{G}|$$
$$|\ker(\psi)| = \frac{p^r}{p^m} = p^{r-m} \tag{3.22}$$

So $\ker(\psi)$ is of order p^{r-m}.

Since $\psi_i(h) = \psi(\alpha^{-i} \cdot h)$, for $\alpha^i \cdot h \in \mathbb{H}$, we have;

$$\psi_i(\alpha^i \cdot h) = \psi(\alpha^{-i} \cdot \alpha^i \cdot h) = \psi(h) \tag{3.23}$$

(3.23) shows that, under homomorphism ψ_i, $\alpha^i \cdot h \in \mathbb{H}$ maps to the same element of $\mathbb{G}$ to which h maps under homomorphism ψ. This means that when $h \in \ker(\psi)$, $\alpha^i \cdot h \in \ker(\psi_i)$. Therefore, we have;

$$\ker(\psi_i) = \alpha^i \cdot \ker(\psi) \tag{3.24}$$

(3.24) shows that $\ker(\psi)$ and $\ker(\psi_i)$ are α-connected subgroups of $\mathbb{H}$, of order p^{r-m}. Then, for any $j, k \in \{0, 1, \ldots, p^r - 2\}$, where $j < k$;

$$\ker(\psi_k) = \alpha^{k-j} \cdot \ker(\psi_j) \tag{3.25}$$

So the set of kernels $\mathcal{S} = \{\ker(\psi_0), \ker(\psi_1), ..., \ker(\psi_{p^r-2})\}$ (with possible duplicates removed), satisfy the first condition for an α-connected set, as given in definition 3.4.

Now consider some $\ker(\psi_j) \in \mathcal{S}$. Any subgroup α-connected with $\ker(\psi_j)$ is of the form $\alpha^k \ker(\psi_j)$. This may be expressed as $\alpha^{k+j-j} \ker(\psi_j)$, where addition is modulus $(p^r - 1)$. Then, according to (3.25), $\alpha^{k+j-j} \ker(\psi_j) = \ker(\psi_{k+j})$. $\mathcal{S}$ contains kernels of all homomorphisms ψ_i, and therefore $\ker(\psi k + j) \in \mathcal{S}$. Thus, $\mathcal{S}$ satisfies the second condition in definition 3.4.

Since both conditions of definition 3.4 are satisfied, it is possible to conclude that the set of kernels of homomorphisms ψ_i create an α-connected set containing subgroups of $\mathbb{H}$, of order p^{r-m}. □

The cardinality of an α-connected set generated as in lemma 3.4 depends on the homomorphism ψ. Therefore, to construct the smallest α-connected set, one must find a suitable homomorphism. The homomorphism we use is based on the representation of $\mathbb{F}_{p^r}$ as an *extension* of the subfield $\mathbb{F}_{p^m}$. Following lemma establishes the structure of $\mathbb{F}_{p^m}$ in $\mathbb{F}_{p^r}$.

Lemma 3.5. *Consider $\mathbb{F}_{p^r}$ and let $m \mid r$. Also, $i' = \frac{p^r-1}{p^m-1}$. Let $\mathbb{S}_{i'}$ be the set of elements in the multiplicative group of $\mathbb{F}_{p^r}$ generated by $\alpha^{i'}$. Then, $\mathbb{S}_{i'} \cup \{0\}$, where 0 is the additive identity of $\mathbb{F}_{p^r}$, is the subfield $\mathbb{F}_{p^m}$.*

Proof. Since $m \mid r$, according to theorem 2.3, $\mathbb{F}_{p^r}$ should contain the subfield $\mathbb{F}_{p^m}$. We denote the multiplicative groups of the two fields with $\mathbb{K} = \{\mathbb{F}_{p^r}, \cdot\}$ and $\mathbb{J} = \{\mathbb{F}_{p^m}, \cdot\}$. $\mathbb{J}$ is a subgroup of $\mathbb{K}$, of order $(p^m - 1)$. As $\mathbb{K}$ is a cyclic group, it contains only a single subgroup of a particular order [7].

The set of elements generated by $\alpha^{i'}$, $\mathbb{S}_{i'}$, is a subgroup containing $(p^m - 1)$ elements, and therefore $\mathbb{J} = \mathbb{S}_{i'}$, which leads to the conclusion that $\mathbb{F}_{p^m} = \mathbb{S}_{i'} \cup \{0\}$. □

Homomorphisms we use to generate α-connected sets of minimum cardinality are based on the *polynomial representation* of $\mathbb{F}_{p^r}$ as an extension of $\mathbb{F}_{p^m}$. As briefly discussed in chapter 2, in such a representation, $\alpha^i \in \mathbb{F}_{p^r}$ is represented with a polynomial $\mathfrak{E}_{\alpha^i}(x)$ over $\mathbb{F}_{p^m}$, of degree at most $(\frac{r}{m} - 1)$. For $\alpha^i \in \mathbb{F}_{p^m}$ (when $i = k\frac{p^r-1}{p^m-1}$), this polynomial will be of degree 0, a constant.

Primitive polynomial for this representation, $\Pi(x)$, is an irreducible polynomial over $\mathbb{F}_{p^m}$, of degree $\frac{r}{m}$. We also observe that since $\Pi(x)$ is irreducible, it must have a non-zero constant term.

To illustrate the above concepts, Table 3.3 presents the polynomial representation of $\mathbb{F}_{2^4}$ as an extension of $\mathbb{F}_{2^2}$. α and ω represent the primitive elements of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^4}$ respectively. Primitive polynomial that has been used here is $\Pi(x) = x^2 + x + \omega$, which is irreducible over $\mathbb{F}_{2^2}$. Note that i' in lemma 3.5 evaluates to $\frac{2^4-1}{2^2-1} = 5$ here, and the set of elements generated by α^5, $\{1, \alpha^5, \alpha^{10}\}$, and 0 form the subfield $\mathbb{F}_{2^2}$. Moreover, the polynomial representations of these four elements, as listed in Table 3.3, are of degree zero.

$0:0$	$\alpha^3 : \omega^2 x + \omega$	$\alpha^7 : \omega x + x^2$	$\alpha^{11} : \omega^2 x$
$1:1$	$\alpha^4 : x + 1$	$\alpha^8 : x + \omega^2$	$\alpha^{12} : \omega^2 x + 1$
$\alpha : x$	$\alpha^5 : \omega$	$\alpha^9 : \omega x + x$	$\alpha^{13} : \omega x + 1$
$\alpha^2 : x + \omega$	$\alpha^6 : \omega x$	$\alpha^{10} : \omega^2$	$\alpha^{14} : \omega^2 x + \omega^2$

Table 3.3: Polynomial Representation of $\mathbb{F}_{2^4}$ over $\mathbb{F}_{2^2}$

Based on the polynomial representation of $\mathbb{F}_{p^r}$ over $\mathbb{F}_{p^m}$, we define a homomorphism ψ^* between $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$ and $\mathbb{G} = \{\mathbb{F}_{p^m}, +\}$ that may be used to generate smallest α-connected sets.

Definition 3.5. *Let $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, and $\mathbb{G} = \{\mathbb{F}_{p^m}, +\}$. The homomorphism $\psi^* : \mathbb{H} \to \mathbb{G}$ maps $h \in \mathbb{H}$ to $g \in \mathbb{G}$ when the constant term in the polynomial representation of $\mathbb{F}_{p^r}$ as an extension of $\mathbb{F}_{p^m}$, $\mathfrak{E}_h(x)$, is g.*

Lemma 3.6 in the following proves that using the method proposed in lemma 3.4 with homomorphism ψ^* defined above generates an α-connected set of minimum cardinality.

Lemma 3.6. *Define a set of homomorphisms from $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$ to $\mathbb{G} = \{\mathbb{F}_{p^m}, +\}$, $\psi_i^*(h) = \psi^*(\alpha^{-i}h)$, for $i = \{0, 1, \ldots, p^r - 2\}$, where ψ^* is from definition 3.5. The set of kernels of these homomorphisms, $\mathcal{S}^*$, form an α-connected set containing additive subgroups of order p^{r-m}, of $\mathbb{H}$, that is of the minimum possible cardinality $\frac{p^r-1}{p^m-1}$.*

Proof. According to definition 3.5, $\psi^* : \mathbb{H} \to \mathbb{G}$, and thus, $\ker(\psi^*)$ is a subgroup of order p^{r-m} of $\mathbb{H}$. Using ψ^* as proposed in Lemma 3.4, it is possible to generate an α-connected set of such subgroups. This set, which we denote $\mathcal{S}^*$, contains the kernels of homomorphisms $\psi_i^*(h) = \psi^*(\alpha^{-i}h)$, for $i = \{0, 1, \ldots, p^r - 2\}$.

Consider generating $\mathcal{S}^*$ using $\ker(\psi^*) = \ker(\psi_0^*)$, as in lemma 3.2. Then, the cardinality of $\mathcal{S}^*$ will be equal to the minimum value of j for which $\ker(\psi_i^*) = \alpha^j \cdot \ker(\psi_0^*) = \ker(\psi_0^*)$. We denote this value with j_m.

Let $g_k \in \ker(\psi_0^*)$ and $\alpha^{j_m} \cdot g_k = \gamma_k$, where $k = 1, ..., p^{r-m}$. Also, let polynomial representations over $\mathbb{F}_{p^m}$ of α^{j_m}, g_k and γ_k, be, respectively, $\mathfrak{E}_{\alpha^{j_m}}(x), \mathfrak{E}_{g_k}(x)$ and $\mathfrak{E}_{\gamma_k}(x)$, and $\Pi(x)$ denote the primitive polynomial of the extended representation. Then, these polynomials are related as;

$$\mathfrak{E}_{\alpha^{j_m}}(x) \times \mathfrak{E}_{g_k}(x) = \Pi(x) \times \mathfrak{P}_k(x) + \mathfrak{E}_{\gamma_k}(x) \tag{3.26}$$

where $\mathfrak{P}_k(x)$ is some polynomial over $\mathbb{F}_{p^m}$.

Since $g_k \in \ker(\psi_0^*)$, the constant term in $\mathfrak{E}_{g_k}(x)$ should be 0. This in turn makes the constant term of $\mathfrak{E}_{\alpha^{j_m}}(x) \times \mathfrak{E}_{g_k}(x)$ zero. As $\alpha^{j_m} \cdot \ker(\psi_0^*) = \ker(\psi_0^*)$, $\alpha^{j_m} \cdot g_k = \gamma_k \in \ker(\psi_0^*)$, which means the constant term of $\mathfrak{E}_{\gamma_k}(x)$ is 0 as well. Also observe that $\Pi(x)$ is an irreducible polynomial, where the constant term is non-zero. Based on these observations, we can arrive at the conclusion that the constant term of $\mathfrak{P}_k(x)$ is 0. As $\alpha^{j_m} \cdot \ker(\psi_0^*) = \ker(\psi_0^*)$, this should be true for $k = 1, \ldots, p^{r-m}$.

Order of $\ker(\psi_0^*)$ is p^{r-m}, and due to how the homomorphism is defined, constant terms in polynomial representations of all $g \in \ker(\psi_0^*)$ are zero. Maximum degree possible for a polynomial representation is $\frac{r}{m} - 1 < p^{r-m}$. Therefore, the set of polynomial representations of the elements $\in \ker(\psi_0^*)$ should contain at least one polynomial of each possible degree, from degree 0 to degree $\frac{r}{m} - 1$.

Now if $\deg(\mathfrak{E}_{\alpha^{j_m}}(x)) > 0$, then for at least one $g_k \in \ker(\psi^*)$, $\mathfrak{E}_{\alpha^{j_m}}(x) \times \mathfrak{E}_{g_k}(x)$ would be of degree $\frac{r}{m}$. But $\Pi(x)$ is also of degree $\frac{r}{m}$, which means that in such a case, $\mathfrak{P}_k(x)$ would be a degree 0 polynomial, a non-zero constant term. Then, from (3.26), constant term of $\mathfrak{E}_{\gamma_k}(x)$ cannot be zero, meaning $\alpha^{j_m} \cdot g_k \notin \ker(\psi_0^*)$. This contradicts our initial statement, $\alpha^{j_m} \cdot \ker(\psi_0^*) = \ker(\psi_0^*)$, and therefore, we can conclude that $\deg(\mathfrak{E}_{\alpha^{j_m}}(x)) = 0$.

As discussed briefly earlier, in the polynomial representation of $\mathbb{F}_{p^r}$ over $\mathbb{F}_{p^m}$, zero degree polynomials are for elements of $\mathbb{F}_{p^m}$. Thus, if $\deg(\mathfrak{E}_{\alpha^{j_m}}(x)) = 0$, then $\alpha^{j_m} \in \mathbb{F}_{p^m}$. According to lemma 3.5, $\mathbb{F}_{p^m}$ in $\mathbb{F}_{p^r}$ is the set of elements generated by $\alpha^{\frac{p^r-1}{p^m-1}}$ and 0. Since j_m is the minimum non-zero power of α satisfying $\alpha^j \cdot \ker(\psi_0^*) = \ker(\psi_0^*)$, $j_m = \frac{p^r-1}{p^m-1}$.

Lemma 3.3 proves that the minimum cardinality of an α-connected set containing subgroups of order p^{r-m}, of $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, is $\frac{p^r-1}{p^m-1}$. As proved above, such a set may be generated by following the procedure in lemma 3.4 with homomorphism ψ^*, given in definition 3.5. □

Now consider some subgroup $\mathbb{G}'$ of $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$. Let $\mathbb{Q}_{\mathbb{G}'} = \{\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{p^m-1}\}$ be the quotient group of $\mathbb{G}'$, where $\mathcal{C}_i$'s represent the p^m cosets. $\mathcal{C}_0$ denotes the trivial coset, $\mathbb{G}'$ itself.

Cosets contain elements of $\mathbb{F}_{p^r}$, and using the multiplicative properties of the field, we define a *multiplication operation* $\odot$ on $\mathbb{Q}_{\mathbb{G}'}$ as follows.

Definition 3.6. *Let $\mathbb{G}'$ be a subgroup of $\mathbb{H}$, and $\mathbb{Q}_{\mathbb{G}'}$ its quotient group. Operation $\beta \odot \mathbb{Q}_{\mathbb{G}'}$, for some $\beta \in \mathbb{F}_{p^r}$, is defined as $\beta \odot Q_\psi = \{\beta \cdot \mathcal{C}_0, \beta \cdot \mathcal{C}_1, \ldots, \beta \cdot \mathcal{C}_{p^m-1}\}$, where $\beta \cdot \mathcal{C}_i$ is multiplying all elements of $\mathcal{C}_i$ with β.*

It turns out that the quotient groups of two α-connected subgroups also have an equivalent relationship, under $\odot$ defined above. This relationship is presented in the following lemma.

Lemma 3.7. *Let $\mathbb{G}_j$ and $\mathbb{G}_k$ be two subgroups of $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, that are α-connected; i.e. $\mathbb{G}_j = \alpha^t \cdot \mathbb{G}_k$ for some α^t. We denote the respective quotient groups with $\mathbb{Q}_{\mathbb{G}_j}$ and $\mathbb{Q}_{\mathbb{G}_k}$. Then, $\alpha^t \odot \mathbb{Q}_{\mathbb{G}_k} = \mathbb{Q}_{\mathbb{G}_j}$.*

Proof. Let $\mathbb{Q}_{\mathbb{G}_j} = \{\mathcal{C}_0^j, \mathcal{C}_1^j, \ldots, \mathcal{C}_{p^m-1}^j\}$ and $\mathbb{Q}_{\mathbb{G}_k} = \{\mathcal{C}_0^k, \mathcal{C}_1^k, \ldots, \mathcal{C}_{p^m-1}^k\}$. Trivial cosets, $\mathcal{C}_0^{\mathrm{i}}$, are the subgroups themselves. A coset $\mathcal{C}_i^{\mathrm{i}}$ may be represented with the subgroup $\mathbb{G}_{\mathrm{i}}$ and a coset leader term l_i^{i}, as $\mathbb{G}_{\mathrm{i}} + l_i^{\mathrm{i}}$. Representing cosets of $\mathbb{Q}_{\mathbb{G}_j}$ and $\mathbb{Q}_{\mathbb{G}_k}$ with this form results in;

$$\begin{aligned} \mathbb{Q}_{\mathbb{G}_j} &= \{\mathbb{G}_j, \mathbb{G}_j + l_1^j, \ldots, \mathbb{G}_j + l_{p^m-1}^j\} \\ \mathbb{Q}_{\mathbb{G}_k} &= \{\mathbb{G}_k, \mathbb{G}_k + l_1^k, \ldots, \mathbb{G}_k + l_{p^m-1}^k\} \end{aligned} \tag{3.27}$$

Using the operation $\odot$ (in definition 3.6) with α^t and $\mathbb{Q}_{\mathbb{G}_k}$ yields;

$$\alpha^t \odot \mathbb{Q}_{\mathbb{G}_k} = \{\alpha^t \cdot \mathbb{G}_k, \alpha^t \cdot \mathbb{G}_k + \alpha^t \cdot l_1^k, \ldots, \alpha^t \cdot \mathbb{G}_k + \alpha^t \cdot l_{p^m-1}^k\} \tag{3.28}$$

Cosets of $\mathbb{G}_k$ are mutually exclusive, and therefore, sets $(\alpha^t \cdot \mathbb{G}_k + \alpha^t \cdot l_i^k)$ are disjoint with each other. Since we have $\mathbb{G}_j = \alpha^t \cdot \mathbb{G}_k$, (3.28) may be written as;

$$\alpha^t \odot \mathbb{Q}_{\mathbb{G}_k} = \{\mathbb{G}_j, \mathbb{G}_j + \alpha^t \cdot l_1^k, \ldots, \mathbb{G}_j + \alpha^t \cdot l_{p^m-1}^k\} \tag{3.29}$$

(3.29) shows that $\alpha^t \odot \mathbb{Q}_{\mathbb{G}_k}$ is a set containing $\mathbb{G}_j$, and its $(p^m - 1)$ proper cosets, similar to $\mathbb{Q}_{\mathbb{G}_j}$. Thus, both $\mathbb{Q}_{\mathbb{G}_j}$ and $\alpha^t \odot \mathbb{Q}_{\mathbb{G}_k}$ represent the *same set*. There could be a difference in the ordering of cosets, or the coset leader terms l_i^{i}. When quotient groups are considered in some specific order, as would be the case in decoding of NB-LDPC codes, $\alpha^t \odot \mathbb{Q}_{\mathbb{G}_k}$ will be some permutation of $\mathbb{Q}_{\mathbb{G}_j}$. □

As was stated a number of times in earlier explanations, kernel of a surjective homomorphism $\psi : \mathbb{H} \to \mathbb{G}$ is a subgroup of order $|\mathbb{G}|$, of $\mathbb{H}$. Then, according to the first isomorphism

theorem [7], also presented in chapter 2, quotient group of $\ker(\psi)$, which we denote $\mathbb{Q}_\psi$, is isomorphic to $\mathbb{H}$. $\mathbb{Q}_\psi$ contains the cosets of $\ker(\psi)$, including the trivial coset, and in the isomorphism between $\mathbb{Q}_\psi$ and $\mathbb{G}$, this trivial coset will map to the identity element of $\mathbb{G}$, and other cosets to the remaining elements.

For the subfield expansion, homomorphisms we consider, presented in definition 3.5 and lemma 3.4, are between the additive group of the original field, $\mathbb{H} = \{\mathbb{F}_{p^r}, +\}$, and the additive group of the subfield, $\mathbb{G} = \{\mathbb{F}_{p^m}, +\}$. The expansion uses the smallest α-connected set containing subgroups of order p^{r-m}, of $\mathbb{H}$, which will be denoted with $\Theta^p_{r,m}$ from here onwards. Subgroups in $\Theta^p_{r,m}$ are kernels of surjective homomorphisms from $\mathbb{H}$ to $\mathbb{G}$, and thus, are isomorphic to $\mathbb{G}$. Also, since subgroups in $\Theta^p_{r,m}$ are α-connected, their quotient groups would also be related as in Lemma 3.7. These observations allow us to represent symbols of $\mathbb{F}_{p^r}$ as matrices over $\mathbb{F}_{p^m}$, and provide the basis for the subfield expansion.

Expansion we propose is presented in detail in the following subsection. It is discussed mainly in the perspective of the factor graph, as that is the more intuitive approach when it comes to decoding NB-LDPC codes. In the examples provided, the graph expansion and the corresponding matrix expansion both are presented for the same NB-LDPC code.

3.2.2 Graph Expansion

Consider a NB-LDPC code ζ over $\mathbb{F}_{p^r}$, and let $m \mid r$. To decode ζ over $\mathbb{F}_{p^m}$, which would provide significant complexity gains, it should be suitably represented over that field. With NB-LDPC codes, this means expanding the PCM of ζ to a matrix over $\mathbb{F}_{p^m}$, which may also be viewed as expanding the factor graph of ζ to a graph over $\mathbb{F}_{p^m}$. The expansion we propose is based on the quotient groups of the subgroups in $\Theta^p_{r,m}$, the smallest α-connected set of subgroups of order p^{r-m}. This set of quotient groups of $\Theta^p_{r,m}$ will be represented with $\mathfrak{Q}^p_{r,m}$. According to lemma 3.3, $\mathfrak{Q}^p_{r,m}$ is of cardinality $\frac{p^r-1}{p^m-1}$. In the following, we first provide some motivation for the expansion, and later present the expansion in a more structured way.

A quotient group $\in \mathfrak{Q}^p_{r,m}$ may be viewed as a way of dividing the symbols of $\mathbb{F}_{p^r}$ in to p^m groups. In decoding NB-LDPC codes, as discussed in chapter 2, each variable node is associated with a *symbol probability vector*, which contains probabilities of the respective value being each symbol of the field. Similarly, given a quotient group, it is also possible to consider probabilities of the said value *belonging* to each grouping of symbols in that quotient group. With a quotient group in $\mathfrak{Q}^p_{r,m}$, such a computation would result in a probability vector of length p^m, whereas a

traditionally used symbol probability vector will be of length p^r. Since the groupings in a quotient group are cosets, we refer to these probability vectors as *coset probability vectors* (CPVs). As $|\mathfrak{Q}^p_{r,m}| = \frac{p^r-1}{p^m-1}$, that many CPVs are required for a single variable node. Complexity advantage of using CPVs in decoding is mainly in the check node computations step, the complexity bottleneck of decoding NB-LDPC codes [33], [28].

As discussed in chapter 2, check node computations step can be considered as a combination of two sub-steps; permutation and convolution of symbol probability vectors (convolution operation was represented in (2.9)) [25]. In the permutation sub-step, the simpler one of the two, symbol probability vectors received by the check node are permuted, where the permutations are determined by the respective edge labels. Since $\mathfrak{Q}^p_{r,m}$ contains the quotient groups of the subgroups in $\Theta^p_{r,m}$, an α-connected set, according to lemma 3.7, CPVs will also have to be permuted similarly. Thus, the permutation sub-step would not be significantly altered by using CPVs. But the impact of this approach on the convolution sub-step is much more significant.

In order to understand how our approach impacts the convolution sub-step, consider the simple case of a degree 3 check node of ζ, with the parity-check equation $v_1 + v_2 + v_3 = 0$ (all edge labels are 1). For computing the estimate for the symbol probability vector of v_3, $\underline{\mathbf{p}}^s_{v_3}$, probability vectors received from the other two variable nodes have to undergo a convolution operation. As discussed in chapter 2, such an operation will be of complexity order $\mathcal{O}(p^{2r})$, since the vectors are of length r.

Now assume we have to compute some i'th CPV of v_3, $\underline{\mathbf{p}}^c_{v_3,i}$. This computation also only requires the i'th CPVs received from the other two nodes. As all quotient groups in $\mathfrak{Q}^p_{r,m}$ are isomorphic to the additive group of $\mathbb{F}_{p^m}$, computation of $\underline{\mathbf{p}}^c_{v_3,i}$ should be similar to the convolution sub-step at a check node of a code over $\mathbb{F}_{p^m}$. This means that $\underline{\mathbf{p}}^c_{v_3,i}$ is the result of a convolution between two probability vectors of length $m < r$, which is of complexity order $\mathcal{O}(p^{2m})$, significantly less than $\mathcal{O}(p^{2r})$ of the traditional approach. However, since $|\mathfrak{Q}^p_{r,m}| = \frac{p^r-1}{p^m-1}$, that many CPVs are required to replace a single symbol probability vector. Therefore, overall complexity of the convolution sub-step when CPVs are used is $\left(p^{2m} \times \frac{p^r-1}{p^m-1}\right) \approx \mathcal{O}(p^{m+r})$, somewhat higher than $\mathcal{O}(p^{2m})$. Nevertheless, when compared with $\mathcal{O}(p^{2r})$, this is a significant reduction of complexity, particularly for cases when $m \ll r$.

These observations act as the motivation to convert ζ, a NB-LDPC code over $\mathbb{F}_{p^r}$, to a form where CPVs can be used for decoding. This form is essentially an expansion of the PCM of ζ, which we analyze from the perspective of the factor graph in the following sections.

3.2.2.1 Initial Expansion

Initial step of the expansion we propose is relatively simple; each check node and variable node of the original factor graph, i.e., the so-called $\mathbb{F}_{p^r}$ nodes, are to be replaced with $\frac{p^r-1}{p^m-1}$ nodes, one for each CPV. Since CPVs are treated as $\mathbb{F}_{p^m}$ probability vectors due to the respective quotient groups being isomorphic to $\{\mathbb{F}_{p^m}, +\}$, we refer to the new nodes as $\mathbb{F}_{p^m}$ nodes. Each $\mathbb{F}_{p^m}$ node of the set which replaced an $\mathbb{F}_{p^r}$ variable node would represent some CPV of the respective variable, whereas the new $\mathbb{F}_{p^m}$ check nodes will be tasked with computing estimates of these CPVs. As a consequence of Lemma 3.7, connections between the new variable and check nodes that replaced a pair of $\mathbb{F}_{p^r}$ nodes depend on the edge label between them in the original factor graph.

Consider a neighboring check node m and a variable node n in the original factor graph, where the edge label is $\alpha^k \in \mathbb{F}_{p^r}$. According to Lemma 3.7, when multiplied with α^k, $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^p_{r,m}$ becomes a permutation of some $\mathbb{Q}_{\psi_j} \in \Theta^{\mathbb{Q}}_m$. Therefore, in the expanded factor graph, the $\mathbb{F}_{p^m}$ variable node representing the i'th CPV of variable n, n_i, should be connected to the $\mathbb{F}_{p^m}$ check node computing estimates of j'th CPVs, m_j. $(\alpha^k \odot \mathbb{Q}_{\psi_i})$ is a *permutation* of $\mathbb{Q}_{\psi_j}$, and thus, the CPV sent from n_i to m_j has to be permuted, similar to symbol probability vectors on the original factor graph. From the perspective of the graph expansion, the requirement for a permutation translates to connecting n_i and m_j with an edge labeled with some symbol from $\mathbb{F}_{p^m}$. Label in such a case is the symbol of $\mathbb{F}_{p^m}$ which induces the reverse permutation of the one on $\mathbb{Q}_{\psi_j}$, on a symbol probability vector.

As a toy example, consider the following degree 2 parity-check equation ρ, which is from a code over $\mathbb{F}_{2^4}$. α here denotes a primitive of that field.

$$\rho : \alpha^4 \cdot v_1 + \alpha \cdot v_2 = 0 \tag{3.30}$$

We consider representing ρ over $\mathbb{F}_{2^2}$, a subfield of $\mathbb{F}_{2^4}$. Set of quotient groups required for the expansion, $\mathfrak{Q}^2_{4,2}$, may be generated using the homomorphism presented in definition 3.5, and by following the procedure outlined in lemma 3.4. Each $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^2_{4,2}$ is isomorphic to $\{\mathbb{F}_{2^2}, +\}$, and thus, cosets of $\mathbb{Q}_{\psi_i}$ can be mapped to symbols of $\mathbb{F}_{2^2}$. Quotient groups of $\mathfrak{Q}^2_{4,2}$, along with the mappings between cosets and $\mathbb{F}_{2^2}$ symbols, are given in the following table, where ω represents a primitive element of $\mathbb{F}_{2^2}$.

Figure 3.1 presents the original factor graph for ρ, along with its expansion over $\mathbb{F}_{2^2}$, which can be obtained by following the procedure discussed so far with quotient groups in Table 3.4. Original factor graph, over $\mathbb{F}_{2^4}$, is shaded in grey, and the expanded graph is below that. In both

	0	1	ω	ω^2
$\mathbb{Q}_{\psi_0}$	$\{0, \alpha, \alpha^6, \alpha^{11}\}$	$\{1, \alpha^4, \alpha^{12}, \alpha^{13}\}$	$\{\alpha^2, \alpha^3, \alpha^5, \alpha^9\}$	$\{\alpha^7, \alpha^8, \alpha^{10}, \alpha^{14}\}$
$\mathbb{Q}_{\psi_1}$	$\{0, 1, \alpha^5, \alpha^{10}\}$	$\{\alpha, \alpha^2, \alpha^4, \alpha^8\}$	$\{\alpha^6, \alpha^7, \alpha^9, \alpha^{13}\}$	$\{\alpha^3, \alpha^{11}, \alpha^{12}, \alpha^{14}\}$
$\mathbb{Q}_{\psi_2}$	$\{0, \alpha^4, \alpha^9, \alpha^{14}\}$	$\{1, \alpha, \alpha^3, \alpha^7\}$	$\{\alpha^5, \alpha^6, \alpha^8, \alpha^{12}\}$	$\{\alpha^2, \alpha^{10}, \alpha^{11}, \alpha^{13}\}$
$\mathbb{Q}_{\psi_3}$	$\{0, \alpha^2, \alpha^7, \alpha^{12}\}$	$\{1, \alpha^8, \alpha^9, \alpha^{11}\}$	$\{\alpha, \alpha^5, \alpha^{13}, \alpha^{14}\}$	$\{\alpha^3, \alpha^4, \alpha^6, \alpha^{10}\}$
$\mathbb{Q}_{\psi_4}$	$\{0, \alpha^3, \alpha^8, \alpha^{13}\}$	$\{1, \alpha^2, \alpha^6, \alpha^{14}\}$	$\{\alpha^4, \alpha^5, \alpha^7, \alpha^{11}\}$	$\{\alpha, \alpha^9, \alpha^{10}, \alpha^{12}\}$

Table 3.4: Quotient Groups in $\mathfrak{Q}^2_{4,2}$

graphs, circles denote variable nodes and squares denote check nodes. In the figure, $\mathbb{F}_{2^2}$ variable and check nodes are considered in the same order as in Table 3.4, as the labeling on the left hand side indicates.

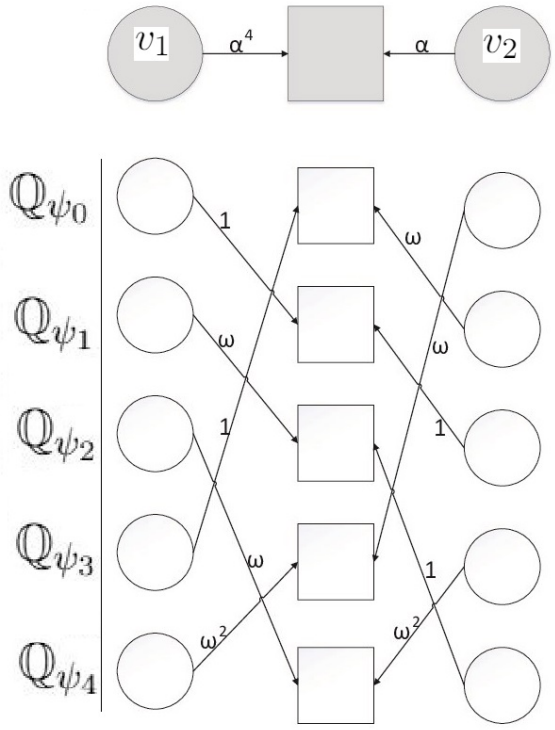

Figure 3.1: Initial Subfield Expansion

As discussed earlier, we have replaced the $\mathbb{F}_{2^4}$ variable nodes and the check node with five nodes over $\mathbb{F}_{2^2}$ each. These represent, or compute, estimates of CPVs for the five quotient groups given in Table 3.4.

To understand how the new nodes are connected, consider the $\mathbb{F}_{2^2}$ variable node that

corresponds to $\mathbb{Q}_{\psi_0}$ of $\mathbb{F}_{2^4}$ node v_1. The original edge label between v_1 and the $\mathbb{F}_{2^4}$ check node is α^4. It is evident that multiplying $\mathbb{Q}_{\psi_0}$ with α^4, as in Definition 3.6, results in $\mathbb{Q}_{\psi_1}$. Interestingly, $\alpha^4 \odot \mathbb{Q}_{\psi_0}$ produces $\mathbb{Q}_{\psi_1}$ in the *same order* as considered in Table 3.4. Representing this, the $\mathbb{Q}_{\psi_0}$ node of v_1 is connected by an edge labeled with 1 to the new check node that computes estimates of CPVs that correspond to $\mathbb{Q}_{\psi_1}$.

Now consider the $\mathbb{Q}_{\psi_1}$ node of v_1. Here, the operation $\alpha^4 \odot \mathbb{Q}_{\psi_1}$ produces the set $\mathbb{Q}_{\psi_2}$ but in a *different order* to the one used in Table 3.4. This variable node is therefore connected to the new check node for $\mathbb{Q}_{\psi_2}$, and the edge label is now ω, the $\mathbb{F}_{2^2}$ element that induces the *reverse permutation* of the one caused by the operation $\alpha^4 \odot \mathbb{Q}_{\psi_1}$.

In this manner, each edge of the original $\mathbb{F}_{2^4}$ graph in Figure 3.1 is replaced with 5 edges labeled with $\mathbb{F}_{2^2}$ symbols. From the perspective of a matrix expansion, this is similar to replacing each element in the original PCM with a 5×5 matrix over $\mathbb{F}_{2^2}$, or in other words, representing $\mathbb{F}_{2^4}$ symbols as 5×5 matrices over $\mathbb{F}_{2^2}$. In the general case of $\mathbb{F}_{p^r}$ and $\mathbb{F}_{p^m}$, each $\beta \in \mathbb{F}_{p^r}$ will be represented with a $(\frac{p^r-1}{p^m-1} \times \frac{p^r-1}{p^m-1})$ matrix over $\mathbb{F}_{p^m}$. This alternative matrix representation can be defined as follows.

Definition 3.7. *Consider $\mathbb{F}_{p^r}$ and $\mathbb{F}_{p^m}$, where $m \mid r$. **Alternative matrix representation** of a symbol $\beta \in \mathbb{F}_{p^r}$ over $\mathbb{F}_{p^m}$ is the matrix h^*_β defined as follows.*

$$h^*_\beta(i,j) \begin{cases} \neq 0; & \beta \cdot \ker(\psi_i) = \ker(\psi_j) \\ = 0; & otherwise \end{cases}, \qquad 0 \leq i,j \leq \left(\frac{p^r-1}{p^m-1}\right) - 1 \tag{3.31}$$

When $\beta \cdot \ker(\psi_i) = \ker(\psi_j)$, then $h^*_\beta(i,j) = \gamma \in \mathbb{F}_{p^m}$, where γ is the element that induces the reverse permutation of the permutation of $\mathbb{Q}_{\psi_j}$ produced by operation $\beta \odot \mathbb{Q}_{\psi_i}$.

For example, consider the case of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$. Alternative matrix representations of $\mathbb{F}_{2^4}$ symbols as 5×5 matrices over $\mathbb{F}_{2^2}$ are given in Table 3.5, which are based on the quotient groups, and homomorphisms, presented in Table 3.3. In Table 3.5, α and ω denote a primitive element of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$, respectively.

Alternative matrix representations of $\mathbb{F}_{2^4}$ over $\mathbb{F}_{2^2}$, as may be observed from Table 3.5, are very sparse. Even though not permutation matrices, they also only contain a single non-zero element per row and column. As the following lemma proves, this property is true for the general case also.

Lemma 3.8. *Consider the alternative matrix representation of $\mathbb{F}_{p^r}$ over $\mathbb{F}_{p^m}$, where $m \mid r$. In any such matrix h^*_β, $\beta \in \mathbb{F}_{p^r}$, only a single position per row and per column is non-zero.*

0	$\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$	α^3	$\begin{bmatrix}0&0&1&0&0\\0&0&0&0&\omega\\0&0&0&\omega^2&0\\0&\omega^2&0&0&0\\\omega&0&0&0&0\end{bmatrix}$	α^7	$\begin{bmatrix}0&0&0&0&\omega\\0&0&0&1&0\\\omega^2&0&0&0&0\\0&0&1&0&0\\0&\omega&0&0&0\end{bmatrix}$	α^{11}	$\begin{bmatrix}0&0&0&1&0\\1&0&0&0&0\\0&\omega^2&0&0&0\\0&0&0&0&\omega\\0&0&\omega^2&0&0\end{bmatrix}$
1	$\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix}$	α^4	$\begin{bmatrix}0&1&0&0&0\\0&0&\omega&0&0\\0&0&0&0&\omega\\1&0&0&0&0\\0&0&0&\omega^2&0\end{bmatrix}$	α^8	$\begin{bmatrix}0&0&\omega&0&0\\0&0&0&0&\omega^2\\0&0&0&1&0\\0&1&0&0&0\\\omega^2&0&0&0&0\end{bmatrix}$	α^{12}	$\begin{bmatrix}0&0&0&0&\omega^2\\0&0&0&\omega&0\\1&0&0&0&0\\0&0&\omega&0&0\\0&\omega^2&0&0&0\end{bmatrix}$
α	$\begin{bmatrix}0&0&0&\omega&0\\\omega&0&0&0&0\\0&1&0&0&0\\0&0&0&0&\omega^2\\0&0&1&0&0\end{bmatrix}$	α^5	$\begin{bmatrix}\omega&0&0&0&0\\0&\omega&0&0&0\\0&0&\omega&0&0\\0&0&0&\omega&0\\0&0&0&0&\omega\end{bmatrix}$	α^9	$\begin{bmatrix}0&\omega&0&0&0\\0&0&\omega^2&0&0\\0&0&0&0&\omega^2\\\omega&0&0&0&0\\0&0&0&1&0\end{bmatrix}$	α^{13}	$\begin{bmatrix}0&0&\omega^2&0&0\\0&0&0&0&1\\0&0&0&\omega&0\\0&\omega&0&0&0\\1&0&0&0&0\end{bmatrix}$
α^2	$\begin{bmatrix}0&0&0&0&1\\0&0&0&\omega^2&0\\\omega&0&0&0&0\\0&0&\omega^2&0&0\\0&1&0&0&0\end{bmatrix}$	α^6	$\begin{bmatrix}0&0&0&\omega^2&0\\\omega^2&0&0&0&0\\0&\omega&0&0&0\\0&0&0&0&1\\0&0&\omega&0&0\end{bmatrix}$	α^{10}	$\begin{bmatrix}\omega^2&0&0&0&0\\0&\omega^2&0&0&0\\0&0&\omega^2&0&0\\0&0&0&\omega^2&0\\0&0&0&0&\omega^2\end{bmatrix}$	α^{14}	$\begin{bmatrix}0&\omega^2&0&0&0\\0&0&1&0&0\\0&0&0&0&1\\\omega^2&0&0&0&0\\0&0&0&\omega&0\end{bmatrix}$

Table 3.5: Alternative Matrix Representations of $\mathbb{F}_{2^4}$

Proof. From definition 3.7, $h^*_\beta(i,j) \neq 0$ only if $\beta \cdot \ker(\psi_i) = \ker(\psi_j)$. Now assume there are two non-zero positions on row i, which we denote j_1 and j_2. Then;

$$\beta \cdot \ker(\psi_i) = \ker(\psi_{j_1}) = \ker(\psi_{j_2}) \tag{3.32}$$

$\ker(\psi_{j_1})$ and $\ker(\psi_{j_2})$ are subgroups of the smallest α-connected set, and therefore, by definition, $\ker(\psi_{j_1}) \neq \ker(\psi_{j_2})$. Thus, there cannot be more than a single non-zero position on any row.

Then we assume there are two non-zero positions i_1 and i_2 on column j. Then;

$$\beta \cdot \ker(\psi_{i_1}) = \beta \cdot \ker(\psi_{i_1}) = \ker(\psi_j) \tag{3.33}$$

Again we have $\ker(\psi_{i_1}) = \ker(\psi_{i_1})$, which is not possible as explained earlier. Thus, there is only a single non-zero position in a column as well. □

So similar to the extended binary representation [40], no cycles can exist in a matrix h^*_β of the alternative binary representation.

As can be observed from Table 3.4, each $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^2_{4,2}$ is a different grouping of the same set of symbols, those of $\mathbb{F}_{2^4}$. Therefore, a CPV associated with some $\mathbb{Q}_{\psi_i}$ would contain some

information about all such CPVs. In other words, CPVs cannot be considered as independent of each other. But the initial expansion, presented in Figure 3.1, does not capture these dependencies, as there are no paths between the $\mathbb{F}_{2^2}$ variable nodes representing CPVs of a single $\mathbb{F}_{2^4}$ node. In general, when expanding a graph over $\mathbb{F}_{p^r}$ to one over $\mathbb{F}_{p^m}$, it should be noted that the set of $\mathbb{F}_{p^m}$ variable nodes of a single $\mathbb{F}_{p^r}$ variable node are dependant on each other. In the following, we first analyze the nature of these dependencies, and then modify the expanded graph to accommodate them.

3.2.2.2 An Alternative Vector Representation

Extended binary representation [40], also discussed at the beginning of this chapter, proposes a new binary vector representation for symbols in $\mathbb{F}_{2^r}$, which, together with the binary matrix representation proposed in the same paper [40], form the basis for that expansion. We also introduce a similar alternative vector representation to better understand the dependencies between CPVs.

Definition 3.8. *Consider $\mathbb{F}_{p^r}$ and $\mathbb{F}_{p^m}$, where $m \mid r$, and the set of quotient groups $\mathfrak{Q}^p_{r,m}$, as defined earlier. Value of $\beta \in \mathbb{F}_{p^r}$, with respect to some $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^p_{r,m}$, is the symbol of $\mathbb{F}_{p^m}$ that maps with the coset of $\mathbb{Q}_{\psi_i}$ containing β. Note that $|\mathfrak{Q}^p_{r,m}| = \frac{p^r-1}{p^m-1}$. **Alternative vector representation** of β over $\mathbb{F}_{p^m}$ is the vector of $\frac{p^r-1}{p^m-1}$ values defined with respect to each $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^p_{r,m}$.*

As an example, consider the case of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$ once more. Alternative vector representations of $\mathbb{F}_{2^4}$ symbols as length $\frac{2^4-1}{2^2-1} = 5$ vectors over $\mathbb{F}_{2^2}$ are given in Table 3.6, which are based on the quotient groups, and homomorphisms, presented in Table 3.4. Note that position i in a vector of Table 3.6 maps to quotient group $\mathbb{Q}_{\psi_i}$ in Table 3.4.

$0 : 00000$	$\alpha^3 : \omega\omega^2 1\omega^2 0$	$\alpha^7 : \omega^2\omega 10\omega$	$\alpha^{11} : 0\omega^2\omega^2 1\omega$
$1 : 10111$	$\alpha^4 : 110\omega^2\omega$	$\alpha^8 : \omega^2 1\omega 10$	$\alpha^{12} : 1\omega^2\omega 0\omega^2$
$\alpha : 011\omega\omega^2$	$\alpha^5 : \omega 0\omega\omega\omega$	$\alpha^9 : \omega\omega 01\omega^2$	$\alpha^{13} : 1\omega\omega^2\omega 0$
$\alpha^2 : \omega 1\omega^2 01$	$\alpha^6 : 0\omega\omega\omega^2 1$	$\alpha^{10} : \omega^2 0\omega^2\omega^2\omega^2$	$\alpha^{14} : \omega^2\omega^2 0\omega 1$

Table 3.6: Alternative Vector Representations of $\mathbb{F}_{2^4}$

It can be observed that the vectors in Table 3.6 form a 2-dimensional vector space over $\mathbb{F}_{2^2}$. In coding theory terminology, these are the codewords of a $(5, 2)$ linear code over $\mathbb{F}_{2^2}$. This means that just two of the values with respect to $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^2_{4,2}$ are sufficient to derive the alternative vector representation of any $\beta \in \mathbb{F}_{2^4}$. The dependent relationships between $\mathbb{Q}_{\psi_i}$'s can be fully captured with the parity-check matrix of this code.

In general, with $\mathbb{F}_{p^r}$ and $\mathbb{F}_{p^m}$, where $m \mid r$, alternative representation vectors would form a $\frac{r}{m}$ dimensional vector space, or in other words a $(\frac{p^r-1}{p^m-1}, \frac{r}{m})$ code, over $\mathbb{F}_{p^m}$. Each set of $\frac{p^r-1}{p^m-1}$ $\mathbb{F}_{p^m}$ nodes of a $\mathbb{F}_{p^r}$ variable node would form this code, and therefore, we refer to it as the *'local code'*. Our proposal is to use the PCM of the local code, $\mathfrak{H}^p_{r,m}$, to represent the dependencies between CPVs. A $(\frac{p^r-1}{p^m-1}, \frac{r}{m})$ code is actually from the family of non-binary simplex codes. Since dual of such a code is the $(\frac{p^r-1}{p^m-1}, \frac{p^r-1}{p^m-1} - \frac{r}{m})$ Hamming code over $\mathbb{F}_{p^m}$ [35], $\mathfrak{H}^p_{r,m}$ would consist of $\frac{p^r-1}{p^m-1} - \frac{r}{m}$ Hamming codewords. As an example, the local PCM for the case of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$, $\mathfrak{H}^2_{4,2}$, which consists of 3 codewords of the $(3, 5)$ Hamming code over $\mathbb{F}_{2^2}$, is given below.

$$\mathfrak{H}^2_{4,2} = \begin{bmatrix} 1\,1\,1\,0\,0 \\ 1\,\omega\,0\,1\,0 \\ 1\,\omega^2\,0\,0\,1 \end{bmatrix} \tag{3.34}$$

(3.34) shows that, when expanding a factor graph over $\mathbb{F}_{2^4}$ into one over $\mathbb{F}_{2^2}$, three parity-check equations are sufficient to capture the dependencies between the CPVs of a single variable node. From the perspective of a factor graph, these equations may be viewed as new check nodes that have to be added to the expanded graph. How that can be done is discussed in the following section.

3.2.2.3 Final Expansion

In the preceding section, it was established that the parity-check equations of the PCM that represents the local code have to be added to the expanded factor graph as new check nodes. An instance of the local code exists among every set of $\mathbb{F}_{p^m}$ nodes that replaced a $\mathbb{F}_{p^r}$ variable node in the expanded graph. Therefore, the number of new check nodes that have to be added would be fairly high, $(\frac{p^r-1}{p^m-1} - \frac{r}{m})$ times the code length. It might initially seem that adding that many new nodes would significantly increase decoding complexity, but these check nodes are of low degrees. As discussed earlier, local code is a non-binary simplex code, and its dual code is a non-binary Hamming code, which contains many codewords of weight 3. Local PCM can be constructed with some of those, and then the new check nodes would be of degree 3. A more in-depth look at decoding complexity on proposed expansions will be given in chapter 4.

So it seems that, in the proposed subfield expansion, there are two distinct types of check nodes. First type are the ones that replace the $\mathbb{F}_{p^r}$ check nodes. These were introduced in the initial expansion section, and from here onwards, we refer to them as *regular check nodes*. The second

type are the ones added to represent dependencies between CPVs, which we will call *local check nodes*. The reason for distinguishing between the two types is more related to decoding on expanded graphs, the focus of the next chapter.

Performance of a NB-LDPC code, with any form of iterative message passing decoding, is dependant on the features of the graph used [13], [14]. As briefly discussed in chapter 2, in the literature, short cycles are considered to be at the root of graphical structures that negatively impact the decoding [13], [14]. These include stopping sets, and absorbing sets, known to cause the undesirable error floor exhibited by some LDPC codes.

When it comes to binary codes, or binary graphs, short cycles are only distinguished by their lengths, as all edge labels are 1. Negative impact of a cycle decreases with the length; length 4 cycles are the most undesirable. Any cycle in a binary graph can create a stopping or an absorbing set, and thus, in LDPC construction methods, special focus is given to increasing the length of the shortest cycle in the graph, also known as the *girth* of the corresponding code [13]. But the situation is slightly different with NB-LDPC codes, as negative impact of a short cycle is also dependent on the edge labels [14], [18]. While the length of a cycle is important here as well, cycles created by sub-matrices in the PCM that are not of full-rank are considered the most troublesome. NB absorbing sets, considered the major cause of error-floor in NB-LDPC codes, are created by such cycles [14]. Thus, in construction of NB-LDPC codes, major focus will be on reducing these type of cycles, rather than increasing the girth as in binary LDPC codes [18], although girth is also of significance. Chapter 4 sheds some more light on graphical structures that impact iterative decoding of NB-LDPC codes.

A factor graph can be made more suitable for iterative decoding by reducing the number of undesirable graphical sub-structures as much as possible. Keeping this in mind, we propose using a matrix with the lowest possible short cycle count as the local PCM, $\mathfrak{H}^{p}_{r,m}$. As the structures such as stopping and absorbing sets are caused by short cycles, this approach would result in a local PCM that is much better suited for iterative decoding. Although one could use a canonical generator matrix of a Hamming code (possibly with column permutations) as the local PCM, the graph induced may contain short cycles. In such a scenario, row operations can be carried out on the matrix until a *better* one, in terms of the number of short cycles, is obtained. This is particularly important when the expansion results in a *binary* graph ($p = 2$ and $m = 1$), since then *any* short cycle is detrimental for decoding. In the non-binary case ($m > 1$), it might not be possible to remove all short cycles, and one might have to settle with a local PCM free only of those not satisfying the

full rank condition [18], such as $\mathfrak{H}^2_{4,2}$ in (3.34). Also, the decoding scheme we propose in the next chapter employs a technique to further reduce the possible negative effects of cycles among local check nodes.

Local check nodes are able to capture the dependencies between CPVs fully, and adding these is the only modification necessary to the initial expansion, discussed in 3.2.2.1. Figure 3.2 presents the final expanded graph for the toy example of $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$, with parity-check equation ρ in (3.30). $\mathfrak{H}^2_{4,2}$ in (3.34) was used for the local code. Initial expansion of ρ, before accounting for the dependencies between CPVs, was given in Figure 3.1. Once more, original graph is shaded grey, and the $\mathbb{F}_{2^2}$ expansion is shown in white. Circles represent variable nodes, squares regular check nodes, and hexagons local check nodes. Note that in the interest of a clearer figure, only the edge labels $\neq 1$ are shown. Order of quotient group assignment to $\mathbb{F}_{2^2}$ nodes is the same as in Figure 3.1 and local check nodes correspond to $\mathfrak{H}^2_{4,2}$ in (3.34).

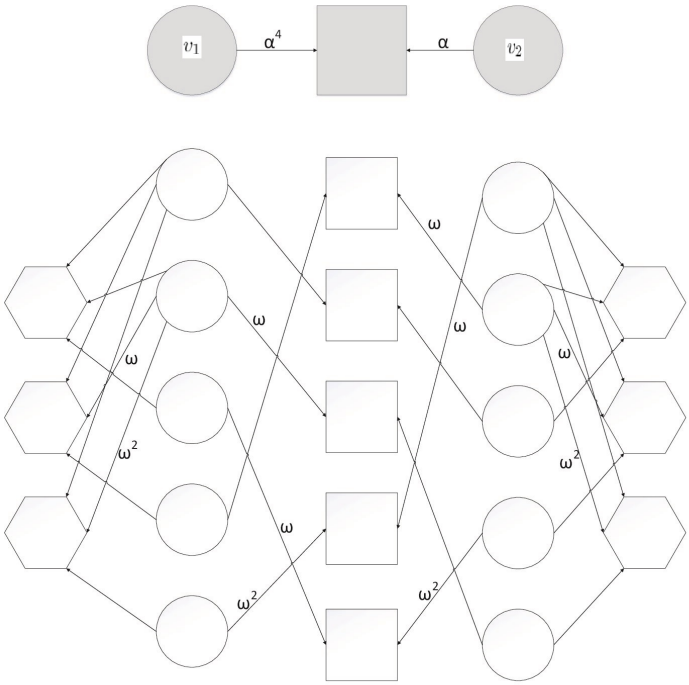

Figure 3.2: Final Subfield Expansion

In the following, we summarize the different steps of the subfield expansion, when expanding a graph over $\mathbb{F}_{p^r}$ into one over $\mathbb{F}_{p^m}$.

1. Obtain the smallest set of α-connected subgroups $\Theta^p_{r,m}$, using the homomorphism in definition 3.5 with the procedure outlined in Lemma 3.4.
2. Derive the set of quotient groups $\mathfrak{Q}^p_{r,m}$, using $\Theta^p_{r,m}$ from step 1.
3. Find the homomorphisms between elements of $\{\mathbb{F}_{p^m}, +\}$, and each $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^p_{r,m}$.
4. Use the homomorphisms found in step 3 to derive the alternative vector representation of $\mathbb{F}_{p^r}$ elements.
5. Find a matrix suitable for iterative decoding, to be used as the PCM of the *local* code formed by alternative representation vectors, derived in step 4.
6. Expand each node in the $\mathbb{F}_{p^r}$ graph into $(\frac{p^r-1}{p^m-1})$ $\mathbb{F}_{p^m}$ nodes. Connect the new nodes with edges labeled with $\mathbb{F}_{p^m}$ elements, based on the original edge labels.
7. Add local check nodes to represent the local PCM found in step 5.

While a number of different expansions are possible for a single factor graph, there are two expansions that are of particular interest; the binary expansion, when $p = 2$ and $m = 1$, and the square-root expansion, with $r = 2m$. Binary expansion offers the highest reduction in complexity, while the square-root expansion is the best in terms of performance. Interestingly, while both are instances of the general subfield expansion explained earlier, each can be derived in a more ad-hoc manner as well. In the following, we present these derivations, as well as more detailed examples, which would also provide a better understanding of the step-wise graph expansion procedure given above.

3.2.3 Special Cases

3.2.3.1 Binary Expansion

Subfield expansion results in a binary graph when the original field is of characteristic 2, or $p = 2$, and subfield is the binary field, or $m = 1$. Procedure outlined at the end of the previous section can be followed to derive this binary graph.

1. Homomorphism ψ^*, in definition 3.5, is now based on the binary polynomial representation of $\mathbb{F}_{2^r}$ symbols, and the smallest α-connected set $\Theta^2_{r,1}$ now contains $\frac{2^r-1}{2^1-1} = 2^r - 1$ subgroups of order 2^{r-1}, of $\{\mathbb{F}_{2^r}, +\}$. Thus, the order of a subgroup used for the expansion is exactly

half of the order of the field's additive group. Cardinality of the smallest α-connected set is one less than the order of the original field.

2. A quotient group in $\mathfrak{Q}^2_{r,1}$ contains only two cosets; the trivial coset, and one proper coset.

3. In a homomorphism between $\{\mathbb{F}_2, +\}$ and a $\mathbb{Q}_{\psi_i} \in \mathfrak{Q}^2_{r,1}$, the trivial coset will be mapped to 0 and the proper coset to 1.

4. Alternative vector representations will be length $(2^r - 1)$ binary vectors, codewords of the $(2^r - 1, r)$ binary simplex code.

5. Local PCM will consist of codewords of the $(2^r - 1, 2^r - r - 1)$ binary Hamming code. In fact, a generator matrix of that code could be used as the PCM. As discussed in the previous section, removing short cycles is quite important for the binary case, and thus, some row operations might have to be carried out to make the local PCM better suited for iterative decoding.

6. In the original factor graph, each non-binary node will be replaced with $(2^r - 1)$ binary nodes. The connections between the new variable and check nodes are dependent on the original edge labels. Since the expansion is a binary graph, new edge labels will all be 1.

7. Local binary check nodes will correspond to the PCM derived in step 5.

The smallest α-connected set used for the binary expansion contains $(2^r - 1)$ subgroups, as mentioned above. These subgroups are of order 2^{r-1}, and it turns out that $\mathbb{F}_{2^r}$ contains only $(2^r - 1)$ such subgroups, and they all are α-connected with each other. Thus, the α-connected set used in the binary expansion is the set containing *all* subgroups of order 2^{r-1}, of $\{\mathbb{F}_{2^r}, +\}$. Following lemma establishes that there exists only $(2^r - 1)$ such subgroups.

Lemma 3.9. *There are* $(2^r - 1)$ *distinct subgroups of order* 2^{r-1} *in* $\mathbb{H} = \{\mathbb{F}_{2^r}, +\}$.

Proof. $\mathbb{H}$ can be considered a binary vector space of dimension r. Similarly, a subgroup of order 2^{r-1} of $\mathbb{H}$ is a subspace of this vector space, of dimension $(r-1)$. Thus, the number of such distinct subgroups is equal to the number of distinct subspaces of dimension $(r-1)$ in a binary vector space of dimension r.

We now count the ways such a subspace can be constructed, which will be denoted with n_w. To construct a subspace of dimension $(r - 1)$, one should simply chose $(r - 1)$ linearly

independent vectors from the original vector space, the set of which is usually referred to as the *basis* of the subspace. For the first choice, any vector other than the all-zero vector is valid. For the second, vectors linearly dependant on the first one picked have to be disregarded, but since the vector space is binary, there are none. In the third choice, the vector resulting from the addition of the first two has to be left-out. Continuing in this manner, we find;

$$n_w = (2^r - 1)(2^r - 2)...(2^r - 2^{r-2}) \tag{3.35}$$

In (3.36), we may have counted the same subspace several times, since one vector space could have a number of different bases. Therefore, in order to find the number of *distinct* subspaces, n_w has to be divided by the number of different bases of a $(r-1)$-dimensional vector space, which we denote with n_b. Following the same approach as before with the $(r-1)$-dimensional space;

$$n_b = (2^{r-1} - 1)(2^{r-1} - 2)...(2^{r-1} - 2^{r-2}) \tag{3.36}$$

Now, the number of distinct subspaces of dimension $(r-1)$, is simply $\frac{n_w}{n_b}$.

$$\frac{n_w}{n_b} = \frac{(2^r - 1)(2^r - 2)...(2^r - 2^{r-2})}{(2^{r-1} - 1)(2^{r-1} - 2)...(2^{r-1} - 2^{r-2})} = \frac{(2^r - 1)2^{r-2}}{(2^{r-1} - 2^{r-2})} = 2^{r-1} \tag{3.37}$$

Thus, the number of distinct subgroups of order 2^{r-1} of $\mathbb{H}$ is $(2^r - 1)$. □

Although above lemma shows that there are only $(2^r - 1)$ subgroups of order 2^{r-1}, they have to form an α-connected set to be used in the expansion. Lemma 3.10 below establishes this.

Lemma 3.10. *Let Θ be the set containing the $(2^r - 1)$ subgroups of order 2^{r-1}, of $\mathbb{H} = \{\mathbb{F}_{2^r}, +\}$. Then, Θ is an α-connected set.*

Proof. Let $\mathbb{G} \in \Theta$. As lemma 3.1 established, multiplying all the elements of $\mathbb{G}$ by some $\beta \in \mathbb{F}_{2^r}$ results in a subgroup $\mathbb{G}_\beta$ of the same order. Since, according to lemma 3.9, Θ contains all possible subgroups of order 2^{r-1} of $\mathbb{H}$, $\mathbb{G}_\beta \in \Theta$.

In order to prove that Θ is an α-connected set, it is sufficient to show that the subgroup $\mathbb{G}_\beta$ produced by multiplying some $\mathbb{G} \in \Theta$ with some $\beta \in \mathbb{F}_{2^r}$, $\beta \neq 0, 1$, is different for each value of β. Since there are $(2^r - 2)$ possible values for β, this would mean that each $\mathbb{G}$ is α-connected with $(2^r - 2)$ subgroups of order 2^{r-1}. Then together with $\mathbb{G}$, there are $(2^r - 1)$ such subgroups, while lemma 3.9 proves that only $(2^r - 1)$ exist in $\mathbb{H}$. Therefore, a subgroup $\mathbb{G} \in \Theta$ is α-connected with all other subgroups of Θ, and any subgroup α-connected with $\mathbb{G}$ should be in Θ. Thus, Θ satisfies the requirements of an α-connected set, as given in definition 3.4.

Now assume that for some $\mathbb{G} \in \Theta$, subgroups generated by multiplying for two different values of β, $\mathbb{G}_{\beta_1}$ and $\mathbb{G}_{\beta_2}$, are not distinct. That is;

$$\begin{aligned} \beta_1 \cdot \mathbb{G} &= \beta_2 \cdot \mathbb{G} \\ \beta_2^{-1} \cdot \beta_1 \cdot \mathbb{G} &= \mathbb{G} \end{aligned} \tag{3.38}$$

Let $\beta_2^{-1} \cdot \beta_1 = \alpha^d$, where α is a primitive of $\mathbb{F}_{2^r}$. Also, let $\mathbb{K}$ denote the set of powers of α, of all elements in $\mathbb{G}$ except 0. $\mathbb{K}$ is a sub-set of the ring of integers modulo $(2^r - 1)$, with $(2^{r-1} - 1)$ elements. From the perspective of $\mathbb{K}$, operation $\alpha^d \cdot \mathbb{G}$ is equivalent to $d \oplus \mathbb{K}$, where $\oplus$ is addition modulo $(2^r - 1)$.

Since according to (3.38) $\alpha^d \cdot \mathbb{G} = \mathbb{G}$, $d \oplus \mathbb{K} = \mathbb{K}$. Without loss of generality, assume $\mathbb{K} = \{x_0, x_1, ..., x_{2^{p-1}-2}\}$ is an ordered set, and then, $d \oplus \mathbb{K}$ should be some i'th cyclic shift of $\mathbb{K}$. This results in the following set of relations.

$$x_j \oplus d = x_{j \circ i}; \qquad j = 1, ..., 2^{r-1} - 1 \tag{3.39}$$

$\circ$ denotes the addition modulo $2^{r-1} - 1$. With normal integer addition instead of modular addition, we can express (3.39) as two sets of equations.

$$\begin{aligned} x_{j \circ i} - x_j &= d; \qquad j = 1, ..., 2^{r-1} - 1 - i \\ x_{j \circ i} - x_j + (2^r - 1) &= d; \qquad j = 2^{r-1} - i, ..., 2^{r-1} - 1 \end{aligned} \tag{3.40}$$

Summing up all $(2^{r-1} - 1)$ equations in the two sets of (3.40) results in;

$$d = \frac{i(2^r - 1)}{2^{r-1} - 1} \tag{3.41}$$

Note that d has to be an integer, and that $(2^r - 1)$ and $(2^{r-1} - 1)$ share no common factors. Thus, $i = (2^{r-1} - 1)$, and $d = (2^r - 1)$. This results in $\beta_2^{-1} \cdot \beta_1 = \alpha^{2^r - 1} = 1$, and $\beta_1 = \beta_2$.

Above shows that all subgroups of the form $\beta \cdot \mathbb{G}$ are distinct, and thus, Θ is an α-connected set. □

Lemmas 3.9 and 3.10 show that the proposed subfield expansion in the binary case may be arrived at in a more direct way. The α-connected set used for the binary expansion is larger than for other cases, with cardinality $(2^r - 1)$, which is expected since binary field is the smallest subfield of $\mathbb{F}_{2^r}$. With $|\Theta^2_{r,1}| = (2^r - 1)$, alternative vector representation of a symbol in $\mathbb{F}_{2^r}$ will be a length $(2^r - 1)$ binary vector, while the alternative matrix representation will be of dimensions

$(2^r - 1) \times (2^r - 1)$. According to lemma 3.8, since the subfield is now $\mathbb{F}_2$, matrix representations will be permutation matrices, same as in the extended binary representation.

Comparing this binary expansion with the extended binary representation [40], discussed in section 3.1.2, it can be observed that they produce essentially the same expanded matrix/factor graph. Matrix and vector representations used in both are of the same dimensions, and in both the additional constraints are those of a binary simplex code. In fact, the only difference between the extended representations and the alternative representations over $\mathbb{F}_2$ is in the labeling of rows/columns, which means that one representation can be considered a permutation of the other. This difference in labeling is partly due to the two significantly different approaches used to arrive at the binary expansion. In the following, we derive the binary expansion of the matrix H, in (3.1) of example 3.1, which would show the similarities between the two expansions.

Example 3.3. *We present matrix H, a 3×4 matrix over $\mathbb{F}_{2^3}$, once more.*

$$H = \begin{bmatrix} 0 & \alpha & \alpha^5 & 0 \\ 1 & 0 & \alpha^3 & \alpha^6 \\ \alpha^2 & \alpha^4 & 0 & 1 \end{bmatrix}$$

Table 3.7 lists the quotient groups used for the subfield expansion over $\mathbb{F}_2$, $\mathfrak{Q}^2_{3,1}$, along with the homomorphisms to $\mathbb{F}_2$.

	0	1
$\mathbb{Q}_{\psi_0}$	$\{0, \alpha, \alpha^2, \alpha^4\}$	$\{1, \alpha^3, \alpha^5, \alpha^6\}$
$\mathbb{Q}_{\psi_1}$	$\{0, 1, \alpha^2, \alpha^6\}$	$\{\alpha, \alpha^3, \alpha^4, \alpha^5\}$
$\mathbb{Q}_{\psi_2}$	$\{0, 1, \alpha, \alpha^3\}$	$\{\alpha^2, \alpha^4, \alpha^5, \alpha^6\}$
$\mathbb{Q}_{\psi_3}$	$\{0, \alpha, \alpha^5, \alpha^6\}$	$\{1, \alpha^2, \alpha^3, \alpha^4\}$
$\mathbb{Q}_{\psi_4}$	$\{0, \alpha^3, \alpha^4, \alpha^6\}$	$\{1, \alpha, \alpha^2, \alpha^5\}$
$\mathbb{Q}_{\psi_5}$	$\{0, 1, \alpha^4, \alpha^5\}$	$\{\alpha, \alpha^2, \alpha^3, \alpha^6\}$
$\mathbb{Q}_{\psi_6}$	$\{0, \alpha^2, \alpha^3, \alpha^5\}$	$\{1, \alpha, \alpha^4, \alpha^6\}$

Table 3.7: Quotient Groups in $\mathfrak{Q}^2_{3,1}$

Table 3.8 presents the alternative vector and matrix representations of $\mathbb{F}_{2^3}$ over $\mathbb{F}_2$, derived using the quotient groups listed in Table 3.7.

Comparing Table 3.8 below with Table 3.2, it is easily observable that the alternative vector and matrix representations permuted versions of the extended representations.

0	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{bmatrix}$
1	$\begin{bmatrix}1\\0\\0\\1\\1\\0\\1\end{bmatrix}$	$\begin{bmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix}$
α	$\begin{bmatrix}0\\1\\0\\0\\1\\1\\1\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\end{bmatrix}$
α^2	$\begin{bmatrix}0\\0\\1\\1\\1\\1\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\end{bmatrix}$
α^3	$\begin{bmatrix}1\\1\\0\\1\\0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\end{bmatrix}$
α^4	$\begin{bmatrix}0\\1\\1\\1\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\end{bmatrix}$
α^5	$\begin{bmatrix}1\\1\\1\\0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\end{bmatrix}$
α^6	$\begin{bmatrix}1\\0\\1\\0\\0\\1\\1\end{bmatrix}$	$\begin{bmatrix}0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\end{bmatrix}$

Table 3.8: Alternative Representations of $\mathbb{F}_{2^3}$ over $\mathbb{F}_2$

Local PCM for the expansion, $\mathfrak{H}^2_{3,1}$, which is a generator matrix for the $(7,4)$ Hamming code, is;

$$\mathfrak{H}^2_{3,1} = \begin{bmatrix}1&0&1&1&0&0&0\\1&1&1&0&1&0&0\\0&1&1&0&0&1&0\\1&1&0&0&0&0&1\end{bmatrix} \tag{3.42}$$

But $\mathfrak{H}^2_{3,1}$ above contains 3 cycles of length 4, and since it is a binary matrix, they all can form graphical substructures undesirable for iterative decoding. Thus, following step 5 of the subfield expansion procedure, we carry out some row operations, and derive the following matrix, $\widetilde{\mathfrak{H}}^2_{3,1}$, which is free of 4-cycles, and hence, is more suitable for iterative decoding.

$$\widetilde{\mathfrak{H}}^2_{3,1} = \begin{bmatrix}1&0&1&1&0&0&0\\1&0&0&0&1&1&0\\0&1&1&0&0&1&0\\1&1&0&0&0&0&1\end{bmatrix} \tag{3.43}$$

Using matrix representations in Table 3.8, and the local PCM in (3.43) *above,* H *in* (3.1) *can be converted to the binary matrix* H_b^* *in* (3.44). *In the bottom half of* H_b^*, $\underline{0}$ *denotes a* 4×7 *all-zero matrix.*

$$\widetilde{H}_b = \left[\begin{array}{cccc}
\begin{smallmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{smallmatrix} & \begin{smallmatrix}0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\end{smallmatrix} & \begin{smallmatrix}0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\end{smallmatrix} & \begin{smallmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{smallmatrix} \\
\hdashline
\begin{smallmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{smallmatrix} & \begin{smallmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{smallmatrix} & \begin{smallmatrix}0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\end{smallmatrix} & \begin{smallmatrix}0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\end{smallmatrix} \\
\hdashline
\begin{smallmatrix}0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\end{smallmatrix} & \begin{smallmatrix}0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\1&0&0&0&0&0&0\\0&1&0&0&0&0&0\end{smallmatrix} & \begin{smallmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{smallmatrix} & \begin{smallmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{smallmatrix} \\
\hline
\widetilde{\mathfrak{H}}_{3,1}^2 & \underline{0} & \underline{0} & \underline{0} \\
\hdashline
\underline{0} & \widetilde{\mathfrak{H}}_{3,1}^2 & \underline{0} & \underline{0} \\
\hdashline
\underline{0} & \underline{0} & \widetilde{\mathfrak{H}}_{3,1}^2 & \underline{0} \\
\hdashline
\underline{0} & \underline{0} & \underline{0} & \widetilde{\mathfrak{H}}_{3,1}^2
\end{array}\right] \tag{3.44}$$

Our approach to derive a binary expansion of a matrix/factor graph over $\mathbb{F}_{2^r}$ is different from the one in [40], and this difference allows us to propose a novel binary decoding scheme for the corresponding code. This scheme, as well as the ones based on the extended representation, will be discussed in the next chapter.

3.2.3.2 Square-root Expansion

Consider a code over $\mathbb{F}_{p^r}$, where $r = 2m$. From theorem 2.3, $\mathbb{F}_{p^m}$ is a subfield of $\mathbb{F}_{p^r}$, and it is possible to use the subfield expansion proposed earlier to expand the PCM or the factor graph of the code to a matrix or graph over $\mathbb{F}_{p^m}$. As $\sqrt{p^r} = \sqrt{p^{2m}} = p^m$, we refer to this expansion as the *square-root expansion*.

It is possible to derive the square-root expansion by following the general procedure outlined at the end of section 3.2.2. But it turns out that, for this case, there is an easier method to find the smallest α-connected set, $\Theta^p_{2m,m}$. Lemma 3.10 below outlines this alternative method.

Lemma 3.11. *Consider $\mathbb{H} = \{\mathbb{F}_{p^{2m}}, +\}$, and $\mathbb{G} = \{\mathbb{F}_{p^m}, +\}$. $\mathbb{F}_{p^m}$ is a subfield of $\mathbb{F}_{p^{2m}}$, and let α denote a primitive element of $\mathbb{F}_{p^{2m}}$. The smallest α-connected set containing subgroups of order p^m, of $\mathbb{H}$, can be generated by multiplying $\mathbb{G}$ with different α^i's.*

Proof. Let $i^* = \frac{p^{2m}-1}{p^m-1} = (p^m + 1)$. According to lemma 3.5, subfield $\mathbb{F}_{p^m}$ consists of the set of elements generated by α^{i^*}, which we denote $\mathbb{S}_{i^*}$, and 0. Powers of α in the elements of $\mathbb{S}_{i^*}$ form a subgroup of the ring of integers modulo $(p^{2m} - 1)$. Let this subgroup be $\mathbb{K}_{i^*}$.

According to lemma 3.1, $\alpha^i \cdot \mathbb{G}$, where $\alpha^i \in \mathbb{F}_{p^{2m}}$, produces a subgroup of order p^m, of $\mathbb{H}$. While $0 \in \mathbb{G}$, $\alpha^i \cdot 0 = 0$. From the perspective of the remaining elements of $\mathbb{G}$, which are those in $\mathbb{S}_{i^*}$, operation $\alpha^i \cdot \mathbb{G}$ is akin to $i \oplus \mathbb{K}_{i^*}$, where $\oplus$ denotes addition modulo $(p^{2m} - 1)$. Thus, the number of different subgroups that can be generated by multiplying $\mathbb{G}$ with α^i is equal to the number of unique sets $i \oplus \mathbb{K}_{i^*}$.

Note that if the elements of $\mathbb{K}_{i^*}$ are arranged in increasing order, then the gap between two consecutive elements and the first and the last is $(p^m + 1)$. Therefore, $(p^m + 1) \oplus \mathbb{K}_{i^*} = \mathbb{K}_{i^*}$, and similarly, for any pair of non-negative integers i, j, $(i + j(p^m + 1)) \oplus \mathbb{K}_{i^*} = i \oplus \mathbb{K}_{i^*}$. Thus, $i \oplus \mathbb{K}_{i^*}$ is can only be unique for $i = 1, \ldots, p^m$. Also taking into account $\mathbb{K}_{i^*}$, there are only $(p^m + 1)$ unique sets $i \oplus \mathbb{K}_{i^*}$, which means the number of unique subgroups of the form $\alpha^i \cdot \mathbb{G}$ is also $(p^m + 1)$. Let this set of subgroups be $\theta^p_{2m,m}$.

Now from lemma 3.3, minimum possible cardinality of an α-connected set containing subgroups of order p^m, of $\mathbb{H}$, is $\frac{p^{2m}-1}{p^m-1} = (p^m + 1)$. Since cardinality of $\theta^p_{2m,m}$ is $(p^m + 1)$, it is such a set.

□

With $\Theta^p_{2m,m}$ generated as in lemma 3.11, the remaining steps of the subfield expansion can be followed to expand a matrix/factor graph over $\mathbb{F}_{p^{2m}}$ to one over $\mathbb{F}_{p^m}$. In fact, the example we developed through the first half of this section, with $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$, is an instance of the square-root expansion.

$\mathbb{F}_{p^m}$ is the largest possible subfield of $\mathbb{F}_{p^{2m}}$. According to theorem 2.3, this is the case where the subfield order, relative to the field order, is at its largest. Consequently, size of the smallest α-connected set is at its lowest in the square-root expansion. While the relatively large size of the

subfield limits the achievable complexity gain, performance loss in this case is the smallest among all subfield expansions, as can be seen in chapter 5.

In the following, we provide a short example of the square-root expansion, once more with $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$.

Example 3.4. *Consider the following matrix H, over $\mathbb{F}_{2^4}$, where α is a primitive element.*

$$H = \begin{bmatrix} \alpha & \alpha^5 & 0 \\ 0 & 1 & \alpha^3 \end{bmatrix} \tag{3.45}$$

H can be represented with the following factor graph, where squares represent rows, or check nodes, and circles represent columns, or variable nodes.

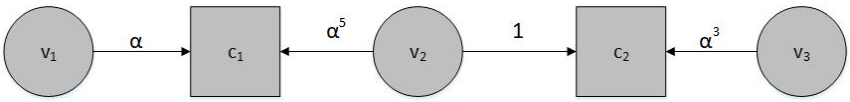

Figure 3.3: Original Factor Graph of H

To derive the expansion over $\mathbb{F}_{2^2}$ of H, we use the set of quotient groups, $\mathfrak{Q}^2_{4,2}$, in Table 3.4, and the alternative matrix representations in Table 3.5. Since $\mathfrak{H}^2_{4,2}$ in (3.34) *does not contain any sub-matrices that does not have full rank, it will be used to represent the local code.*

Expanded factor graph of H, constructed in this manner, is presented in Figure 3.4, while the expanded matrix, H^*, is given in (3.46)*. Once more, in the figure, squares represent check nodes, circles variable nodes, and hexagons local check nodes. $v_{n,i}$ and $c_{m,i}$ denote, respectively, the $\mathbb{F}_{2^2}$ nodes of the n'th variable node and the m'th check node of Figure 3.3 that compute the CPV corresponding to $\mathbb{Q}_{\psi_i}$ of Table 3.4. Local parity-check equation represented by the i'th row of $\mathfrak{H}^2_{4,2}$ (in of* (3.34)*), for the set of nodes of the n'th variable node of Figure 3.3, is denoted with $L_{n,i}$.*

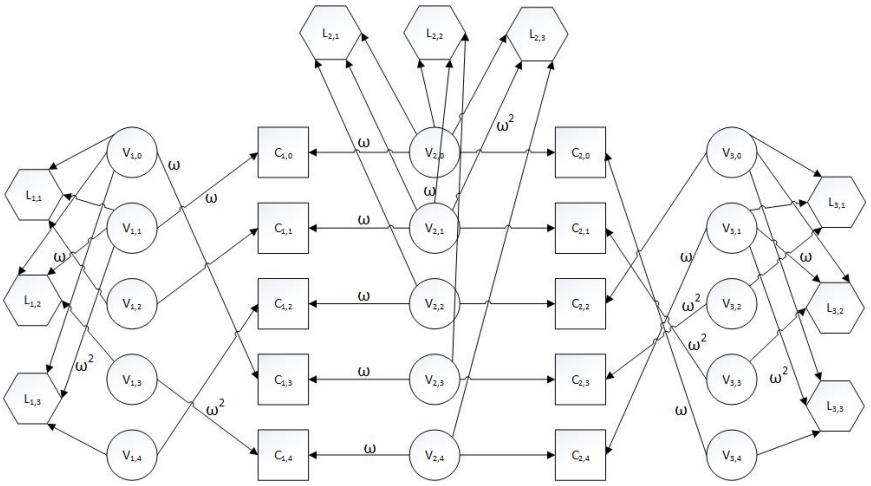

Figure 3.4: Expanded Factor Graph of H

$$H^* = \left[\begin{array}{ccc}
\begin{matrix} 0&0&0&\omega&0 \\ \omega&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&0&0&\omega^2 \\ 0&0&1&0&0 \end{matrix} & \begin{matrix} \omega&0&0&0&0 \\ 0&\omega&0&0&0 \\ 0&0&\omega&0&0 \\ 0&0&0&\omega&0 \\ 0&0&0&0&\omega \end{matrix} & \begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} \\
\hline
\begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} & \begin{matrix} 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{matrix} & \begin{matrix} 0&0&1&0&0 \\ 0&0&0&0&\omega \\ 0&0&0&\omega^2&0 \\ 0&\omega^2&0&0&0 \\ \omega&0&0&0&0 \end{matrix} \\
\hline
\begin{matrix} 1&1&1&0&0 \\ 1&\omega&0&1&0 \\ 1&\omega^2&0&0&1 \end{matrix} & \begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} & \begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} \\
\hline
\begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} & \begin{matrix} 1&1&1&0&0 \\ 1&\omega&0&1&0 \\ 1&\omega^2&0&0&1 \end{matrix} & \begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} \\
\hline
\begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} & \begin{matrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{matrix} & \begin{matrix} 1&1&1&0&0 \\ 1&\omega&0&1&0 \\ 1&\omega^2&0&0&1 \end{matrix}
\end{array}\right] \tag{3.46}$$

Chapter 4

Decoding Based on Expanded Graphs

In the last chapter, we presented overviews of the two most well-known binary expansions of NB-LDPC codes, the binary image and the extended binary representation, along with a novel approach that allows expanding a code over any subfield of the field used in construction. Major purpose of these expansions, or representing a code over a smaller field, is to use in decoding, so that the complexity can be reduced.

Since decoding complexity is directly dependant on the size of the field used in decoding, representing codes over a smaller field can offer significant advantages. But the PCM, or the factor graph, produced by an expansion may have some features that make it unsuitable for using with an existing decoding algorithm. Therefore, any algorithm will have to be modified in certain ways before using it on an expanded PCM/factor graph. Focus of this chapter is on such decoding schemes.

In the literature, it is well established that the most harmful graph substructures for iterative decoding are short cycles [46], [47]. As was briefly discussed in Chapter 3, short cycles could be created in graph expansions even though none exist in the original graph. Interestingly, when it comes to NB-LDPC codes, the impact of these structures can also depend on the associated edge labels. In order to better understand the various decoding schemes discussed in this Chapter, all of which use different strategies to mitigate the effects of cycles, some background knowledge on them and their impact is essential.

In the following, we first present an overview of short cycles, undesirable graph structures created by them, and their impact on iterative decoding. As this dissertation mainly focuses on NB-LDPC codes, non-binary cycles and associated structures are discussed in more detail. We also provide a summary of decoding strategies that have been suggested in the literature for using with

binary images and extended binary representations. Then we introduce a novel approach for modifying any iterative decoding algorithm of NB-LDPC codes for using with the subfield expansions proposed in the last chapter. While this decoding strategy does not lose the complexity advantage of decoding on a smaller subfield, performance losses are minimal, as the presented simulation results would show. We give more focus to decoding on the binary expansion, since that offers the highest complexity reduction, and also propose a novel majority logic decoding algorithm on that expansion, which manages to outperform existing MLgD algorithms in many cases.

4.1 Short Cycles in NB-LDPC Codes

As discussed in Chapter 2, decoding of LDPC codes is generally viewed from the perspective of the factor graph representations of their PCMs. The sparseness of these matrices lead to factor graphs that are almost free of cycles; the primary reason for the exceptional performance of LDPC codes with iterative decoding algorithms. In [46], it is proved that such decoding algorithms, when implemented on cycle-free graphs, will produce the same estimates as MAP decoding. While it is not practical to construct codes with PCMs that produce cycle-free factor graphs, when it comes to LDPC codes, the sparse PCMs ensure that very few cycles will exist in the said graph. In this case, it is understood that iterative algorithms will closely approximate MAP decoding [46].

Performance of LDPC codes is closely tied to the lengths and number of cycles present in the factor graphs, with shorter cycles and higher cycle counts widening the gap between performance of iterative and MAP decoding. When constructing LDPC codes, particular focus is placed on increasing the length of the shortest cycle, otherwise known as the *girth* of the code [48]. All constructions, random or structured, ensure that no cycle of length four exists, and most try to avoid length six cycles as well [49], [50].

Cycles in a factor graph particularly impact decoding at low bit error rates. There, cycles would cause the respective curve to flatten, and an increase in bit energy would not result in an equivalent decrease of BER. This phenomenon is known as the *error floor* in the literature [13], and codes with shorter cycles have been observed to exhibit error floors at relatively high BERs. In [13], authors show that the error floors are not primarily caused by cycles, but are rather due to the graphical sub-structures created by them, such as trapping sets and stopping sets. However, as removing short cycles ensure that such undesirable sub-structures are not created, primary focus in construction of LDPC codes is still to increase the girth.

When it comes to NB-LDPC codes, where a number of different non-zero edge labels are possible, effects of cycles become a bit more complicated. Edge labels, or rather the relationships between them, determine to a level how harmful a short cycle could be for iterative decoding [14], [18]. Similar to in the binary domain, here also it is the graph sub-structures created by the short cycles that are actually harmful, and in [51], it has been observed that the most harmful type of sub-structure depends on the channel being used. For the most commonly used AWGN channel, the most harmful structure is known as a *non-binary absorbing set* [14], [51]. These absorbing sets are created by short cycles where the respective edge labels create a sub-matrix in the PCM that is rank deficient [14]. In the literature, such cycles are sometimes referred to as cycles not satisfying the full rank condition (non-FRC), and as expected, absorbing sets created by shorter non-FRC cycles are known to be particularly harmful for iterative decoding. This relationship between edge labels and decoding performance has added another degree of flexibility to the construction of NB-LDPC codes, which has been successfully explored in [15], [17].

In the following, we first give a brief overview of the graphical sub-structures considered harmful for iterative decoding of NB-LDPC codes. As the AWGN channel is the most widely used channel model in the field, we give particular focus to non-binary absorbing sets. Then we explore how these undesirable graph structures are changed by the expansions discussed in Chapter 3.

4.1.1 Harmful Graph Structures for Iterative Decoding

The existence of graphical sub-structures that negatively impact iterative decoding was first noticed in [47], when analyzing performance of LDPC codes over the binary erasure channel (BEC). In that paper, authors presented the definition of a *stopping set*, the structure primarily responsible for the error floors of LDPC codes on the BEC. For the sake of completeness, we present this definition in the following.

Definition 4.1. *Let $\mathfrak{G}$ be the factor graph of a binary LDPC code, and $\mathcal{V}$ be the set of variable nodes. A **stopping set** $\mathcal{S}$ is a subset of $\mathcal{V}$ such that all neighboring check nodes of $\mathcal{S}$ are connected to $\mathcal{S}$ at least twice.*

The subgraph induced by a stopping set consisting of three variable nodes, $v1$, $v2$ and $v3$, is given in Figure 4.1. In this subgraph, each of the three neighboring check nodes, $c1$, $c2$ and $c3$, is connected to exactly two of the three variable nodes.

In the literature, it is accepted that performance of LDPC codes over the BEC is completely determined by stopping sets, with smaller ones, containing a lesser number of nodes, lead-

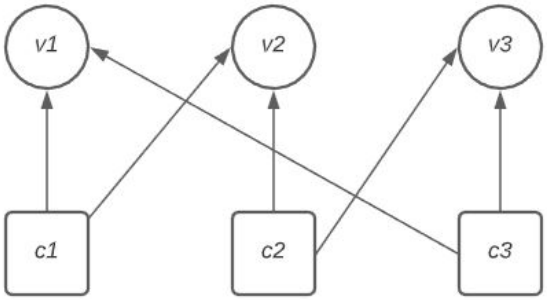

Figure 4.1: Factor Graph of a Stopping Set

ing to higher error floors [52]. But when it comes to the binary symmetric channel (BSC) and the AWGN channel, the structures known as trapping sets are known to be more harmful [52]. Existence of trapping sets was first noticed in [53], when investigating the reason for the high error floor of LDPC codes from the Margulis construction. In [53], trapping sets are actually referred to as *near codewords*. The term trapping set was formally introduced in [13], which presents a deeper exploration of the effects of trapping sets on iterative decoding. We present the definition of a trapping set in the following.

Definition 4.2. *Let $\mathfrak{G}$ be the factor graph of a binary LDPC code, and $\mathcal{V}$ be the set of variable nodes. Also, let $\mathcal{S}$ be a subset of $\mathcal{V}$, and $\mathfrak{G}_\mathcal{S}$ be the subgraph induced by $\mathcal{S}$. $\mathcal{O}(\mathcal{S})$ and $\mathcal{E}(\mathcal{S})$ denote, respectively, the sets of check nodes of $\mathfrak{G}_\mathcal{S}$ connected an odd number of times and an even number of times to $\mathcal{S}$. Then, $\mathcal{S}$ is an (a, b)* ***trapping set*** *if $|\mathcal{S}| = a$ and $|\mathcal{O}(\mathcal{S})| = b$.*

Figure 4.2 presents the factor graph of a $(3, 1)$ trapping set. The impact of these structures can be more easily understood from the perspective of hard decision based bit-flipping decoding. Observe that when using such an algorithm, if only the variable nodes in the trapping set in Figure 4.2 are received in error, then check nodes $c2$ and $c3$ will be satisfied, as they are connected an even number of times to erroneous nodes. $c1$ will be the only unsatisfied node, but its estimates might not be enough to correct the error in $v1$, as $v1$ is also connected to $c3$, which is satisfied.

The sub-class of the most harmful trapping sets for iterative decoding is known as *absorbing sets* [14]. We present the definition of a binary absorbing set in the following, as a continuation of definition 5.2.

Definition 4.3. *$\mathcal{S}$ is an (a, b)* ***absorbing set*** *if $\mathcal{S}$ is an (a, b) trapping set, and each node in $\mathcal{S}$ has strictly more neighbors in $\mathcal{E}(\mathcal{S})$ than in $\mathcal{O}(\mathcal{S})$.*

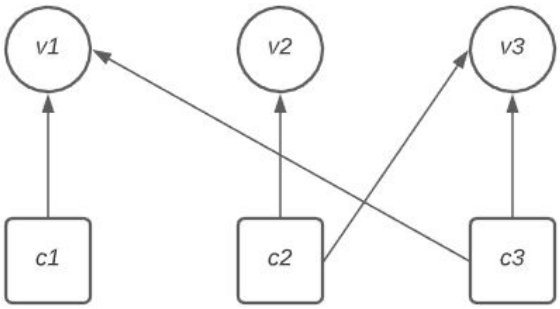

Figure 4.2: Factor Graph of a (3, 1) Trapping Set

Note that the additional constraint of each node in $\mathcal{S}$ having more neighboring check nodes in $\mathcal{E}(\mathcal{S})$ than in $\mathcal{O}(\mathcal{S})$ makes it even harder to recover from a scenario where all variable nodes in the absorbing set are in error. For example, consider the absorbing set in Figure 4.3, under bit-flipping decoding. There, if only the nodes in the set are in error, check nodes $c2, c3$ and $c4$ will still be satisfied, since each has two erroneous nodes connected. $c1$ will not be satisfied, but it cannot correct the error in $v1$, as its estimate will be *out-voted* by those of $c2$ and $c3$.

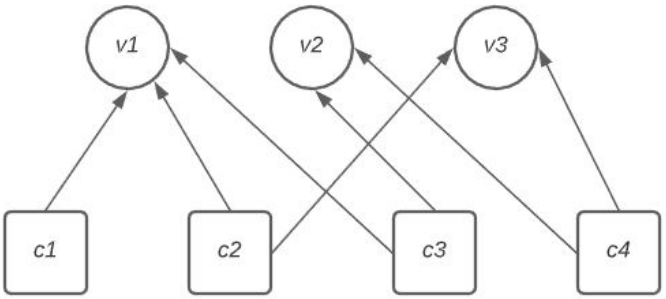

Figure 4.3: Factor Graph of a (3, 1) Absorbing Set

Above insights on trapping and absorbing sets make it easier to understand how they may create an error floor in decoding. If all the bits, or variable nodes, belonging to such a set are received in error, then the decoder would find it a very difficult state to recover from. Of course, in the case of bit-flipping decoding and absorbing sets, such errors would be unrecoverable. For the sake of brevity, we refer to these kind of errors as *set errors* from hereon. Increasing bit energy reduces the likelihood of all errors, including these, but whenever a set error occurs, the code would be unable to correct it. Therefore, at high bit energy, the slope of the bit error rate curve would reflect the slowly decreasing probability of set errors occurring, and when considered relative to

other parts, the curve would look almost parallel to the horizontal axis. This is the phenomenon referred to as the error floor.

If the number of variable nodes in a trapping or absorbing set is relatively small, then there is a higher probability of a corresponding set error occurring. Therefore, in the literature, smaller trapping/absorbing sets are considered more harmful for decoding [52], and more focus is given to removing those when designing LDPC codes. Small trapping/absorbing sets are most often of an elementary form, with the neighboring check nodes connected to one or two of the nodes in the set, and are sometimes referred to as *elementary trapping/absorbing sets* [52], [14]. Figure 4.2 and Figure 4.3 both represent such elementary sets.

Different from the binary counterparts, in NB-LDPC codes, an edge label may take one of many possible values. Then, for a graph sub-structure to affect decoding in a similar way to a binary trapping/absorbing set, certain conditions have to be met with regard to edge labels, in addition to the more topographical requirements given in Definitions 4.2 and 4.3. It has been observed that the performance of NB-LDPC codes on the AWGN channel is mostly determined by the existence of non-binary absorbing sets [14]. In the following, we present the definition of such a set.

Definition 4.4. *Let H and $\mathfrak{H}$ be, respectively, the PCM and the factor graph of a NB-LDPC code, and $\mathcal{V}$ be the set of variable nodes. Also, let $\mathcal{S}$ be a subset of $\mathcal{V}$, of cardinality 'a', and $\underline{S}$ be the $m \times a$ sub-matrix containing the columns of H that correspond to the nodes in $\mathcal{S}$. For $\mathcal{S}$ to be an (a, b) absorbing set;*

1. *$\underline{S}$ could be partitioned into two sub-matrices $\underline{E}$ and $\underline{O}$, of dimensions $(m-b) \times a$ and $b \times a$ respectively, such that $\underline{E}$ is not of full rank but $\underline{O}$ is.*

2. *Each node in $\mathcal{S}$ should be connected to strictly more nodes that correspond to $\underline{E}$ than those that correspond to $\underline{O}$.*

In the above definition, the sub-matrix $\underline{E}$ corresponds to the set of check nodes in the sub-graph induced by $\mathcal{S}$ that will be satisfied by certain configurations of non-zero values of the variable nodes in $\mathcal{S}$. $\underline{O}$ represents the remaining, un-satisfied check nodes. Thus, $\underline{E}$ and $\underline{O}$ are analogous to the sets $\mathcal{E}(\mathcal{S})$ and $\mathcal{O}(\mathcal{S})$ in Definition 4.3. First requirement above is simply the existence of some satisfied check nodes, which is represented by $\underline{E}$ not being of full rank. Second requirement is the same as that of a binary absorbing set; each variable node has to be connected to more satisfied check nodes than un-satisfied ones.

Factor graph of a $(3, 1)$ absorbing set is presented in Figure 4.4, along with the relationship between edge labels w_i. Topographically, it is of the same structure as the binary absorbing set in Figure 4.3, but unless the given relationship is satisfied, it would not be considered an absorbing set in a non-binary setting. Certain configurations of values for $v1, v2$ and $v3$ would satisfy the check nodes $c2, c3$ and $c4$, while $c1$ would remain un-satisfied. Observe that all check nodes in Figure 4.4 are of degree one or two, and as such, the absorbing set is an elementary one [14]. Similar to with binary codes, small absorbing sets, which are quite often also elementary, are the most harmful for iterative decoding even in the non-binary domain.

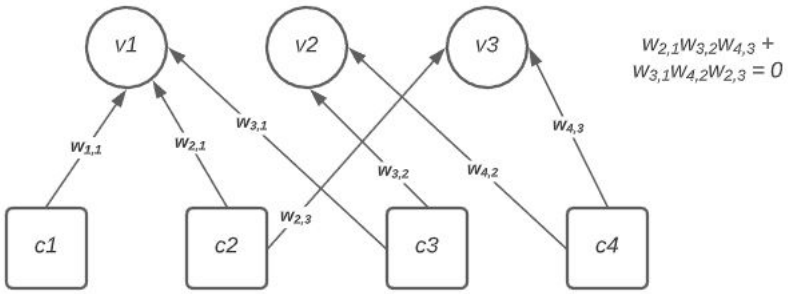

Figure 4.4: Factor Graph of a (3, 1) Non-binary Absorbing Set

A binary trapping set may not always involve a short cycle, and Figure 4.2 is an example of this. Still, a short cycle would always create a trapping/absorbing set, and it is generally considered that such structures containing short cycles are more detrimental to iterative decoding [18]. Because of this, as we explained earlier as well, significant focus is still placed on increasing the girth when designing LDPC codes. With NB-LDPC codes though, not all short cycles may create an absorbing set. Ones that do create such structures will have to be labelled with values that satisfy the relationship involving the edge labels. For example, in Figure 4.4, the $(3, 1)$ absorbing set contains a cycle of length 6, involving variable nodes $v1, v2, v3$ and check nodes $c2, c3, c4$, and the relationship given is between the edge labels of that 6-cycle. From the perspective of the PCM, this relationship may be viewed as the condition to be satisfied for the sub-matrix induced by the 6-cycle to be rank deficient. As such, only short cycles that correspond to rank deficient sub-matrices of the PCM can create non-binary absorbing sets. Also, in [15] it is shown that these non-FRC cycles lead to the code having codewords of a low binary weight, which is another cause of error floor. Because of this, more focus is given to limiting the non-FRC cycles when designing NB-LDPC codes, and [14] reports significant gains by removing such cycles by changing the edge labels. Still, it should be noted that short cycles of any type are undesirable when it comes to iterative decoding, as they

create correlations between messages passed along the graph [54].

FRC and non-FRC cycles play a prominent role in the remainder of this Chapter, and also in Chapter 5, where the distinction between the two types is used to design parity-check matrices for soft decoding of Reed-Solomon codes. Definition 4.5 below formally defines these structures.

Definition 4.5. *Let* $\mathfrak{c}$ *be a cycle in the factor graph* $\mathcal{G}$ *of an NB-LDPC code defined by the parity-check matrix* H*. Also, let* $\mathcal{V}$ *and* $\mathcal{C}$ *denote the sets of variable and check nodes involved in* $\mathfrak{c}$*. We denote the sub-matrix of* H *that corresponds to* $\mathfrak{c}$ *with* $H_{\mathfrak{c}}$*.* $H_{\mathfrak{c}}$ *contains the labels of the edges that connect nodes in* $\mathcal{V}$ *to those in* $\mathcal{C}$*. Then;*

1. $\mathfrak{c}$ *is called a cycle that does not satisfy the full rank condition, or a* **non-FRC cycle***, if* $H_{\mathfrak{c}}$ *is rank deficient.*

2. $\mathfrak{c}$ *is called a cycle that satisfies the full rank condition, or an* **FRC cycle***, if* $H_{\mathfrak{c}}$ *is of full rank.*

In the following section, we observe how short cycles of both types get transformed from the expansions discussed in Chapter 3. This section was primarily for establishing the necessary theoretical background for that and subsequent sections, by providing brief overviews of the related concepts. A deeper discussion of graph sub-structures in NB-LDPC codes is presented in [14] and [51].

4.1.2 Short Cycles in Expanded Graphs

As discussed in Chapter 3, it is possible to represent a non-binary factor graph over one of its subfields, which may allow decoding schemes of lower complexities to be designed. However, the topographical changes introduced by an expansion may form undesirable graph sub-structures that were not present in the original graph. If such structures existed in the original graph, then there is also a chance of those being removed by the expansion. Since most undesirable graph structures are created due to cycles, in this section, we explore how graph expansions create or eliminate short cycles.

Chapter 3 presented three approaches for expanding a factor graph; the binary image representation and the extended binary representation to derive a binary graph, and our proposal, subfield expansions, that allow expanding a graph over any subfield of the original field. Later in the chapter it was shown that extended binary representation produces the same factor graph as the subfield expansion over the binary subfield. Therefore, here we focus only on the binary image representation and subfield expansions.

4.1.2.1 Cycles in Binary Image Representations

Binary image representation makes use of the matrix representation of a finite field, where each element over $\mathbb{F}_{2^r}$ is represented as an $r \times r$ binary matrix. Here, it is possible that some field elements are represented with dense matrices, sometimes even containing four-cycles. Thus, the binary image representation of a non-binary factor graph will contain many short cycles, even though the original graph may not have any. Example 3.1, in Chapter 3, provides evidence of this. There, the original graph contains a single four-cycle, whereas several exist in the binary expansion.

Since the expanded graph is binary, effects of a cycle depend only on its length and any can create structures such as stopping sets, trapping sets, or absorbing sets. Therefore, a binary image representation may contain several graph sub-structures harmful for iterative decoding, and in the literature, binary image representations are not directly used for decoding. Several techniques are usually applied to mitigate the effects of short cycles and associated stopping/trapping sets, such as changing the binary graph per decoding iteration [43], adding redundant binary check nodes [42], [55], and using message passing schedules different from the conventional one [56]. Section 4.2.1 presents a more detailed review of decoding with binary image representations.

4.1.2.2 Cycles in Subfield Expansions

Subfield expansions, introduced in Chapter 3, allow expanding an $\mathbb{F}_{p^r}$ graph over any subfield $\mathbb{F}_{p^m}$, where $m \mid r$. As explained in that chapter, such an expansion can be viewed as consisting of two principal stages; initial stage where variables and check nodes of the $\mathbb{F}_{p^r}$ graph are expanded and connected, and then the second stage, where the additional check nodes necessary from a decoding perspective are added. However, all the modifications to short cycles that exist in the original graph are captured in the initial stage, and therefore, in the following, our focus is primarily on that.

Subfield expansion of a $\mathbb{F}_{p^r}$ graph over $\mathbb{F}_{p^m}$ makes use of the alternative matrix representation of $\mathbb{F}_{p^r}$, where each element is represented as a $\frac{p^r-1}{p^m-1} \times \frac{p^r-1}{p^m-1}$ matrix over $\mathbb{F}_{p^m}$. This representation is introduced in Definition 3.7, and it has a unique structure that is advantageous in terms of the density of the expanded graph. According to Lemma 3.8, there is exactly a single non-zero position in each row and column of the alternative matrix representation of any symbol $\in \mathbb{F}_{p^r}$. Thus, different from the binary image representation, no cycle of any length can exist in any of the matrices used for the expansion. However, any short cycle that exists in the original graph may create one or more such cycles in the expanded graph.

A subfield expansion may be carried over a non-binary subfield as well, and in such an instance, two types of short cycles need to be considered. As discussed earlier, the first type, which we refer to as non-FRC cycles, can create non-binary absorbing sets, and the second, FRC cycles, cannot, but still negatively impact decoding by making the messages passed along the graph correlated. However, in the following lemma, which presents the conditions for a short cycle to exist in a subfield expansion, we do not distinguish between the two types. Also, it should be noted that although the Lemmas presented in this section focus on four-cycles, they may be generalized to different lengths.

Lemma 4.1. *Consider the expansion of a $\mathbb{F}_{p^r}$ graph over $\mathbb{F}_{p^m}$, where $m \mid r$. After the initial stage, a four-cycle may exist in the expanded graph only if there existed a non-FRC four-cycle between the corresponding positions of the original graph.*

Proof. Assume that a four-cycle exists between four $\mathbb{F}_{p^m}$ nodes in the expansion, one each from the sets that replaced variable nodes $v1, v2$, and check nodes $c1, c2$ in the original graph. It is clear that for this to happen, a four-cycle must exist in the original graph between the same nodes. As it makes identification of non-FRC cycles easier, we chose to view the expansion from the perspective of the PCM. (4.1) presents the four cycle in the original PCM, where the corresponding edge labels are a_1, a_2, b_1 and b_2.

$$\begin{array}{c} \\ c1 \\ c2 \end{array} \begin{array}{c} \begin{array}{ccccc} & v1 & & v2 & \end{array} \\ \begin{pmatrix} \dots & a_1 & \dots & b_1 & \dots \\ \dots & a_2 & \dots & b_2 & \dots \end{pmatrix} \end{array} \tag{4.1}$$

How the cycle exists in the expanded PCM is given in Figure 4.5, where it is shown in red, and the alternative representation matrices that replaced the four elements a_1, a_2, b_1 and b_2 in the original PCM are denoted with black squares. Moreover, the four-cycle is assumed to involve the (i, k)'th element of the alternative matrix representation of a_1, the (j, k)'th element of a_2, the (i, l)'th position of b_1, and the (j, l)'th position of b_2.

Now, according to definition 3.7, relationships in (4.2) should hold. ψ_i's are the homomorphisms between the additive groups of $\mathbb{F}_{p^r}$ and $\mathbb{F}_{p^m}$ that are used for the subfield expansion.

$$\begin{aligned} a_1 \cdot \ker(\psi_i) &= \ker(\psi_k) & \qquad a_2 \cdot \ker(\psi_j) &= \ker(\psi_k) \\ b_1 \cdot \ker(\psi_i) &= \ker(\psi_l) & \qquad b_2 \cdot \ker(\psi_j) &= \ker(\psi_l) \end{aligned} \tag{4.2}$$

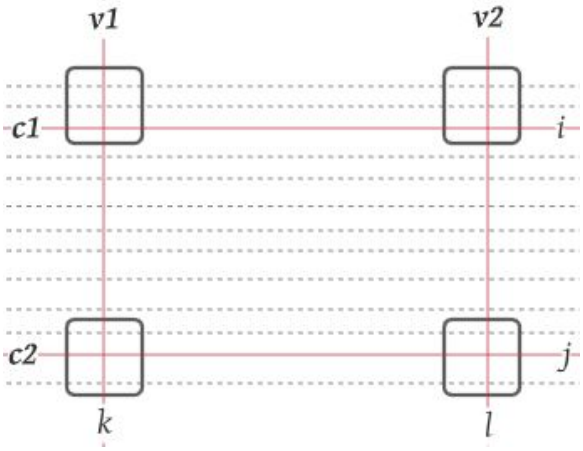

Figure 4.5: A four-cycle in a subfield expansion

Simplifying the above leads to the following relationships.

$$\begin{aligned} a_1 \cdot \ker(\psi_i) &= a_2 \cdot \ker(\psi_j) \\ b_1 \cdot \ker(\psi_i) &= b_2 \cdot \ker(\psi_j) \end{aligned} \tag{4.3}$$

Taking the division of the two equations in (4.3) allows us to arrive at the following relationship between the edge labels.

$$\frac{a_1 \cdot \ker(\psi_i)}{b_1 \cdot \ker(\psi_i)} = \frac{a_2 \cdot \ker(\psi_j)}{b_2 \cdot \ker(\psi_j)} \Longrightarrow \frac{a_1}{b_1} = \frac{a_2}{b_2} \tag{4.4}$$

Now consider the determinant $\mathfrak{D}$ of the sub-matrix induced by the four-cycle in the original PCM, given in (4.1).

$$\mathfrak{D} = a_1 \cdot b_2 - a_2 \cdot b_1 \tag{4.5}$$

But due to (4.4), $\mathfrak{D} = 0$. Thus, the four-cycle in the original matrix is a non-FRC cycle. □

Lemma 4.1 establishes that only a non-FRC cycle in the original graph can lead to a cycle in the expanded graph. But it is possible that such a cycle creates more than a single cycle in the expansion, which is explored in the following lemma.

Lemma 4.2. *Consider the expansion of a $\mathbb{F}_{p^r}$ graph over $\mathbb{F}_{p^m}$, where $m \mid r$. A non-FRC four-cycle in the original graph will create $\frac{p^r-1}{p^m-1}$ four-cycles in the expanded graph.*

Proof. We recall that in the subfield expansion, each element of the original PCM is replaced with a $\frac{p^r-1}{p^m-1} \times \frac{p^r-1}{p^m-1}$ matrix over $\mathbb{F}_{p^m}$. According to Lemma 3.8, these matrices have only a single non-zero entry per row or column. Then, if $\frac{p^r-1}{p^m-1}$ four-cycles are to be created, every row of a matrix that

replaced one of the elements in the original cycle should be involved in a four-cycle. Once more, we use (4.1) to represent the original cycle.

Consider the i'th row of the matrix that replaced a_1, and assume its non-zero entry is on the k'th column, similar to in Figure 4.5. By Lemma 3.8, i'th row of b_1 must have a non-zero entry, and let that be in the l'th column. Similarly, let the non-zero entry in the k'th column of a_2 be at the position corresponding to row j. Note that $0 \leq i, j, k, l \leq \frac{p^r-1}{p^m-1} - 1$. This configuration completes two sides of the four-cycle, and three of the relationships in (4.2), which are given below, become valid.

$$a_1 \cdot \ker(\psi_i) = \ker(\psi_k) \qquad a_2 \cdot \ker(\psi_j) = \ker(\psi_k) \qquad b_1 \cdot \ker(\psi_i) = \ker(\psi_l) \quad (4.6)$$

For a four-cycle, it is required that (j, l)'th position of the matrix that replaced b_2 is non-zero. According to Definition 3.7, the non-zero position of the j'th row will be on the column t, $0 \leq t \leq \frac{p^r-1}{p^m-1} - 1$, that satisfies the following condition.

$$b_2 \cdot \ker(\psi_j) = \ker(\psi_t) \quad (4.7)$$

Substituting for the RHS of (4.7) with the relationships in (4.6);

$$\frac{b_2 \cdot a_1}{a_2} \cdot \ker(\psi_i) = \ker(\psi_t) \quad (4.8)$$

The four-cycle in the original PCM is non-FRC, and the edge labels satisfy (4.4). Using that with (4.8);

$$b_1 \cdot \ker(\psi_i) = \ker(\psi_t) \quad (4.9)$$

But from (4.6);

$$\ker(\psi_l) = \ker(\psi_t) \quad (4.10)$$

Thus, the non-zero entry of the j'th row of b_2 is on the l'th column, which completes the four-cycle. Since no special constraints were placed on the row i, above proof is valid for any of the $\frac{p^r-1}{p^m-1}$ rows of a_1. This proves that the single non-FRC four cycle in the original PCM will create $\frac{p^r-1}{p^m-1}$ four-cycles in the subfield expansion. □

It turns out that any four-cycle created in the subfield expansion during the initial stage is non-FRC. This is proved in Lemma 4.3.

Lemma 4.3. *Consider the expansion of a $\mathbb{F}_{p^r}$ graph over $\mathbb{F}_{p^m}$, where $m \mid r$. Any four-cycle that exists in the expansion after the initial stage does not satisfy the full rank condition.*

Proof. As established in Lemma 4.1, a four-cycle will only exist in the expansion if there existed a non-FRC four-cycle between the corresponding positions of the original graph. In such a scenario, edge labels of the original graph should satisfy (4.4). Using that relationship, we represent the original four-cycle as follows, where $c \in \mathbb{F}_{p^r}$.

$$\begin{array}{c} \\ c1 \\ c2 \end{array} \begin{array}{c} \begin{array}{ccccc} & v1 & & v2 & \\ \end{array} \\ \begin{pmatrix} \dots & a_1 & \dots & b_1 & \dots \\ \dots & c \cdot a_1 & \dots & c \cdot b_1 & \dots \end{pmatrix} \end{array} \tag{4.11}$$

We focus on one of the four-cycles created in the expansion, involving positions (i, k) of the matrix that replaced a_1, (j, k) of $c \cdot a_1$, (i, l) of b_1, and (j, l) of $c \cdot b_2$. While this four-cycle is graphically represented in Figure 4.5, here we chose to view it as follows, where $\gamma_1, \gamma_2, \delta_1, \delta_2 \in \mathbb{F}_{p^m}$.

$$\begin{array}{c} \\ c1, i \\ \\ c2, j \end{array} \begin{array}{c} \begin{array}{ccccc} & v1, k & & v2, l & \\ \end{array} \\ \begin{pmatrix} \dots & \gamma_1 & \dots & \delta_1 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \gamma_2 & \dots & \delta_2 & \dots \end{pmatrix} \end{array} \tag{4.12}$$

According to Definition 3.7, γ_1 is the element that induces the reverse permutation of the permutation of $\mathbb{Q}_{\psi_k}$ produced by operation $a_1 \odot \mathbb{Q}_{\psi_i}$. Similarly, γ_2 should reverse the permutation of $\mathbb{Q}_{\psi_k}$ produced by $a_2 \odot \mathbb{Q}_{\psi_j}$, or with the representation in (4.11), $(c \cdot a_1) \odot \mathbb{Q}_{\psi_j}$. We recall that $a_1 \odot \mathbb{Q}_{\psi_i}$ is multiplying all elements in all cosets of $\mathbb{Q}_{\psi_i}$ with a_1. Then it is clear that γ_2 could be written as $\kappa \cdot \gamma_1$, where $\kappa \in \mathbb{F}_{p^m}$ reverses the permutation due to $c \in \mathbb{F}_{p^r}$. Applying the same modification to δ_2, (4.12) becomes;

$$\begin{array}{c} \\ c1, i \\ \\ c2, j \end{array} \begin{array}{c} \begin{array}{ccccc} & v1, k & & v2, l & \\ \end{array} \\ \begin{pmatrix} \dots & \gamma_1 & \dots & \delta_1 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \kappa \cdot \gamma_1 & \dots & \kappa \cdot \delta_1 & \dots \end{pmatrix} \end{array} \tag{4.13}$$

Computing the determinant of the sub-matrix corresponding to the four-cycle in (4.13);

$$\mathfrak{D} = \gamma_1 \cdot \kappa \cdot \delta_1 - \delta_1 \cdot \kappa \cdot \gamma_1 = 0 \tag{4.14}$$

Therefore, the four-cycle in (4.14) does not satisfy the full rank condition. □

As established by Lemmas 4.1, 4.2, and 4.3, after the initial stage, the expanded graph would contain only non-FRC four-cycles, $\frac{p^r-1}{p^m-1}$ cycles each for any such four-cycle in the original

graph. FRC four-cycles, if they existed in the original, are removed in the expansion. This seems advantageous in terms of iterative decoding, but any gain is offset by the increase of non-FRC cycles, the most harmful type of the two.

In the second stage of the subfield expansion, new check nodes are added to account for the dependencies between the coset probability vectors used for decoding. Chapter 3 proposes conducting row operations to remove the cycles that might exist in the PCM used for the local check nodes, but it might not be possible to remove all short cycles. For example, consider the local PCM in (3.34), which contains three FRC four-cycles. Still, it should be possible to remove any non-FRC short cycle.

Above discussions show that with regard to short cycles, the graphs produced by subfield expansions may not necessarily be better than the original graphs. Still, the expanded graphs will contain a significantly lesser number of cycles than the graphs produced by the binary image representation. Thus, it is possible to conclude that subfield expansions produce graphs that are more suited for iterative decoding.

Section 4.2 focuses on some of the more well-known decoding schemes proposed for expanded graphs in the literature. Certain strategies are used in these algorithms to mitigate the effects of short cycles, and the undesirable graph structures they create, which were reviewed previously. Later, in Section 4.3, we propose novel decoding schemes for subfield expansions. Insights on short cycles and their impact on iterative decoding, gained in this Section, are also made use of there.

4.2 Existing Decoding Schemes

4.2.1 Decoding on Binary Images

As discussed in the previous chapter, major obstacle in using the binary image representation for iterative decoding is the higher number of short cycles it contains. In the literature, different strategies have been proposed to use this representation for decoding NB-LDPC codes [43], [41], [42].

One of the more well-known decoding schemes suggested for the binary image representation is *adaptive belief propagation* (ABP). This was initially proposed in [43] for soft decision decoding of Reed-Solomon (RS) codes. We also note that the soft decoding algorithm proposed in [57] for RS codes also makes use of ABP. ABP considers using SPA [4] on the binary image

representation of a non-binary PCM. In order to negate the impact of short cycles, the binary matrix used for decoding is changed *per iteration*. We give a brief overview of how this is done in the following.

In variable node operations of SPA, analogous to (2.4) of QSPA, a full LLR estimate is computed for each node, by taking the sum of the initial channel estimate and neighboring check node estimates [4]. If the magnitude of this value is close to zero for some node, then it is an indication of significant differences between channel and check node estimates for that node, which may be due to the effects of short cycles. Thus, such variable nodes are considered less reliable and the LLR estimates sent out by them are more likely to be erroneous [43]. It would be advantageous to limit their involvement in the decoding process, and with iterative decoding, one way to do that is by reducing the node degree, which, in the perspective of the PCM, is the number of 1's in the respective column.

Now consider an $m \times n$ matrix over $\mathbb{F}_{2^r}$. Binary image of this would be a $mr \times nr$ matrix. When conducting ABP on such a matrix, at the end of each decoding iteration, the $(nr - mr)$ least reliable variable nodes are found by considering the magnitudes of the final LLR estimates, and their degrees for the next iteration are made one by *adapting* the binary PCM in such a way that the corresponding columns contain only a single 1. This matrix adaption is usually carried out with row operations [43].

Simulation results given in [43] show that ABP offers very attractive performance gains over hard decision decoding of RS codes, with gains at times exceeding 5dB. But the disadvantage here is the equally significant increase of complexity, which is mainly due to the matrix adaption procedure. Furthermore, ABP is not suitable for hardware implementations, since the PCM used in decoding is changing each iteration.

[55] proposes some improvements to ABP in terms of complexity; instead of changing the PCM per iteration, it uses a fixed binary matrix. This matrix is constructed by adding some redundant parity-check equations to the binary image, and carrying out row operations to reduce the number of 4-cycles. Additionally, [55] also proposes changing the sign of the channel estimates of least reliable bits in each iteration. It should also be noted that [56] suggests a novel message update scheme for ABP, which further improves performance.

A scheme sharing many similarities to [55] has been proposed in [41], where the focus is on NB-LDPC codes rather than RS codes. The scheme proposes *enhancing* the binary image representations by adding some redundant parity-check equations, but different from [55] which targets 4-cycles, these are selected to remove the low-weight stopping sets. Some simulation results

with short length NB-LDPC codes over small fields are presented in [41], which show the proposed scheme performing very close to QSPA.

A somewhat different approach to decoding with the binary image representation is suggested in [42]. In the binary image of a matrix over $\mathbb{F}_{2^r}$, each row is replaced by r binary rows. These r rows can be considered as a basis of an r dimensional binary vector space, which contains 2^r vectors. [42] proposes using all those vectors, except the all-zero one, when constructing the binary image. Thus, in [42], each row of the original matrix is replaced by $(2^r - 1)$ binary rows. This results in a significantly larger matrix than the traditional binary image, but it should also be noted that the number of rows in this new matrix is equal to that in the extended binary representation. [42] uses SPA on this larger binary image, but introduces some modifications to reduce the impact of short cycles. In check node and variable node computations, rather than using all the estimates received from neighboring nodes, only some are used. Particularly, estimates of smaller magnitudes are not used since they are considered less reliable, as explained earlier. Simulation results show that this decoding scheme manages to perform within a fraction of a dB of QSPA, even with codes over large sized field, such as $\mathbb{F}_{2^8}$.

As discussed above, while some of the strategies in the literature manage to perform satisfactorily, decoding NB-LDPC codes with the binary image is yet to become popular. This is mostly due to the performance-complexity trade-offs offered by the approaches not being very attractive; performance losses tend to be heavy when complexity gains are significant.

4.2.2 Decoding on Extended Binary Representations

Extended binary representation is an alternative binary representation free of the major shortcoming in the binary image, the large number of short cycles [40]. Extended representation of an $m \times n$ matrix over $\mathbb{F}_{2^r}$ is a $(2^r - 1)m \times (2^r - 1)n$ matrix, significantly larger than the $mr \times nr$ binary image. Although this matrix is quite sparse, generic decoding algorithms cannot be implemented in a straight-forward manner on it either, since, as we explained in the last chapter, the matrix contains more variable nodes than the number of bits transmitted through the channel. However, a few approaches have been proposed for decoding NB-LDPC codes using the extended binary representation in the literature [40], [44], [45].

[40] presents an iterative hard decision decoding algorithm with the extended binary representation, for using with the binary erasure channel (BEC). In that algorithm, only r of the $(2^r - 1)$ binary nodes of a single $\mathbb{F}_{2^r}$ node is initialized with channel information; the rest are marked as era-

sures. An iterative soft decoding algorithm for general channels is suggested in [44], where initial estimates for the set of $(2^r - 1)$ binary nodes are derived using the channel estimate for the corresponding $\mathbb{F}_{2^r}$ node, based on simplex code constraints of the extended vector representation (see definition 3.2). But in this approach, binary Fourier transform has to be applied at variable node operations, which makes it of the complexity order $\mathcal{O}(r2^r)$ [44], same as FFT-QSPA [25]. Thus, the complexity advantage of using a binary representation is lost.

Two decoding strategies for the extended binary representation is proposed in [45], a hard decision decoding strategy, and a hybrid decoding strategy, which has components of both hard and soft decoding. [45] considers the vector space spanned by the rows of the extended binary matrix of the non-binary PCM, and picks a set of vectors that satisfy certain criteria to form the matrix used for decoding. In the proposed iterative hard decision decoder, simplex constraints are used to derive the initial estimates, similar to [44]. On the other hand, for the hybrid decoding scheme, which conducts a number of soft decoding iterations and a number of hard decoding iterations consecutively, the authors propose transmitting all bits of the extended vector representation. While the simulation results presented show good performance, this decoding strategy actually changes the rate of the code, and may not be appropriate in many situations.

4.3 Subfield Decoding

Consider decoding an NB-LDPC code over $\mathbb{F}_{p^r}$ using one of its subfield expansions, proposed in Chapter 3. New factor graphs produced by the expansion fully capture the relationships in the original graph, and any iterative, message-passing decoding algorithm proposed for NB-LDPC codes could be adapted for using on such an expanded graph. Advantage herein is the expansion being over a smaller field $\mathbb{F}_{p^m}$, where $m \mid r$, than the original, which results in a significantly lower decoding complexity, since the complexity order is quadratic in field size. The original factor graph could be expanded over any subfield of $\mathbb{F}_{p^r}$, and this produces a few different graphs that could be used for decoding. Each graph could also be used with any decoding algorithm, given that the algorithm is suitably modified. Decoding algorithms include simplifications of more complex ones, such as EMSA and QSPA, introduced in Chapter 2. Pairing such an algorithm with an expanded graph would offer the combined complexity gains of the simplification and the graph expansion both. With a few different expanded graphs and a few different decoding algorithms, a number of different pairings are possible for decoding a particular NB-LDPC code, and each of these would offer a unique performance-complexity trade-off.

Due to their certain special characteristics, graphs obtained through the proposed subfield expansion cannot be used with a decoding algorithm of NB-LDPC codes in a straight-forward manner. Some modifications need to be applied to such an algorithm in order to make it compatible with an expanded graph. Although there could be differences in implementations, the modifications necessary are common to any algorithm. In the following, we present these modifications, and explain why they are required. It should be noted that the explanations are from the perspective of a soft decision decoding (SDD) algorithm, such as QSPA and its many variations, but the reasons for the modifications are valid for different types of algorithms as well, such as majority-logic decoding.

1. Computing Initial Estimates

Any SDD algorithm has to be initialized with probability estimates computed based on information received from the channel. As explained in Chapter 2, in QSPA and its variants, these initial probability estimates are computed per variable node, and they take the form of symbol probability vectors, as a variable node represents a position in the codeword. In the subfield expansions we propose, variable nodes represent different CPVs. Therefore, when using those for decoding, channel information has to be converted to initial estimates for CPVs.

Consider some quotient group $\mathbb{Q}_{\psi_i}$, used for expanding a graph over $\mathbb{F}_{p^r}$ over some subfield $\mathbb{F}_{p^m}$. Each coset $\mathcal{C}_j^i$ of $\mathbb{Q}_{\psi_i}$ contains a subset of symbols in $\mathbb{F}_{p^r}$. This observation makes computing initial estimates for CPVs quite straight-forward, i.e., probability of the value of a $\mathbb{F}_{p^r}$ code symbol being from a particular sub-set of $\mathbb{F}_{p^r}$ symbols is simply the sum of the individual probabilities of those symbols that belong to the subset. These symbol probabilities can be computed using channel information, similar to in QSPA and its variants, which were reviewed in Chapter 2. (4.15) in the following presents the computation of the initial estimate for the i'th CPV of some $\mathbb{F}_{p^r}$ variable node n. $\underline{\mathbf{p}}_{n,i}^{c}(j)$ represents the j'th coset probability of that CPV, $\underline{\mathbf{p}}_{n}^{s}$ represents the initial symbol probability vector for node n, and β_k denotes $\mathbb{F}_{p^r}$ symbols in the coset under consideration, $\mathcal{C}_j^i$.

$$\underline{\mathbf{p}}_{n,i}^{c}(j) = \sum_{\beta_k \in \mathcal{C}_i^j} \underline{\mathbf{p}}_{n}^{s}(\beta_k) \qquad j = 0, 1, \ldots, (p^m - 1) \tag{4.15}$$

As explained in Chapter 2, decoders operate on either log or log-likelihood ratio (LLR) domain in most practical applications, due to hardware stability concerns [29]. Therefore, an initial CPV $\underline{\mathbf{p}}_{n,i}^{c}$, computed as in (4.15), has to be converted to one of those domains, for the decoding algorithm to be usable in practice. For example, (4.16) below presents the conversion to LLR domain. Note that this computation is the same as that of converting a symbol probability vector

over $\mathbb{F}_{p^m}$ to LLR domain.

$$\underline{L}^c_{n,i}(j) = \log \frac{\underline{\mathbf{p}}^c_{n,i}(j)}{\underline{\mathbf{p}}^c_{n,i}(0)} \qquad j = 0, 1, \ldots, (p^m - 1) \tag{4.16}$$

When expanding an $\mathbb{F}_{p^r}$ graph to an $\mathbb{F}_{p^m}$ one, each node in the original graph will be replaced by $\frac{p^r-1}{p^m-1}$ nodes, as discussed in chapter 3. Each new node of a $\mathbb{F}_{p^r}$ variable node would represent one of its CPVs, and has to be initialized as in (4.15) and (4.16). This may not seem problematic, but observe that only a single symbol probability vector $\underline{\mathbf{p}}^s_n$ will be used for initializing $\frac{p^r-1}{p^m-1}$ nodes. This symbol probability vector corresponds to a single $\mathbb{F}_{p^r}$ symbol transmitted through the channel. Channel information of such a symbol contains information equivalent to r symbols of $\mathbb{F}_p$, or $\frac{r}{m}$ symbols of $\mathbb{F}_{p^m}$. Therefore, since $\underline{\mathbf{p}}^s_n$ is derived from channel information of one such $\mathbb{F}_{p^r}$ symbol, it should only be sufficient to compute initial estimates of $\frac{r}{m}$ $\mathbb{F}_{p^m}$ nodes/CPVs. But if we proceed as in (4.15), $\frac{p^r-1}{p^m-1}$ CPVs are initialized. This means that directly applying (4.15) to compute initial estimates of all CPVs *duplicates* channel information of a single $\mathbb{F}_{p^r}$ node. This also creates dependencies between initial CPV estimates. Furthermore, any error present in channel information would get magnified, and can propagate through the graph in subsequent decoding iterations, leading to significant performance losses.

One seemingly straight-forward way to reduce error propagation is to initialize only $\frac{r}{m}$ nodes out of the $\frac{p^r-1}{p^r+1}$ with channel information as in (4.15), and set initial estimates for all other CPVs to zero. But this could also impact decoding performance, since error-free portions of channel information that could have been duplicated without impacting decoding negatively end up being used only once. Therefore, a method with characteristics of both the approaches discussed so far, initializing all CPVs and only a limited number, may be more appropriate for decoding on subfield expansions. What we propose is using an optimized scaling factor δ ($0<\delta<1$) in (4.16). All initial CPV estimates will be derived as in (4.15), in probability domain, but when converting to LLR domain, $(\frac{p^r-1}{p^m-1} - \frac{r}{m})$ are multiplied with δ, reducing their magnitudes. Thus, magnitudes of any errors present in those CPVs also get reduced, and impact of error propagation would not be as severe as before. Also, we end up using full channel information as in a majority of SDD algorithms, since $\frac{r}{m}$ nodes are initialized without the scaling factor. The following equation shows how scaling is used with (4.16). Note that this does not apply for $\frac{r}{m}$ CPVs, and that scaling factor δ has to be optimized per code.

$$\underline{L}^c_{n,i}(j) = \delta \cdot \log \frac{\underline{\mathbf{p}}^c_{n,i}(j)}{\underline{\mathbf{p}}^c_{n,i}(0)} \qquad j = 0, 1, \ldots, (p^m - 1) \tag{4.17}$$

Recall the concept of the *local code*, introduced in Section 3.2.2.2, in the previous chapter. This allows us to view the $\frac{p^r-1}{p^m-1}$ nodes of a $\mathbb{F}_{p^r}$ variable node as representing code positions of a $(\frac{p^r-1}{p^m-1}, \frac{r}{m})$ code over $\mathbb{F}_{p^m}$. Then, in turn, we can consider $\frac{r}{m}$ of the new nodes as representing *information positions*, and the others as representing *parity positions*. Our method of initializing is appropriate in this perspective as well, where it may be interpreted as using full channel information for the $\frac{r}{m}$ nodes representing information positions and using a scaling factor with the remaining nodes, those that represent parity positions. Any suitable set of $\frac{r}{m}$ nodes/CPVs could be chosen for representing information positions, and (4.17) should only be used with the rest. Simulation results we present later in the chapter validate modifying the initialization step in this manner.

2. Distinguishing Local Checks from Regular Checks

As discussed in Section 3.2.2.3, expanded graphs contain two different types of check nodes; *local* check nodes that represent dependencies between CPVs, and *regular* check nodes, those that replaced check nodes in the original graph. A local check node is only connected with some of the $\left(\frac{p^r-1}{p^m-1}\right)$ $\mathbb{F}_{p^m}$ nodes that were added for a single $\mathbb{F}_{p^r}$ variable node, whereas a regular check node will be connected with at most one node of each such set.

The two types of check nodes actually represent constraints of two different codes, the code represented by the original factor graph, and the simplex code of the alternative vector representation, presented in Definition 3.8. As seen in Section 3.2.2, an instance of the same simplex code is created between the set of $\mathbb{F}_{p^m}$ nodes of each $\mathbb{F}_{p^r}$ variable node, and local check nodes are added for each such set. Thus, estimates from a local check node can be considered to only depend on a single variable of the original code. These observations suggest that treating probability estimates from the two types of check nodes similarly may not be the best approach to follow.

When expanding a graph, the initial step we proposed in Section 3.2.2 is replacing each original node with a set of nodes from the subfield, and connecting these new nodes based on the original edge label. In the perspective of a matrix expansion, this corresponds to replacing each value in the original PCM with their alternative matrix representation (see Definition 3.7) over the subfield. These alternative matrix representations are very similar to permutation matrices, as proved in Lemma 3.8. Due to special characteristics of the alternative matrix representation, the initial expansion would contain short-cycles only if the original graph contained short-cycles that satisfy certain conditions, as was seen in Lemma 4.1, and therefore, the factor graph after the initial expansion could be sparser than the original.

However, the local PCM, which is a generator matrix of a Hamming code, could contain

short cycles, and in many cases it does. Thus, the final expansion will be more dense than the original graph. Short-cycles added will only involve the local check nodes, and, as discussed in Section 3.2.2.3, they may not even be of the most damaging type. Still, estimates computed by a certain check node involved in such a cycle in two consecutive decoding iterations will be correlated to some degree, which could make check node estimates *over-confident* of a variable node taking a particular value. This is only an issue with local check nodes though.

Taking into consideration the short cycles in the local PCM, and the differences between estimates computed by the two types of check nodes, we propose using another scaling factor ψ ($0<\psi<1$) for estimates of local check nodes. Similar approaches have been suggested in the literature to reduce the impact of short cycles in iterative decoding, for example in [43].

(4.18) in the following shows how check node estimates are combined at a variable node n in the expanded graph, with this modification. It is presented for an algorithm implemented in LLR-domain, such as LLR-QSPA, discussed in Section 2.2.1. Similar to (2.11), here $\underline{L}_n^{(k)}$ represents the combined estimate for node n in decoding iteration k, $\underline{L}_n^{(0)}$ is the channel estimate for that node, and $\underline{P}_{j,n}^{(k)}$ are estimates from neighboring check nodes j. $M_r(n)$ denotes the set of regular check nodes in the neighborhood of node n, while $M_l(n)$ denotes the set of local check nodes.

$$\underline{L}_n^{(k)} = \underline{L}_n^{(0)} + \sum_{j \in M_r(n)} \underline{P}_{j,n}^{(k)} + \psi \cdot \sum_{j \in M_l(n)} \underline{P}_{j,n}^{(k)} \tag{4.18}$$

(4.19) shows how the message for a connected check node m, $\underline{R}_{n,m}^{(k)}$, is computed with this modification, for the two possible cases of m; a regular check node, and a local check node.

$$\begin{aligned} \underline{R}_{n,m}^{(k)} &= \underline{L}_n^{(k)} - \underline{P}_{m,n}^{(k)}; \qquad && m \in M_r(n) \\ \underline{R}_{n,m}^{(k)} &= \underline{L}_n^{(k)} - \psi \cdot \underline{P}_{m,n}^{(k)}; \qquad && m \in M_l(n) \end{aligned} \tag{4.19}$$

Similar to δ in initialization, ψ also has to be optimized per code.

3. Testing for Convergence

An important part in iterative decoding of NB-LDPC codes is testing whether the decoder has converged to a valid codeword. If so, the decoding process can be terminated early, which increases the decoding efficiency. As explained in Section 2.2, this test for convergence is carried out after variable node operations, with the vector of tentative decisions taken by each such node. In the test, constraints in the PCM, which are represented by check nodes in the factor graph, are evaluated, since for the decoder to converge, all such constraints have to be satisfied.

It is possible to follow the same approach when decoding on subfield expansions. Consider decoding on a graph over $\mathbb{F}_{p^m}$, obtained by expanding a $\mathbb{F}_{p^r}$ graph. Tentative decision at each $\mathbb{F}_{p^m}$ variable node will be some $\mathbb{F}_{p^m}$ symbol that is the most likely for that node. Constraints represented by both regular check nodes and local check nodes have to be evaluated in the convergence test. With $\left(\frac{p^r-1}{p^m-1} - \frac{r}{m}\right)$ local check nodes per variable node of the original graph, number of constraints will be fairly high here. If all constraints are satisfied, meaning that the decoder has converged, the output will be a vector of $\mathbb{F}_{p^m}$ elements that is $\frac{p^r-1}{p^m-1}$ times longer than the original code length. For recovering the original codeword, each set of consecutive $\frac{p^r-1}{p^m-1}$ symbols in the output vector can be mapped to a single $\mathbb{F}_{p^r}$ symbol, via the alternative vector representation, discussed in Section 3.2.2.2.

An alternative approach for testing convergence when decoding over subfield expansions is to first map each set of $\mathbb{F}_{p^m}$ symbols to a $\mathbb{F}_{p^r}$ symbol, and then use the constraints of the original graph for the test. Since validating constraints only involves simple field arithmetic, complexity of the operation does not depend much on the field size. But the original graph will have a much lower number of constraints, which makes this approach advantageous in terms of complexity. Field arithmetic is usually carried out with look-up tables, and this requires tables for both $\mathbb{F}_{p^r}$ and $\mathbb{F}_{p^m}$ to be available in a decoder, which seems a disadvantage when it comes to hardware resource requirements. Still, since $\mathbb{F}_{p^m}$ is a subfield of $\mathbb{F}_{p^r}$, sections of the larger field's look-up tables could be used for the smaller one.

Straight-forward way to map $\left(\frac{p^r-1}{p^m-1}\right)$ $\mathbb{F}_{p^m}$ symbols to a single $\mathbb{F}_{p^r}$ symbols is through the alternative vector representation introduced in the last chapter. But recall that the alternative vector representations are codewords of a $\left(\frac{p^r-1}{p^m-1}, \frac{r}{m}\right)$ simplex code, and thus, if the decoder had converged, only the $\frac{r}{m}$ information symbols of the $\frac{p^r-1}{p^m-1}$ are required to map to a $\mathbb{F}_{p^r}$ symbol. Therefore, we propose only using the information symbols when mapping between the two fields for convergence testing. This is advantageous when one or more of the parity nodes out of a set of $\frac{p^r-1}{p^m-1}$ are not converging to the proper $\mathbb{F}_{p^m}$ symbol. Using only the information nodes could allow decoding to be terminated a few iterations early in such a case. Note that certain information nodes converging to inaccurate symbols would result in the decoder termination only if they converge to specific symbols that still form a codeword. Probability of such a case is directly dependent on the minimum distance properties of the code, as is common with any type of decoding, using subfield expansions or otherwise.

With the modifications proposed above, any iterative soft-decoding algorithm proposed

for NB-LDPC codes, such as the ones in [26], [25], [28], [33], may be used with expanded graphs. A case of special interest here is when the subfield used is the binary field, or $\mathbb{F}_2$. In such a case, quotient groups used for the expansion are isomorphic to $\mathbb{F}_2$, and CPVs are only of length 2. Thus, decoding can be conducted with algorithms proposed for binary LDPC codes [4], [31], given that the modifications proposed are applied appropriately. A brief overview of this case is provided in the following.

4.3.1 Decoding on Binary Expansions

With binary expansions, each node over $\mathbb{F}_{2^r}$ is expanded to $(2^r - 1)$ binary nodes, as seen in Section 3.2.3. CPVs these nodes represent contain only two elements; the probability of the corresponding variable being from a subgroup of size 2^{r-1}, and the probability of it being from that subgroup's proper coset. Therefore, it is not necessary to use probability vectors in computations. Instead, scalar LLR values can be used, similar to decoding of binary LDPC codes [4]. We chose to define the LLR represented by the i'th binary node of the n'th $\mathbb{F}_{2^r}$ variable node as follows. $\mathbb{G}_i$ and $\mathbb{C}_i$ represent the corresponding subgroup and its coset, whereas $p(n \in \mathbb{G}_i)$ and $p(n \in \mathbb{C}_i)$ represent the probabilities of the symbol for node n being from $\mathbb{G}_i$ and $\mathbb{C}_i$ respectively.

$$L_{n,i} = \log \frac{p(n \in \mathbb{C}_i)}{p(n \in \mathbb{G}_i)} \tag{4.20}$$

Unlike NB-LDPC codes, only a few algorithms have been suggested in the literature for binary LDPC codes. This is probably due to the best known decoder, binary sum-product algorithm (SPA) [4], not been as complex as QSPA, and its simplification, min-sum algorithm (MSA) [31], not loosing much on performance. In this section, we focus on how SPA is modified for decoding on binary expansions. To make the discussions more related to the practical case, we assume data is transmitted through the binary-input, additive white Gaussian noise (BI-AWGN) channel, with BPSK modulation.

1. Initialization

Although each $\mathbb{F}_{2^r}$ node is replaced with $(2^r - 1)$ binary nodes, when using a binary-input channel, only r bits are transmitted for each $\mathbb{F}_{2^r}$ symbol. The bits transmitted are the ones that constitute the traditional binary representation of a symbol. The set of $\mathbb{F}_{2^r}$ symbols for which a specific bit in that representation is high should form a subgroup of order 2^{r-1} of $\mathbb{H} = \{\mathbb{F}_{2^r}, +\}$. According to Lemmas 3.9 and 3.10, such subgroups belong in the set that is used for the binary expansion, and

the expanded graph contains nodes that represent probabilities of $\mathbb{F}_{2^r}$ variables belonging or not belonging to them.

As discussed earlier, the $(2^r - 1)$ binary nodes of a non-binary variable node can be divided into two groups; ones that represent information bits of the local code, and the ones that represent parity bits. In this case, the local code, first introduced in Section 3.2.2.2, is a $(2^r - 1, r)$ binary simplex code. We propose choosing the nodes that map to the bits transmitted through the channel as the ones that represent information bits. This allows us to initialize them in a very simple manner, as follows. $L_{n,i}^{(0)}$ denotes the initial estimate for the i'th binary node of the n'th $\mathbb{F}_{2^r}$ variable node, and $y_{n,i}$ represents the value received from the BI-AWGN channel for the corresponding bit, while σ^2 is the noise variance.

$$L_{n,i}^{(0)} = \frac{2 \cdot y_{n,i}}{\sigma^2}; \qquad i = 1, \ldots, r \tag{4.21}$$

Remaining $(2^r - r - 1)$ binary nodes represent the parity bits of the local code. This means that each of them can be written as a linear combination of the r information bits, which are initialized with channel values, as in (4.21). These relationships are represented with some codeword in the dual code of the local simplex code, and can be used with the initial estimates of information bits to initialize the parity bits. But the first modification to iterative decoding algorithms we proposed earlier applies here. Simply duplicating channel information can lead to error propagation and inferior decoding performance, and therefore, an optimized scaling factor ψ has to be used in the computation. (4.22) below represents this computation, for a node j that represents a parity bit. $\mathfrak{I}_j$ denotes the set of information bits involved in generating the parity bit j.

$$L_{n,j}^{(0)} = \delta \cdot 2 \tanh^{-1} \prod_{k \in \mathfrak{I}_j} \tanh \frac{L_{n,k}^{(0)}}{2}; \qquad j = r + 1, \ldots, 2^r - 1 \tag{4.22}$$

One significant advantage in this approach is that symbol probability vectors now do not have to be computed with channel information, unlike when using the original graph for decoding. Therefore, using the binary expansion results in complexity gains even in the initialization step.

2. Check Node Operations

Subfield expansions contain two types of check nodes; regular ones and local ones. One of the modifications proposed earlier was for distinguishing between estimates from these two types of check nodes. This was done with another scaling factor ψ, which was applied at variable nodes. Thus, when it comes to check node operations, both regular and local nodes operate the same way,

as a check node in a binary LDPC decoder [4]. Following equation represents how such a node m computes the probability estimate for some variable node n in its neighborhood during k'th decoder iteration, using the probability estimates sent from other variable nodes in the previous iteration.

$$P_{m,n}^{(k)} = 2 \tanh^{-1} \prod_{i \in N(m);\ i \neq n} \tanh \frac{R_{i,m}^{(k-1)}}{2} \tag{4.23}$$

3. Variable Node Operations

Variable node operations in binary SPA [4] and QSPA [26] take almost the same form, with the only difference being QSPA using probability vectors instead of single probability values, as in binary SPA. First operation at variable nodes in both cases is combining the probability estimates received from neighboring check nodes with the initial estimate, which, in LLR domain, amounts to simple additions of the respective values. When it comes to decoding on binary expansions though, a slight modification is necessary to distinguish between estimates of regular check nodes and local check nodes. This operation, which is analogous to (4.18), is given in the following, for some binary variable node n of the expanded graph. Note that the symbols used represent the same quantities as in (4.18).

$$L_n^{(k)} = L_i^{(0)} + \sum_{j \in M_r(n)} P_{j,n}^{(k)} + \psi \cdot \sum_{j \in M_l(n)} P_{j,n}^{(k)} \tag{4.24}$$

Next operation at variable nodes is the test for convergence, which is carried out with their tentative decisions, taken based on $L_n^{(k)}$ values above. In traditional decoding of a code over $\mathbb{F}_{2^r}$, where its original factor graph will be used, 2^r comparisons between real numbers are necessary at each variable node to take a tentative decision. As proposed earlier, when decoding on subfield expansions, only the r nodes that represent information symbols of the local code need to be considered for that, and with a binary node, only a comparison against zero is necessary for deciding on the bit value [4]. Therefore, only r such comparisons are now required, as opposed to the 2^r if the original graph was used.

In the event that the test for convergence failed, decoding will continue for another iteration, and the final step of variable node operations is computing the probability estimates for each neighboring check node, to be used in that iteration. When using the binary expansion, due to the modification in (4.24), this operation is slightly different for the two types of check nodes, as shown in the following.

$$\begin{aligned} R_{n,m}^{(k)} &= L_n^{(k)} - P_{m,n}^{(k)}; && m \in M_r(n) \\ R_{n,m}^{(k)} &= L_n^{(k)} - \psi \cdot P_{m,n}^{(k)}; && m \in M_l(n) \end{aligned} \tag{4.25}$$

4. Simplifications

Although complexity of binary SPA is not at levels that discourage practical usage, operations such as $\tanh$ and $\tanh^{-1}$ computations, and real number multiplications, all necessary at check node operations, are considered complex at hardware level. The well-known simplification of binary SPA, min-sum algorithm (MSA) [31], reduces these computations to some comparisons, without losing much on performance.

MSA can also be used with binary expansions, granted that the modifications proposed are first applied. Since the sole difference between SPA and MSA is in check node operations, in the following we only present the respective computation, which is a simplification of (4.23). Note that since no modifications are proposed for this step, computation is exactly the same as with a binary LDPC code.

$$P_{m,n}^{(k)} = \prod_{i \in N(m);\ i \neq n} \operatorname{sign}(R_{i,m}^{(k-1)}) \cdot \min_{i \in N(m);\ i \neq n} \{|R_{i,m}^{(k-1)}|\} \tag{4.26}$$

In the case of binary LDPC codes, it has been identified that the small performance loss in MSA when compared to SPA is due to the check node estimates computed as in (4.26) being overestimates of the actual value [58]. Various methods to correct this overestimation error has been suggested in the literature [58], [59], [60], among which the simplest is to use a constant scaling factor ω ($0 < \omega \leq 1$) with check node estimates [58]. We propose using this approach to further improve performance of MSA on the binary expansions. With this simple modification, (4.26) takes the following form where, once more, ω represents the scaling factor.

$$P_{m,n}^{(k)} = \omega \cdot \prod_{i \in N(m);\ i \neq n} \operatorname{sign}(R_{i,m}^{(k-1)}) \cdot \min_{i \in N(m);\ i \neq n} |R_{i,m}^{(k-1)}| \tag{4.27}$$

An operation similar to (4.26) is carried out during initialization as well, in (4.22), to derive initial estimates for the set of nodes that represent parity bits of the local code. It is possible to use the min-sum simplification there as well, but the resulting complexity gain will be minimal, since the operation is only necessary for initialization. Further, this would also result in SPA and MSA using different initial probabilities for decoding the same vector. Therefore, we propose leaving (4.22) as is when using MSA for decoding on binary expansions.

Subfield expansions we introduced in Chapter 3 can be used with most decoding algorithms suggested for NB-LDPC codes in the literature, after applying the modifications proposed in this section. With a few expansions available for a single NB-LDPC code, a large number of

decoding strategies become possible. For applications where decoding latency is of primary concern, either a simplification of QSPA, such as min-max decoding [28], can be used with an expanded graph, or if the field is of characteristic 2, the binary expansion can be used with MSA. Such options offer the combined complexity gains of both the simplification and the expansion. In the following section, we present some results from decoding simulations where a few of these different strategies were implemented.

4.3.2 Performance

In this section, we evaluate performance of some decoding schemes on subfield expansions, implemented with the modifications proposed earlier. Since NB-LDPC codes are most often constructed over finite fields of characteristic 2 [5], only such codes were used for the performance evaluation. We consider different expansions of the same Tanner graph (different m for a fixed r), and use QSPA [5] and min-max algorithm [28] when $m > 1$. With binary expansions ($m = 1$), SPA [4] and MSA [31] are used instead. Frame error rate (FER) performance of these schemes are compared with using QSPA, min-max algorithm, and max-log-SPA [26] (also see Section 2.2.3) on the original graph. All algorithms are implemented in LLR domain, and it is assumed that data is transmitted over a BI-AWGN channel, using traditional BPSK modulation where $0 \rightarrow +1$ and $1 \rightarrow -1$. Scaling factors δ, ψ and ω, introduced in the previous section, were optimized for each code and each algorithm using Monte-Carlo simulations. When analyzing performance of different decoding schemes in the following, we use the algorithm along with the field size to refer to each; i.e. $\mathbb{F}_{p^r}$-QSPA when using QSPA on a $\mathbb{F}_{p^r}$ graph.

Figure 4.6 presents performance of some decoding schemes with $\mathfrak{C}_1$, a rate 0.88 NB-LDPC code over $\mathbb{F}_{2^4}$, of 1998 symbols in length. Code was generated through random re-labeling of a regular graph of variable degree 4 and check degree 36, from [61]. We consider using two expansions of $\mathfrak{C}_1$, with the two subfields of $\mathbb{F}_{2^4}$, $\mathbb{F}_{2^2}$, and $\mathbb{F}_2$, the binary field. Maximum number of decoding iterations was set to 50 for all schemes.

It can be observed from the figure that the decoding schemes over expanded graphs manage to perform quite close to the same algorithms implemented on the original graph. For example, QSPA over the $\mathbb{F}_{2^2}$ expansion manages to perform within 0.2dB of $\mathbb{F}_{2^4}$-QSPA, at a FER of 10^{-4}. Gap between $\mathbb{F}_{2^4}$-min-max and $\mathbb{F}_{2^2}$-min-max is also around 0.2dB. As expected, when it comes to the binary expansion, performance loss relative to using the original graph widens. Still, gap of SPA with $\mathbb{F}_{2^4}$-QSPA is less than 0.4dB, while for MSA, it is about 0.55dB.

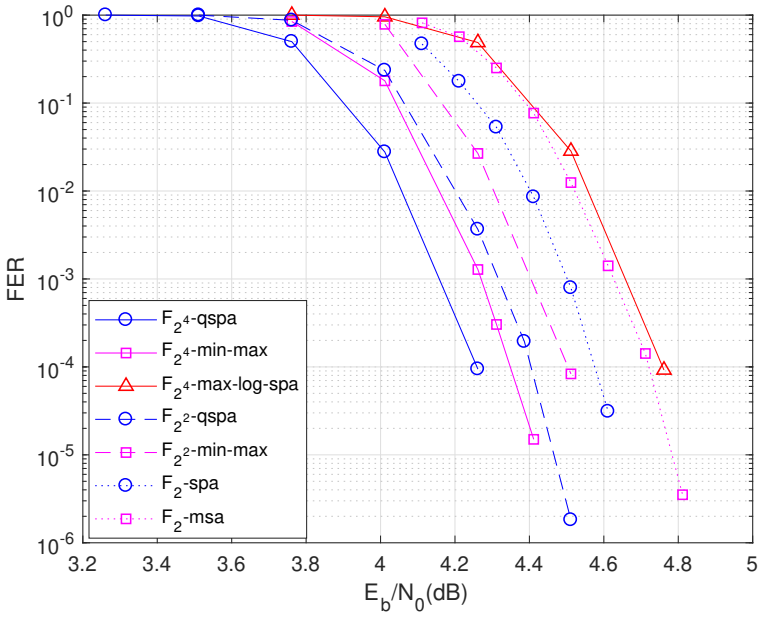

Figure 4.6: FER Performance with a (1998, 1776) Code over $\mathbb{F}_{2^4}$

When considering the gap with the optimal decoding scheme, QSPA on the original graph, closest is $\mathbb{F}_{2^4}$-min-max, but it should be noted that this is over the original graph, and thus is more complex than even QSPA on expanded graphs, as will be made evident in the next section. Interestingly, the second simplification of QSPA, max-log-SPA, although it is used with the original graph, is outperformed by *all* decoding schemes on expanded graphs. $\mathbb{F}_{2^4}$-max-log-SPA has a gap of almost 0.4dB with $\mathbb{F}_{2^2}$-QSPA and, and even MSA on the binary expansion, the least complex decoding scheme of all, marginally outperforms it. Results shows that the information loss in max-log-SPA, due to discarding all the LLR values other than the highest [26], can be costly. As results with other codes would show, the loss becomes more significant when the variable degree increases. If the variable degree is relatively higher, then the variable nodes would receive a higher number of PMF estimates (in LLR form), all of which are suffering from the information loss due to the approximation at check nodes. Therefore, the combined estimate at a given variable node, and the PMF estimates it sends out in the next iteration, could be significantly different from the ones in QSPA, which leads to the observed performance degradation.

As mentioned earlier, scaling factors used for decoding schemes on expanded graphs were all optimized with simulations. For $\mathbb{F}_{2^2}$-QSPA, optimum values of δ and ψ were found to be,

respectively, 0.75 and 0.25, while for $\mathbb{F}_{2^2}$-min-max, these were 0 and 0.35. Optimum values of δ and ψ for SPA and MSA were 0.7 and 0.25, and $\omega = 0.9$ was found to be the best for MSA.

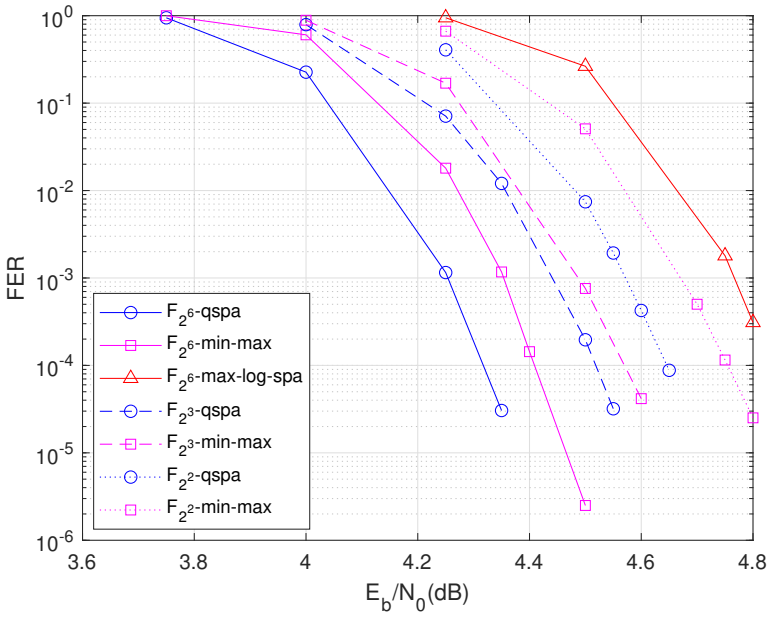

Figure 4.7: FER Performance with a (1998, 1776) Code over $\mathbb{F}_{2^6}$

Same graph as the one used for $\mathfrak{C}_1$ was re-labelled over $\mathbb{F}_{2^6}$, resulting in the code $\mathfrak{C}_2$, which is of the same rate, symbol length, and node degrees as $\mathfrak{C}_1$. Performance of various decoding schemes with $\mathfrak{C}_2$ are given in Figure 4.7. In this case, we considered $\mathbb{F}_{2^3}$ and $\mathbb{F}_{2^2}$ expansions of $\mathfrak{C}_2$, and maximum number of decoding iterations was set to 50, same as with $\mathfrak{C}_1$ earlier.

Figure shows that decoding schemes on expanded graphs perform quite close to those on the original graph in this case as well. Gap between $\mathbb{F}_{2^6}$-QSPA and $\mathbb{F}_{2^3}$-QSPA is only about 0.2dB, at a FER of 10^{-4}. For min-max algorithm, moving from the original $\mathbb{F}_{2^6}$ graph to the $\mathbb{F}_{2^3}$ expansion results in an even smaller performance loss, about 0.15dB. With the $\mathbb{F}_{2^2}$ expansion, gap between using the original and expanded graph is around 0.35dB for QSPA, and 0.4dB for min-max.

With $\mathfrak{C}_2$ also, closest algorithm to the optimal scheme, $\mathbb{F}_{2^6}$-QSPA, is min-max on the original graph. But the performance gap between that scheme and $\mathbb{F}_{2^3}$-QSPA is only 0.1dB, while the second is of much lower complexity. Once more, we find that max-log-SPA on the original graph performs worse than the considered decoding schemes using expansions. $\mathbb{F}_{2^3}$-QSPA manages to outperform this scheme by close to 0.4dB, while the decoding scheme with the lowest complexity,

$\mathbb{F}_{2^2}$-min-max, outperforms it by 0.1dB.

Optimum values for δ and ψ were found to be 0.75 and 0.25 with $\mathfrak{C}_2$, both for $\mathbb{F}_{2^3}$-QSPA and $\mathbb{F}_{2^2}$-QSPA. For both $\mathbb{F}_{2^3}$-min-max and $\mathbb{F}_{2^2}$-min-max, $\delta = 0$ was the optimum, while ψ was 0.3 for the first and 0.4 for the second.

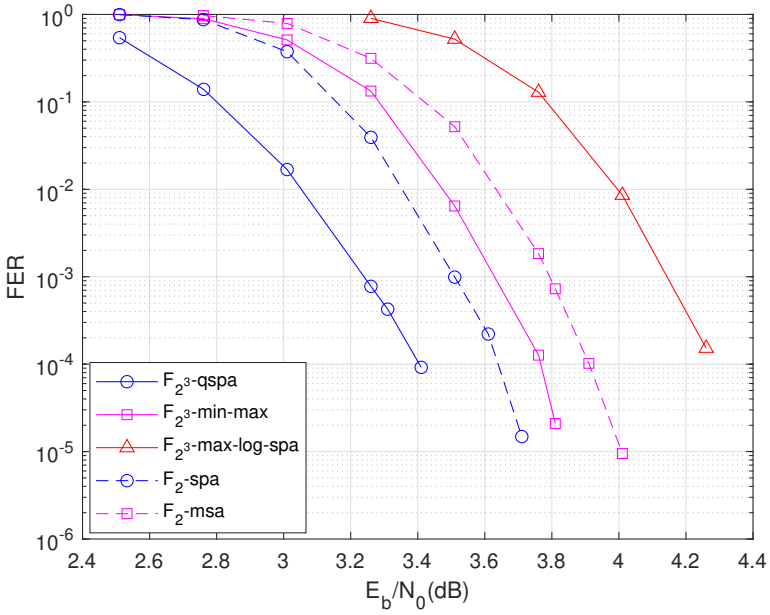

Figure 4.8: FER Performance with a (816, 408) Code over $\mathbb{F}_{2^3}$

Figure 4.8 presents performance of different decoding schemes with $\mathfrak{C}_3$, a rate $\frac{1}{2}$ code over $\mathbb{F}_{2^3}$, of 816 symbols in length. This code was generated by randomly re-labeling a regular graph, with column degree 5 and check degree 10, also obtained from [61]. $\mathbb{F}_{2^3}$ contains a single subfield, $\mathbb{F}_2$, and thus, only subfield decoding schemes considered are SPA and MSA on the binary expansion. Maximum number of decoding iterations was set to 50 for all cases.

With $\mathfrak{C}_3$, SPA on the binary expansion is the closest performing scheme to the optimum one, $\mathbb{F}_{2^3}$-QSPA. Gap between the two is around 0.25dB, at a FER of 10^{-4}. At the same FER level, $\mathbb{F}_{2^3}$-min-max has a gap of close to 0.4dB with $\mathbb{F}_{2^3}$-QSPA. A loss of about 0.3dB from SPA can be observed for MSA on the binary graph. Once more, subfield decoding schemes manage to outperform max-log-SPA on the original graph; SPA on the binary graph outperforms it by about 0.6dB, and MSA by close to 0.3dB.

Optimum values for δ and ψ were, respectively, 0.5 and 0.25, with both $\mathbb{F}_2$-SPA and

$\mathbb{F}_2$-MSA, while for ω in MSA, it was found to be 0.7.

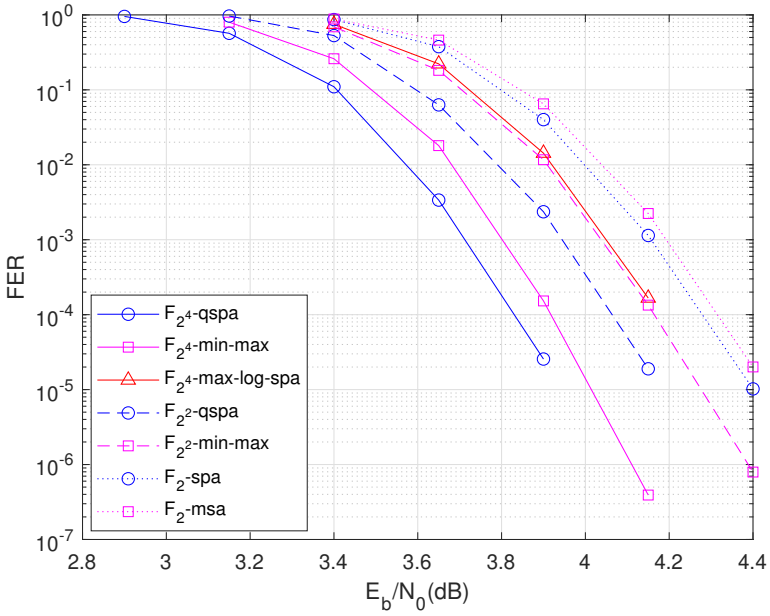

Figure 4.9: FER Performance with a (1000, 816) Code over $\mathbb{F}_{2^4}$

Figure 4.9 illustrates FER performance of some decoding schemes with $\mathfrak{C}_4$, a rate 0.861 code over $\mathbb{F}_{2^4}$, of 1000 symbols in length. $\mathfrak{C}_4$ was constructed by re-labeling a factor graph of regular variable degree 3 and average check degree 22, generated with the progressive edge growth algorithm [48]. For this code, we consider expansions over the two subfields of $\mathbb{F}_{2^4}$, $\mathbb{F}_{2^2}$ and $\mathbb{F}_2$, the binary field. Once more, maximum number of decoding iterations was set to 50 for all schemes.

It is observable from the figure that the decoding schemes on graph expansions perform almost as well as those on the original graph. Gap between using QSPA on the original graph and its $\mathbb{F}_{2^2}$ expansion is less than 0.3dB at a FER of 10^{-4}. Performance loss of replacing QSPA by its simplification min-max algorithm is about 0.1dB for both original and expanded graphs. With $\mathfrak{C}_4$, performance of max-log-SPA on the original, $\mathbb{F}_{2^4}$ graph, is very close to $\mathbb{F}_{2^2}$-min-max, although its complexity is much higher. When compared with $\mathbb{F}_{2^4}$-QSPA, SPA on the binary graph results in a 0.5dB loss in performance. Simplifying SPA to MSA only loses a further 0.05dB. Although a 5.5dB loss seems significant, using MSA on the binary expansion provides even more significant advantages in complexity and hardware implementation costs, as will be explained in the next section.

Optimum values for scaling factors δ and ψ were 0.5 and 0.25 for $\mathbb{F}_{2^2}$-QSPA, SPA, and MSA, while for $\mathbb{F}_{2^2}$-min-max, they were 0 and 0.3. Best value of ω, in MSA, was found to be 1.

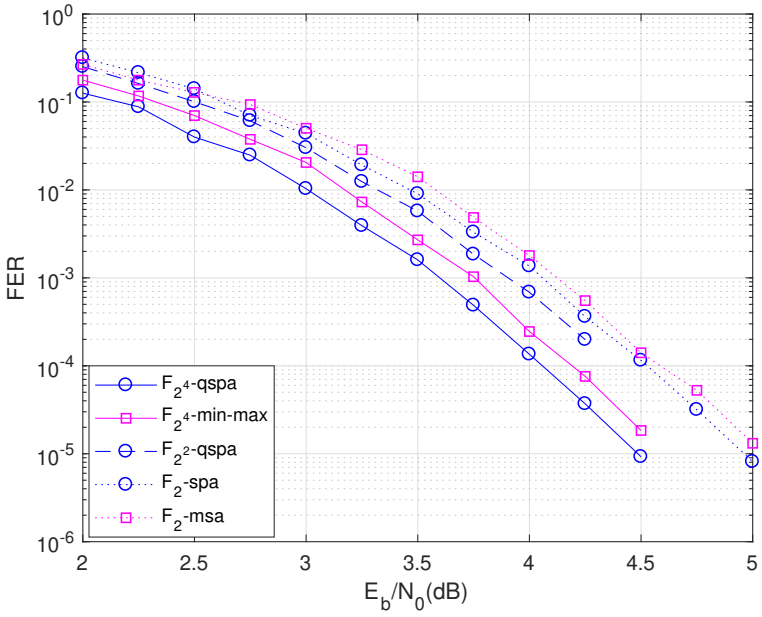

Figure 4.10: FER Performance with a (32, 16) Code over $\mathbb{F}_{2^4}$

FER performance of a few decoding strategies with $\mathfrak{C}_5$, a rate 0.5 code over $\mathbb{F}_{2^4}$, 32 symbols in length, is presented in Figure 4.10. This code was originally proposed in [16] based on an irregular protograph, and is also presented as a candidate for short block length transmissions in space data systems [62]. Once more we consider expansions over $\mathbb{F}_{2^2}$ and the binary field $\mathbb{F}_2$. As performance of $\mathbb{F}_{2^2}$-min-max was found to be very close to SPA, the respective curve was removed in the interest of a clearer figure. Further, performance of max-log-SPA on the original graph was identical to that of $\mathbb{F}_{2^4}$-min-max, and therefore that curve was also removed. Here, we set the maximum number of decoding iterations to 100 for all decoding schemes.

It can be seen from the figure that with $\mathfrak{C}_5$, $\mathbb{F}_{2^4}$-min-max performs closest to the optimal scheme, $\mathbb{F}_{2^4}$-QSPA, within 0.15dB at a FER of 10^{-5}. At the same level, $\mathbb{F}_{2^2}$-QSPA has a gap of about 0.4dB. This widens to 0.5dB when the binary expansion is used with SPA, while the min-sum simplification results in a further 0.075dB loss.

Optimum values for δ and ψ were found to be, respectively, 0.2 and 0.3, for all three decoding schemes on graph expansions, $\mathbb{F}_{2^2}$-QSPA, $\mathbb{F}_2$-SPA, and $\mathbb{F}_2$-MSA. Best value for ω, in

MSA, was 1.

As discussed at the start of Section 4.3, scaling factors play an important role when using subfield expansions for decoding. While they were optimized through Monte-Carlo simulations for obtaining the results presented in this section, a rough idea on the suitable ranges for the scaling factors could be obtained by analyzing the results.

The motivation for scaling factor δ, used in initialization (see (4.17) and (4.22)), is to mitigate the effects of erroneous channel estimates. When using subfield expansions for decoding, initial estimates for the parity symbols of the local code are computed using channel estimates for the information symbols. Therefore, any error in them could get duplicated a number of times and then significantly damage decoding performance. In the expansion of a code with a relatively low check degree, a higher percentage of neighboring variable nodes of a check node could represent parity symbols. In such a scenario, check node computations get *dominated* by estimates sent from those nodes, and duplicated errors in channel estimates would have a larger negative impact on decoding. Therefore, the optimum value of δ will generally be small for codes with relatively low check degrees. Moreover, due to the different form of LLR computation in min-max decoding (see (2.21)), where the most likely symbol is given more importance, erroneous initial estimates could be especially damaging. Therefore, the best option when using min-max decoding on subfield expansions is to set the initial estimates for parity symbols to zero, or $\delta = 0$.

The second scaling factor ψ is used to reduce the negative impacts of short cycles among the *local* check nodes. These checks have no relation to the constraints of the NB-LDPC code. Rather, they ensure that the set of new nodes that replaced a single variable node of the original graph converge to a valid codeword of the local code. As discussed earlier as well, considering the impact of short cycles, and the fact that the local code is of less importance in the decoding process, a small value should be picked for ψ. In the results presented here also, optimum value for ψ was found to be 0.25 in many cases.

Simulation results show that decoding algorithms implemented on proposed graph expansions are capable of performing quite close to those that use the original graph. For any algorithm, performance gap of decoding on the expanded graph and using the original widens when the size of the field used for the expansion decreases. With a few different graph expansions possible, many decoding options are available for any given code. As discussed in the next section, all these decoding schemes provide attractive complexity gains, with different levels of performance-complexity trade-offs.

4.3.3 Complexity

In the following, we analyze the complexities of some decoding schemes on expanded graphs. The two popular versions of QSPA, LLR-QSPA [26] and FFT-QSPA [25], are of different complexity orders and therefore, they are considered separately in the complexity analysis. We also consider min-max algorithm [28], the best performing simplification of QSPA as was seen in the previous section. Decoding complexities of using these algorithms on the original graph, and on subfield expansions with the modifications proposed in Section 4.3, are compared since the complexity gain offered by an expansion is dependant on the algorithm used. As NB-LDPC codes are most often defined over finite fields of characteristic 2 [5], a code over $\mathbb{F}_{2^r}$ is used in the complexity analysis, and m is used to denote a factor of r.

Similar to the previous section, we consider using SPA and MSA on binary expansions. Although they are special cases of QSPA and EMSA [33], SPA and MSA have a particularly unique feature that is absent in a direct implementation of QSPA or EMSA over $\mathbb{F}_2$. LLR domain versions of SPA and MSA use a single LLR value, instead of a vector of length 2. This makes the convolution operation unnecessary at check nodes of SPA [4]. Thus, complexities for a general m do not fully represent the complexities of these algorithms, and we present them separately.

Complexities of the two major steps in iterative decoding, check node operations and variable node operations, are compared separately. For the comparison, we consider operations at a single node of each type, in the original graph, during one iteration. Since the proposed expansions replace each node over $\mathbb{F}_{2^r}$ with $\frac{2^r-1}{2^m-1}$ nodes, complexity of that many nodes together is considered for decoding schemes on expanded graphs. Also, as explained in the previous chapter, these graphs have the additional feature of *local* check nodes. Since $(\frac{2^r-1}{2^m-1} - \frac{r}{m})$ such nodes are added per variable node of the original graph, variable node complexities of schemes on expanded graphs include complexities of local check nodes.

At hardware level, apart from the number of operations, the type of operation also affects the complexity. It is well-known that operations such as multiplications are more complex than comparisons [29]. Therefore, we consider the number of operations of a few different types; comparisons (Comp), additions/subtractions (Add), multiplications/divisions (Mult) and table look-ups (LUT). Note that $\max^*$ operation in LLR-QSPA can be performed with one comparison, two additions, and one table look-up [26]. Transformation between $\log$ and probability domains, required in FFT-QSPA, and $\tanh$ and $\arctanh$ computations in SPA can be carried out with look-up tables. It has also been assumed that the forward-backward approach [28] is used in check node operations of

the three algorithms. Further, cost of permuting probability vectors has been disregarded, since its impact on total complexity is negligible. We also disregard the complexities of multiplication and division by 2, used in (4.23) of SPA, and computation of the sign, in (4.26) of MSA, as at hardware level, these only amount to some bit-wise operations.

When representing complexities, we use d_c and d_v to denote the average degrees of a check node and a variable node in the original graph over $\mathbb{F}_{p^r}$, while d_l denotes the degree of a *local* check node in an expanded graph. Due to these new check nodes, average variable node degree would be slightly higher than d_v in the expanded graphs, and we denote this new value with $\widetilde{d_v}$, which can be computed as;

$$\widetilde{d_v} = d_v + d_l \times (\frac{2^r - 1}{2^m - 1} - \frac{r}{m}) \times (\frac{2^r - 1}{2^m - 1})^{-1} \tag{4.28}$$

As discussed in the previous chapter, local check nodes represent constraints of a simplex code, which are codewords of its dual, a Hamming code. Therefore, it could always be ensured that all local check nodes are of degree 3, and the complexity analysis we present here is for such a case. Simplifying (4.28) and substituting $d_l = 3$ result in;

$$\widetilde{d_v} = d_v + 3 - 3 \times \frac{r(2^m - 1)}{m(2^r - 1)} \tag{4.29}$$

Because of the higher complexity reductions it offer, we wil focus more on the binary expansion in this section. For the binary case, average variable node degree will be represented with $\widetilde{d_v^*}$, which takes the following value.

$$\widetilde{d_v^*} = d_v + 3 - 3 \times \frac{r}{2^r - 1} \tag{4.30}$$

Note that the average degree of *regular* check nodes in any expanded graph remains d_c.

Table 4.1 below lists complexities of check node operations of each decoding setup, while Table 4.2 considers variable node operations. In order to make the tables more succinct, we use E_f and L_f to denote, respectively, the number of $\mathbb{F}_{p^m}$ nodes per $\mathbb{F}_{p^r}$ node, and the number of local check nodes per $\mathbb{F}_{p^r}$ variable node. We remind that the values represented are

$$E_f = \frac{2^r - 1}{2^m - 1} \qquad\qquad L_f = \frac{2^r - 1}{2^m - 1} - \frac{r}{m} \tag{4.31}$$

With the binary expansion, where $m = 1$, the values are;

$$E_f^* = 2^r - 1 \qquad\qquad L_f^* = 2^r - 1 - r \tag{4.32}$$

Algorithm	*Check Node Operations*			
	Comp	Add	Mult	LUT
$\mathbb{F}_{2^r}$-LLR-QSPA	$(3d_c-4)\times 2^r(2^r-1)$	$(3d_c-4)\times 2^r(3.2^r-2)$	0	$(3d_c-4)\times 2^r(2^r-1)$
$\mathbb{F}_{2^m}$-LLR-QSPA	$E_f(3d_c-4)\times 2^m(2^m-1)$	$E_f(3d_c-4)\times 2^m(3.2^m-2)$	0	$E_f(3d_c-4)\times 2^m(2^m-1)$
$\mathbb{F}_{2^r}$-FFT-QSPA	0	$2d_c\times 2^r r$	$(2d_c-1)\times 2^r$	$2d_c\times 2^r$
$\mathbb{F}_{2^m}$-FFT-QSPA	0	$E_f.2d_c\times 2^m m$	$E_f(2d_c-1)\times 2^m$	$E_f.2d_c\times 2^m$
$\mathbb{F}_{2^r}$-Min-Max	$(3d_c-4)\times 2^r(2.2^r-1)$	0	0	0
$\mathbb{F}_{2^m}$-Min-Max	$E_f(3d_c-4)\times 2^m(2.2^m-1)$	0	0	0
$\mathbb{F}_2$-SPA	0	0	$(2^r-1)\times(2d_c-1)$	$(2^r-1)\times 2d_c$
$\mathbb{F}_2$-MSA	$(2^r-1)\times(2d_c-3)$	0	0	0

Table 4.1: Comparison of Check Node Complexity

Table 4.1 shows that using expanded graphs to implement a decoding algorithm offers complexity reductions when compared to using the same algorithm on the original graph. The significance of this complexity gain depends on the algorithm itself. For both LLR-QSPA and min-max decoding, using an expanded graph instead of the original results in a significant reduction in complexity, while for FFT-QSPA, the gains are more modest.

In the case of LLR-QSPA, using the original graph requires approximately $3d_c \times 2^{2r}$ comparisons, additions, and table look-ups, which results in an overall complexity of $\mathcal{O}(2^{2r})$. However, with the expansion over $\mathbb{F}_{2^m}$, only around $3d_c \times 2^{r+m}$ operations of each type become necessary, which results in an overall complexity of $\mathcal{O}(2^{r+m})$. This is a significant gain, especially in the cases with a large r, and we feel that, as a trade-off, the small performance losses observed in the previous section are justifiable. A similar level of complexity reduction can also be observed for the min-max algorithm, from $\mathcal{O}(2^{2r})$ to $\mathcal{O}(2^{r+m})$. Although they are of the same complexity order, it should be noted that min-max decoding is significantly simpler than LLR-QSPA, since only comparisons are required. Using min-max algorithm on an expanded graph offers combined complexity gains of both the simplified algorithm and the graph expansion. The performance-complexity trade-off offered by this particular scheme is very attractive, especially since the performance loss relative to

the best performing algorithm is less than 0.5dB, as seen earlier. Gains of using expanded graphs reduce in the case of FFT-QSPA. Here, the number of multiplications and table look-ups required are almost the same, approximately $2d_c \times 2^r$, when using the original graph or one of its expansions. There is a slight reduction in the number of additions though, from approximately $2d_c \times 2^r r$ to $2d_c \times 2^r m$. Thus, the overall complexity of FFT-QSPA on an expanded graph is $\mathcal{O}(2^r m)$, slightly lower than $\mathcal{O}(2^r r)$ on the original graph.

Table also shows that using SPA and MSA on the binary expansion is the most advantageous in terms of complexity. Although FFT-QSPA on the original graph and SPA on the binary graph require roughly the same number of multiplications and table look-ups for check node operations, SPA does not require any additions. Its complexity is linear in field size 2^r, $\mathcal{O}(2^r)$, a very significant reduction when compared with the usual quadratic complexity, $\mathcal{O}(2^{2r})$. MSA on the binary expansion is of the same linear complexity class, but since only comparisons are required, it would be even less complex at hardware level. If compared with using min-max algorithm on the original graph, which also only uses comparisons at check nodes, it can be observed that MSA requires only a $\frac{1}{2^r}$ fraction of the comparisons necessary for min-max decoding.

As explained earlier as well, we include the complexity of the L_f local check nodes added for each $\mathbb{F}_{2^r}$ variable node when computing variable node complexities of decoding schemes on expanded graphs. Note that complexity of one such node can be derived by substituting $d_l = 3$ as the node degree, and 2^m as the field size, in the expressions for the respective algorithm in Table 4.1. Due to this additional cost, complexity at variable nodes are slightly *higher* in proposed schemes, but not high enough to completely offset the complexity gain obtained at check node operations, especially when using LLR-QSPA and min-max decoding, or binary expansions.

Table 4.2 presents variable node complexities of the decoding schemes considered in Table 4.1. There, for the schemes that use expanded graphs, complexity at new variable nodes and local check nodes are given separately, but should be considered together when comparing different decoding strategies. As the table shows, complexity at variable nodes for all algorithms on the original graph is $\mathcal{O}(2^r)$. By substituting for $L_f, \widetilde{d}_v$ and $\widetilde{d}_v^*$ from (4.29) and (4.30), it can be seen that for algorithms on expanded graphs, complexity is $\mathcal{O}(2^{r+m})$. But as explained with Table 4.1, complexity order changes from $\mathcal{O}(2^{2r})$ to $\mathcal{O}(2^{r+m})$ ($\mathcal{O}(2^r)$ with the binary expansion) at check nodes when using expanded graphs with LLR-QSPA and min-max algorithm, a more significant reduction than the increase at variable nodes. Thus, for those algorithms, the overall complexity gain would still be quite significant, while for FFT-QSPA, which had a more modest gain at check nodes, overall gain could be minimal.

Algorithm	*Variable Node Operations*			
	Comp	Add	Mult	LUT
$\mathbb{F}_{2^r}$-LLR-QSPA	$2^r - 1$	$2d_v \times 2^r$	0	0
$\mathbb{F}_{2^m}$-LLR-QSPA	$(r/m) \times (2^m - 1)$	$E_f.2\widetilde{d_v} \times 2^m$	0	0
Local Checks	$5L_f \times 2^m(2^m - 1)$	$5L_f \times 2^m(3.2^m - 2)$	0	$5L_f \times 2^m(2^m - 1)$
$\mathbb{F}_{2^r}$-FFT-QSPA	$2^r - 1$	$2d_v \times 2^r$	0	0
$\mathbb{F}_{2^m}$-FFT-QSPA	$(r/m) \times (2^m - 1)$	$E_f.2\widetilde{d_v} \times 2^m$	0	0
Local Checks	0	$6L_f \times 2^m m$	$5L_f \times 2^m$	$6L_f \times 2^m$
$\mathbb{F}_{2^r}$-Min-Max	$(d_v + 1) \times 2^r$	$3d_v \times 2^r$	0	0
$\mathbb{F}_{2^m}$-Min-Max	$(E_f.\widetilde{d_v} + r/m) \times 2^m$	$E_f.3\widetilde{d_v} \times 2^m$	0	0
Local Checks	$5L_f \times 2^m(2.2^m - 1)$	0	0	0
$\mathbb{F}_2$-SPA	r	$(2^r - 1) \times 2\widetilde{d_v^*}$	0	0
Local Checks	0	0	$(2^r - r - 1) \times 5$	$(2^r - r - 1) \times 6$
$\mathbb{F}_2$-MSA	r	$(2^r - 1) \times 2\widetilde{d_v^*}$	0	0
Local Checks	$(2^r - r - 1)$	0	0	0

Table 4.2: Comparison of Variable Node Complexity

Table 4.1 and Table 4.2 demonstrate that decoding on expanded graphs is advantageous in terms of asymptotic complexity, while the actual performance gains would depend on parameters of the code used, such as field sizes, code length, rate, and average node degrees. In Table 4.3, we present complexities in terms of the number of operations required per iteration for decoding two different NB-LDPC codes; a (1998, 1776) code over $\mathbb{F}_{2^6}$ ($\mathfrak{C}_2$ in previous section) and a (816, 408) code over $\mathbb{F}_{2^3}$ ($\mathfrak{C}_3$ in previous section). Decoding schemes considered are LLR-QSPA, FFT-QSPA, and min-max decoding on the original graph, and the ones on expanded graphs for which FER performances were depicted in Figure 4.7 and Figure 4.8.

Due to different operations being of different complexities when implemented in hardware, Table 4.3 presents number of operations required of four different types; comparisons (Comp),

Code	*Algorithm*	*Number of Operations* ($\times 10^5$)			
		Comp	Add	Mult	LUT
$\mathfrak{C}_2$	$\mathbb{F}_{2^6}$-LLR-QSPA	932.17	2817.73	-	930.91
	$\mathbb{F}_{2^3}$-LLR-QSPA	155.8 ($\approx 17\%$)	507.01 ($\approx 18\%$)	-	155.52 ($\approx 17\%$)
	$\mathbb{F}_{2^2}$-LLR-QSPA	79.94 ($\approx 8\%$)	287.93 ($\approx 10\%$)	-	79.76 ($\approx 8\%$)
	$\mathbb{F}_{2^6}$-FFT-QSPA	1.26	71.61	10.09	10.23
	$\mathbb{F}_{2^3}$-FFT-QSPA	0.28 ($\approx 22\%$)	72.89 ($\approx 101\%$)	16.94 ($\approx 168\%$)	18.22 ($\approx 178\%$)
	$\mathbb{F}_{2^2}$-FFT-QSPA	0.18 ($\approx 14\%$)	66.17 ($\approx 92\%$)	20.43 ($\approx 202\%$)	22.06 ($\approx 215\%$)
	$\mathbb{F}_{2^6}$-Min-Max	1882.99	15.35	-	-
	$\mathbb{F}_{2^3}$-Min-Max	342.7 ($\approx 18\%$)	27.33 ($\approx 178\%$)	-	-
	$\mathbb{F}_{2^2}$-Min-Max	197.38 ($\approx 10\%$)	33.09 ($\approx 215\%$)	-	-
$\mathfrak{C}_3$	$\mathbb{F}_{2^3}$-LLR-QSPA	59.98	193.23	-	59.4
	$\mathbb{F}_{2^3}$-FFT-QSPA	0.58	26.11	6.2	6.53
	SPA	0.24 ($\approx 41\%$)	7.67 ($\approx 29\%$)	7.06 ($\approx 114\%$)	7.67 ($\approx 118\%$)
	$\mathbb{F}_{2^3}$-Min-Max	131.21	9.79	-	-
	MSA	5.43 ($\approx 4\%$)	7.67 ($\approx 78\%$)	-	-

Table 4.3: Number of Operations per Iteration with $\mathfrak{C}_2$ and $\mathfrak{C}_3$

additions/subtractions (Add), multiplications/divisions (Mult), and table look-ups (LUT). Of these, comparisons are the least complex in practical implementations, while multiplications/divisions are the most complex. Table look-ups are not very complex operations, but require special hardware for implementing. For decoding schemes over expansions, we also present the number of operations required as a percentage of the requirement when using the same algorithm with the original graph. Since SPA and MSA do not have direct equivalents in the non-binary domain, we compare with, respectively, FFT-QSPA and min-max algorithm instead.

In Table 4.3, we observe that, in the case of $\mathfrak{C}_2$, using LLR-QSPA on expanded graphs offer exceptional complexity gains. Less than 20% of the operations required for decoding on the original graph are necessary when using the $\mathbb{F}_{2^3}$ expansion. This reduces further with the $\mathbb{F}_{2^2}$ expansion, to less than 10%. These gains correspond to speed-ups of *more than 5 times* in the $\mathbb{F}_{2^3}$ case, and *more than 10 times* in the $\mathbb{F}_{2^2}$ case. Considering that the performance losses, as shown in

Figure 4.7, are only 0.2dB and 0.3dB, the complexity gains here are very attractive. Implementing FFT-QSPA on expanded graphs are not particularly advantageous though. Only gain with the $\mathbb{F}_{2^3}$ expansion, when compared with using the same algorithm on the original $\mathbb{F}_{2^6}$ graph, is in the number of comparisons required. Both decoding schemes use a similar number of additions, while the setup on the expanded graph needs significantly more multiplications and table look-ups. This is due to the operations of local check nodes, which are absent in the original graph. With $\mathbb{F}_{2^2}$ expansion, the number of comparisons required reduces further, and the number of additions is also slightly less than in the $\mathbb{F}_{2^6}$ case. Since $\mathbb{F}_{2^2}$ expansion has more local check nodes than the $\mathbb{F}_{2^3}$ one, the number of multiplications and table look-ups have increased even further. Thus, for $\mathfrak{C}_2$, using FFT-QSPA with any of the two expansions considered is not advantageous in terms of complexity. The case of min-max decoding is very similar to that of LLR-QSPA; complexity gains are significant, and they are higher when the size of the subfield used for the expansion is smaller. Once more, due to local check nodes, the number of additions when using expanded graphs is higher than with the original graph. Nevertheless, since the reduction in the number of comparisons is much higher in magnitude, min-max decoding on expanded graphs is significantly less complex for $\mathfrak{C}_2$.

For $\mathfrak{C}_3$, only one graph expansion is possible, which is over the binary subfield $\mathbb{F}_2$. With this expansion, decoding algorithms operating in the binary domain can be used, and in Figure 4.8, we considered SPA and MSA. Number of operations required in these schemes are given in Table 4.3, and it can be seen that both provide exceptional complexity gains. SPA requires significantly less comparisons and additions, and slightly more multiplications and table look-ups than FFT-QSPA on the original graph. Since the reduction in the number of additions is quite large in magnitude, overall complexity advantage here is still very significant. The gain is even bigger if we compare with LLR-QSPA, the version of QSPA that is better suited for hardware implementations [29]. Largest gain in complexity is offered by MSA on the binary expansion. It can be observed from the table that this scheme needs only 4% of the comparisons required by its closest non-binary counterpart, min-max algorithm on the original graph. It requires significantly less additions as well. As seen in Figure 4.8, SPA and MSA on binary expansions have gaps of 0.25dB and 0.55dB with QSPA on the original graph, and when considering the gains in complexity, performance losses of this level would be readily accepted in most applications.

Combining above complexity analysis with the results presented in the previous section, it is clear that the strategy of using expanded graphs for decoding is capable of offering very attractive performance-complexity trade-offs. Most decoding algorithms proposed for NB-LDPC codes in the literature are of complexity $\mathcal{O}(2^{2r})$ for a code over $\mathbb{F}_{2^r}$, and implementing those algorithms

on expanded graphs results in significant complexity gains with minimal performance losses. For algorithms where complexity is not polynomial in field size, such as FFT-QSPA, the proposed strategy may not be advantageous. But as [29], [63] and [64] point out, LLR-QSPA is the more suitable version of QSPA for hardware implementations. It does not require any multiplications, and as the algorithm operates entirely in LLR domain, normalization is unnecessary, which improves stability [64]. Further, in practical implementations of decoders, any value will be represented with a finite number of bits, and therefore, channel information will have to undergo some form of quantization. It has been observed that LLR-QSPA is less sensitive to effects of quantization, when compared with FFT-QSPA [64]. Therefore, from the perspective of practical usage, decoding on expanded graphs can be considered a strategy that offers exceptional complexity gains.

Also, NB-LDPC codes over a field larger than $\mathbb{F}_{2^8}$ are not widely adopted in practice due to complexity issues as well as the fact that permutation of probability vectors at check nodes is only practically feasible for small field sizes [29]. But, with the proposed strategy, a code defined over a large field could be decoded over a much smaller subfield, thereby allowing us to circumvent these practical issues. Thus, it enables realizing the promised gains of NB-LDPC codes over large fields in practice.

Out of all decoding schemes considered, most attractive complexity gains are offered by those on binary expansions. Since SPA and MSA are decoding algorithms of binary LDPC codes, which are widely used in practice, very efficient hardware implementations for them are already available commericially. They have been optimized over many years, and would be considerably less costly to implement than the ones for QSPA or any of its simplifications. Our proposal allows these technologies to be used for decoding NB-LDPC codes as well.

In conclusion, decoding schemes on expanded graphs that we propose could be used to reduce decoding complexity in most practical applications that adopt NB-LDPC codes. In particular, our proposed strategy enables decoding a code defined over a large field using a graph over a much smaller field, while providing a good performance and complexity tradeoff, leading to a practical solution to decoding NB-LDPC codes.

4.4 Coset Reliability based Majority Logic Decoding

For NB-LDPC codes, majority logic decoding (MLgD) is the approach closest to bit flipping decoding that is used with its binary counterparts. As discussed in Chapter 2, main focus of MLgD algorithms is to reduce the decoding complexity of NB-LDPC codes so that they may also be

used in resource-constrained systems. Many such algorithms have been proposed in the literature, for example in [24], [37], [38], [39], [23], and these are of very low complexity when compared with any QSPA variant. Moreover, they require only additions and finite field arithmetic, and as such, can be very efficiently implemented in hardware. But the performance losses of MLgD algorithms, relative to QSPA, are also very significant, particularly with randomly constructed NB-LDPC codes, where gaps in excess of 1dB has been observed. Further, many instances where these algorithms exhibit significantly high error floors have also been reported in the literature [39].

In this section we propose a novel MLgD algorithm based on the *binary* expansion of a code over $\mathbb{F}_{2^r}$, presented in Chapter 3. The new algorithm falls into the category of soft reliability based MLgD algorithms, meaning some soft information from the channel is required for decoding. We specifically chose the binary expansion since this allows us to devise an algorithm that operates in the binary domain. This is advantageous due to two reasons. First is in relation to the error floor. High error floors of MLgD algorithms for NB-LDPC codes are often attributed to not all symbol reliabilities getting updated in every iteration [39]. An algorithm operating in the binary domain would not have this short coming. The second reason is in relation to hardware implementations. Many calculations in a binary decoding algorithm can be reduced to bit-wise operations, which makes the algorithm more efficiently implementable in practice. It should also be noted that bit-wise operations are also advantageous in terms of complexity.

As explained in Chapter 3, binary expansion we propose for a code over $\mathbb{F}_{2^r}$ is based on additive subgroups of order 2^{r-1} and their proper cosets. Section 4.3.1 presented the soft decoding scheme on this expansion, which operates in the LLR domain, with the LLRs defined as in (4.20). Similarly, with MLgD, we propose using reliability values of variables over $\mathbb{F}_{2^r}$ belonging to the proper cosets of the subgroups used for the expansion. Therefore, the novel algorithm will be referred to as *coset reliability based majority-logic decoding* (CRB-MLgD). In the following, we outline the three major steps of CRB-MLgD, initialization, check node operations, and variable node operations. For the explanation, we consider a code over $\mathbb{F}_{2^r}$, transmitted over the BI-AWGN channel, with traditional BPSK modulation, where $0 \rightarrow +1$ and $1 \rightarrow -1$. Also, it is assumed that the system on which the novel algorithm is implemented has the capacity to use integers of a maximum bit length b_l. Similar to other soft reliability based MLgD algorithms [24], [38], at the receiver, values sampled from the channel will be uniformly quantized into $(2^{b_l} - 1)$ intervals, with symmetrical clipping with respect to the origin. Each interval can then be represented with an integer of b_l bits.

1. Initialization

In the expansion, each $\mathbb{F}_{2^r}$ variable node is represented with $(2^r - 1)$ binary variable nodes. As explained in Chapter 3, each set of such binary nodes can be considered to form an instance of a $(2^r - 1, r)$ linear code. Thus, similar to soft decision decoding (see Section 4.3.1), r of the nodes can be chosen to represent the information bits of this local code. These r nodes are initialized with scaled versions of their quantized channel information, where the scaling factor w_c has to be optimized per code. The following represents this operation for the n'th $\mathbb{F}_{2^r}$ variable node. $\mathfrak{C}_{n,i}$ is the quantized channel information for the i'th binary node of node n, and $\mathfrak{V}_{n,i}^{(0)}$ represents the reliability value of this binary node after initialization.

$$\mathfrak{V}_{n,i}^{(0)} = w_c \cdot \mathfrak{C}_{n,i} \qquad i = 0, 1, \dots r-1 \tag{4.33}$$

Keeping in mind that BPSK modulation is used for data transmission, the most likely value for each of the r bits in (4.33) can be simply obtained by comparing $\mathfrak{V}_{n,i}^{(0)}$ (or $\mathfrak{C}_{n,i}$) with 0; if $\mathfrak{V}_{n,i}^{(0)} \geq 0$, the most likely value for the i'th bit of node n, $\mathfrak{e}_{n,i}^{(0)}$, is 0, else it is 1.

Remaining $(2^r - 1 - r)$ binary nodes represent the parity bits of the local code, and therefore, the most likely values $\mathfrak{e}_{n,i}^{(0)}$ for $i\text{`} = r, \dots, 2^r - 2$, can be derived using the constraints of that code. This operation is given in the following, where $\mathfrak{I}_{i\text{`}}$ denotes the set of information bits i involved in generating parity bit i`, and $\oplus$ represents binary addition.

$$\mathfrak{e}_{n,i\text{`}}^{(0)} = \bigoplus_{i \in \mathfrak{I}_{i\text{`}}} \mathfrak{e}_{n,i}^{(0)} \qquad i\text{`} = r, \dots, 2^r - 2 \tag{4.34}$$

Sign of the reliability values of the parity bits will be decided by the most likely values computed as in (4.34); if $\mathfrak{e}_{n,i\text{`}}^{(0)} = 0$, reliability will be positive, else negative. We propose computing a magnitude for the reliability values, as in (4.35) given below, where the constraints of the local code are used once more.

$$|\mathfrak{V}_{n,i\text{`}}^{(0)}| = \text{ceil}\left(\delta \cdot \min_{i \in \mathfrak{I}_{i\text{`}}} |\mathfrak{V}_{n,i}^{(0)}|\right) \qquad i\text{`} = r, \dots, 2^r - 2 \tag{4.35}$$

where δ is another scaling factor ($0<\delta<1$), which plays the same role as δ in (4.22), reducing error propagation caused by duplication of channel information. Similar to w_c in (4.33), δ also has to be optimized per code.

After initialization, each binary node will send its estimated value, along with the reliability, to neighboring check nodes. The connection between a $\mathbb{F}_{2^r}$ node, and the binary nodes that replaced it, is mostly used during the initialization. Therefore, we change our notations from hereon,

in order to make the expressions more succinct. Instead of referring to a binary node in relation to the $\mathbb{F}_{2^r}$ node it replaced, as (n, i), each binary check node and variable node would be considered on its own, and referred to with, respectively, m and n.

2. Check Node Operations

Task of a binary check node in the expansion is to compute a bit estimate for each neighboring variable node, along with a value that reflects its reliability. It is possible to compute these without violating the extrinsic principle of message passing, while ensuring that complexity or resource requirements are not increased significantly when compared with existing soft MLgD algorithms [24], [38].

(4.36) in the following represents the computation of ${}^{\mathfrak{e}}\mathfrak{p}_{m,n}^{(k)}$, the bit estimate for the n'th variable node at the m'th check node, during the k'th decoding iteration. ${}^{\mathfrak{e}}\mathfrak{r}_{n,m}^{(k)}$ denotes the estimate sent from variable node n to check node m, and $N(m)$ is the neighborhood of that check node.

$$ {}^{\mathfrak{e}}\mathfrak{p}_{m,n}^{(k)} = \bigoplus_{i \in N(m);\, i \neq n} {}^{\mathfrak{e}}\mathfrak{r}_{i,m}^{(k-1)} \tag{4.36} $$

Similar to the initialization, sign of the reliability value will be decided by the bit estimate, ${}^{\mathfrak{e}}\mathfrak{p}_{m,n}^{(k)}$. Its magnitude can be computed as;

$$ |{}^{\mathfrak{v}}\mathfrak{p}_{m,n}^{(k)}| = \min_{i \in N(m);\, i \neq n} |{}^{\mathfrak{v}}\mathfrak{r}_{i,m}^{(k-1)}| \tag{4.37} $$

where ${}^{\mathfrak{v}}\mathfrak{p}_{m,n}^{(k)}$ and ${}^{\mathfrak{v}}\mathfrak{r}_{n,m}^{(k)}$ are the reliability values sent from, respectively, check node m to variable node n, and variable node n to check node m, in iteration k.

3. Variable Node Operations

Upon receiving messages from the neighboring check nodes, each variable node n will update its reliability value, $\mathfrak{V}_n^{(k)}$, as;

$$ \mathfrak{V}_n^{(k)} = \mathfrak{V}_n^{(0)} + w_e \cdot \sum_{j \in M(n)} {}^{\mathfrak{v}}\mathfrak{p}_{j,m}^{(k)} \tag{4.38} $$

(4.38) is very similar to the operation depicted in (4.24), of soft decoding on the binary expansion. Similar to that, updating the reliability value consists of simply adding the initial reliability, computed as in (4.33)-(4.35), to those provided by the check nodes in the particular node's neighborhood $M(n)$. But unlike in (4.24), here we do not differentiate between the estimates of regular and local check nodes of the expansion, since due to the simplistic nature of the decoding

algorithm, all reliability estimates are lacking in accuracy. Instead, we use a scaling factor w_e with the combined estimates of all neighboring check nodes, which has to be optimized per code.

After updating, $\mathfrak{V}_n^{(k)}$ is used to take a tentative decision on the bit represented with node n. Similar to any iterative decoding algorithm, the tentative decisions are used to test for convergence. As we proposed in Section 4.3, this test can be conducted with only the r information bits, and mapping each set to $\mathbb{F}_{2^r}$ symbols through the alternative vector representation. But this requires conducting calculations over $\mathbb{F}_{2^r}$, for which specialized hardware, such look-up tables, are necessary. Therefore, since MLgD is targeted primarily for resource constrained systems, better option for convergence test is to simply use the constraints in the binary graph. Only bit-wise XOR operations are necessary to test each such constraint, and therefore, this approach does not lose much in terms of complexity.

In the majority of MLgD algorithms, extrinsic principle is violated at variable node operations [24], [38]. This is mostly due to the gain in complexity by not having to compute a separate message for each neighboring variable node. Since most MLgD algorithms *accumulate* reliability values through iterations [24], [23], adhering to the extrinsic principle would also require more memory. But even in MLgD algorithms that do not accumulate reliability values, such as IISRB-MLgD [38], extrinsic principle is not followed at variable nodes. As these algorithms still operate in the non-binary domain, not all symbol reliability values will be updated in each iteration, and therefore, by following the extrinsic principle, there is a risk that the symbol reliability vector sent to some neighboring check nodes being very similar to the one sent in the earlier iteration, which can impact the speed of convergence. Out of these reasons, only the issue of complexity applies to CRB-MLgD, since it is an algorithm that operates in the binary domain, without accumulating reliability values. Nevertheless, that complexity gain could be quite significant since savings are per iteration, and therefore, we take the approach of simply sending $\mathfrak{V}_n^{(k)}$, in (4.38), as the reliability value to all neighboring check nodes. The tentative decision taken based on $\mathfrak{V}_n^{(k)}$ will be the bit estimate.

Similar to any soft reliability based MLgD algorithm, CRB-MLgD could be used in scenarios where the channel offers some soft information, but available hardware resources do not permit using a more established soft decoding algorithm. In the following, we compare FER performance of CRB-MLgD with some other soft reliability based MLgD algorithms proposed in the literature. Later on, we analyze the complexity of the proposed scheme, and compare with existing algorithms.

4.4.1 Performance

We compare performance of CRB-MLgD with the two most well-known soft reliability based MLgD algorithms in the literature, ISRB-MLgD [24] and IISRB-MLgD [38]. Both systematically constructed codes with larger variable degrees, and randomly constructed codes, are used in the comparison. Performance of QSPA is used as the benchmark to compare performance of MLgD algorithms. BI-AWGN channel was used for the simulations, with BPSK modulation. All algorithms use 12-bit uniform quantization, with symmetrical clipping with respect to the origin, which allows reliability values to be represented with 12-bit integers. Scaling factors λ in ISRB-MLgD [24], ξ_1 and ξ_2 in IISRB-MLgD [38], and w_c, w_e and δ in CRB-MLgD were optimized for best performance through simulations. Maximum number of decoding iterations was set to 50 for every algorithm.

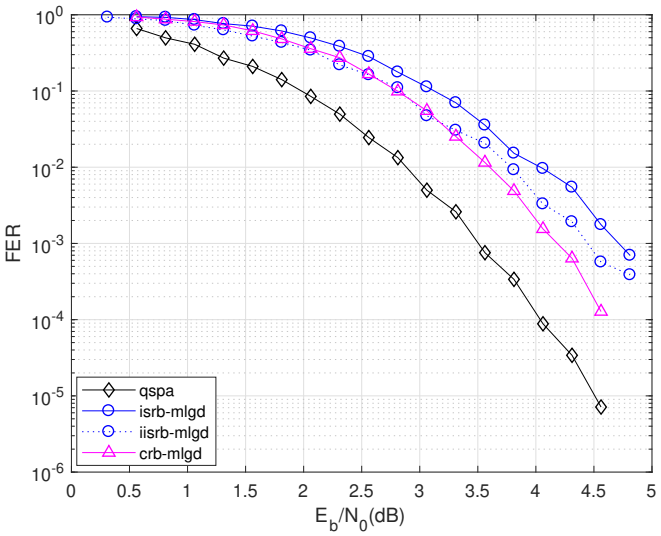

Figure 4.11: FER Performance with a (63, 37) Code over $\mathbb{F}_{2^8}$

Figure 4.11 shows FER performance of the MLgD algorithms considered, and QSPA, with a (63, 37) code over $\mathbb{F}_{2^3}$, the same code used in Figure 2.5, constructed following the method proposed in [23]. The code has a relatively large variable degree of 8, and thus, is better suited for MLgD. This can be observed from the figure, where all three MLgD algorithms can be seen to perform within 1dB of QSPA at a FER of 10^{-4}. Proposed scheme is the closest to QSPA, with a gap of just over 0.5dB. Another interesting observation could be made regarding the error floor. Both existing MLgD algorithms, particularly IISRB-MLgD, show early signs of error floor, while

the curves for QSPA and CRB-MLgD exhibit similarly sharp descents. This is an advantage of decoding over the binary expansion. There, in every decoding iteration, all bit reliability values are updated, whereas in other algorithms, only a few symbol reliability values will be updated, a reason for an early error floor [39].

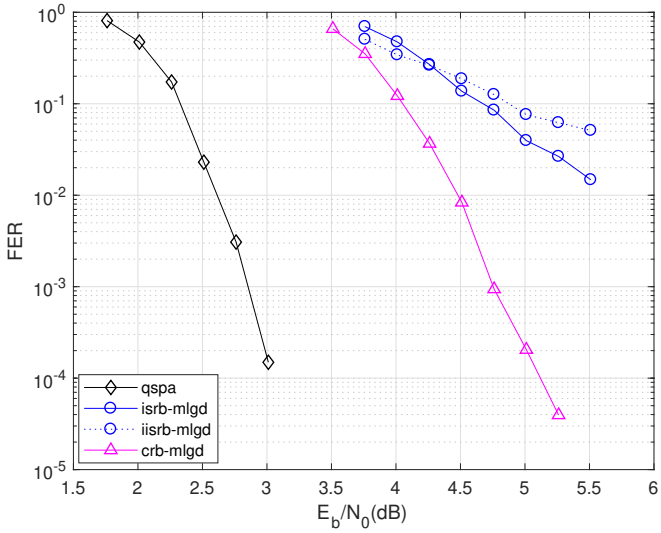

Figure 4.12: FER Performance with a (384, 256) Code over $\mathbb{F}_{2^4}$

Performance of MLgD algorithms with a (384, 256) NB-LDPC code over $\mathbb{F}_{2^4}$ is given in Figure 4.12. This is the same code we used for Figure 2.6, and was constructed by randomly re-labelling a binary graph obtained from [65]. Since the code is not from a structured construction method, its variable degree is relatively low, 3.375. MLgD approach is considered not quite suitable for using with such codes, for which Figure 4.12 supplies evidence. There is a very significant gap, several dBs large, between QSPA and MLgD algorithms. Here also the smallest gap is with the proposed approach, which is just over 2dB at a FER of 10^{-4}. Both ISRB-MLgD and IISRB-MLgD have gaps over 3dB with QSPA, at a quite high FER of 10^{-2}. Also, both algorithms show very early signs of error floor, while the curve for the proposed scheme almost parallels that of QSPA.

Figure 4.13 presents FER performance of the algorithms with a $(255, 175)$ code over $\mathbb{F}_{2^4}$. This code was also constructed using the method proposed in [23], and is of variable degree 16, which is fairly high when compared with randomly constructed codes. The high variable degree has resulted in reducing the gap between QSPA and MLgD algorithms, as the figure shows. At a FER of 10^{-4}, both CRB-MLgD and IISRB-MLgD have gaps around 0.25dB with QSPA. ISRB-MgD shows

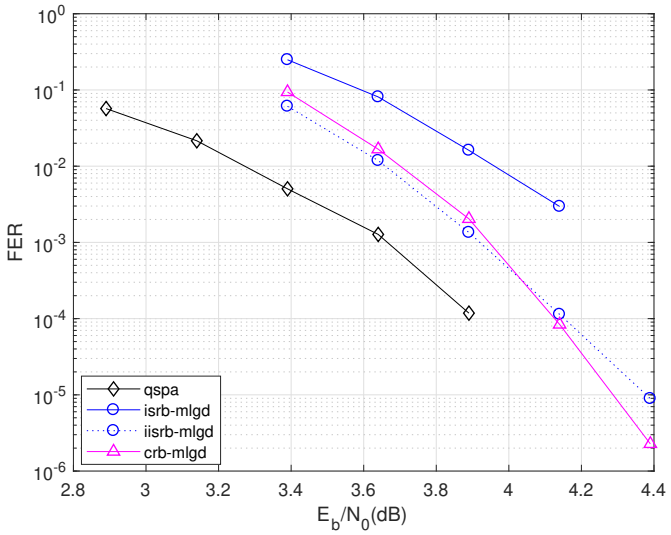

Figure 4.13: FER Performance with a (255, 175) Code over $\mathbb{F}_{2^4}$

a significantly higher gap though, about 0.6dB at a FER of 10^{-3}. It can also be seen that at higher error rates, IISRB-MLgD marginally outperforms the proposed approach. But around 10^{-4} FER, the two curves cross over, and at a FER of 10^{-5}, the proposed approach is leading IISRB-MLgD by about 0.1dB.

Simulation results presented show that CRB-MLgD is successful in reducing the gap between MLgD and QSPA, for both high variable degree constructions and random constructions. But in the case of random constructions, it does not manage to reduce the said gap to a level which makes MLgD practically feasible for such codes. Nevertheless, performance gains the proposed approach offers for high variable degree codes are very attractive, particularly with regard to the error floor. This gain in performance does not incur any significant complexity cost, as the analysis in the next section shows.

4.4.2 Complexity and Hardware Requirements

In the following, we analyze the decoding complexities and the resource requirements of ISRB-MLgD, IISRB-MLgD, and CRB-MLgD. As complexity is a primary concern when it comes to this class of algorithms, cost of initialization should also be considered in any analysis [39]. As such, we compare the complexity of the three algorithms during initialization, check node operations and variable node operations separately. Similar to in Section 4.3.3, the number of operations

required of a few different types, comparisons (Comp), additions/subtractions (Add), scaling operations (Scal), table look-ups (LUT), and XOR operations, are considered. It should be noted that all operations except XOR are carried out with integer operands. Scaling operations are the multiplications with optimized scaling factors, necessary to complete calculations such as (2.27), (4.33), (4.35), and (4.38). In most cases, these scaling factors would be *hardware-friendly* values, such as powers of 2, which allows multiplications to be completed with bit shifts. To reflect this, we chose to refer to them as scaling operations, rather than as multiplications. A code over $\mathbb{F}_{2^r}$ is used for the complexity analysis, which is assumed to be represented with a $n \times m$ PCM. Further, it is assumed that data is transmitted over the BI-AWGN channel.

Here also we consider the operations required to complete one decoding iteration, at one check/variable node, similar to in 4.2.3. Since it is more natural to view initialization from the perspective of variable nodes, complexity of that step is presented per one such node. In the case of CRB-MLgD, complexity of $(2^r - 1)$ nodes together are considered, as the expansion replaces one non-binary node with that many binary nodes. Further, although here we do not distinguish between regular and local check nodes, complexity of local check nodes are added to that of variable nodes, for the sake of convenience. In the following, we use d_c and d_v to represent check and variable degrees, and $\widetilde{d}_v^*$, given by (4.30), for variable degree in the binary expansion, which is larger than d_v due to additional connections with local check nodes. All local check nodes are considered to be of degree 3, which, as explained earlier, should always be possible. Complexity of testing decoder convergence is disregarded, since the operations required are almost the same for every algorithm.

Algorithm	*Initialization*			
	XOR	Comp	Add	Scal
ISRB-MLgD	-	$2^r - 1 + 2d_c \cdot \frac{m}{n}$	$2^r \cdot r$	2^r
IISRB-MLgD	-	$2^r - 1 + 2d_c \cdot \frac{m}{n}$	$2^r \cdot r$	2^r
CRB-MLgD	$(2^r - r - 1) \cdot \lvert\widetilde{\mathfrak{I}}\rvert$	$(2^r - r - 1) \cdot \lvert\widetilde{\mathfrak{I}}\rvert$	-	$2^r - 1$

Table 4.4: Initialization Complexity of MLgD Algorithms

Table 4.4 lists the complexities of initialization, for the three algorithms. In ISRB-MLgD and IISRB-MLgD, during initialization, computation of edge reliability values is carried out at check nodes, considering initial reliability vectors of all neighboring variable nodes [24], [38]. As the focus is on per variable node complexities, these values have been normalized by n, the number of variable nodes. In CRB-MLgD, since decoding is in binary domain, channel information received could be directly used to initialize the nodes representing the information bits of the local code. But

to derive the initial estimates for the remaining nodes, computations in (4.34) and (4.35) have to be carried out. When considering the complexities of those computations, we have used $|\widetilde{\mathfrak{I}}|$ to denote the average number of information bits involved in generating a parity bit in the local code. Note this is different from the degree of a local check node, as one such node could be connected to more than a single node representing parity bits.

Table 4.4 shows that during initialization, CRB-MLgD has a slight edge in complexity when compared with the other two algorithms. Both ISRB-MLgD and IISRB-MLgD require a fairly high number of comparisons, mainly due to the computation of edge reliability values. In CRB-MLgD, most of the operations required is for computing initial estimates for the nodes representing parity bits of the local code, and a significant number of those are XOR operations, which are very simple at hardware level.

Algorithm	*Check Node Operations*		
	XOR	Comp	LUT
ISRB-MLgD	-	-	$4d_c$
IISRB-MLgD	-	-	$4d_c$
CRB-MLgD	$(2^r-1)\cdot 2d_c$	$(2^r-1)\cdot 2d_c$	-

Table 4.5: Check Node Complexity of MLgD Algorithms

At check nodes though, the proposed approach is slightly more complex than the others. While only field arithmetic is required for ISRB-MLgD and IISRB-MLgD, which can be carried out using look-up tables, CRB-MLgD requires a number of comparisons, since the reliability update sent by check nodes change from iteration to iteration there. But we feel that the complexity cost of these dynamic reliability updates is justified by the gain in performance, particularly at low error rates, where the other two algorithms show error floor effects. Apart from comparisons, CRB-MLgD also requires a fair number of XOR operations, but their impact on complexity will be minimal.

Algorithm	*Variable Node Operations*			
	XOR	Comp	Add	Scal
ISRB-MLgD	-	2^r-1	$2d_v$	-
IISRB-MLgD	-	2^r-1	$2d_v$	d_v
CRB-MLgD	$6(2^r-r-1)$	$6(2^r-r-1)+r$	$(2^r-1)\cdot 2\widetilde{d}_v^*$	$(2^r-1)\cdot \widetilde{d}_v^*$

Table 4.6: Variable Node Complexity of MLgD Algorithms

Table 4.6 shows that when it comes to variable node operations, CRB-MLgD is at a distinct disadvantage in terms of complexity. This is mainly due to operations of local check nodes, which we have considered together with variable node operations. As the table shows, proposed approach requires significantly more additions and scaling operations compared to the other two, and a higher number of comparisons as well. CRB-MLgD also needs a number of XOR operations, which is not necessary for ISRB-MLgD and IISRB-MLgD, but their impact on complexity will be minimal, as mentioned earlier as well.

Apart from decoding complexity, hardware resources required is also a critical consideration when it comes to MLgD. In Table 4.7, we present the amount of memory necessary for the operations of the three algorithms. We consider the size of a single message exchanged between a pair of neighboring variable and check nodes, and the size of the values that have to be stored in memory during the decoding process. This is given per variable node, since no values are being stored at check nodes. Similar to the complexity comparison, we consider messages/memory of $(2^r - 1)$ binary nodes together for CRB-MLgD. In the instances where values being exchanged/stored are integers, the number of bits utilized would depend on the system.

Algorithm	*Resource Requirements*	
	Message	Memory
ISRB-MLgD	r bits	$2^r + 2 \cdot \frac{m}{n}$ integers
IISRB-MLgD	r bits	$2^r + 2 \cdot \frac{m}{n}$ integers
CRB-MLgD	$(2^r - 1)$ bits $(2^r - 1)$ integers	$(2^r - 1)$ integers

Table 4.7: Resource Requirements of MLgD Algorithms

Table 4.7 shows that the messages transmitted in CRB-MLgD are significantly larger than in the other two algorithms, primarily because a reliability value is transmitted along with the bit estimate, whereas ISRB-MLgD and IISRB-MLgD transmit a symbol estimate only. But CRB-MLgD requires less memory, since there only the initial reliability values of the binary nodes need to be stored. In ISRB-MLgD and IISRB-MLgD, in addition to the symbol reliability vectors, individual edge reliability values also have to be stored. Although every edge in the graph is associated with such a value, all of them would not be unique. In fact, due to the nature of the computation [24], there are only two different edge reliability values possible per check node. This has been taken into account in Table 4.7.

CRB-MLgD offers a few advantages in hardware implementation that may not be imme-

diately apparent from the above comparisons. Since it requires only integer additions, integer comparisons, scaling operations and XOR operations, relatively simpler hardware could be designed for the algorithm. Also, due to the low hardware resources of the systems that these algorithms are usually implemented on, integer overflow is a very common issue [24]. This is guaranteed with algorithms that accumulate reliability values, such as ISRB-MLgD, but is also possible in other cases as well. With symbol reliability vectors, a normalization operation has to be carried out if one value exceeds the capacity of the system. In CRB-MLgD though, since bit reliability values are not considered relative to each other, a simple clipping operation is sufficient to deal with integer overflow.

From simulation results in the earlier section, and the analysis on complexity and resource requirements above, it can be seen that CRB-MLgD offers significant performance gains at a somewhat higher complexity than existing soft reliability based algorithms. Hardware requirement of the new approach is fairly similar to existing ones. Thus, it could be used to bridge the significant gap between QSPA and MLgD, and used in practical applications with high column weight NB-LDPC codes.

Chapter 5

Iterative Soft Decoding of Reed-Solomon Codes

Reed-Solomon (RS) codes are a very well-known class of algebraic codes, first introduced in 1960 [6] and still widely used in communication and storage systems. Their popularity is largely due to the excellent distance properties they posses as maximum distance separable (MDS) codes, which allow them to correct a large number of channel-induced errors. In an overwhelming majority of the applications where RS codes are used, they are decoded via the popular Berlekamp-Massey (BM) algorithm [66], [67], an algebraic HDD method.

The BM algorithm is very efficient, and therefore, fits the requirements of many practical applications that employ RS codes. The algorithm is classified as *syndrome based decoding*, since decoding procedure is based on the syndrome of the received vector. It is capable of correcting up to $\lfloor \frac{d_{min}-1}{2} \rfloor$ errors, where d_{min} denotes the minimum distance of the code. As RS codes are MDS, for an (n, k) RS code, $d_{min} = n - k + 1$, and the BM algorithm can correct at most $\frac{n-k}{2}$ errors [35].

In 1999, Guruswami and Sudan proposed a different HDD algorithm for RS codes [68], based on an alternative approach to syndrome based decoding. This algorithm perceives the received values as points on a two dimensional plane, and computes a *list* of possible transmitted codewords by curve-fitting and interpolation on that set of points. Although computationally more expensive than BM algorithm, the Guruswami-Sudan (GS) algorithm is capable of correcting more than $\frac{n-k}{2}$ errors. It is classified as *interpolation based decoding*.

Soft information could be made available in many applications that use RS codes, and it is generally accepted that SDD has the potential to outperform HDD by several dBs. In order to

realize these gains, many algorithms for SDD of RS codes have been proposed over the years, such as the well-known Koetter-Vardy (KV) algorithm, proposed in 2003 [69]. KV algorithm could be considered as the soft decoding version of the GS algorithm, and offers significant gains over both GS and BM algorithm, especially with lower rate codes. This interpolation based soft decoding approach has been further developed in [70] and [71]. Syndrome based soft decoding algorithms have also been proposed for RS codes, such as [72] that makes use of the generalized minimum distance (GMD) decoding strategy, originally proposed for linear codes in [73]. Chase decoding [35] has also been considered in the literature, using both syndrome based algorithms and interpolation based algorithms for the selection of the most likely codeword, for example refer [74]. Still, none of these SDD schemes are widely used in practice yet, particularly due to the high decoding complexity and the requirement for specialized hardware resources.

More recently, schemes based on iterative decoding, made popular by Turbo and LDPC codes, have been considered for SDD of RS codes, primarily due to the approach being relatively low in complexity when compared with others. Main obstacle here is the high density of the PCMs, which negatively impacts decoding through the creation of a large number of short cycles and other graph sub-structures, which were reviewed in Section 4.1. The adaptive belief propagation (ABP) [43], one of the earliest iterative decoding schemes proposed for RS codes, uses binary image representations of non-binary PCMs, and circumvents the issue of cycles by changing, or *adapting*, the matrix through Gaussian operations in each decoding iteration. A more detailed overview of ABP, and its improvements [55], [56], were given in Section 4.2.1. Although ABP offers impressive performance gains, often outperforming KV algorithm, it is even more complex, and is also not well suited for hardware implementations, as the PCM changes in each iteration. Thus, ABP or related schemes have not been considered for practical applications so far. Somewhat related, but a significantly simpler scheme is proposed in [54], where a fixed binary PCM with a large number of redundant parity-check equations is used. Still, sorting is a primary operation in the algorithm, which increases its complexity at hardware levels. Also, an enhanced hard decision decoder has to be used after each decoding iteration.

Based on the observations on non-binary absorbing sets in Chapter 4, in this Chapter we propose a novel approach for soft decoding of RS codes. In this strategy, we consider constructing non-binary PCMs better suited for iterative decoding of these codes. As discussed previously, effects of short cycles are more complex when it comes to non-binary codes. Although no short cycle is desirable, most harmful ones are non-FRC cycles, or those that induce rank deficient sub-matrices in the PCM (see Definition 4.5). Reducing these non-FRC cycles has led to improved performances

with NB-LDPC codes, as seen in [18], [15]. We take a similar approach here, and construct alternative non-binary PCMs for RS codes without any non-FRC cycle, or a few at most. The construction considers only the sparsest possible matrices, thus reducing the overall number of short cycles, and then attempts to reduce the number of non-FRC cycles. Results show that the new method is able to improve significantly over hard decision decoding (HDD), while its complexity, relative to different soft decision decoding (SDD) strategies proposed in the literature, is low. In the following, we first discuss the construction of suitable PCMs, with some examples, and then present simulation results and a complexity analysis.

5.1 Construction of PCMs Suitable for Iterative Decoding

Traditionally, RS codes are represented with PCMs that are of the so called *Vandermonde* form. These matrices do not contain a single zero, and are thus not suitable for iterative decoding at all. Cycles of many lengths exist in them, starting from the shortest possible four-cycles [43], [54]. For iterative SDD of RS codes, we propose constructing alternative PCMs where the number of short cycles has been reduced as much as possible. As explained in section 5.1, shorter cycles are much more harmful for iterative decoding, and therefore, in the following, our focus will be on reducing the number of shortest cycles, the cycles of length four and six. Initially, we do not differentiate between FRC and non-FRC cycles, but attempt to reduce the overall number of cycles, prior to shifting our focus to limiting the number of non-FRC cycles.

5.1.1 Reducing Short Cycles

The straight-forward approach to limit the number of short cycles in the PCM of an RS code is to build the said matrix using the minimum-weight codewords of the dual code. It is well known that the dual code of the (n, k) RS code is the $(n, n-k)$ RS code over the same field [30]. Following lemma on the dual codewords establishes the platform for constructing PCMs with the minimum possible short cycle count.

Lemma 5.1. *Consider the (n, k) RS code over $\mathbb{F}_q$. In the dual code of this code;*

1. *A codeword can contain at most $n-k-1$ zeros.*
2. *A codeword exists with zeros at any given $n-k-1$ positions.*

Proof. We adopt the original definition of RS codes, the one based on polynomial evaluation [6], to prove the two statements. As previously noted, the dual code of the (n, k) RS code is the $(n, n-k)$ RS code over $\mathbb{F}_q$. According to the original definition, any codeword $\underline{c}_d$ of the dual code may be expressed as;

$$\underline{c}_d = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \deg(f) < n-k\} \tag{5.1}$$

Here $F_q[x]$ denotes the set of functions with coefficients from $\mathbb{F}_q$.

It can be easily seen that the first statement is true, as a function of degree at most $(n-k-1)$ can have at most that many zeros. Further, since (5.1) does not put any constraint on the function f other than the one on degree, it should always be possible to find some codeword $\underline{c}_d$ with zeros at any given $(n-k-1)$ positions by using a suitable function f with roots at the corresponding values. This proves that the second statement is also true. □

From the above lemma, it is easy to see that the sparsest PCM possible for an (n, k) RS code will have only $(n-k) \times (n-k-1)$ zeros, i.e., $(n-k-1)$ per row. But the count of short cycles is not only impacted by the number of zeros in the PCM; their relative positions also matter. One possible approach to construct a PCM with the least count of short cycles is to first construct a binary, $(n-k) \times n$ matrix with constant row weight $(k+1)$, using a suitable LDPC construction method, and then create the required RS PCM by picking the dual codewords with zeros in the exact same positions as that first matrix. Second statement in Lemma 5.1 guarantees the existence of such codewords. As discussed ealier, LDPC construction methods are focused on limiting the number of short cycles as much as possible [48], [75], and therefore, this approach would produce a matrix with the least cycle count. Note that it is important to evaluate the rank of the produced matrix, as it is possible that some of the dual codewords used are linearly dependent. If it is found that the matrix is not of full rank, same procedure could be repeated with the zeros at different positions.

With high rate RS codes, where n is not significantly greater than k, the number of zeros in the sparsest possible PCM would be quite small. In such a case, it is possible to construct the matrix with the least cycle count by limiting the number of zeros per column. That is, if $n \geq (n-k) \times (n-k-1)$, the best possible matrix should have at most one zero per column. As an example, imagine that two zeros have to be placed in a $2 \times n$ matrix of all ones, one per each row. Prior to placing the zeros, the matrix would have $\binom{n}{2}$ four-cycles. Placing the first zero anywhere would eliminate $\binom{n-1}{1}$ of these. If the second one is placed in the same column, no additional four-cycles will be removed, but if it is placed in a separate column, $\binom{n-2}{1}$ more will get eliminated.

For a given (n, k) RS code, there will be a number of matrices with the least possible count of four-cycles and six-cycles. But the number of non-FRC cycles will not be the same in all these. This is due to the fact that satisfying the full rank condition is dependant on the edge labels in the matrix, as well as the topology of the underlying graph. But as discussed in Section 4.1, the number of non-FRC cycles can be expected to significantly impact iterative decoding performance with any chosen PCM. Therefore, in the next section, we attempt to reduce the count of non-FRC cycles in the sparsest possible PCMs, those that were discussed in this section, for a given RS code. Once more, we focus our study only on non-FRC cycles of lengths four and six, since they are the most detrimental for iterative decoding.

5.1.2 Reducing Non-FRC Short Cycles

While our goal is to reduce the count of both non-FRC four-cycles and six-cycles, it is not guaranteed that the lowest count of both will be with the same matrix. Four-cycles are considered the most harmful for decoding, but if the matrix with the lowest four-cycle count has a significant number of six-cycles, then that will not be quite suitable for iterative decoding. Therefore, some quantitative measure other than simply the count of non-FRC cycles is necessary for picking the best-suited matrix. For this purpose, we define a *cost* $\mathcal{C}_m$ for a matrix as follows.

$$\mathcal{C}_m = wn_4 + n_6 \tag{5.2}$$

n_4 and n_6 are the number of non-FRC four-cycles and the number of non-FRC six-cycles, respectively, and $w \geq 1$ is parameter that depends on the code. Primary purpose of w is to reflect that four-cycles are more damaging to iterative decoding than six-cycles.

The matrix with the least cost will be considered the one best-suited for decoding. Best value for w for any code is to be selected by first decoding that code with a few matrices for which the cycle counts are known. By comparing the BER performances of these, a suitable value for w may be selected. From our simulations, it was observed that w should increase both with the rate of the code and the size of the field over which the code is defined. w values used for some RS codes are presented in the next section.

The search for the matrix with the least cost should ideally consider all possible PCMs for the given RS code. But such an approach is computationally infeasible even for very short codes. Therefore, we limit our search to the matrices comprising of only the minimum-weight codewords of the dual, like those discussed in the previous section. Apart from the complexity issue, we take

such an approach since it is reasonable to expect that the number of non-FRC cycles will decrease with the number of short cycles, and also since any short cycle can affect iterative decoding, as discussed at the beginning of this chapter.

For an (n, k) RS code over $\mathbb{F}_q$, the search for the best PCM formed by minimum-weight dual codewords will have to consider a vector space of size $q\binom{n}{n-k-1}$. But one useful observation on non-FRC cycles allows us to reduce this size significantly. The count or even the positions of such cycles in a given matrix will not change if the rows were multiplied with constants. This helps reduce the size of the vector space for the search to $\binom{n}{n-k-1}$. Also, since RS codes are cyclic [30], and dual codes themselves are RS codes, the set of matrices produced by cyclically shifting a single matrix would have the same number of non-FRC cycles.

Even with making use of the aforementioned properties, it is impossible to reduce the complexity of the search to a computationally feasible level, especially for RS codes over higher order fields. In the following lemma, we present a condition on the existence of non-FRC four-cycles between a pair of dual codewords, which leads to a simpler method of constructing a PCM free of such cycles for any RS code.

Lemma 5.2. *Consider two minimum-weight dual codewords of an (n, k) RS code over $\mathbb{F}_q$, each with $(n-k-1)$ zeros. If the two share $(n-k-2)$ of the zeros, then between the codewords, there does not exist any cycle not satisfying the full rank condition.*

Proof. Let the two dual codewords be $\underline{c}_1$ and $\underline{c}_2$, which are from the $(n, n-k)$ RS code. Following the classical definition, $\underline{c}_1$ and $\underline{c}_2$ could be represented as polynomials $f_1, f_2 \in F_q[x]$ as follows. $z_i^{(j)}$'s denote the zeros of the two polynomials in $\mathbb{F}_q$.

$$f_1(x) = \prod_{i=0}^{n-k-2} (x - z_i^{(1)})$$
$$f_2(x) = \prod_{i=0}^{n-k-2} (x - z_i^{(2)}) \tag{5.3}$$

Now consider the $2 \times n$ matrix created by $\underline{c}_1$ and $\underline{c}_2$, and any 2×2 sub-matrix $\mathbb{S}$ of it. Let the columns corresponding to $\mathbb{S}$ be j_1 and j_2. The determinant of $\mathbb{S}$ may be expressed as in (5.4),where α denotes a primitive element of $\mathbb{F}_q$.

$$|\mathbb{S}| = f_1(\alpha^{j_1}) f_2(\alpha^{j_2}) - f_2(\alpha^{j_1}) f_1(\alpha^{j_2}) \tag{5.4}$$

For the existence of a non-FRC cycle, $\mathbb{S}$ should be singular. Then;

$$|\mathbb{S}| = f_1(\alpha^{j_1}) f_2(\alpha^{j_2}) - f_2(\alpha^{j_1}) f_1(\alpha^{j_2}) = 0 \tag{5.5}$$

Substituting to (5.5) $f_1(\alpha^{j_i})$ and $f_2(\alpha^{j_i})$, for $i = 1, 2$, evaluated with (5.3);

$$\prod_{i=0}^{n-k-2}(\alpha^{j_1} - z_i^{(1)}) \prod_{i=0}^{n-k-2}(\alpha^{j_2} - z_i^{(2)}) - \prod_{i=0}^{n-k-2}(\alpha^{j_1} - z_i^{(2)}) \prod_{i=0}^{n-k-2}(\alpha^{j_2} - z_i^{(1)}) = 0 \tag{5.6}$$

Since f_1 and f_2 share $(n-k-2)$ zeros, (5.6) can be simplified further as follows, where $z_u^{(1)}$ and $z_u^{(2)}$ denote the zeros that are not shared between the two polynomials.

$$(\alpha^{j_1} - z_u^{(1)})(\alpha^{j_2} - z_u^{(2)}) - (\alpha^{j_1} - z_u^{(2)})(\alpha^{j_2} - z_u^{(1)}) = 0 \tag{5.7}$$

Then (5.7) may be further simplified to

$$(\alpha^{j_1} - \alpha^{j_2})(z_u^{(1)} - z_u^{(2)}) = 0 \tag{5.8}$$

(5.8) will only be satisfied when $j_1 = j_2$, which contradicts the assumption of distinct columns. This shows that no 2×2 sub-matrix of the $2 \times n$ matrix created by $\underline{c}_1$ and $\underline{c}_2$ is singular. Thus, there cannot exist cycles not satisfying the full rank condition between the pair of codewords. □

If a PCM for an RS code consists only of minimum-weight dual codewords, then Lemma 5.2 guarantees that it would be free of non-FRC four-cycles if every pair of rows share all but one zero. This is a simpler method to construct a matrix suitable for iterative decoding, since such a matrix would also be free of the smallest possible non-binary absorbing sets [14]. A number of matrices, all free of non-FRC four-cycles, may be constructed from following this approach, but the number of non-FRC six-cycles would not be the same in every such matrix. If that number was also minimized, iterative decoding performance can be expected to improve further. We propose evaluating the cost of each matrix constructed following Lemma 5.2 using (5.2), and choosing the one with the lowest cost for decoding.

In the following section, we present some examples of alternative PCMs constructed for different RS codes, along with their non-FRC cycle counts, and associated costs.

5.1.3 Some Alternative Parity-check Matrices

In Section 5.1.1, constructing PCMs for RS codes with the lowest short cycle count was discussed, while in Section 5.1.2, focus was on reducing the non-FRC cycle count in the sparsest possible matrices. While the construction inspired by Lemma 5.2 produces a matrix with zero non-FRC four-cycles, it may not be the best matrix for iterative decoding, as many non-FRC six-cycles

could exist. Therefore, in our simulations, we consider the following two types of matrices for each RS code, results of which are provided in the next section. It is expected that a comparison of the BER performances with the two matrices would provide better insight to refine the cost function defined in (5.2).

- *Type-A*: The matrix with the lowest cost (according to (5.2)) among those constructed with minimum-weight dual codewords, found through a computer-based search
- *Type-B*: The matrix with the lowest non-FRC six-cycle count, among those with zero non-FRC four-cycles, found through a computer-based search

Note that when searching for the best matrices, care should be taken to eliminate any rank-deficient matrix. Such a matrix would not be a full representation of the code, and in fact, eliminating them can actually make the computer-based search faster. For example, for an (n, k) code where $n - k > 3$, all matrices containing three linearly dependant dual codewords could be eliminated during intermediate steps of the search, saving a lot on compelxity.

In the following, we present as examples the alternative type-A and type-B PCMs constructed for (7, 3) and (15, 11) RS codes.

Example 5.1. *Traditional PCM for the (7, 3) RS code, which is of the Vandermonde form, contains 4 non-FRC four-cycles, and 72 non-FRC six-cycles, apart from a large number of FRC short cycles. This matrix, $H^O_{7,3}$ is given in* (5.9)*, where ω denotes a primitive of $\mathbb{F}_{2^3}$.*

For this code, search for the best type-A and type-B matrices produced the same matrix, which does not contain any non-FRC cycle of length four or six. It does contain a few FRC short-cycles though. (5.10) *presents this matrix, which we denote $H^A_{7,3}$.*

$$H^O_{7,3} = \begin{bmatrix} 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 \\ 1 & \omega^3 & \omega^6 & \omega^2 & \omega^5 & \omega & \omega^4 \\ 1 & \omega^4 & \omega^8 & \omega^5 & \omega^2 & \omega^6 & \omega^3 \end{bmatrix} \tag{5.9}$$

$$H^A_{7,3} = \begin{bmatrix} 0 & \omega & \omega^6 & \omega^4 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ \omega^5 & \omega^4 & \omega^6 & 0 & 0 & 0 & \omega^2 \\ \omega^3 & 0 & \omega & 0 & \omega^5 & 0 & 0 \end{bmatrix} \tag{5.10}$$

Example 5.2. *When it comes to the (15, 11) RS code, the traditional PCM contains significantly more non-FRC short cycles, which is to be expected due to its larger dimensions. It contains 15 non-FRC four-cycles and 780 non-FRC six-cycles. The matrix is given in the following, where α denotes a primitive of $\mathbb{F}_{2^4}$.*

$$H^{O}_{15,11} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \alpha^{10} & \alpha^{12} & \alpha^{14} & \alpha & \alpha^3 & \alpha^5 & \alpha^7 & \alpha^9 & \alpha^{11} & \alpha^{13} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \\ 1 & \alpha^4 & \alpha^8 & \alpha^{12} & \alpha & \alpha^5 & \alpha^9 & \alpha^{13} & \alpha^2 & \alpha^6 & \alpha^{10} & \alpha^{14} & \alpha^3 & \alpha^7 & \alpha^{11} \end{bmatrix} \tag{5.11}$$

Our search for the best type-A PCM produced a matrix with 8 non-FRC four-cycles and 134 non-FRC six-cycles. While the count of non-FRC cycles is not low by any measure, it is still a significant reduction from the traditional PCM. Value of w, in (5.2)*, was set to 5 for the search. As discussed earlier, this value was based on a comparison of the BER performances of a few PCMs with known cycle counts. The matrix, $H^{A}_{15,11}$, is given in* (5.12)*.*

$$H^{A}_{15,11} = \begin{bmatrix} 0 & 1 & 1 & \alpha^{10} & 1 & 0 & \alpha^5 & \alpha^5 & 1 & \alpha^5 & 0 & \alpha^{10} & \alpha^{10} & \alpha^5 & \alpha^{10} \\ \alpha^5 & 0 & \alpha^6 & \alpha^5 & \alpha^5 & \alpha^6 & \alpha^9 & 0 & \alpha^9 & \alpha^5 & \alpha^9 & \alpha^9 & \alpha^6 & \alpha^6 & 0 \\ \alpha & \alpha^{14} & \alpha^5 & 0 & \alpha^2 & \alpha^{11} & \alpha^{11} & \alpha^3 & 0 & \alpha^{12} & \alpha^{10} & 0 & \alpha^4 & \alpha & \alpha^3 \\ \alpha^3 & \alpha & \alpha^{14} & \alpha^5 & 0 & \alpha^2 & \alpha^{11} & \alpha^{11} & \alpha^3 & 0 & \alpha^{12} & \alpha^{10} & 0 & \alpha^4 & \alpha \end{bmatrix} \tag{5.12}$$

While the best type-B matrix, $H^{B}_{15,11}$, does not contain any non-FRC four-cycles, as guaranteed by lemma 5.5, it has 189 non-FRC six-cycles. This results in $H^{B}_{15,11}$ having a marginally higher cost than $H^{A}_{15,11}$, and how big a difference in performance this cost difference would lead to can be observed from the simulation results provided in the next section. $H^{B}_{15,11}$ is presented in (5.13)*.*

$$H^{B}_{15,11} = \begin{bmatrix} 0 & 1 & 1 & \alpha^{10} & 1 & 0 & \alpha^5 & \alpha^5 & 1 & \alpha^5 & 0 & \alpha^{10} & \alpha^{10} & \alpha^5 & \alpha^{10} \\ \alpha^9 & \alpha^{14} & \alpha^2 & 1 & \alpha^{12} & \alpha^{10} & \alpha^9 & \alpha^{12} & 0 & 1 & \alpha^{12} & 0 & 0 & \alpha^5 & \alpha^4 \\ \alpha^7 & \alpha^2 & \alpha^7 & \alpha^6 & \alpha^6 & \alpha^2 & 0 & \alpha^{13} & \alpha^9 & \alpha^{10} & \alpha^5 & 0 & 0 & \alpha^4 & \alpha^8 \\ \alpha^6 & \alpha^{12} & \alpha^{13} & 0 & 0 & \alpha^{14} & \alpha^9 & \alpha^{13} & \alpha^7 & 0 & \alpha^6 & 1 & \alpha^5 & \alpha^{11} & \alpha^{11} \end{bmatrix} \tag{5.13}$$

5.2 Performance and Complexity

As discussed in the previous sections, best *type-A* and *type-B* PCMs found for a given RS code would have properties that make them more suited for iterative decoding. We propose

simply using QSPA on these matrices for SDD of RS codes. In comparison with existing soft decision decoders, such as [69], [43], [56], this approach seems significantly simpler and much more straightforward. However, we suggest two minor modifications to QSPA that would improve performance of the new decoding scheme significantly. The modifications are presented in some detail in the following. Note that similar approaches are commonly used for improving performance of iterative decoding algorithms, particularly when it comes to NB-LDPC codes.

1) *Scaling factors with check node estimates:*
Although our search procedure attempts to reduce the number of cycles as much as possible, due to the nature of RS codes, some cycles, of both FRC and non-FRC types, will still be present in the matrices selected. Also, as we put more focus on reducing non-FRC cycles, most of the remaining would be FRC cycles. These do impact iterative decoding, particularly by making the messages passed in the graph highly correlated. This correlation can make the estimates computed at nodes somewhat unreliable, and in particular, they could be *over-confident* of a variable node taking a particular symbol value. Thus, in such a situation, it is better to use them as mere metrics indicating the *possibility* of a variable node taking some value, rather than as close approximations of the MAP value[54]. Considering all these, we suggest using a scaling factor κ, $0 < \kappa \leq 1$, with the estimates received from check nodes, before combining them at variable nodes. Scaling factors have been used to improve performance of iterative decoders quite often in the literature, such as in [42], [33], and [76]. In fact, we also proposed the use of scaling factors with the novel decoding algorithms presented for NB-LDPC codes, in Chapter 4.

2) *Redundant rows in the PCM:*
In [77], it has been observed that for iterative decoding of an (n, k) code, using a PCM with more than $(n - k)$ rows can offer significant performance improvements, particularly when the code cannot be represented with a sparse PCM. These additional rows are usually referred to as redundant rows, since they do not affect the row space of the matrix, and adding them would make the PCM rank deficient. As reviewed in Chapters 2 and 4, such PCM representations have also been used for decoding NB-LDPC codes constructed with structured methods, which are mainly targeted for iterative, majority-logic based approaches. A larger number of rows would naturally increase the number of estimates received by variable nodes, which may lead to faster convergence. But at the same time, these new rows may create additional short cycles as well. We suggest adding a few redundant rows to the best PCMs produced by the search. These rows should also be minimum-

weight codewords of the dual, so as not to significantly add to the cycle counts.

Optimum damping coefficient κ, and the optimum number of redundant parity-check equations to add, would depend on the code and to a degree on the PCM selected for decoding. These parameters can be optimized through simulations.

5.2.1 Performance

In the following, we present simulation results of our scheme with some RS codes. For each code, FER performance with both *type-A* and *type-B* matrices are given. We also evaluate the performance gain by increasing the maximum number of decoding iterations allowed, and in the figures, plots are labelled with the matrix type and this number, for example *type-A-20* when using a *type-A* matrix with maximum number of iterations set to 20. Further, FER performance with a matrix with redundant rows is given for most of the codes, which is labelled *type-R*. As proposed earlier, the optimum number of such rows was found through simulations.

Figure 5.1 presents performances of our schemes, along with that of HDD with the BM algorithm, and ML decoding, for the (7, 3) RS code. As explained in the previous section, the best *type-A* and *type-B* matrices happen to be the same one for this code, which is $H^A_{7,3}$, given in (5.10). In the figure, we have labelled the corresponding plots as *type-A*. It can be seen that our scheme is able to achieve performance gains of more than 2dB over HDD, at a FER of 10^{-5}, even with just 20 decoding iterations. This increases by a further 0.25dB, at 10^{-6} FER, when maximum decoding iterations is increased to 100. When compared with performance of ML decoding, our scheme is only about 0.3dB away, at a FER of 10^{-4}. Exceptional performance in this case is primarily due to the PCM used being free from all length four and six non-FRC cycles. The presence of a few FRC cycles does not seem to affect decoding performance much, as the FER curves do not exhibit an error floor, running almost parallel to the curve of HDD. In this case we did not apply the modifications to QSPA discussed previously, due to the exceptional performance and the PCM being quite sparse.

Figure 5.2 shows FER performance in decoding the (15, 11) RS code with traditional HDD, some existing SDD schemes, and different variations of the proposed scheme. The PCMs used for the novel schemes are $H^A_{15,11}$ and $H^B_{15,11}$, given in, respectively, (5.12) and (5.13). In this case we applied the modifications proposed earlier, and the optimum scaling factor κ was found to be 0.65 for both matrices. Decoding was attempted with redundant rows added to $H^A_{15,11}$, and it was

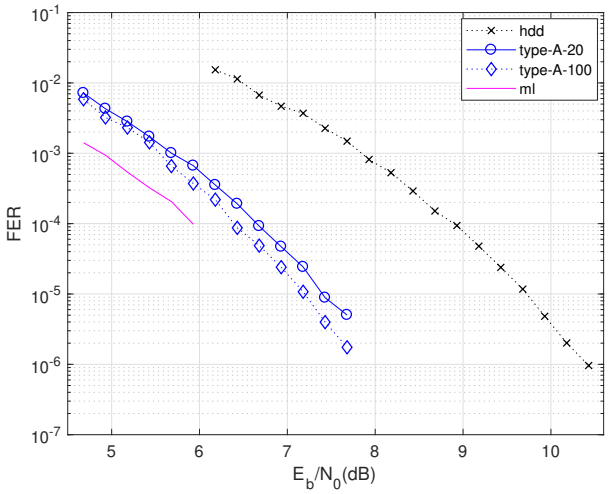

Figure 5.1: FER Performance with (7, 3) RS Code

observed that adding more than two such rows does not offer significant performance gains.

Out of the SDD schemes, ABP performs the best, offering gains in excess of 2dB over HDD with 20 decoding iterations and one decoding round [43]. KV algorithm [69], when used with no complexity limitations, is able to offer around 1dB gain over HDD, while the proposed scheme, with both *type-A* and *type-B* matrices, offer gains in excess of 1.3dB, at a FER of 10^{-5}. As discussed in example 3.2, the value of w used in the search for the best *type-A* matrix was 5, which, according to (5.2), results in the cost of $H^A_{15,11}$ being 174, and that of $H^B_{15,11}$ being 189. It is observable from the figure that the slightly low cost of $H^A_{15,11}$ has resulted in the corresponding decoding scheme performing marginally better than the one using $H^B_{15,11}$, thus supporting our choice for w. Interestingly, increasing the maximum number of decoding iterations to 40 from 20 does not seem to have any impact on iterative decoding performance with the *type-A* matrix, which suggests that QSPA is converging in less than 20 iterations in most cases. Further, there is no discernible error floor here as well, with sharper slopes than HDD at lower error rates, despite the two matrices used having quite a few short cycles of both types. This could be due to the use of the optimized scaling factor κ, which reduces the correlation among messages that is created by the cycles.

Performances with both matrices has gaps around 1dB with ABP, which decreases to about 0.7dB when using the *type-A* matrix with redundant rows. Novel schemes manage to outperform KV algorithm by more than 0.3dB in all cases. Complexity-wise, it has significant advantages over both ABP and KV algorithm, which will be briefly discussed in the next section.

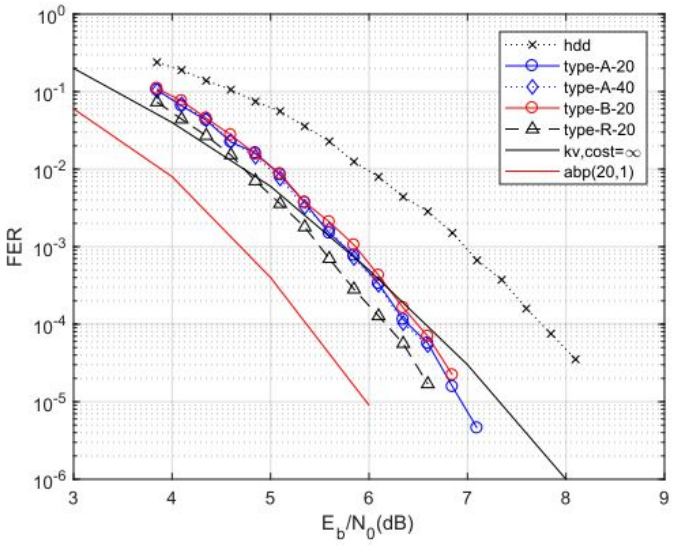

Figure 5.2: FER Performance with (15, 11) RS Code

Figure 5.3, Figure 5.4, and Figure 5.5 present decoding performances of our schemes and HDD with, respectively, (15, 7), (31, 27), and (63, 59) RS codes. In interest of brevity, we present the various parameters associated with iterative SDD of these codes in Table 5.1. There, the optimum value of w in (5.2) is listed for each code, along with the non-FRC four-cycle and six-cycle counts (n_4 and n_6) in the traditional Vandermonde PCM, the best *type-A* PCM, and the best *type-B* PCM. The costs $\mathcal{C}_m$ of the two alternative PCMs are also given, computed as in (5.2), using the listed value of w. Since we apply both modifications proposed when decoding, the optimum values of scaling factor κ, and the optimum number of redundant rows in the PCMs (r_N) are listed as well. Note that as adding redundant rows is considered only for the best *type-A* matrix, r_N is given only for that.

Code	w	*type-A*					*type-B*				*Traditional*	
		n_4	n_6	$\mathcal{C}_m$	κ	r_N	n_4	n_6	$\mathcal{C}_m$	κ	n_4	n_6
RS (15, 7)	4	6	134	158	0.65	4	0	900	900	0.5	195	10320
RS (31, 27)	5	32	1810	1970	0.6	3	0	2412	2412	0.6	0	3720
RS (63, 59)	7	131	10964	11881	0.6	2	0	12790	12790	0.6	63	15372

Table 5.1: Decoding Parameters for Some RS Codes

Figures show that proposed decoding scheme is capable of offering gains in excess of 1dB over HDD with all three codes. In fact, for (15, 7) and (31, 27) codes, gains of the best performing

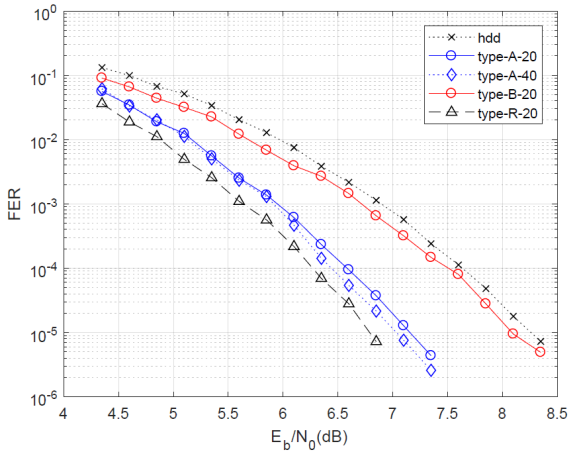

Figure 5.3: FER Performance with (15, 7) RS Code

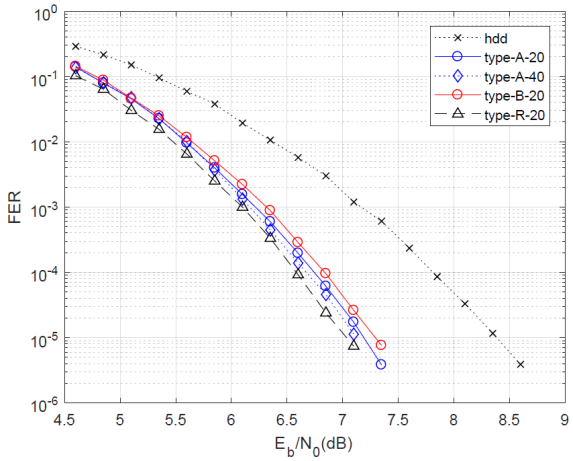

Figure 5.4: FER Performance with (31, 27) RS Code

variant of the novel strategy is very close to 1.5dB, while for (63, 59) code, it is closer 1.25dB. Interestingly, there is a big difference between performance of *type-A* and *type-B* matrices with the (15, 7) code. There, *type-B* matrix is only able to offer marginal gains over HDD. This is due to the significantly large cost of that matrix, caused by the large number of non-FRC six-cycles present. As given in Table 5.1, the cost of the best *type-A* matrix is 134, while that of the *type-B* matrix is 900, a percentage increase of almost 700%. For the other two codes, cost difference between the two types of matrices is not that significant, and this is further validated by the almost similar performances of associated decoding schemes. But there also, *type-A* matrices have lower costs, and decoding performance with them is marginally better.

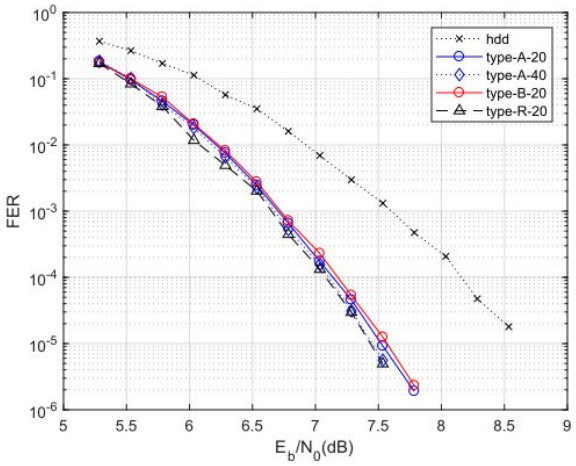

Figure 5.5: FER Performance with (63, 59) RS Code

In all three cases, performance can be improved through adding some redundant rows to the PCM. Optimum number of rows to add depends on the code itself, as values presented in Table 5.1 show. Significance of this gain also depends on the code, which is highest for the (15, 7) code, where it is close to 0.5dB. For the (31, 27) code, this gain is about 0.15dB, and in the case of the (63, 59) code, it is marginal, less than 0.1dB. Performance gain by increasing the maximum number of decoding iterations follows a similar trend. Increasing this number from 20 to 40 results in a gain of around 0.15dB for the (15, 7) code, while the gains for the other two codes are negligible. Once more, we observe no error-floor, which we attribute to the first proposed modification to QSPA, use of optimized scaling factors with check node estimates.

During our simulations, it was observed that the gains offered by the new scheme for RS codes over fields larger than $\mathbb{F}_{2^5}$, decreases when the dimension, or rate, of the dual code increase. This behaviour is to be expected, as the number of rows in the PCM would also increase with the dual code rate, which results in more short cycles, including non-FRC ones. How to improve performance of the scheme in such cases is still an open problem.

5.2.2 Complexity

As seen from the simulation results presented in the previous section, the novel iterative SDD strategy is able to offer impressive performance gains over HDD in most instances. It was also compared with ABP and KV algorithm, two of the most well-known SDD schemes for RS codes, in Figure 5.2. There it was seen that the proposed scheme has a significant gap with ABP, while it

outperforms KV algorithm comfortably. But when it comes to decoding complexity and hardware implementations, the new scheme has a number of advantages over any SDD scheme proposed for RS codes in the literature. In this section, we present a brief overview of those advantages, starting with a rough complexity comparison with ABP, which is the best performing SDD scheme for RS codes in the literature.

Since ABP uses the binary image representation of the PCM [43], for an RS code over $\mathbb{F}_q$, it is actually using $q \log q$ variable nodes in decoding (as in most cases code length $\approx q$). This makes its decoding complexity in one iteration to be of order $O(q^3 \log^3 q)$, due to the matrix adaption operation carried out through Gaussian elimination. In contrast, dominant complexity term of the proposed algorithm is of order $O(q^2 \log q)$, which corresponds to the complexity of all the check node operations in one iteration. Additionally, ABP also has complexity terms of orders $O(q \log q \log(q \log q))$, for sorting bits based on LLR values, and $O(q \log q)$, for operations of SPA. Proposed method has only one additional complexity term, of order $O(q^2)$, for variable node operations. Thus, when used with the same number of iterations, the proposed method has significantly lower complexity when compared to ABP.

Use of QSPA, which is generally considered to be of a high complexity level, might seem a disadvantage of the proposed scheme, when it comes to decoding complexity. But the fast convergence of QSPA, when used with the alternative PCMs we propose, offsets any complexity increase due to the algorithm. As an example, in Figure 5.6, we present the mean and standard deviation of the number of iterations taken for QSPA to converge when decoding the (15, 11) RS code, using the best *type-A* matrix found. Maximum number of iterations was set to 20 here, and the figure shows that in most cases, QSPA converges rapidly, in much less than 20 iterations.

The primary reason for SDD not been used with RS codes in practical applications is the lack of an algorithm that can be efficiently implemented in hardware. KV algorithm contains many operations that are significantly complex at hardware level, which makes efficient implementations of the algorithm impossible. Most iterative SDD schemes for RS codes employ equivalent binary PCMs for decoding [43], [55], [56], obtained through the binary image representation of the traditional PCM. But as discussed earlier in the chapter, these contain many short cycles, and to make them better suited for iterative decoding, these strategies use various techniques, such as changing the matrix in each iteration [43], or based on the properties of the noisy vector received from the channel [54]. But in the proposed scheme, the PCM is fixed at all times, and it does not require expensive operations, such as Gaussian elimination, necessary in ABP and its many variants, or sorting, used in almost every iterative soft decision decoder proposed for the codes. Thus,

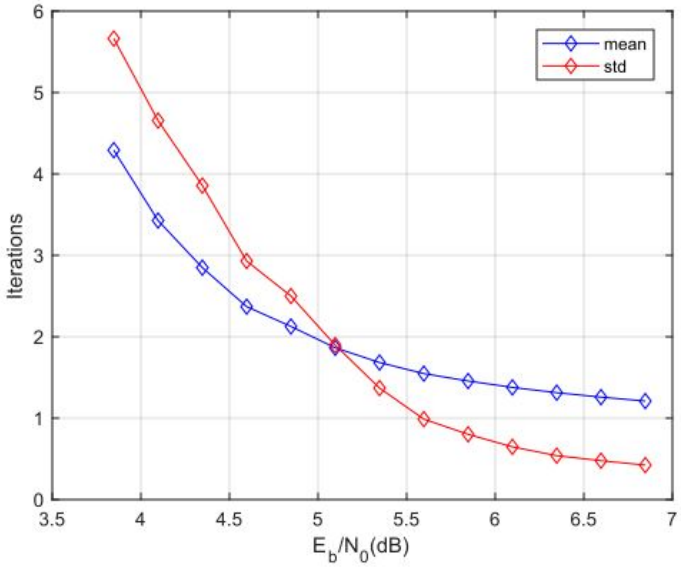

Figure 5.6: Convergence of QSPA with (15, 11) RS Code

with the new scheme, efficient hardware implementations, similar to those suggested in [29] for short-to-medium length codes over fields of characteristic 2, become possible for SDD of RS codes.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, p. 379–423, July 1948.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error correcting coding and decoding," in *Proceedings of IEEE International Conference on Communications*, (Geneva, Switzerland), May 1993.

[3] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, Jan 1962.

[4] D. J. C. Mackay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 46, pp. 399–431, Mar 1999.

[5] M. C. Davey and D. J. C. Mackay, "Low-density parity check codes over $gf(q)$," *IEEE Communication Letters*, vol. 2, pp. 165–167, Jun 1998.

[6] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300–304, June 1960.

[7] J. J. Rothman, *Advanced modern algebra*. Prentice Hall, 2003.

[8] W. P. Wardlaw, "Matrix representation of finite fields," *Mathematics Magazine*, vol. 67, pp. 289–293, Oct 1994.

[9] S. Huczynska and M. Neunhoffer, "Finite fields." `http://www.math.rwth-aachen.de/~Max.Neunhoeffer/Teaching/ff2013/ff2013.pdf`, accessed 2020-08-18.

[10] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533–547, Sep 1981.

[11] M. Korb and A. Blanksby, "Non-binary ldpc codes over finite division near rings," in *Proceedings of IEEE International Conference on Communications*, (Thessaloniki, Greece), May 2016.

[12] D. Sridhara and T. E. Fuja, "Ldpc codes over rings for psk modulation," *IEEE Transactions on Information Theory*, vol. 51, pp. 3209–3220, Aug 2005.

[13] T. J. Richardson, "Error floors of ldpc codes," in *Proceedings of Annual Allerton Conference on Communication, Control and Computing*, (Monticello, Illinois), Oct 2003.

[14] B. Amiri, J. Kliewer, and L. Dolecek, "Analysis and enumeration of absorbing sets for non-binary graph-based codes," *IEEE Transactions on Communications*, vol. 62, pp. 398–409, Feb 2014.

[15] C. Poulliat, M. Fossorier, and D. Declercq, "Design of regular (2,d_c)-ldpc codes over $gf(q)$ using their binary images," *IEEE Transactions on Communications*, vol. 56, pp. 1626–1635, Oct 2008.

[16] L. Dolecek, D. Divsalar, Y. Sun, and B. Amiri, "Non-binary protograph-based ldpc codes: Enumerators, analysis, and designs," *IEEE Transactions on Information Theory*, vol. 60, pp. 3913–3941, Jul 2014.

[17] E. Boutillon, "Optimization of non binary parity check coefficients," *IEEE Transactions on Information Theory*, vol. 65, pp. 2092–2100, Apr 2019.

[18] S. Cho, K. Cheun, and K. Yang, "A message-passing algorithm for counting short cycles in nonbinary ldpc codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Vail, Colorado), Jun 2018.

[19] 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*, vol. 47, pp. 619–637, Feb 2001.

[20] L. Zeng, L. Lan, Y. Y. Tai, S. Song, S. Lin, and K. Abdel-Ghaffar, "Constructions of nonbinary quasi-cyclic ldpc codes: A finite field approach," *IEEE Transactions on Communications*, vol. 56, pp. 545–554, Apr 2008.

[21] L. Zeng, L. Lan, Y. Y. Tai, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "Construction of nonbinary cyclic, quasi-cyclic and regular ldpc codes: A finite geometry approach," *IEEE Transactions on Communications*, vol. 56, pp. 378–387, Mar 2008.

[22] J. Li, K. Liu, S. Lin, and K. Abdel-Ghaffar, "A matrix-theoretic approach to the construction of non-binary quasi-cyclic ldpc codes," *IEEE Transactions on Communications*, vol. 63, pp. 1057–1068, Apr 2015.

[23] C. Chao, B. Bai, X. Wang, and M. Xu, "Nonbinary ldpc codes constructed based on a cyclic mds code and a low-complexity nonbinary message-passing decoding algorithm," *IEEE Communication Letters*, vol. 14, pp. 239–241, Mar 2010.

[24] C.-Y. Chen, Q. Huang, C.-C. Chao, and S. Lin, "Two low-complexity reliability-based message-passing algorithms for decoding non-binary ldpc codes," *IEEE Transactions on Communications*, vol. 58, pp. 3140–3147, Nov 2010.

[25] L. Barnault and D. Declerq, "Fast decoding algorithm for ldpc over *GF(*2^q*)*," in *Proceedings of IEEE Information Theory Workshop*, (Paris, France), Apr 2003.

[26] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of ldpc codes over *GF(q)*," in *Proceedings of IEEE International Conference on Communications*, (Paris, France), Jun 2004.

[27] H. Song and J. R. Cruz, "Reduced-complexity decoding of *Q*-ary ldpc codes for magnetic recording," *IEEE Transactions on Magnetics*, vol. 39, pp. 1081–1087, Apr 2003.

[28] V. Savin, "Min-max decoding for non binary ldpc codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Toronto, Canada), Jul 2008.

[29] C. Spagnol, E. Popovici, and W. Marnane, "Hardware implementation of $gf(2^m)$ ldpc decoders," *IEEE Transactions on Circuits and Systems I*, vol. 56, pp. 2609–2620, Mar 2009.

[30] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Amsterdam: North-Holland, 1977.

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

[32] J. Sayir, "Non-binary ldpc decoding using truncated messages in the walsh-hadamard domain," in *Proceedings of International Symposium on Information Theory and its Applications*, (Melbourne, Australia), Oct 2014.

[33] D. Declerq and M. Fossorier, "Extended minsum algorithm for decoding ldpc codes over *GF(q)*," in *Proceedings of IEEE International Symposium on Information Theory*, (Adelaide, Australia), Sep 2005.

[34] E. Li, D. Declercq, and K. Gunnam, "Trellis-based extended min-sum algorithm for non-binary ldpc codes and its hardware structure," *IEEE Transactions on Communications*, vol. 61, pp. 2600–2611, Jul 2013.

[35] S. Lin and D. J. Costello, *Error control coding*. Pearson Education, 2004.

[36] S. Haykin, *Communication system*. John Wiley and Sons, Inc, 2001.

[37] X. Zhang, F. Cai, and S. Lin, "Low-complexity reliability-based message-passing decoder architectures for non-binary ldpc codes," *IEEE Transactions on VLSI Systems*, vol. 20, pp. 1938–1950, Nov 2012.

[38] C. Xiong and Z. Yan, "Improved iterative hard- and soft-reliability based majority-logic decoding algorithms for non-binary low-density parity-check codes," *IEEE Transactions on Signal Processing*, vol. 62, pp. 5449–5457, Aug 2014.

[39] L. Song, M. Zhang, Q. Huang, and Z. Wang, "Low error-floor majority-logic decoding based algorithm for non-binary ldpc codes," in *Proceedings of IEEE International Conference on Communications*, (London, UK), Jun 2015.

[40] V. Savin, "Binary linear-time erasure decoding for non-binary ldpc codes," in *Proceedings of IEEE Information Theory Workshop*, (Taormina, Italy), Oct 2009.

[41] A. Bhatia, A. R. Iyengar, and P. H. Siegel, "Enhancing binary images of non-binary ldpc codes," in *Proceedings of IEEE Global Communications Conference*, (Houston, Texas), Dec 2011.

[42] M. Zhang, K. Cai, Q. Huang, and S. Yuan, "On bit-level decoding of nonbinary ldpc codes," *IEEE Transactions on Communications*, vol. 66, pp. 3736–3748, Sep 2018.

[43] J. Jiang and K. R. Narayanan, "Iterative soft-input soft-output decoding of reed-solomon codes by adapting the parity-check matrix," *IEEE Transactions on Information Theory*, vol. 52, pp. 3746–3756, Aug 2006.

[44] V. Savin, "Fourier domain representation of non-binary ldpc codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Cambridge, Massachusetts), Jul 2012.

[45] Y. Yu, W. Chen, J. Li, X. Ma, and B. Bai, "Generalized binary representation for the nonbinary ldpc code with decoder design," *IEEE Transactions on Communications*, vol. 62, pp. 3070–3083, Sep 2014.

[46] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, Feb 2001.

[47] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, pp. 1570–1579, June 2002.

[48] X.-Y. Hu, E. Eleftheriou, and D. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, pp. 386–398, Jan 2005.

[49] H. Chen and Z. Cao, "A modified peg algorithm for construction of ldpc codes with strictly concentrated check-node degree distributions," in *Proceedings of IEEE Wireless Communication and Networking Conference*, (Kowloon, China), Mar 2007.

[50] Y. Wang, J. S. Yedidia, and S. C. Draper, "Construction of high-girth qc-ldpc codes," in *Proceedings of International Symposium on Turbo Codes and Related Topics*, (Lausanne, Switzerland), Sep 2008.

[51] A. Hareedy, C. Lanka, N. Guo, and L. Dolecek, "A combinatorial methodology for optimizing non-binary graph-based codes: Theoretical analysis and applications in data storage," *IEEE Transactions on Information Theory*, vol. 65, pp. 2128–2154, Apr 2019.

[52] A. Price and J. Hall, "A survey on trapping sets and stopping sets," *arXiv e-prints*, p. arXiv:1705.05996, May 2017.

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

[54] J. Bellorado, A. Kavcic, and L. Ping, "Soft-input, iterative, reed-solomon decoding using redundant parity-check equations," in *Proceedings of IEEE Information Theory Workshop*, (Tahoe City, CA, USA), Sep 2007.

[55] F. Shayegh and M. R. Soleymani, "Iterative soft decoding of reed-solomon codes using information correction," in *Proceedings of International Symposium on Telecommunications*, (Tehran, Iran), Dec 2010.

[56] H.-C. Lee, G.-X. Huang, C.-H. Wang, and Y.-L. Ueng, "Iterative soft-decision decoding of reed-solomon codes using informed dynamic scheduling," in *Proceedings of IEEE International Symposium on Information Theory*, (Hong Kong, China), Jun 2015.

[57] M. El-Khamy and R. J. McEliece, "Iterative algebraic soft-decision list decoding of reed-solomon codes," *IEEE Journal on Selected Areas in Communications*, vol. 24, pp. 481–490, Mar 2006.

[58] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, pp. 406–414, Aug 2002.

[59] V. Savin, "Self-corrected min-sum decoding of ldpc codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Toronto, Canada), Jul 2008.

[60] X. Wu, Y. Song, M. Jiang, and C. Zhao, "Adaptive-normalized/offset min-sum algorithm," *IEEE Communication Letters*, vol. 14, pp. 667–669, Jul 2010.

[61] D. J. C. Mackay, "Encyclopedia of sparse graph codes." `http://www.inference.org.uk/mackay/codes/data.html`. (2020, Sep 16).

[62] Consultative Committee for Space Data Systems, *Short block length LDPC codes for TC synchronization and channel coding*, Apr 2015.

[63] C.-L. Wang, X. Chen, Z. Li, and S. Yang, "A simplified min-sum decoding algorithm for non-binary ldpc codes," *IEEE Transactions on Communications*, vol. 61, pp. 24–32, Oct 2012.

[64] L. Bhargava, R. Bose, and M. Balakrishnan, "Novel hardware implementation of llr-based non-binary ldpc decoders," in *Proceedings of National Conference on Communications*, (New Delhi, India), Feb 2013.

[65] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of channel codes and ml simulation results." `www.uni-kl.de/channel-codes`. (2020, Oct 12).

[66] E. R. Berlekamp, "Nonbinary bch decoding," in *Proceedings of IEEE International Symposium on Information Theory*, (San Remo, Italy), Sep 1967.

[67] J. L. Massey, "Shift-register synthesis and bch decoding," *IEEE Transactions on Information Theory*, vol. 15, pp. 122–127, Jan 1969.

[68] V. Guruswami and M. Sudan, "Improved decoding of reed-solomon and algebraic-geometry codes," *IEEE Transactions on Information Theory*, vol. 45, pp. 1757–1767, Sep 1999.

[69] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of reed-solomon codes," *IEEE Transactions on Information Theory*, vol. 49, pp. 2809–2825, Nov 2003.

[70] Y. Liu and L. Chen, "A new progressive algebraic soft decoding algorithm for reed-solomon codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Honolulu, HI, USA), Aug 2014.

[71] J. Xing, L. Chen, and M. Bossert, "Progressive module minimization for re-encoding transformed soft decoding of rs codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Paris, France), July 2019.

[72] G. Forney, "Fast generalized minimum-distance decoding of algebraic-geometry and reed-solomon codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 721–737, May 1996.

[73] G. Forney, "Generalized minimum distance decoding," *IEEE Transactions on Information Theory*, vol. 12, pp. 125–131, Apr 1966.

[74] J. Xing, L. Chen, and M. Bossert, "Low-complexity chase decoding of reed-solomon codes through basis reduction," in *Proceedings of IEEE International Symposium on Information Theory*, (Los Angeles, CA, USA), June 2020.

[75] G. Liva, S. Song, L. Lan, Y. Zhang, S. Lin, and W. E. Ryan, "Design of ldpc codes: A survey and new results," *Journal of Communications Software and Systems*, vol. 2, pp. 191–211, Sep 2006.

[76] J. O. Lacruz, F. Garcia-Herrero, D. Declercq, and J. Valls, "Simplified trellis min–max decoder architecture for nonbinary low-density parity-check codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, pp. 1783–1792, Sep 2015.

[77] T. Hehn, J. B. Huber, S. Laendner, and O. Milenkovic, "Mulitiple-bases belief-propagation for decoding of short block codes," in *Proceedings of IEEE International Symposium on Information Theory*, (Nice, France), June 2007.