CONTRIBUTIONS TO THE CONSTRUCTION AND DECODING OF NON-BINARY LOW-DENSITY PARITY-CHECK CODES

NG KHAI SHENG

NATIONAL UNIVERSITY OF SINGAPORE

2005

CONTRIBUTIONS TO THE CONSTRUCTION AND DECODING OF NON-BINARY LOW-DENSITY PARITY-CHECK CODES

NG KHAI SHENG

(B.Eng.(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2005

Acknowledgements

First of all, I would like to express my sincere thanks and gratitude to my supervisor, Dr. Marc Andre Armand, for his invaluable insights, patience, guidance and generosity throughout the course of my candidature. This thesis would not have been possible without his support.

My thanks also go out to my friends and lab mates in the Communications Laboratory, for the many enjoyable light-hearted moments and the occasional get-togethers. In particular, I would like to extend my thanks to Tay Han Siong and Thomas Sushil John, whose friendship, great company and encouragement have helped me through some rough times; and to Zhang Jianwen, for the many thought provoking and fruitful discussions. I would also like to thank my pals from my undergraduate days, Koh Bih Hian and Ng Kim Piau for their friendship.

My gratitude goes to the Department of Electrical and Computer Engineering, National University of Singapore, for providing all the necessary resources and giving me the opportunity to conduct such exciting and cutting edge research.

Last, but not least, I would like to thank my parents for their unwavering support and love.

Contents

Summary

Low-density parity-check (LDPC) codes are well known for their near Shannon limit performance and are at the forefront of research. Much of the earlier existing work done on LDPC codes in the literature involved large block lengths over binary alphabets. Richardson and Urbanke showed that increasing the size of the alphabet of the LDPC code leads to a corresponding improvement in bit error rate (BER) performance. Indeed, the computer simulation results of Davey and MacKay have shown that LDPC codes over GF(4) and GF(8) outperformed their binary counterparts over an additive-white-Gaussian-noise (AWGN) channel.

In the first part of this thesis, we present a novel method of constructing LDPC codes over mixed alphabets. In this method, we take a sparse matrix consisting of disjoint submatrices defined over the distinct subfields of a given field and link their associated subgraphs together. This is done by adding non-zero entries to the matrix. We also present a modified message passing algorithm (MPA), which takes into account the different row and column subgraph alphabets. This will reduce the number of redundant computations during decoding. Simulation results show that the codes constructed using the proposed method yields slight improvement in BER performance over their single alphabet counterparts with slight increase in decoding complexity.

In the second part, we present a multistage decoding approach for decoding of LDPC codes defined over the integer ring $\mathbb{Z}_q$, where $q = p^m$, p is a prime and

$m > 1$. We make use of the property that for an integer ring $\mathbb{Z}_q$, the natural ring epimorphism can be applied $\mathbb{Z}_q \mapsto \mathbb{Z}_{p^l} : r \mapsto \sum_{i=0}^{l-1} r^{(i)} p^i$ with kernel $p^l \mathbb{Z}_q$ for each l, $1 \leq l \leq m$, where $\sum_{i=0}^{m-1} r^{(i)} p^i$ is the p-adic expansion of r. Then we perform decoding using a modified MPA on each homomorphic image of the code. Computer simulations on codes over $\mathbb{Z}_4$ and $\mathbb{Z}_8$ of moderate length and rate half over the AWGN channel with binary-phase shift-keying (BPSK) modulation show that this multi-stage approach offers a coding gain of about 0.1 dB over a single stage decoding approach. For the case of a m-ary PSK modulation, we observe a slightly smaller coding gain (compared to BPSK modulation) over the single stage approach.

List of Figures

List of Tables

Chapter 1

Introduction

In 1948, Shannon published his seminal work on the Noisy Channel Coding Theorem [40]. In it, he proved that if information is properly coded and transmitted below the channel capacity, the probability of decoding error can be made to be arbitrarily small. Since then, much research has been devoted to finding codes which can be transmitted at as close to the the channel capacity as possible. In the remaining of this chapter, we briefly review several known constructions of earlier error-correcting codes as well as the current state-of-the-art codes, putting Low-Density Parity-Check (LDPC) codes in perspective. We then follow with the scope of work, the contribution of this thesis as well as the thesis outline.

1.1 Early Codes

One of the earliest papers on the construction of codes was presented by R. W. Hamming in [20], 2 years after Shannon's paper. In it, Hamming demonstrated a method for the construction of single error detecting and single error correcting systematic linear block codes. He defined systematic block codes as codes in which an input block of K (information) symbols is mapped to an output block of N (code) symbols.

The first K symbols of the output block is associated with the input block, while the remaining $N - K$ output symbols are used for error detection and correction. This class of codes are known today as Hamming codes.

Since then, some of the other codes discovered include the Bose-Chaudhuri-Hocquenghem (BCH) codes [6] [5] as well as the ubiquitous Reed-Solomon (RS) codes [36], which is a special case of BCH codes. Unlike Hamming codes, both BCH and RS codes are multiple-error-correcting codes. Both codes are popular due to their ease of implementation and good performance.

Convolutional codes were first introduced by Elias in 1955 [13]. The convolution code is similar to linear block codes in that they map an input block of K symbols to an output block of N symbols. However, the output block depends not only on just the inputs, but also on previous input blocks. This means that the encoder has memory. The maximum number of previous input blocks which an output symbol is dependent upon is known as the constraint length. Constraint length 7 convolutional codes have been used for satellite communications [28].

Convolutional codes can approach the Shannon limit as the constraint length increases, but the computational complexity of the (Viterbi) decoding algorithm is exponential in the constraint length.

Later, information was first encoded using a RS code, with the resulting codeword encoded via the convolutional encoder. Constructions such as the above where the output of one encoder is encoded again by another are known as concatenated codes [15]. For several years, these RS outer codes concatenated with convolutional codes gave the best practical performance for the Gaussian channel.

1.2 State-of-the-Art Error Correction

Turbo codes were discovered by Berrou et al. [3] in 1993. Their near Shannon limit performance over the additive white Gaussian noise (AWGN) channel brought about a renewed vigour in the search for other such high-performance codes. In [3], the turbo encoder consists of two binary rate $\frac{1}{2}$ convolutional encoders in parallel. The input to one of the encoders is a pseudo-random permutation of the input to the other. The constituent convolutional codes are systematic. During turbo-encoding, the systematic bits produced by one of the convolutional codes are discarded.

The decoding algorithm consists of the modified Viterbi decoding algorithm applied to each constituent code, with the output *a posteriori* estimates from one decoder being used as input to the other. Decoding consists of several iterations of this message passing algorithm.

Low-Density Parity-Check (LDPC) codes were first discovered by Gallager more than four decades ago in 1962 [16]. He also gave a description of an iterative decoding algorithm for such codes. However, due to its high decoding complexity (relative to the technology then), it remained largely forgotten until its recent rediscovery by MacKay [31]. LDPC codes have a simple description and a largely random structure. Its impressive performance, coupled with a relatively low decoding complexity (compared to Turbo codes) has attracted much attention from the research community. In fact, the world's best code is an irregular LDPC code (with block length $N = 10^7$) of rate $\frac{1}{2}$, falling short of the Shannon limit by just 0.04dB [9].

Another class of high performance codes are the repeat and accumulate (RA) codes that were studied by Divsalar et al. [12]. The encoding of RA codes comprises of two parts. The first part repeats a length K information sequence w times and performing pseudo-random permutation of the length wK sequence. The resultant block is then encoded by a rate 1 accumulator. The code can then be decoded using

a belief propagation decoder.

Such codes provide surprisingly good performance, although the repetition code is useless on its own. The RA code can perform to within 1 dB of capacity of an AWGN channel when the rate approaches zero and the block length is increased [12].

These state-of-the-art codes have several characteristics in common. They have a strong pseudo-random element in their construction and can be decoded via an iterative belief propagation decoding algorithm. Also, they have shown near Shannon limit error-correction capabilities.

1.3 Scope of Work

In the first part of this thesis, a method of constructing LDPC codes over mixed alphabets is proposed. This is done using a sparse matrix containing disjoint submatrices over distinct subfields of a given field and linking the associated subgraphs together by adding non-zero entries to this matrix.

We also present a modified decoding algorithm which takes into account the different alphabet of distinct code word coordinates.

The codes constructed here are of rate $R = 0.5$ and of short block length where $N = 1000$ and 2000 bits. We investigate their bit error rate (BER) performance over the AWGN channel with binary-phase shift-keying (BPSK) modulation. The BER results are compared against those of their single alphabet counterparts.

In the second part, we present a multi-stage decoding approach for LDPC codes defined over the integer ring $\mathbb{Z}_q$, where $q = p^m$, p is a prime and $m > 1$. We make use of the property that the natural ring epimorphism $\mathbb{Z}_q \mapsto \mathbb{Z}_{p^l} : r \mapsto \sum_{i=0}^{l-1} r^{(i)} p^i$

with kernel $p^l\mathbb{Z}_q$ for each l, $1 \leq l \leq m$, where $\sum_{i=0}^{m-1} r^{(i)}p^i$ is the p-adic expansion of $r \in \mathbb{Z}_q$.

We apply the multi-stage decoding algorithm to LDPC codes over $\mathbb{Z}_4$ and $\mathbb{Z}_8$ of block length $N = 500$ and 1000 symbols. We investigate the BER performance of this decoding approach over the AWGN channel with both BPSK as well as q-ary PSK modulation. The BER results are compared against those of the conventional single-stage approach.

1.4 Contribution of Thesis

The contribution of this thesis is the presentation of a class of mixed alphabet codes and the study of their performance against their single alphabet counterparts. Another contribution is the modified decoding algorithm. This modified decoding algorithm helps to streamline the decoding process and eliminates redundant computations.

Another major contribution of this thesis is the presentation of the multi-stage approach to decode LDPC codes over $\mathbb{Z}_q$. We also present a method to partition the q-ary signal space such that the elements of $\mathbb{Z}_{p^m}$ coinciding modulo p^{l+1} are grouped together, as this will minimise the probability of decoder error in the multi-stage approach.

1.5 Thesis Outline

In Chapter 2, a basic description of binary and non-binary LDPC codes will be presented. It summarises the fundamentals of LDPC codes as well as their representations via the Tanner Graph (Bipartite Graph) as well as the properties of good

LDPC codes. Some popular methods of constructing good LDPC codes as well as current popular research areas will also be discussed.

Chapter 3 describes the decoding of binary and non-binary LDPC codes via the message-passing algorithm (MPA). The MPA will be described in detail for the non-binary case. For the nonbinary MPA, the fast Fourier transform is used to reduce decoding complexity. The decoding complexity (in terms of the number of arithmetic operations required per iteration)the Fourier transform method is discussed as well.

Chapter 4 starts off with a brief exposition on mixed alphabet codes. Two currently existing codes over mixed alphabets are presented and discussed. The method of constructing the proposed novel mixed alphabet code is presented in detail. Also, we demonstrate that for such codes, distinct code coordinates are defined over different alphabets. A modified MPA which takes into account the different row and column alphabet sizes to reduce the number of redundant computations is then presented. A brief description of the system model, simulation set-up as well as the simulation results of the BER for the proposed mixed alphabet codes against their single alphabet counterparts for different block lengths is presented.

In Chapter 5, we begin by giving a brief exposition on the structure of codes defined over the integer ring $\mathbb{Z}_q$. An MPA (modified from that presented in Chapter 3) for decoding LDPC codes over $\mathbb{Z}_q$ is shown. The multi-stage decoding algorithm based on this modified MPA is then presented. Computer simulation results of the BER for codes over $\mathbb{Z}_4$ and $\mathbb{Z}_8$ of moderate lengths and rate half over AWGN with BPSK as well as q-ary modulation decoded using our multi-stage approach are shown and compared against the BER of the same codes decoded using the conventional single-stage MPA.

Chapter 6 concludes the thesis and recommends possibilities for future work.

Chapter 2

Low-Density Parity-Check Codes

2.1 Background

LDPC codes are a class of linear error-correcting block codes. Linear codes use a $K \times N$ generator matrix $\mathbf{G}$ to map blocks of length K messages $\mathbf{m}$ to blocks of length N codewords $\mathbf{c}$. The set of codewords $\mathcal{C}$ are defined as the null space of the $(N-K) \times N$ parity-check matrix $\mathbf{H}$ of full rank, i.e. $\mathbf{cH}^T = \mathbf{0}$, $c \in \mathcal{C}$.

2.2 LDPC Fundamentals

2.2.1 Regular LDPC Codes

As the name suggests, LDPC codes are defined in terms of their parity-check matrices $\mathbf{H}$ which contain mostly zeroes and only a small number of non-zero elements. In his paper [16], Gallager defined regular binary (N, j, k) LDPC codes to have block length N with exactly j ones in each column and exactly k ones in each row.

A regular non-binary (or q-ary) LDPC code can be defined in a similar manner to the regular binary LDPC code, with the only difference being that for a non-binary (N, j, k) LDPC code $\mathcal{C}$ defined over $\mathbb{F}_q = \mathrm{GF}(q = p^m)$, the code coordinates $c_j \in \{0, 1, \ldots, \alpha^{q-2}\}$, $1 \leq j \leq N$, where α is primitive in $\mathbb{F}_q$.

In this case, every parity-check equation (a row of the parity-check matrix $\mathbf{H}$) involves exactly k code symbols, and every code symbol is involved in exactly j parity-check equations. The restriction that $j < k$ is needed to ensure that more than just the all-zero codeword satisfies all of the constraints. The total number of non-zero elements in $\mathbf{H}$ is $Nj = (N - K)k$. For a full-ranked $\mathbf{H}$, the code rate is then $R = 1 - \frac{j}{k}$. For $R > 0$, it is important that $j < k$. The regular $(20, 3, 4)$ binary LDPC parity check matrix provided by Gallager [16] is shown in Figure 2.1.

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right].$$

Figure 2.1: Parity-check matrix for Gallager's (20, 4, 3) code

The two lower sections of $\mathbf{H}$ are column permutations of the first section. Note that for the given matrix, not all rows are linearly independent since rows 10 and 15 are linearly dependent on the remaining rows. The remaining 13 rows are linearly independent and hence, the rank of $\mathbf{H}$ is 13.

A new full ranked parity-check matrix $\mathbf{H}'$ can be defined by eliminating the redundant rows from $\mathbf{H}$. However, the number of ones in k columns of $\mathbf{H}'$ would decrease each time a redundant row is removed so that $\mathbf{H}'$ would no longer obey the regularity of a regular LDPC matrix. Hence, an LDPC code could often be described by a rank-deficient but regular parity-check matrix.

By studying the ensemble of all matrices formed by such column permutations, Gallager proved several important results. These include the fact that the error probability of the optimum decoder decreases exponentially for sufficiently low noise and sufficiently long block length, for fixed j. Also, the typical minimum distance increases linearly with block length.

2.2.2 Irregular LDPC Codes

For binary irregular LDPC [37] codes, the matrix is still sparse, however not all rows and columns contain the same number of ones. Every code node (please refer to Section 2.3 for explanation on Tanner graph terminology) has a certain number of edges which connect to check nodes, similarly so for check nodes. For an irregular code's parity-check matrix as well as its bipartite graph, we say that an edge has degree i on the left (respectively, right) if the code (respectively, check) node it is connected to has degree i. Suppose that an irregular graph has some maximum left degree d_l and some maximum right degree d_r. The irregular graph can be specified by the sequence $(\lambda_1, \lambda_2, \ldots, \lambda_{d_l})$ and $(\rho_1, \rho_2, \ldots, \rho_{d_r})$ where λ_i and ρ_i are the fractions of edges belonging to degree-i code and check nodes, respectively.

Further a pair of polynomials $\lambda(x) = \sum_{i=2}^{d_l} \lambda_i x^{i-1}$ and $\rho(x) = \sum_{i=2}^{d_r} \rho_i x^{i-1}$ can be defined to be the generating functions of the degree distributions for the code and check nodes, respectively. The nominal expression for rate R of the code is given by $R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}$.

2.3 Tanner Graph Representation of LDPC Codes

Any parity-check code (including an LDPC code) may be specified by a Tanner graph [45] [27]. For an (N, K) code, the Tanner graph is a bipartite graph consisting of N "code" nodes associated with the code symbols, and at least $N - K$ "check" nodes, associated with the parity-check symbols. Each code node, (respectively, check node), corresponds to a particular column, (respectively, row), of $\mathbf{H}$. For an (N, j, k) parity-check matrix, each code node has degree j is connected to j check nodes, while each check node has degree k and is in turn connected to k code nodes. An edge exists between the ith check node and the lth code node if and only if $h_{il} \neq 0$, where h_{il} denotes the entry of $\mathbf{H}$ at the ith row, lth column.

The Tanner graph for the LDPC matrix provided by Gallager is illustrated below.

In Figure 2.2, the code nodes (also known as variable nodes) are circular and denoted by x_j for $0 \leq j \leq N - 1$ (in this case, $N = 20$) and the check nodes are squares and denoted by c_i for $0 \leq i \leq N - K - 1$ (in this case, $N - K = 15$). The connection between code node j and check node i is called an edge and denoted e_{ji}

For the case of a non-binary LDPC matrix (respectively, Generator matrix) $\mathbf{H}$ (respectively, $\mathbf{G}$) defined over $\mathbb{F}_{2^m}$, each non-zero $h_{i,j}$ (respectively , $g_{i,j}$) $\in \mathbb{F}_{2^m}$ can be represented by its $m \times m$ binary matrix [10]. Multiplication of a symbol x_j by $h_{i,j}$ is equivalent to matrix multiplication (mod 2) of the binary string for x_j by

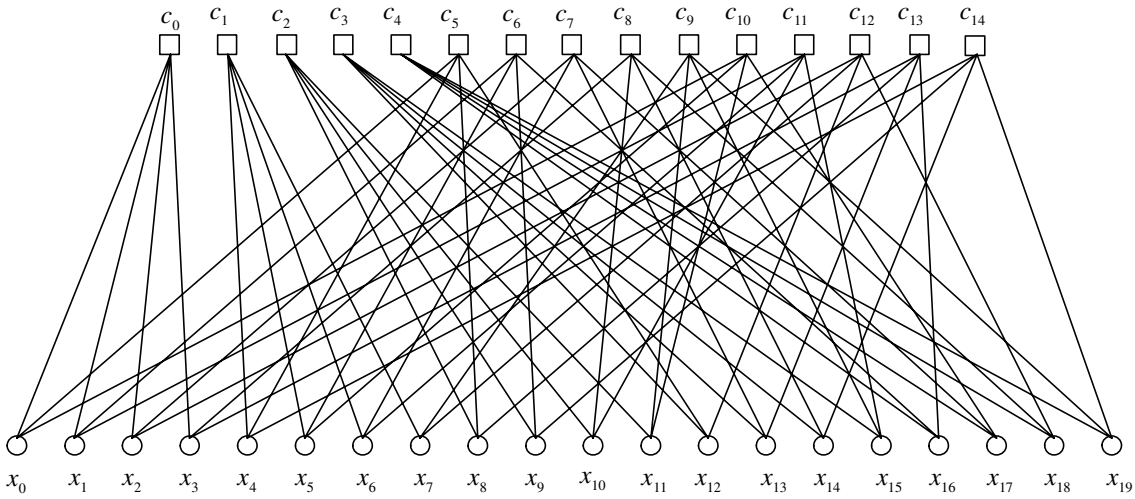

Figure 2.2: Tanner graph for $(20, 3, 4)$ Gallager LDPC matrix

the matrix associated with $h_{i,j}$. By replacing each symbol in the q-ary matrix $\mathbf{H}$ (respectively, $\mathbf{G}$) by the associated binary $m \times m$ blocks, the binary matrix $\mathbf{H}_2$ (respectively, $\mathbf{G}_2$ that is m times as large in each direction is obtained. To multiply a q-ary message $\mathbf{m}$ by $\mathbf{G}$, we can form the binary representation of $\mathbf{m}$, multiply by $\mathbf{G}_2$ and take the q-ary representation of the resulting binary vector.

Figure 2.3 shows a fragment of a non-binary matrix over $\mathbb{F}_4$ and its equivalent binary representation over $\mathbb{F}_2$ as well as their respectively Tanner graphs.

The Tanner graph gives a complete description of the structure of the LDPC matrix $\mathbf{H}$. It will be shown in Chapter 3 that the decoding algorithms work directly on this bipartite graph.

2.4 Some Factors Affecting Performance

Since its rediscovery, LDPC codes have been the subject of intense research. However, they are still not well understood. However, there are a few parameters that will improve the performance of the code.

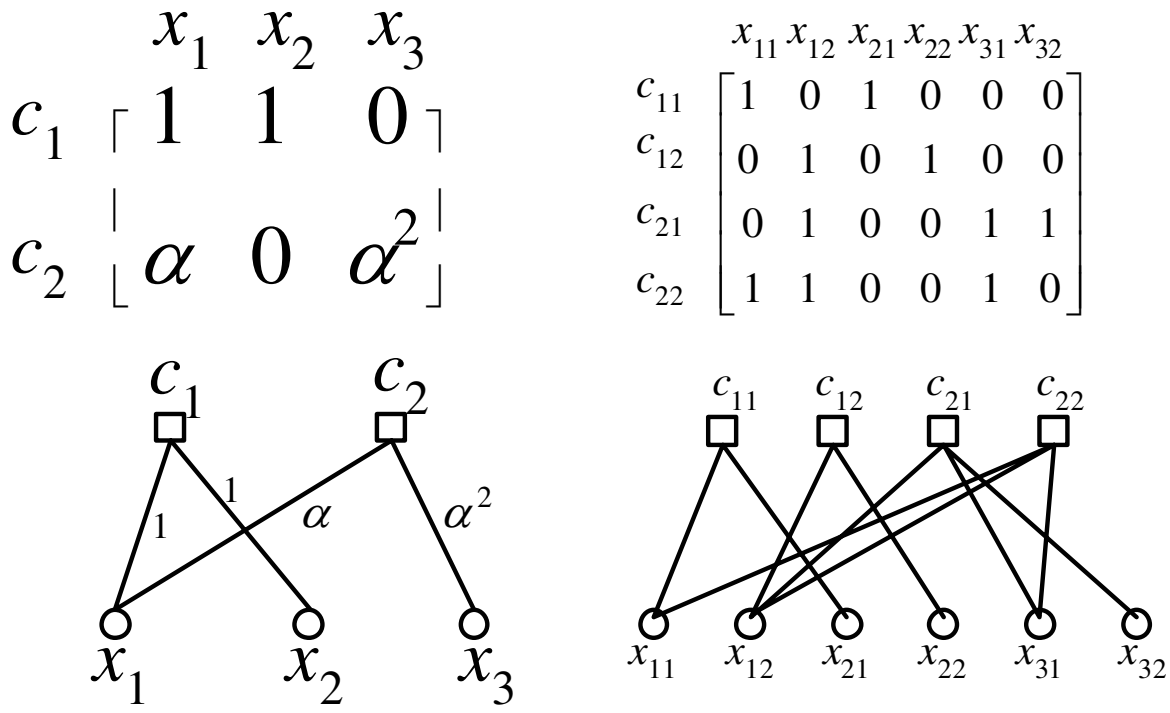

Figure 2.3: Fragments of equivalent parity-check matrices over (left) $\mathbb{F}_4$ and (right) $\mathbb{F}_2$ and comparison of their corresponding graph structure [10]

2.4.1 Sparsity

The decoding computational complexity is proportional to the sparsity of the parity-check matrix. The lesser the number of non-zero elements in the row (and column), the fewer the computations required for decoding. However this is subjected to $j \geq 2$. Another inherent reason why $j \geq 2$ will be apparent in the following sections.

The increase in the row weight k will also impair the performance of the LDPC code. This is because each check node now has more neighbours and is less confident about each neighbour's state [10].

2.4.2 Girth

A cycle in a bipartite graph is defined as a closed path of distinct edges that begins and ends at the same node. The girth of the graph is defined as the smallest cycle in the graph.

The shortest possible cycle in a bipartite graph is a length 4 cycle. Length 4 cycles are manifested in $\mathbf{H}$ as four '1's that lie at the corners of one of its submatrices. This means that between 2 columns of $\mathbf{H}$, there are 2 pairs of entries coinciding on the same row. As shown in Figure 2.4, $\mathbf{H}_4$ has a cycle of length 4. On the other hand, $\mathbf{H}_6$ shows a matrix representation of a cycle of length 6. After all short cycles of length 4 are removed from a random parity-check matrix $\mathbf{H}$, the girth of the resultant parity-check matrix $\mathbf{H}'$ is at least 6.

$$\mathbf{H}_4 = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 1 & \cdots & 1 & \cdots \\ \cdots & \vdots & \ddots & \vdots & \cdots \\ \cdots & 1 & \cdots & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \quad \mathbf{H}_6 = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \cdots & 1 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \cdots & \cdots & \cdots & 1 \\ \cdots & \cdots & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

Figure 2.4: Matrix representation of cycles of length 4 ($\mathbf{H}_4$) and 6 ($\mathbf{H}_6$)

It is important to remove short cycles in the LDPC matrix as they have a negative impact on the decoding algorithm. The decoding algorithm used (to be explained in greater detail in Chapter 3) attempts to calculate the posterior probability in an iterative fashion. These short cycles cause the results to be highly skewed after a few iterations, since the same information is reused. Thus, the estimated posterior probabilities are not accurate. On the other hand, large girth results in reduced dependency in the decoding algorithm and also allows for better approximation to the true posterior probability [27].

The removal of short cycles is particularly crucial in the case of $j = 2$ columns. In such cases, the minimum Hamming distance d_{min} will be severely affected. A code has minimum distance d_{min} if and only if every $d_{min} - 1$ columns of $\mathbf{H}$ are linearly independent and some d_{min} columns are linearly dependent [33]. [47] states that for $j = 2$, $d_{min} = G_{min}/2$ for a binary LDPC code, where G_{min} is the girth. If short cycles of length 4 are not removed for $j = 2$, then $d_{min} = 2$. Low minimum distance will also degrade the performance of the codes.

2.4.3 Size of Code Alphabet

Richardson and Urbanke showed in [38] that increasing the size of the code alphabet could lead to a corresponding increase in performance in terms of BER improvement. Davey and MacKay [10] [11] constructed codes over GF(8) and GF(4) of rates 1/4 to 1/2 and showed via simulations that such codes offer up to 0.4dB and 0.2dB of coding gain over their binary counterparts respectively on an AWGN channel. It is reasonable to expect the results to hold for other code rates as well.

Nevertheless, such improvement in performance comes at the expense of increased decoding complexity.

2.5 Construction of LDPC Codes

LDPC codes can be described in terms of their random sparse parity-check matrices, making it easy to construct LDPC codes of any rate. Many good codes can simply be constructed by specifying the column and row weight, and creating a random matrix subject to those constraints. To design good LDPC codes, we need to consider the factors listed in Section 2.4 as well. Here, we review some construction methods.

2.5.1 Gallager's Constructions

In [16], Gallager constructed regular LDPC codes where the columns and rows had fixed weights j and k respectively. The parity-check matrix was divided into j equal sized submatrices, each containing a '1' in each column. The first submatrix was constructed in some predetermined manner. The subsequent $j-1$ submatrices were random permutations of the first. An example of Gallager's original (20, 3, 4) code was shown in Section 2.2.

2.5.2 MacKay's Constructions

In [31], MacKay wanted to keep the number of short cycles present in the bipartite graph representing the parity-check matrix to a minimum. Short cycles of length 4 was removed by ensuring that any pair of columns in $\mathbf{H}$ has an overlap of at most one non-zero entry.

MacKay also showed that reducing the overall weight of the matrx via the introduction of some weight 2 columns can improve decoding, but measures must be taken to reduce the probability of low-weight codewords. MacKay described the following construction methods for matrices with no cycles of length 4.

Construction 1A In this particular construction, the column weight j are fixed at a constant value (say, $j = 3$). $\mathbf{H}$ is then constructed at random keeping the weight per row as uniform as possible. Overlap of non-zero entries between any pair of columns is kept to a maximum of one.

Construction 2A As per 1A, except up to $(N-K)/2$ of the columns have weight 2. These weight 2 columns are constructed in the form of two (possibly truncated) identity matrices of size $(N-K)/2 \times (N-K)/2$, one above the other.

Construction 1B, 2B Some carefully chosen columns from a 1A or 2A matrix are deleted, so that the bipartite graph of the matrix has no short cycles of length less than some length G_{min} (say, $G_{min} = 6$)

2.5.3 Ultra-light Matrices

In Construction 2A and 2B, MacKay used a maximum of $(N - K)/2$ weight 2 columns. Any more makes low weight codewords unacceptably likely, leading to increased decoding errors (undetected errors). With non-binary codes, more weight 2 columns can be included before encountering such problems. The constructions we will be using to construct our codes is a modification of Construction 2A mentioned above, arranging $N - K$ weight 2 columns in a staircase fashion [35].

2.5.4 Geometric Approach

Yu Kou proposed a geometric approach for constructing LDPC codes [26]. In it, four classes of codes are constructed based on the lines and points of Euclidean and projective geometries over finite fields. The codes constructed using these methods had good minimum distances and their corresponding Tanner graphs had girth of size 6.

Heng Tang et al. [44] also proposed using algebraic methods for constructing LDPC codes based on parallel and cyclic properties of lines of Euclidean and projective geometries. Five classes of quasi-cyclic and cyclic codes were generated. These codes have large girth and various minimum distances. They performed well under iterative decoding and have low encoding complexity.

For more information on finite geometries, the reader is referred to [28].

2.5.5 Combinatorial Approach

In recent years, the combinatorial approach for constructing LDPC codes have been gaining popularity. The codes are well structured, and have low-complexity implementation.

Vasic [46] makes use of balanced incomplete block designs (BIBDs) for the construction of LDPC codes. Johnson [24] constructs irregular quasi-cyclic LDPC codes derived from difference families. In [25] she constructs high-rate LDPC codes based on the incidence matrices of unital designs, making use of the fact that unital designs exist with incidence matrices which are rank deficient, giving rise to the high-rate LDPC codes with large number of parity-check equations.

Several common characteristics in LDPC codes constructed using combinatorial design are that their corresponding bipartite graphs have girth 6, they can be designed to be of very high rate ($R \geq 0.8$) and of relatively short length. They also perform well under iterative decoding.

2.5.6 Progressive Edge-Growth (PEG) Tanner Graphs

Xiao-Yu Hu [23] presented a simple but efficient method for constructing Tanner graphs having a large girth in a best-effort sense by progressively establishing edges between code and check nodes in an edge-by-edge fashion, also known as the PEG construction.

Given the number of code nodes N, the number of check nodes (say $N - K$), and the code-node-degree sequence, an edge-selection procedure is started such that the placement of a new edge on the graph has as small an impact on the girth as possible.

The PEG construction is a general, non-algebraic method for constructing graphs with large girth. Simulation results show that LDPC codes from PEG construction significantly outperform randomly constructed ones [23]. A construction similar to the PEG method was presented in [47].

2.6 Research Trends

2.6.1 Codes over Larger Alphabets

Prior to [11], [10] and [38], studies were on LDPC codes defined over the binary alphabet. The improvement obtained in increasing alphabet size has motivated research in the design of LDPC codes defined over various alphabets. Sridhara and Fuja [43] studied the performance of LDPC codes over groups and rings with coded modulation, while Erez and Miller [14] focused on the code construction techniques of LDPC codes over $\mathbb{Z}_q$ and their corresponding maximum-likelihood performance.

2.6.2 Reduction of Encoding and Decoding Complexity

One major criticism concerning LDPC codes have been their apparent high encoding complexity. This is due to their random nature, which means they have no specific characteristics that may be exploited by hardware. The several works [41] [29] [32] have attempted to address this issue. However, the methods proposed results in performance loss when compared to a standard LDPC code.

In [39], an efficient encoding method was proposed and has linear encoding complexity in the block length (compared to the previous quadratic complexity in the block length).

Decoding of non-binary LDPC codes over GF(q) becomes prohibitively complex as the size of the alphabet increases. The decoding complexity is $\mathcal{O}(q^2/p)$ where q is the size of the field, and p its characteristic. Currently, the largest field of practical interest for q-ary LDPC codes is GF(16) [42].

However, several recent works have proposed to reduce the decoding complexity of non-binary LDPC codes. Barnault [2] modified the MPA such that the computational complexity of decoding LDPC codes over GF(q) so that complexity scales as $q \log_2(q)$. Using such an algorithm, he was able to simulate the performance of LDPC codes defined over GF(256), however details of his algorithm were not given in [2]. Wymeersch proposes a log-domain decoding algorithm for LDPC codes over GF(q). This is similar to the log-likelihood ratio (LLR) decoding for binary LDPC codes. He also proposed a non-binary analogue of the of the min-sum algorithm using LLRs. [42] proposes a log-domain FFT decoding algorithm to reduce decoding complexity.

2.6.3 Implementation and Application

LDPC codes show a lot of promise and potential due to its good performance as well as low decoding complexity which is linear in the block length. There has been a lot of interest in the VLSI implementation of the LDPC decoder. Another area of interest is in magnetic recording as well as in wireless communications.

Chapter 3

Decoding of LDPC Codes

3.1 Gallager's Original Decoding Algorithm

In [16], Gallager described a simple iterative hard-decision binary decoding scheme (also known as the bit-flipping algorithm). This scheme, while not performing as well as the MPA, is less computationally complex and requires lesser memory and might still be useful in practice.

For such a decoding scheme, the decoder computes all the parity checks and then changes any code coordinate that is contained in more than some fixed number of unsatisfied parity-check equations. Using these new values, the parity checks are recomputed, and the process is repeated until the parity checks are all satisfied. This means that if the number of unsatisfied parity-check equations which a code symbol x_j participates in exceed a threshold, then the symbol is "flipped", $x_j = x_j \oplus 1$.

3.2 The Non-Binary MPA

An iterative probabilistic decoding algorithm known as the MPA or sum-product algorithm (SPA) is used to decode LDPC codes. The MPA works by iteratively

passing probabilistic messages in a graph and can be used to evaluate extrinsic and posterior probabilities based on intrinsic probabilities and the structure of the Tanner graph representation of the parity-check matrix $\mathbf{H}$.

During the decoding process, the code and check nodes receive messages from their neighbouring nodes via the edges between them and perform computations subject to the constraint of the node at hand before passing the computed messages to the next set of neighbours. For a regular LDPC code, a code node, say x_j, waits for messages to arrive from $j-1$ check nodes say $c_{i_1}, c_{i_2}, \ldots, c_{i_{j-1}}$, along $j-1$ edges, computes a corresponding message and sends it to the 0th check node c_{i_0} via the one remaining edge (i.e. the 0th edge). It then waits for c_{i_0} to send a return message. Upon the arrival of this message, it proceeds to send further messages to $c_{i_1}, c_{i_2}, \ldots, c_{i_j-1}$. The MPA is said to complete one iteration once two messages has passed over every edge, one in each direction [27].

For an LDPC code over $\mathbb{F}_q$, a parity-check constraint corresponding to the ith row of the parity-check matrix $\mathbf{H}$ has the form

$$h_{i,j_0}x_{j_0} + h_{i,j_1}x_{j_1} + \cdots + h_{i,j_{k-1}}x_{j_{k-1}} = c_i, \tag{3.1}$$

where x_{j_l} and c_i are variables associated with the code nodes and check nodes respectively; all scalar additions and multiplications are over $\mathbb{F}_q$. Equation (3.1) is represented graphically in Figure 3.1.

3.2.1 The Row Step

The decoding process involves two main steps: a column step and a row step. It is convenient to express the channel output, i.e. the *a priori* probabilities of the code nodes, as column vectors denoted by $\mathbf{p}_j = [p_j^0\ p_j^1 \ldots p_j^{\alpha^{q-2}}]^T$, where p_j^ξ is the

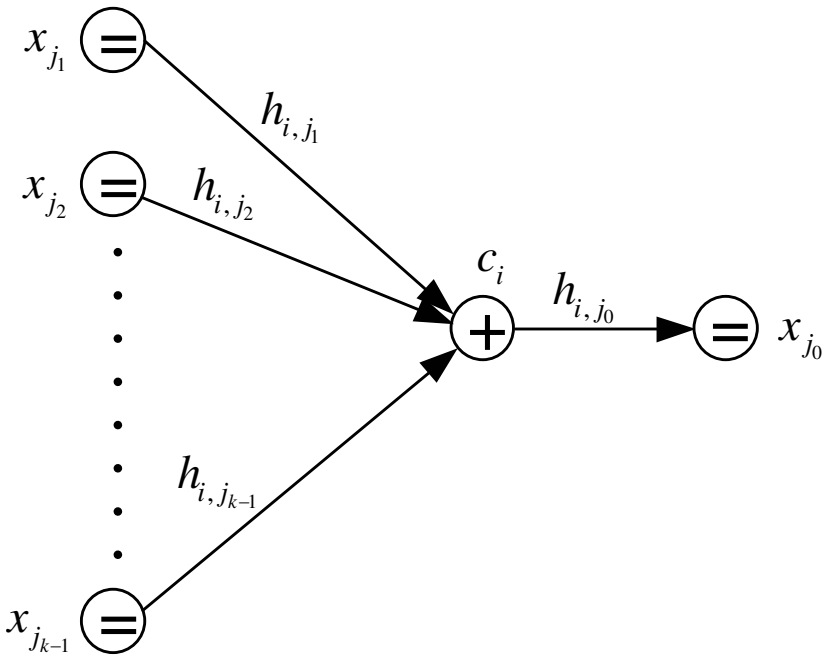

Figure 3.1: Check node c_i with k code nodes x_{j_l} connected to it.

probability that the jth code node is equal to $\xi \in \mathbb{F}_q$. Column operations involve the computation of code-to-check node messages $\mathbf{q}_{ji}$ from check-to-code node messages $\mathbf{r}_{ji}$ (both interpreted as column vectors of length q) and the $\mathbf{p}_j$. Row operations involve the computation of check-to-code node messages $\mathbf{r}_{ji}$ from the code-to-check node messages $\mathbf{q}_{ji}$ for which there is no closed-form expression when $q \neq 2$.

To compute the $\mathbf{r}_{ji}$, the edges of a parity-check node are considered in pairwise fashion. Consider the code nodes x_{j_1} and x_{j_2} in Figure 3.1. Labelling their combined output as $S_{h_{i,j_1}x_{j_1}+h_{i,j_2}x_{j_2}}$, the probability that $S_{h_{i,j_1}x_{j_1}+h_{i,j_2}x_{j_2}} = \xi_j \in \mathbb{F}_q$ may be expressed as

$$P(S_{h_{i,j_1}x_{j_1}+h_{i,j_2}x_{j_2}} = \xi_j) = \sum_{\xi_k \in \mathbb{F}_q} P(S_{h_{i,j_1}x_{j_1}} = \xi_k) P(x_{j_2} = h_{i,j_2}^{-1}(\xi_j - \xi_k)). \qquad (3.2)$$

Extending (3.2) to include all edges yields the recursive relation,

$$P(S_{h_{i,j_1}x_{j_1}+h_{i,j_2}x_{j_2}+\cdots+h_{i,j_{k-1}}x_{j_{k-1}}} = \xi_j)$$
$$= \sum_{\xi_k \in \mathbb{F}_q} P(S_{h_{i,j_1}x_{j_1}+h_{i,j_2}x_{j_2}+\cdots+h_{i,j_{k-2}}x_{j_{k-2}}} = \xi_k) P(x_{j_{k-1}} = h_{i,j_{k-1}}^{-1}(\xi_j - \xi_k)). \quad (3.3)$$

Fourier Transform Decoding

The left hand side of (3.3) is equivalent to computing $r_{j_0 i}^{\xi_j}$. The right hand side of (3.2) is essentially a discrete convolution. Hence a suitable transform can be used to reduce the number of computations required, since it is well-known that convolution is converted to point-wise multiplication in the transform domain. The choice of transform operator depends on the code alphabet. For $\mathbb{F}_{2^m}$, the appropriate transform is the Hadamard transform matrix $\mathbf{H}_{2^m}$. Correspondingly, for the check node shown in Figure 3.1, the transformed version $\mathbf{R}_{j_0 i}$ of the message $\mathbf{r}_{j_0 i}$ to be passed from c_i to x_{j_0} is

$$\mathbf{R}_{j_0 i} = \prod_{l=1}^{k-1} \mathbf{H}_{2^m} \mathbf{q}_{j_l i}, \quad (3.4)$$

where multiplication is taken component-wise. To obtain the $\mathbf{r}_{j_0 i}$, one simply performs the inverse transform

$$\mathbf{r}_{j_0 i} = \mathbf{H}_{2^m}^{-1} \mathbf{R}_{j_0 i}. \quad (3.5)$$

In (3.4) and (3.5), the weights $h_{i,j_1}, h_{i,j_2}, \ldots, h_{i,j_{k-1}}$ of the edges were assumed to be unity and thus omitted. In the case of edges with non-unity weights, the vector elements in $\mathbf{q}_{j_l i}$ for $l \in \{0, 1, \ldots, k-1\}$ are appropriately rearranged according to h_{i,j_l} and the resulting vector has the form $\mathbf{q}'_{j_l i} = [q_{j_l i}^0 \; q_{j_l i}^{h_{i,j_l}^{-1}} \; \ldots \; q_{j_l i}^{h_{i,j_l}^{-1}\alpha^{q-2}}]^T$. Hence, for the case of non-unity edge weights, we rewrite (3.4) as

$$\mathbf{R}_{j_0 i} = \prod_{l=1}^{k-1} \mathbf{H}_{2^m} \mathbf{q}'_{j_l i}. \quad (3.6)$$

Example 1 *A message* $\mathbf{q}_{j_l i} = [q_{j_l i}^0 \, q_{j_l i}^1 \, q_{j_l i}^{\alpha} \, q_{j_l i}^{\alpha^2}]^T$ *over* $\mathbb{F}_4$ *(where* α *is primitive) is passed through an edge* $h_{i,j_l} = \alpha^2$*. The resultant message after passing through the edge is* $\mathbf{q}'_{j_l i} = [q_{j_l i}'^0 \, q_{j_l i}'^1 \, q_{j_l i}'^{\alpha} \, q_{j_l i}'^{\alpha^2}]^T = [q_{j_l i}^0 \, q_{j_l i}^{\alpha} \, q_{j_l i}^{\alpha^2} \, q_{j_l i}^1]^T$.

As can be seen from the example above, the effect of h_{i,j_l} performs a cyclic shift of the elements in the message vector $\mathbf{q}_{j_l i}$ when the message elements are arranged in ascending powers of α.

The arrangement of the message elements (after rearrangement by non-unity weights of the edges) however need to be changed when performing the Fourier transform. Consider the simplest case of a check node c_i connected to 3 code nodes x_{j_1}, x_{j_2}, and x_{j_3} with unit edge weights defined over $\mathbb{F}_8$ where the primitive polynomial $p(x) = x^3 + x + 1$. If the messages are arranged in ascending powers of α (where α is primitive), i.e. $\mathbf{q}_{j_l i} = \left[q^0_{j_l i}\, q^{\alpha}_{j_l i}\, q^{\alpha^2}_{j_l i}\, q^{\alpha^3}_{j_l i}\, q^{\alpha^4}_{j_l i}\, q^{\alpha^5}_{j_l i}\, q^{\alpha^6}_{j_l i}\, q^{\alpha^7}_{j_l i}\right]^T$ for $0 \leq l \leq 2$. Using (3.5) will not provide a consistent solution for $\mathbf{r}_{j_0 i}$.

Example 2 *Consider the case where $x_{j_1} = \alpha$ and $x_{j_2} = 0$. At the zeroth iteration, we have $\mathbf{p}_{j_l} = \mathbf{q}_{j_l i}$ for $0 \leq l \leq 2$, i.e. $\mathbf{q}_{j_1 i} = [0\,1\,0\,0\,0\,0\,0\,0]^T$ and $\mathbf{q}_{j_2 i} = [1\,0\,0\,0\,0\,0\,0\,0]^T$. From (3.5), we have $\mathbf{r}_{j_0 i} = [0\,1\,0\,0\,0\,0\,0\,0]^T$ as expected.*

However, when $x_{j_1} = \alpha$ and $x_{j_2} = 1$, $\mathbf{q}_{j_1 i} = [0\,1\,0\,0\,0\,0\,0\,0]^T$ and $\mathbf{q}_{j_2 i} = [0\,0\,0\,0\,0\,0\,0\,1]^T$. (3.5) gives $\mathbf{r}_{j_0 i} = [0\,0\,0\,0\,0\,0\,1\,0]^T$ instead of the correct $[0\,0\,0\,1\,0\,0\,0\,0]^T$ since $\alpha + 1 = \alpha^3$. (3.5) also yields the same result ($\mathbf{r}_{j_0 i} = [0\,0\,0\,0\,0\,0\,1\,0]^T$) for when $x_{j_1} = \alpha^2$ and $x_{j_2} = \alpha^4$, where $\alpha^2 + \alpha^4 = \alpha$.

From example 2, the ordering of message elements is non-trivial. We solve a system of equations to find an appropriate arrangement of message elements for $\mathbb{F}_8$. We first set a message vector to be $\left[q^{b_0}_{j_l i}\, q^{b_1}_{j_l i}\, q^{b_2}_{j_l i}\, q^{b_3}_{j_l i}\, q^{b_4}_{j_l i}\, q^{b_5}_{j_l i}\, q^{b_6}_{j_l i}\, q^{b_7}_{j_l i}\right]^T$ for $0 \leq l \leq 2$, where $b_i \in \mathbb{F}_8$ are unknowns for $0 \leq i \leq 7$. We want to arrange the message elements such that (3.5) gives $\mathbf{r}_{j_0 i} = \left[r^{b_0}_{j_0 i}\, r^{b_1}_{j_0 i}\, r^{b_2}_{j_0 i}\, r^{b_3}_{j_0 i}\, r^{b_4}_{j_0 i}\, r^{b_5}_{j_0 i}\, r^{b_6}_{j_0 i}\, r^{b_7}_{j_0 i}\right]^T$.

Solving the following

$$\mathbf{r}_{j_0 i} \;=\; \mathbf{H}_8^{-1}\left(\mathbf{Q}_{j_1 i}\mathbf{Q}_{j_2 i}\right) \tag{3.7}$$

for b_i given the constraint that $\mathbf{r}_{j_0 i} = \left[r_{j_0 i}^{b_0} r_{j_0 i}^{b_1} r_{j_0 i}^{b_2} r_{j_0 i}^{b_3} r_{j_0 i}^{b_4} r_{j_0 i}^{b_5} r_{j_0 i}^{b_6} r_{j_0 i}^{b_7}\right]^T$ and $\left[q_{j_l i}^{b_0} q_{j_l i}^{b_1} q_{j_l i}^{b_2} q_{j_l i}^{b_3} q_{j_l i}^{b_4} q_{j_l i}^{b_5} q_{j_l i}^{b_6} q_{j_l i}^{b_7}\right]^T$ where $\mathbf{Q}_{j_l i} = \mathbf{H}_{2^m} \mathbf{q}_{j_l i}$ for $l = 1, 2$ gives the following solution: $b_0 = 0$, $b_1 = 1$, $b_2 = \alpha$, $b_3 = \alpha^3$, $b_4 = \alpha^2$, $b_5 = \alpha^6$, $b_6 = \alpha^4$ and $b_7 = \alpha^5$.

For the case of $\mathbb{F}_{16}$, we can define a message vector as $[q_{ji}^{b_0} q_{ji}^{b_1} q_{ji}^{b_2} q_{ji}^{b_3} q_{ji}^{b_4} q_{ji}^{b_5} q_{ji}^{b_6} q_{ji}^{b_7}$ $q_{ji}^{b_8} q_{ji}^{b_9} q_{ji}^{b_{10}} q_{ji}^{b_{11}} q_{ji}^{b_{12}} q_{ji}^{b_{13}} q_{ji}^{b_{14}} q_{ji}^{b_{15}}]^T$, where $b_i \in \mathbb{F}_{16}$, for $0 \leq i \leq 15$. The appropriate arrangement of vector elements for Fourier transform is $b_0 = 0$, $b_1 = 1$, $b_2 = \alpha$, $b_3 = \alpha^4$, $b_4 = \alpha^2$, $b_5 = \alpha^8$, $b_6 = \alpha^5$, $b_7 = \alpha^{10}$, $b_8 = \alpha^3$, $b_9 = \alpha^{14}$, $b_{10} = \alpha^9$, $b_{11} = \alpha^7$, $b_{12} = \alpha^6$, $b_{13} = \alpha^{13}$, $b_{14} = \alpha^{11}$, $b_{15} = \alpha^{12}$. This arrangement is for when the primitive polynomial $p(x) = x^4 + x + 1$.

An element $a \in \mathbb{F}_{2^m}$ can be expressed in several ways: in terms of powers of the primitive element α, in polynomial form, $a = \sum_{i=0}^{m-1} a_i x^i$, where $a_i \in \mathbb{F}_2$ and as a m element binary vector, $[a_0\, a_1\, \ldots\, a_{m-1}]$ where the elements are the polynomial coefficients.

Instead of determining the appropriate arrangement of elements of the message vectors by solving (3.7) for the required field order, we realise that we can also arrange the elements in their ascending binary order. This is apparent when we look at the arrangement of elements of the message vectors for $\mathbb{F}_8$ and $\mathbb{F}_{16}$. If we rewrite the elements of the fields in their binary representations (instead of the powers of the primitive element), we can see that the it is ascending in the binary representation. Tables 3.1 and 3.2 illustrate this.

From the above, we see that 2 rearrangements of the message vector elements are required per iteration. The first rearrangement is due to the non-binary edge weights $h_{i,j}$, to obtain $\mathbf{q}'_{ji}$. The second is required for the Fourier transform decoding to function properly, as illustrated above. With a slight abuse in notation, the rearranged vector is denoted as $\mathbf{q}'_{ji}$ as well. Similarly, upon the inverse Fourier transform, the elements of check-to-code node message (again, with slight abuse in notation, we denote this as $\mathbf{r}_{ji}$) are arranged in ascending binary value. We

	power of α	binary form
$q_{ji}^{b_0}$	q_{ji}^{0}	q_{ji}^{000}
$q_{ji}^{b_1}$	q_{ji}^{1}	q_{ji}^{100}
$q_{ji}^{b_2}$	q_{ji}^{α}	q_{ji}^{010}
$q_{ji}^{b_3}$	$q_{ji}^{\alpha^3}$	q_{ji}^{110}
$q_{ji}^{b_4}$	$q_{ji}^{\alpha^2}$	q_{ji}^{001}
$q_{ji}^{b_5}$	$q_{ji}^{\alpha^6}$	q_{ji}^{101}
$q_{ji}^{b_6}$	$q_{ji}^{\alpha^4}$	q_{ji}^{011}
$q_{ji}^{b_7}$	$q_{ji}^{\alpha^5}$	q_{ji}^{111}

Table 3.1: Arrangement of message vector elements for $\mathbb{F}_8$.

rearrange the elements of the check-to-code node message in ascending powers of α, to obtain the $\mathbf{r}_{ji}$ as defined in the earlier part of this section. We choose to arrange the elements in our messages in ascending powers of α because of the convenience it gives when dealing with edges of non-unity weights.

The transform operator is dependent on the non-binary alphabet used. As we saw from above, for alphabets defined over Galois fields of characteristic 2, $\mathbb{F}_{2^m}$, the transform operator is the Hadamard transform of size 2^m. Alternatively, for an alphabet defined over the cyclic additive group $\mathbb{Z}_q$, the appropriate transform is a discrete Fourier transform. This will be further discussed in Chapter 5. An algorithm for decoding LDPC codes over $\mathbb{Z}_q$ obtained from modifying the MPA for decoding LDPC codes over $\mathbb{F}_q$ is presented as well.

3.2.2 The Column Step

Figure 3.2 gives a graphical representation of a code node x_j connected to j check nodes.

	power of α	binary form
$q_{ji}^{b_0}$	q_{ji}^{0}	q_{ji}^{0000}
$q_{ji}^{b_1}$	q_{ji}^{1}	q_{ji}^{1000}
$q_{ji}^{b_2}$	q_{ji}^{α}	q_{ji}^{0100}
$q_{ji}^{b_3}$	$q_{ji}^{\alpha^4}$	q_{ji}^{1100}
$q_{ji}^{b_4}$	$q_{ji}^{\alpha^2}$	q_{ji}^{0010}
$q_{ji}^{b_5}$	$q_{ji}^{\alpha^8}$	q_{ji}^{1010}
$q_{ji}^{b_6}$	$q_{ji}^{\alpha^6}$	q_{ji}^{0110}
$q_{ji}^{b_7}$	$q_{ji}^{\alpha^{10}}$	q_{ji}^{1110}
$q_{ji}^{b_8}$	$q_{ji}^{\alpha^3}$	q_{ji}^{0001}
$q_{ji}^{b_9}$	$q_{ji}^{\alpha^{14}}$	q_{ji}^{1001}
$q_{ji}^{b_{10}}$	$q_{ji}^{\alpha^9}$	q_{ji}^{0101}
$q_{ji}^{b_{11}}$	$q_{ji}^{\alpha^7}$	q_{ji}^{1101}
$q_{ji}^{b_{12}}$	$q_{ji}^{\alpha^6}$	q_{ji}^{0011}
$q_{ji}^{b_{13}}$	$q_{ji}^{\alpha^{13}}$	q_{ji}^{1011}
$q_{ji}^{b_{14}}$	$q_{ji}^{\alpha^{11}}$	q_{ji}^{0111}
$q_{ji}^{b_{15}}$	$q_{ji}^{\alpha^{12}}$	q_{ji}^{1111}

Table 3.2: Arrangement of message vector elements for $\mathbb{F}_{16}$.

A code node specifies all relevant variables to be equal. For the code node x_j connected to j check nodes and $\xi \in \mathbb{F}_q$, this condition is

$$h_{i_0,j}^{-1} c_{i_0} = h_{i_1,j}^{-1} c_{i_1} = \ldots = h_{i_{j-1},j}^{-1} c_{i_{j-1}} = \xi. \tag{3.8}$$

The code-to-check node message $\mathbf{q}_{ji_0}$ from x_j to c_{i_0} can be easily computed and is given by the expression

$$\mathbf{q}_{ji_0} = \gamma \mathbf{p}_j \prod_{l=1}^{j-1} \mathbf{r}_{ji_l}, \tag{3.9}$$

where γ denotes an appropriate normalisation constant used throughout this thesis. Again, the edge weights are all assumed to be unity and thus omitted. In the case

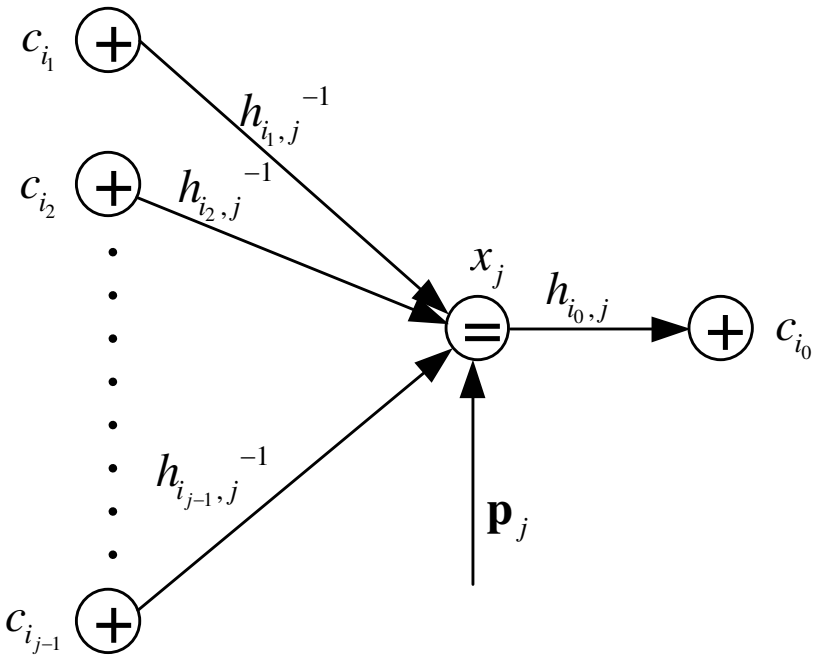

Figure 3.2: Code node x_j with j check nodes c_{i_l} connected to it.

of edges with non-unity weights, the vector elements in $\mathbf{r}_{ji_l}$ for $l \in \{0, 1 \ldots, j-1\}$ are appropriately rearranged according to the $h_{i_l,j}^{-1}$ and has the form $\mathbf{r}'_{ji_l} = [r_{ji_l}^0 \, r_{ji_l}^{h_{i_l,j}} \ldots r_{ji_l}^{h_{i_l,j}\alpha^{q-2}}]^T$. We can then rewrite (3.9) as

$$\mathbf{q}_{ji_0} = \gamma \mathbf{p}_j \prod_{l=1}^{j-1} \mathbf{r}'_{ji_l}, \tag{3.10}$$

The main steps of the MPA for a code over $\mathbb{F}_{2^m}$ may be summarised as follows:

- Step 0. *Initialisation.* The intrinsic probability vector $\mathbf{p}_j$ and the check-to-code node message vectors $\mathbf{r}_{ji}$ are initialised. Set $\mathbf{r}_{ji} = [r_{ji}^0 \, r_{ji}^1 \ldots r_{ji}^{\alpha^{2^m-2}}]^T = \frac{1}{2^m}[1\,1\ldots 1]^T$, and iteration number $l = 1$.

- Step 1. *Column Step.* The code-to-check node message $\mathbf{q}_{ji}$ is given by

$$\mathbf{q}_{ji} = \gamma \mathbf{p}_j \prod_{i' \in M(j)\setminus\{i\}} \mathbf{r}'_{ji'}, \tag{3.11}$$

where $M(j) = \{i : h_{i,j} \neq 0\}$ denotes the set of indices of the c_i corresponding to the parity-check constraints which the code node x_j participates. Then q_{ji}^{ξ} denotes the probability that $x_j = \xi \in \mathbb{F}_{2^m}$, conditioned on the information obtained via the $c_{i'}$ for $i' \in M(j) \setminus \{i\}$ (i.e., the check nodes connected to x_j other than c_i) and channel output $\mathbf{p}_j$.

- Step2. *Row Step.* The check-to-code node messages $\mathbf{r}_{ji}$ are computed as follows:

$$\mathbf{R}_{ji} = \prod_{j' \in L(i) \setminus \{j\}} \mathbf{H}_{2^m} \mathbf{q}'_{j'i}, \tag{3.12}$$

and

$$\mathbf{r}_{ji} = \mathbf{H}_{2^m}^{-1} \mathbf{R}_{ji}$$

where $L(i) = \{j : h_{i,j} \neq 0\}$ denotes the set of indices of the x_j that participate in the ith parity-check constraint.

- Step 3. *Computing the output.* An estimate of the posterior probabilities is first computed by setting

$$\mathbf{q}_j = \mathbf{p}_j \prod_{i \in M(j)} \mathbf{r}'_{ji}$$

An estimate $\hat{x}_j$ of the jth code symbol is then given by $\hat{x}_j = \arg\max_{0 \leq \xi \leq \alpha^{2^m-2}} \{q_j^{\xi}\}$.

- Step 4. *Checking the halting condition.* The decoder checks if the codeword estimate $\hat{\mathbf{x}}$ found in Step 3 is in the null space of $\mathbf{H}$, i.e., $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$. If so, the decoding process is terminated, otherwise, $l = l + 1$ and Steps 1 to 3 are repeated. If the algorithm completes a prescribed maximum number of iterations (i.e., $l = l_{\max}$) and no valid codeword has been found, a decoding failure is declared.

3.2.3 Worked Example

In this section, the iterative decoding over $\mathbb{F}_{2^m}$ will be described in detail with a worked example. The example will illustrate the process of obtaining a systematic generator $\mathbf{G}$ from a parity-check matrix $\mathbf{H}$ over $\mathbb{F}_4$ as well as the decoding process when transmitting over the AWGN channel.

Considering the sparse $N = 10$, $K = 5$, $j = 2$ parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & \alpha & 0 & 0 & \alpha & 0 & 0 & 1 & 0 \\ \alpha & 0 & 0 & 1 & \alpha^2 & 0 & \alpha & 0 & 0 & 0 \\ \alpha^2 & 0 & \alpha & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \alpha & \alpha^2 \\ 0 & \alpha & 0 & 0 & 1 & 0 & \alpha^2 & 0 & 0 & 1 \end{bmatrix},$$

which yields the corresponding systematic generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \alpha^2 & 1 & 0 & 1 & \alpha^2 \\ 0 & 1 & 0 & 0 & 0 & \alpha & 0 & \alpha & \alpha & \alpha \\ 0 & 0 & 1 & 0 & 0 & \alpha^2 & 0 & 1 & \alpha^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \alpha^2 & 0 & 0 & \alpha \\ 0 & 0 & 0 & 0 & 1 & 0 & \alpha & 0 & 0 & 0 \end{bmatrix}.$$

Assuming that a length 5 message vector $\mathbf{m} = [1\,\alpha^2\,\alpha^2\,0\,\alpha^2]$ is encoded to a length 10 codeword vector $\mathbf{x} = [1\,\alpha^2\,\alpha^2\,0\,\alpha^2\,0\,0\,\alpha\,\alpha\,\alpha]$. This codeword is then mapped onto bits for binary-phase shift-keying (BPSK) transmission over the AWGN channel. The probability of each received symbol being one of the four possible states is calculated using the soft channel output and listed in Table 3.3. The first column on the left is the codeword coordinate index, the first row on top lists all the possible states. At the end of every iteration, for a received coordinate, the symbol state having the highest estimated posterior probability is chosen as the estimated

j	0	α	α^2	$\alpha^3 = 1$	Hard Decision Value	Result
0	0.0161	0.0003	0.0174	0.9662	1	Correct
1	0.0094	0.0104	0.5173	0.4629	α^2	Correct
2	0.0487	0.0068	0.1159	0.8285	1	Error
3	0.8099	0.1676	0.0039	0.0187	0	Correct
4	0.0005	0.1296	0.8662	0.0036	α^2	Correct
5	0.9831	0.0003	0.0001	0.0165	0	Correct
6	0.5506	0.4452	0.0019	0.0023	0	Correct
7	0.0276	0.9201	0.0508	0.0015	α	Correct
8	0.5554	0.0596	0.0373	0.3477	0	Error
9	0.2609	0.7230	0.0118	0.0043	α	Correct

Table 3.3: Intrinsic symbol probabilities $\mathbf{p}_j$ calculated using channel's soft output.

transmitted symbol (i.e., a hard decision is made) and is compared with the actual transmitted one to see if it is the correct decision.

The hard decision received word is determined as $\mathbf{r} = [1\,\alpha^2\,1\,0\,\alpha^2\,0\,0\,\alpha\,0\,\alpha]$. As can be seen, the third and ninth symbols are erroneous.

Using the parity-check matrix $\mathbf{H}$ and the intrinsic probability vectors $\mathbf{p}_j$, the code-to-check node message vectors $\mathbf{q}_{ji}$ are initialised to the respective $\mathbf{p}_j$ since the initial check-to-code node message vectors $\mathbf{r}_{ji}$ are uniformly distributed.

The next step is to compute the check-to-node messages $\mathbf{r}_{ji}$. The computation of the check-to-node messages of the first row of $\mathbf{H}$ (i.e. $\mathbf{r}_{j1}$ for $j = 1$, 2, 5 and 8) is shown. First, the involved $\mathbf{q}_{j1}$ are rearranged according to the non-zero edge weights $h_{1,j}$. Table 3.4 shows the rearranged elements in $\mathbf{q}_{j1}$ for $j = 1$, 2, 5 and 8.

Prior to the FFT of the above messages, the elements are permuted to the appropriate arrangement to perform the FFT. The results of the FFT of the code-to-check node messages are shown in Table 3.5. The transformed code-to-check node

j	0	α	α^2	$\alpha^3 = 1$
1	0.0094	0.0104	0.5173	0.4629
2	0.0487	0.8285	0.0068	0.1159
5	0.9831	0.0165	0.0003	0.0001
8	0.5554	0.0596	0.0373	0.3477

Table 3.4: $\mathbf{q}_{j1}$ values of entries in the first row of $\mathbf{H}$ after rearrangement.

messages are denoted as $\mathbf{Q}_{ji}$ where

$$\mathbf{Q}_{ji} = \mathbf{H}_4 \mathbf{q}_{ji}. \tag{3.13}$$

We can compute $\mathbf{Q}_{11}$ as follows,

$$\mathbf{Q}_{11} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.4629 \\ 0.0104 \\ 0.5173 \end{bmatrix} = \begin{bmatrix} 1 \\ -0.9604 \\ -0.0554 \\ 0.0532 \end{bmatrix}.$$

This constitutes the first row of Table 3.5.

j	Q_{j1}^0	Q_{j1}^1	Q_{j1}^α	$Q_{j1}^{\alpha^2}$
1	1	-0.9604	-0.0554	0.0532
2	1	0.7545	-0.6707	-0.8890
5	1	0.9993	0.9663	0.9669
8	1	0.2300	0.8063	0.1854

Table 3.5: Results of FFT on $\mathbf{q}_{j1}$ for $j = 1$, 2, 5 and 8.

After the FFT, the transformed check-to-code node messages $\mathbf{R}_{ji}$ are computed according to (3.4). The results are given in Table 3.6. The inverse FFT is then performed according to (3.5) to obtain the check-to-code node messages $\mathbf{r}_{ji}$. The elements of $\mathbf{r}_{ji}$ are then rearranged so that it is in ascending powers of α.

j	R_{j1}^{0}	R_{j1}^{1}	R_{j1}^{α}	$R_{j1}^{\alpha^2}$
1	1	0.1734	-0.5226	-0.1594
2	1	-0.2207	-0.0432	0.0095
5	1	-0.1666	0.0300	-0.0088
8	1	-0.7241	0.0359	-0.0457

Table 3.6: Transformed check-to-code node messages $\mathbf{R}_{j1}$ for $j = 1$, 2, 5 and 8.

When the above described process has been performed repeatedly for all rows, the update of the all the check-to-code node messages $\mathbf{r}_{ji}$ for all i is complete for this iteration.

The next step calculates the estimated posterior probabilities $\mathbf{q}_j$. The posterior probability for the jth code symbol is the product of all the check-to-code node messages $\mathbf{r}_{ji}$ in the column as well as the intrinsic probability $\mathbf{p}_j$ from the channel. Table 3.7 shows the estimated posterior probabilities obtained after the first iteration.

j	0	α	α^2	$\alpha^3 = 1$	Hard Decision Value	Result
0	0.0256	0.0000	0.0101	0.9643	1	Correct
1	0.0057	0.0103	0.8861	0.0979	α^2	Correct
2	0.0216	0.0018	0.8908	0.0858	α^2	Correct
3	0.9820	0.0063	0.0108	0.0009	0	Correct
4	0.0001	0.0994	0.8998	0.0007	α^2	Correct
5	0.9900	0.0002	0.0001	0.0097	0	Correct
6	0.9362	0.0629	0.0001	0.0008	0	Correct
7	0.0090	0.9257	0.0405	0.0248	α	Correct
8	0.0540	0.0533	0.0464	0.8462	1	Error
9	0.5402	0.4218	0.0359	0.0021	0	Error

Table 3.7: Estimated posterior probabilities $\mathbf{q}_j$ after one iteration.

Comparing the results between Tables 3.3 and 3.7, we can observe that the third erroneous symbol has been corrected. However, eighth symbol remains incorrect while the ninth symbol becomes erroneous as well. However, the eighth symbol is decoded as a 1 while the ninth symbol can be considered to be marginally incorrect since q_9^0 and q_9^α values are quite close to each other. Since the estimated codeword $\hat{\mathbf{x}}$ is not a valid codeword, the decoder performs another iteration. The estimated posterior probabilities after the second iteration are listed in Table 3.8.

j	0	α	α^2	$\alpha^3 = 1$	Hard Decision Value	Result
0	0.0218	0.0000	0.0048	0.9734	1	Correct
1	0.0092	0.0065	0.7368	0.2475	α^2	Correct
2	0.0334	0.0019	0.8571	0.1076	α^2	Correct
3	0.9797	0.0141	0.0020	0.0043	0	Correct
4	0.0001	0.0427	0.9561	0.0011	α^2	Correct
5	0.9915	0.0004	0.0000	0.0081	0	Correct
6	0.9123	0.0865	0.0005	0.0007	0	Correct
7	0.0036	0.9404	0.0494	0.0066	α	Correct
8	0.0647	0.7244	0.0009	0.2100	α	Correct
9	0.4406	0.5462	0.0108	0.0025	α	Correct

Table 3.8: Estimated posterior probabilities $\mathbf{q}_j$ after two iterations.

From Table 3.8, the second iteration has corrected the remaining erroneous symbols left over from the first iteration. The codeword estimate $\hat{\mathbf{x}}$ satisfies the parity-check matrix $\mathbf{H}$ (i.e., $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$). Thus, the decoder will claim that a valid codeword has been found, terminating the decoding process.

3.2.4 Complexity of the FFT

The use of the FFT technique in the decoding of non-binary LDPC codes reduces the complexity significantly. This section evaluates the complexity in terms of the number of additions and multiplications required.

The FFT technique is used mainly in the row step in the updating of the check-to-code node messages $\mathbf{r}_{ji}$, in 2 steps: the FFT and inverse FFT of the code-to-check node messages $\mathbf{q}_{ji}$ (to $\mathbf{Q}_{ji}$) and $\mathbf{R}_{ji}$ (to $\mathbf{r}_{ji}$), respectively, and the computation of $\mathbf{R}_{ji}$ from the various $\mathbf{Q}_{ji}$. We shall first look at the computation complexity required for the FFT and the inverse FFT operations.

FFT and Inverse FFT Complexity

The Fourier transform F of a function f over $\mathbb{F}_2$ is given by $F^0 = f^0 + f^1$, $F^1 = f^0 - f^1$. This can be rewritten in matrix form as

$$\begin{bmatrix} F^0 \\ F^1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f^0 \\ f^1 \end{bmatrix}.$$

Transforms over $\mathbb{F}_{2^m}$ can be viewed as a sequence of binary transforms in each of m dimensions. For $\mathbb{F}_8$ we have

$$\begin{bmatrix} F^0 \\ F^1 \\ F^2 \\ F^3 \\ F^4 \\ F^5 \\ F^6 \\ F^7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} f^0 \\ f^1 \\ f^2 \\ f^3 \\ f^4 \\ f^5 \\ f^6 \\ f^7 \end{bmatrix}. \tag{3.14}$$

Due to the symmetric property of the matrix in (3.14), the complexity of implementing FFTs can be done using $2^m \log_2(2^m)$ additions [18]. For $\mathbb{F}_8$, $8 \times 3 = 24$ additions are required. The derivation of this figure is as follows. Firstly, the following quantities are computed,

$$\begin{aligned}
&(f^0 + f^1)\\
&(f^2 + f^3)\\
&(f^4 + f^5)\\
&(f^6 + f^7)\\
&(f^0 - f^1)\\
&(f^2 - f^3)\\
&(f^4 - f^5)\\
&(f^6 - f^7).
\end{aligned} \tag{3.15}$$

The operations listed in (3.15) requires 8 additions. The quantities computed in (3.15) are then used to compute the following next set of quantities,

$$\begin{aligned}
&(f^0 + f^1) + (f^2 + f^3)\\
&(f^0 + f^1) - (f^2 + f^3)\\
&(f^4 + f^5) + (f^6 + f^7)\\
&(f^4 + f^5) - (f^6 + f^7)\\
&(f^0 - f^1) + (f^2 - f^3)\\
&(f^0 - f^1) - (f^2 - f^3)\\
&(f^4 - f^5) + (f^6 - f^7)\\
&(f^4 - f^5) - (f^6 - f^7).
\end{aligned} \tag{3.16}$$

The second set of operations listed in equation (3.16) requires another 8 additions. Finally, the quantities obtained in equation (3.16) can be used to compute

the final result of the FFT.

$$\begin{aligned}
&\big((f^0+f^1)+(f^2+f^3)\big)+\big((f^4+f^5)+(f^6+f^7)\big)\\
&\big((f^0-f^1)+(f^2-f^3)\big)+\big((f^4-f^5)+(f^6-f^7)\big)\\
&\big((f^0+f^1)-(f^2+f^3)\big)+\big((f^4+f^5)-(f^6+f^7)\big)\\
&\big((f^0-f^1)-(f^2-f^3)\big)+\big((f^4-f^5)-(f^6-f^7)\big)\\
&\big((f^0+f^1)+(f^2+f^3)\big)-\big((f^4+f^5)+(f^6+f^7)\big)\\
&\big((f^0-f^1)+(f^2-f^3)\big)-\big((f^4-f^5)+(f^6-f^7)\big)\\
&\big((f^0+f^1)-(f^2+f^3)\big)-\big((f^4+f^5)-(f^6+f^7)\big)\\
&\big((f^0-f^1)-(f^2-f^3)\big)-\big((f^4-f^5)-(f^6-f^7)\big).
\end{aligned} \tag{3.17}$$

The final step in (3.17) requires another 8 additions. Summing all additions required in the 3 steps, the total number of additions is $2^3 \log_2(2^3) = 24$. Likewise, the inverse FFT will also require the same number of additions. However, normalisation is required during the inverse FFT since $\mathbf{H}_{2^m}^{-1} = \frac{1}{2^m}\mathbf{H}_{2^m}$. Hence a further 2^m multiplications are required. Thus, the overall complexity required in the FFT and inverse FFT of a message is $2^{m+1}\log_2(2^m)$ additions and 2^m multiplications.

Computation of $\mathbf{R}_{ji}$

After all the code-to-check node messages are transformed, the transformed check-to-code node messages $\mathbf{R}_{ji}$ can then be computed using (3.4), which can be more efficiently computed using forward-backward multiplications. This method is illustrated below.

Given a 4-element vector denoted by $[A_{old}\, B_{old}\, C_{old}\, D_{old}]$, we would like the update each element in the vector with the product of other elements within the vector

to obtain an updated vector $[A_{new}\, B_{new}\, C_{new}\, D_{new}]$, where

$$\begin{aligned} A_{new} &= B_{old} \cdot C_{old} \cdot D_{old}, \\ B_{new} &= A_{old} \cdot C_{old} \cdot D_{old}, \\ C_{new} &= A_{old} \cdot B_{old} \cdot D_{old}, \\ D_{new} &= A_{old} \cdot B_{old} \cdot C_{old}. \end{aligned} \tag{3.18}$$

Direct evaluation of equation (3.18) requires $(k-2) \cdot k$ multiplications, where k is the number of elements of the vector, in this case $k = 4$ and 8 multiplications are required. Note that equation (3.18) is analogous to the computation of $\mathbf{R}_{ji}$. We can see that the number of multiplications required in direct evaluation increases drastically as k increases. If the forward-backward multiplication technique is used, the total number of multiplications required is $3k$. The number of multiplications increases linearly with respect the number of elements in the vector. The linear complexity with respect to the number of vector elements makes this technique very suitable for the computation of $\mathbf{R}_{ji}$ of LDPC codes. The computation of the updated vector using forward-backward multiplication is shown in Table 3.9. Note that the technique requires a temporary variable, denoted *temp*. Summing the number of multiplications in Table 3.9, a total of $3k$ multiplications are required to compute a particular state. For $\mathbb{F}_{2^m}$, there are 2^m states, a total of $3(2^m)k$ multiplications are required.

For a regular (N, K) LDPC code over $\mathbb{F}_{2^m}$, a single row of $\mathbf{H}$ has $|L(i)|$ non-zero entries; hence a total of $4|L(i)|(2^m)$ multiplications and $2^{m+1}|L(i)|\log_2(2^m)$ additions are required to compute the $\mathbf{r}_{ji}$. For all $N-K$ rows, a grand total of $4(N-K)|L(i)|(2^m)$ multiplications and $2^{m+1}(N-K)|L(i)|\log_2(2^m)$ additions are required to compute all the $\mathbf{r}_{ji}$ per iteration. Since we have $(N-K)|L(i)| = N|M(j)|$, the total number of computations can also be re-expressed as $4N|M(j)|(2^m)$ multiplications and $2^{m+1}N|M(j)|\log_2(2^m)$ additions.

The update of the code-to-check node messages $\mathbf{q}_{ji}$ will also involve forward

Operation	Value of *temp*	Value of updated entry
Initialise $temp = 1$	1	–
$A_{new} = temp$	1	$A_{new} = 1$
$temp = temp \cdot A_{old}$	A_{old}	
$B_{new} = temp$	A_{old}	$B_{new} = A_{old}$
$temp = temp \cdot B_{old}$	$A_{old} \cdot B_{old}$	
$C_{new} = temp$	$A_{old} \cdot B_{old}$	$C_{new} = A_{old} \cdot B_{old}$
$temp = temp \cdot C_{old}$	$A_{old} \cdot B_{old} \cdot C_{old}$	
$D_{new} = temp$	$A_{old} \cdot B_{old} \cdot C_{old}$	$D_{new} = A_{old} \cdot B_{old} \cdot C_{old}$
$temp = temp \cdot D_{old}$	$A_{old} \cdot B_{old} \cdot C_{old} \cdot D_{old}$	
Reset $temp = 1$	1	
$D_{new} = D_{new} \cdot temp$	1	$D_{new} = A_{old} \cdot B_{old} \cdot C_{old}$
$temp = temp \cdot D_{old}$	D_{old}	
$C_{new} = C_{new} \cdot temp$	D_{old}	$C_{new} = A_{old} \cdot B_{old} \cdot D_{old}$
$temp = temp \cdot C_{old}$	$D_{old} \cdot C_{old}$	
$B_{new} = B_{new} \cdot temp$	$D_{old} \cdot C_{old}$	$B_{new} = A_{old} \cdot C_{old} \cdot D_{old}$
$temp = temp \cdot B_{old}$	$D_{old} \cdot C_{old} \cdot B_{old}$	
$A_{new} = A_{new} \cdot temp$	$D_{old} \cdot C_{old} \cdot B_{old}$	$A_{new} = B_{old} \cdot C_{old} \cdot D_{old}$
$temp = temp \cdot A_{old}$	$D_{old} \cdot C_{old} \cdot B_{old} \cdot A_{old}$	

Table 3.9: Process of forward backward multiplication for 4-element vector.

backward multiplication. Thus, another $3|M(j)|(2^m)$ multiplications are required per column of $\mathbf{H}$. A total of $3|M(j)|(2^m)N$ multiplications is required to compute all $\mathbf{q}_{ji}$ per iteration.

On average, the computational complexity associated with the decoding of one code symbol is $7|M(j)|(2^m)$ multiplications and $2^{m+1}|M(j)|\log_2(2^m)$ additions.

In this chapter, we have shown how non-binary LDPC codes over $\mathbb{F}_q$ can be decoding using the MPA. We have also shown the decoding complexity (in terms of

multiplications and additions) required per iteration of such codes. In the following chapter, we will present our work on mixed alphabet codes and show how particular combinations of alphabets and constructions can lead to improved performance over their single alphabet counterparts.

Chapter 4

Mixed Alphabet LDPC Codes

4.1 Background

In the most general form, the consideration for the MPA assumes that the messages to each edge come from symbols defined over different alphabets [27]. However, most codes studied (e.g. BCH and RS codes) have symbols of different coordinates defined over the same alphabet.

One of the factors affecting the performance of LDPC codes listed in Chapter 2.4 is the size of the code alphabet. [11] [10] showed via simulations that LDPC codes defined over $\mathbb{F}_4$ and $\mathbb{F}_8$ outperformed their binary counterparts at the expense of higher decoding complexity.

Motivated by the potential of obtaining higher coding gains from larger code alphabets, while keeping the corresponding increase in decoding complexity manageable, we introduce a novel code construction method for LDPC codes over mixed alphabets. For such codes, the parity-check equations and the code coordinates (i.e. the rows and columns of $\mathbf{H}$, respectively) can be defined over different alphabets. The code is constructed such that there are a small number of code coordinates defined over a larger alphabet, keeping the remaining over a smaller one. A modified

decoding algorithm to exploit this property to eliminate redundant computations is proposed in Section 4.5.

4.2 Some Earlier Mixed Alphabet Codes

Two earlier codes defined over mixed alphabets were found in a literature review. They are the Chinese Remainder Theorem (CRT) code [17] and the Mixed Covering codes [34] [21].

The CRT code is one which is defined over mixed number fields. The CRT states that a positive integer m is uniquely specified by its remainder modulo k relatively prime integers p_1, p_2, ..., p_k, provided $m < \prod_{i=1}^{k} p_i$. Consequently, a "message" (an integer) m can be encoded by its list of residues modulo N relatively prime integers $p_1 < p_2 < \ldots < p_N$. This redundant expression of m yields an (N, K) CRT code which can correct up to $e < \frac{N-K}{2}$ errors. While such a code has different coordinates defined over different alphabets, its parity-check matrix is not sparse. Consequently, application of the MPA to decode such codes would require too much computation to be practically viable.

Another example of earlier mixed alphabets codes are mixed covering codes [21] [34], where the focus was mainly on single-error-correcting perfect codes over mixed fields of short block length. It was proposed in [34] that coordinates defined over different alphabets be grouped together giving a parity-check matrix $\mathbf{H}$ of the form shown in Figure 4.1. The M sub-matrices $\mathbf{A}_1, \ldots \mathbf{A}_m$ are defined over distinct alphabets. Thus the null space of each sub-matrix is a code over a distinct alphabet. As $\mathbf{H}$ has a disjoint structure, the associated Tanner graph comprises of M disjoint subgraphs, each corresponding to one of the M sub-matrices. Consequently, the code symbols over the different alphabets do not interact, and in turn, there is no

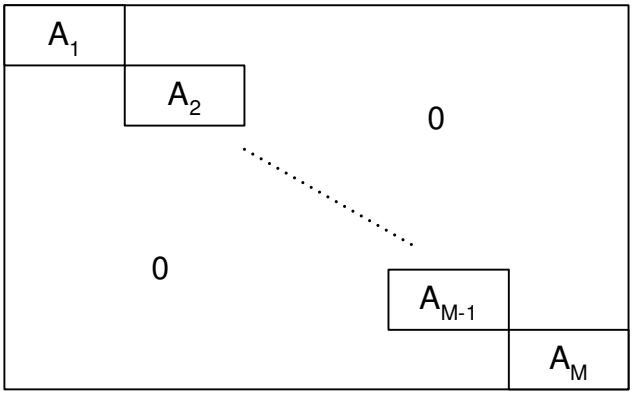

Figure 4.1: Parity check matrix form of a grouped mixed code.

change in individual code characteristics. The parameters of the code defined by $\mathbf{H}$ are

$$N = \sum_{i=1}^{M} N_i, \qquad K = \sum_{i=1}^{M} K_i, \quad \text{and} \quad d = \min_{1 \le i \le M} \{d_i\}, \tag{4.1}$$

where N_i, K_i and d_i are the length, dimension and minimum distance, respectively of the code corresponding to $\mathbf{A}_i$.

Mixed covering codes can be applied in problems of distribution of resources, possibly in coding of speech (although it is not clear whether it has been used in practice). Such codes (systems) are also widely used especially among football pool gamblers in the Nordic countries [34].

4.3 Construction of Mixed Alphabet LDPC Codes

One major difficulty of designing codes over mixed alphabets is in ensuring closure of the underlying pair of binary operations ("+" and "×") over the various fields concerned. Our approach is to construct codes with information symbols defined over $\mathbb{F}_{2^b}$ and $\mathbb{F}_{2^d}$ with some of the parity check symbols over their smallest common extension field, $\mathbb{F}_{2^m}$. The extended field is a much larger alphabet than its subfield and a constraint defined over such an alphabet would require greater computation

during decoding. To keep decoding complexity at a manageable level, the number of parity check elements defined over the extended field is therefore kept low.

Consider $(N_1-K_1)\times N_1$ and $(N_2-K_2)\times N_2$ matrices $\mathbf{A}_1$ and $\mathbf{A}_2$. The elements of $\mathbf{A}_1$ are from $\mathbb{F}_{2^b}$, while the elements of $\mathbf{A}_2$ are from $\mathbb{F}_{2^d}$. Also, consider a matrix $\mathbf{H}=[h_{i,j}]$ comprising of $\mathbf{A}_1$ and $\mathbf{A}_2$ arranged such that $\mathbf{H}=\begin{pmatrix}\mathbf{A}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{A}_2\end{pmatrix}$, $K_1<N_1$ and $K_2<N_2$. It is convenient to write $N=N_1+N_2$ and $K=K_1+K_2$. We will assume $\mathbf{A}_1$ and $\mathbf{A}_2$ to have full rank with the the linearly independent columns of $\mathbf{A}_1$, (respectively, $\mathbf{A}_2$) located in the last N_1-K_1, (respectively, N_2-K_2) columns. $\mathbf{A}_1$ and $\mathbf{A}_2$ are interpreted as the parity-check matrices of two codes, one over $\mathbb{F}_{2^b}$, the other over $\mathbb{F}_{2^d}$.

Suppose the 2 disjoint subgraphs of $\mathbf{H}$ are connected by adding edges from the code nodes of the subgraph associated with $\mathbf{A}_1$ to the check nodes of the subgraph associated with $\mathbf{A}_2$ until each check node of the latter subgraph is connected to at least one code node of the former subgraph, by taking the edges $h_{i,j}$ to be non-zero elements of the alphabet over which $\mathbf{A}_1$ or $\mathbf{A}_2$ is defined, where $0\le j\le N_1-1$ and $N_1-K_1\le i\le N_1+N_2-(K_1+K_2+1)$. With a slight abuse of notation, we denote the resulting matrix $\begin{pmatrix}\mathbf{A}_1 & \mathbf{0}\\ \mathbf{H}_E & \mathbf{A}_2\end{pmatrix}$ by $\mathbf{H}$ as well. The columns of $\mathbf{H}$ are then permuted such that the resulting matrix $\mathbf{H}'$ has the form $[\mathbf{C}_1 \mid \mathbf{C}_2]$ where $\mathbf{C}_1$ and $\mathbf{C}_2$ are $(N-K)\times K$ and $(N-K)\times(N-K)$ matrices, respectively. More specifically, $\mathbf{C}_1$ has the form $\begin{pmatrix}\mathbf{C}_{1\mathbf{A}_1} & \mathbf{0}\\ \mathbf{H}_{E_1} & \mathbf{C}_{1\mathbf{A}_2}\end{pmatrix}$ such that $[\mathbf{C}_{1\mathbf{A}_1}\,\mathbf{H}_{E_1}]^T$ is composed of K_1 columns from $[\mathbf{A}_1\,\mathbf{H}_E]^T$ while $[\mathbf{0}\,\mathbf{C}_{1\mathbf{A}_2}]^T$ is composed of K_2 columns from $[\mathbf{0}\,\mathbf{A}_2]^T$. Similarly, $\mathbf{C}_2$ has the form $\begin{pmatrix}\mathbf{C}_{2\mathbf{A}_1} & \mathbf{0}\\ \mathbf{H}_{E_2} & \mathbf{C}_{2\mathbf{A}_2}\end{pmatrix}$ such that $[\mathbf{C}_{2\mathbf{A}_1}\,\mathbf{H}_{E_2}]^T$ is composed of the remaining N_1-K_1 columns from $[\mathbf{A}_1\,\mathbf{H}_E]^T$ and $[\mathbf{0}\,\mathbf{C}_{2\mathbf{A}_2}]^T$ is composed of the remaining N_2-K_2 columns from $[\mathbf{0}\,\mathbf{A}_2]^T$. Moreover, $\mathbf{A}_1=[\mathbf{C}_{1\mathbf{A}_1}|\mathbf{C}_{2\mathbf{A}_1}]$ and $\mathbf{A}_2=[\mathbf{C}_{1\mathbf{A}_2}|\mathbf{C}_{2\mathbf{A}_2}]$. The inverse of $\mathbf{C}_2$ has the form $\begin{pmatrix}\mathbf{C}_{2\mathbf{A}_1}^{-1} & \mathbf{0}\\ \mathbf{B} & \mathbf{C}_{2\mathbf{A}_2}^{-1}\end{pmatrix}$ where $\mathbf{B}=-\mathbf{C}_{2\mathbf{A}_2}^{-1}\mathbf{H}_{E_2}\mathbf{C}_{2\mathbf{A}_1}^{-1}$ with

entries coming from $\mathbb{F}_{2^m}$. (Note: By expressing all matrix entries as elements of $\mathbb{F}_{2^m}$, all scalar multiplications and additions involved in evaluating a product or summation of matrices obey the multiplicative and additive laws of $\mathbb{F}_{2^m}$.) By our previous assumption, $\mathbf{C}_2^{-1}$ is well-defined.

The mixed code $\mathcal{C}$ can be defined by parity-check matrix $\mathbf{H}'$ and has K_1 messages over $\mathbb{F}_{2^b}$ and K_2 messages over $\mathbb{F}_{2^d}$. We can reduce $\mathbf{H}'$ to its systematic form $[\mathbf{I}_{N-K} \mid -\mathbf{P}^T]$ from $[\mathbf{C}_1 \mid \mathbf{C}_2]$, where $\mathbf{I}_{N-K}$ is the $N \times K$ by $N \times K$ identity matrix and

$$-\mathbf{P}^T = \mathbf{C}_2^{-1}\mathbf{C}_1 = \begin{pmatrix} \mathbf{C}_{2\mathbf{A}_1}^{-1}\mathbf{C}_{1\mathbf{A}_1} & \mathbf{0} \\ \mathbf{B}\mathbf{C}_{1\mathbf{A}_1} + \mathbf{C}_{2\mathbf{A}_2}^{-1}\mathbf{H}_{E_1} & \mathbf{C}_{2\mathbf{A}_2}^{-1}\mathbf{C}_{1\mathbf{A}_2} \end{pmatrix}.$$

Correspondingly, we can write the systematic generator $\mathbf{G}$ as $\mathbf{G} = [\mathbf{I}_K \mid \mathbf{P}]$, where $\mathbf{I}_K$ is the $K \times K$ identity matrix. Hence, we can express $\mathbf{G}$ as

$$\begin{aligned} \mathbf{G} &= \begin{pmatrix} \mathbf{I}_{K_1} & \mathbf{0} & (\mathbf{C}_{2\mathbf{A}_1}^{-1}\mathbf{C}_{1\mathbf{A}_1})^T & (\mathbf{B}\mathbf{C}_{1\mathbf{A}_1} + \mathbf{C}_{2\mathbf{A}_2}^{-1}\mathbf{H}_{E_1})^T \\ \mathbf{0} & \mathbf{I}_{K_2} & \mathbf{0} & (\mathbf{C}_{2\mathbf{A}_2}^{-1}\mathbf{C}_{1\mathbf{A}_2})^T \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{I}_{K_1} & \mathbf{0} & \mathbf{P}_{\mathbf{A}_1} & \mathbf{P}' \\ \mathbf{0} & \mathbf{I}_{K_2} & \mathbf{0} & \mathbf{P}_{\mathbf{A}_2} \end{pmatrix} \end{aligned} \tag{4.2}$$

where $[\mathbf{I}_{K_1}|\mathbf{P}_{\mathbf{A}_1}]$ and $[\mathbf{I}_{K_2}|\mathbf{P}_{\mathbf{A}_2}]$ are the systematic generator matrices of the codes corresponding to $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively and $\mathbf{P}' = -(\mathbf{B}\mathbf{C}_{1\mathbf{A}_1} + \mathbf{C}_{2\mathbf{A}_2}^{-1}\mathbf{H}_{E_1})^T$ is defined over $\mathbf{F}_{2^m}$.

One implicit assumption made above is that the mixed code $\mathcal{C}$ is linear. The individual code coordinates are restricted and defined over certain alphabets (i.e., $\mathbb{F}_{2^b}$, $\mathbb{F}_{2^d}$ or $\mathbb{F}_{2^m}$). In our case, the first K_1 code coordinates are defined over $\mathbb{F}_{2^b}$, the subsequent K_2 code coordinates are defined over $\mathbb{F}_{2^d}$ with the remaining $N_1 - K_1$ and $N_2 - K_2$ code coordinates defined over $\mathbb{F}_{2^b}$ and $\mathbb{F}_{2^m}$, respectively. If we define $\mathcal{G}_q$ as the additive subgroup of $\mathbb{F}_q$, then $\mathcal{C}$ is a subgroup of the additive abelian group $V = \mathcal{G}_{2^b}^{K_1}\mathcal{G}_{2^d}^{K_2}\mathcal{G}_{2^b}^{N_1-K_1}\mathcal{G}_{2^m}^{N_2-K_2}$. By strict definition, a linear block code of length N with K message symbols form a K-dimensional subspace of the vector space of all the N-tuples over the field. We may, however, view our code $\mathcal{C}$ as linear, even though

$\mathcal{C}$ does not have a basis. The inner product of the elements of V, $(u_0\, u_1\, \ldots\, u_{N-1})$ and $(v_0\, v_1\, \ldots\, v_{N-1})$ is well defined and can be expressed as the sum $\sum_{i=0}^{N-1} u_i v_i$ and obeys the additive and multiplicative laws of $\mathbb{F}_{2^m}$. Thus, we can invoke the idea of the dual code in this way where the generator matrix of the dual code of $\mathcal{C}$ can be completely specified by $\mathbf{H}'$ which is also the valid parity-check matrix for $\mathcal{C}$[1]. The code rate of $\mathcal{C}$ is given by $\frac{\sum_{j=0}^{K-1} \log_2 s_{m_j}}{\sum_{j=0}^{N-1} \log_2 s_{x_j}}$ where s_{m_j} (respectively, s_{x_j}) denotes the order of the alphabet of the jth message (respectively, code) coordinate.

Example 3 *Let α be primitive in $\mathbb{F}_{64}$ so that* $\mathbf{A}_1 = \begin{bmatrix} 1 & \alpha^{21} & 0 & 1 & 0 \\ 0 & \alpha^{42} & \alpha^{21} & 0 & 1 \end{bmatrix}$ *and* $\mathbf{A}_2 \begin{bmatrix} \alpha^9 & 1 & 0 & 1 & 0 \\ 0 & \alpha^{45} & \alpha^{54} & 1 & \alpha^{27} \end{bmatrix}$ *are full-ranked matrices over $\mathbb{F}_4$ and $\mathbb{F}_8$, respectively.*

We use $\mathbf{H}_E = \begin{bmatrix} 0 & 0 & 0 & \alpha^{42} & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$ *to connect the code nodes of the subgraph associated with $\mathbf{A}_1$ to the check nodes of the subgraph associated with $\mathbf{A}_2$ and obtain*

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha^{21} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha^{42} & \alpha^{21} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha^{42} & 0 & \alpha^9 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & \alpha^{45} & \alpha^{54} & 1 & \alpha^{27} \end{bmatrix}.$$

The columns of $\mathbf{H}$ are then appropriately permuted so that $\mathbf{C}_2$ is non-singular. This yields

$$\mathbf{H}' = \begin{bmatrix} 1 & \alpha^{21} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \alpha^{42} & \alpha^{21} & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha^9 & \alpha^{42} & 0 & 1 & 1 \\ 1 & 0 & 0 & \alpha^{54} & \alpha^{27} & 0 & 0 & 0 & 1 & \alpha^{45} \end{bmatrix}.$$

[1]The inferences, insights and equations of the above 2 paragraphs, of viewing the mixed codes as linear codes, were contributed and derived, respectively, by Dr. M. A. Armand. The author gives his thanks and gratitude to Dr. M. A. Armand for allowing him to use his work in this thesis.

The corresponding generator matrix in standard form is

$$\mathbf{G} = \left[\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \alpha^{58} & \alpha^{12} \\ 0 & 1 & 0 & 0 & 0 & 0 & \alpha^{21} & \alpha^{42} & \alpha^{36} & \alpha^{54} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \alpha^{21} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \alpha^{45} & \alpha^{45} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \alpha^{18} & \alpha^{18} \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \alpha^{45} & 1 \end{array}\right].$$

The equivalent bipartite graph is shown in Figure 4.2. Note that $c_0, c_1 \in \mathbb{F}_4$ *while* $c_2, c_3 \in \mathbb{F}_{64}$.

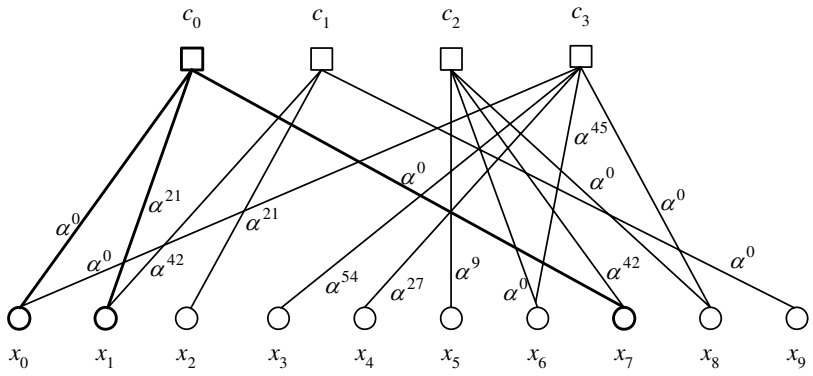

Figure 4.2: Equivalent bipartite graph.

Message blocks have the form $(m_0, m_1, m_2, m_3, m_4, m_5)$ *where* $m_0, m_1, m_2 \in \mathbb{F}_4$ *and* $m_3, m_4, m_5 \in \mathbb{F}_8$. *From the non-zero elements of* $\mathbf{G}$, *it can be observed that the codeword* $(x_0, x_1, \ldots, x_9) \in \mathcal{C}$ *has* $x_0, x_1, x_2, x_6, x_7 \in \mathbb{F}_4$, $x_3, x_4, x_5 \in \mathbb{F}_8$ *and* $x_8, x_9 \in \mathbb{F}_{64}$. *Hence,* $\mathcal{C}$ *has rate* 0.6 *and it's codewords are elements of* $\mathcal{G}_4^3\mathcal{G}_8^3\mathcal{G}_4^2\mathcal{G}_{64}^2$.

In practice, we may generate $\mathbf{H}$ as a single random binary matrix with certain constraints on the column and row weights. To construct codes with good performance, [35] suggests the use of $\frac{(N-K)}{2}$ to $N-K$ weight 2 columns arranged in a staircase fashion. For rate half codes, if we take $N-K$ columns to be of weight 2,

the remaining K columns should be of weight 3, while the row weight should be set to 5, to give good performance, [37]. This binary matrix can be generated in 2 steps. Firstly, we generate a binary matrix where the non-zero elements are arranged in a staircase fashion [35]. Then we randomly generate another matrix, having column weight 3 and row weight of approximately 3. The binary $\mathbf{H}$ is then obtained by concatenating these 2 submatrices together. We check that this binary matrix is of full rank [2]. If the matrix is not of full rank, we regenerate the matrix until the full rank condition is achieved. If the binary $\mathbf{H}$ is full rank, then this means that the binary $\mathbf{A}_1$ and $\mathbf{A}_2$ are full rank as well. To obtain a non-binary matrix, the non-zero elements of a small section of the submatrix of $\mathbf{H}$ formed by the weight 2 columns are replaced by non-zero elements of $\mathbb{F}_{2^d}$ to give us the non-binary $\mathbf{A}_2$. The remaining non-zero elements of $\mathbf{H}$ are then replaced by non-zero elements of $\mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^b}$ or $\mathbb{F}_{2^d}$) to yield the non-binary submatrix $\mathbf{A}_1$ (respectively, non-binary $\mathbf{H}_E$). The non-zero elements are drawn from their respective fields based on a uniform distribution. It is important to ensure that $\mathbf{A}_1$ and $\mathbf{A}_2$ are of full-rank.

For the $(N-K)\times(N-K)$ matrix $\mathbf{C}_2$, there always exists an $(N-K)\times(N-K)$ matrix $\mathbf{\Omega}$ such that $\mathbf{\Omega}\mathbf{C}_2 = \mathbf{C}_{2R}$, where $\mathbf{C}_{2R}$ is the reduced matrix row equivalent to $\mathbf{C}_2$. We can find $\mathbf{\Omega}$ by adjoining $\mathbf{I}_{N-K}$ to the left of $\mathbf{C}_2$ to form an $(N-K)\times 2(N-K)$ matrix $[\mathbf{I}_{N-K} \mid \mathbf{C}_2]$. We can then reduce $\mathbf{C}_2$ by performing elementary row operations on all of $[\mathbf{I}_{N-K} \mid \mathbf{C}_2]$ to eventually arrive at $[\mathbf{\Omega} \mid \mathbf{C}_{2R}]$. This produces $\mathbf{\Omega}$ with $\mathbf{\Omega}\mathbf{C}_2 = \mathbf{C}_{2R}$. If $\mathbf{C}_{2R} = \mathbf{I}_{N-K}$, then $\mathbf{\Omega} = \mathbf{C}_2^{-1}$ and this means that both $\mathbf{A}_1$ and $\mathbf{A}_2$ are full rank. If $\mathbf{C}_{2R} \neq \mathbf{I}_{N-K}$, $\mathbf{C}_2$ has no inverse and is not of full rank. This means that $\mathbf{A}_1$ and $\mathbf{A}_2$ are not of full rank. In the case where $\mathbf{A}_1$ and $\mathbf{A}_2$ are not of full rank, the above steps of regenerating non-binary $\mathbf{A}_1$ and $\mathbf{A}_2$ are repeated again.

In the MPA, the concept of the girth of the graph is crucial in estimating the posterior probabilities. If $\mathbf{A}_1$ and $\mathbf{A}_2$ do not have short cycles of length 4, careful selection of code nodes from $\mathbf{A}_1$ to connect to the check nodes of $\mathbf{A}_2$ will not affect

[2] This can be done by using the "gfrank" function in MATLAB.

the girth of the resulting graph.

4.4 Determining Column and Row Subgraph Alphabet

From example 3, we see that code coordinates are defined over different alphabets. In this subsection, a method to determine the column (code) and row (parity-check) alphabet information.

4.4.1 Column Alphabet Information

Prior to combining the codes and adding edges between the disjoint subgraphs defined by $\mathbf{A}_1$ and $\mathbf{A}_2$, we have the alphabet information of both codes say $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_1$ (respectively, $\mathcal{A}_2$) is an N_1 (respectively, N_2) element vector. $\mathcal{A}_1 = [\mathbb{F}_{2^b} \dots \mathbb{F}_{2^b}]$, while $\mathcal{A}_2 = [\mathbb{F}_{2^d} \dots \mathbb{F}_{2^d}]$. Upon combining the codes, we have $\mathcal{A} = [\mathcal{A}_1 | \mathcal{A}_2]$.

We apply the same permutation on $\mathcal{A}$ as applied on $\mathbf{H}$ to obtain $\mathbf{H}'$, giving us $\mathcal{A}'$. Upon obtaining $\mathbf{G}$ from $\mathbf{H}'$, we can proceed to determine the code coordinate alphabet of the code. To achieve this, we consider the first K elements of $\mathcal{A}'$ and examine the non-zero elements in each of the columns of the systematic $\mathbf{G}$ (from the Kth to $(N-1)$th column). For the jth column, for $0 \leq i \leq K-1$, if all the non-zero $g_{i,j} \in \mathbb{F}_{2^b}$ (respectively $\mathbb{F}_{2^d}$) and the corresponding $\mathcal{A}'(i) = \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$), then $\mathcal{A}'(j) = \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$), where $g_{i,j}$ is a non-zero entry in the ith row, jth column of $\mathbf{G}$.

On the other hand, if any $g_{i,j} \in \mathbb{F}_{2^d}$ while the corresponding $\mathcal{A}'(i) = \mathbb{F}_{2^b}$ and vice versa, then $\mathcal{A}'(j) = \mathbb{F}_{2^m}$.

Repeating this for $K \leq j \leq N-1$, we obtain all the code coordinate alphabet information.

4.4.2 Row Alphabet Information

To determine the alphabet which c_i (i.e. the ith row of $\mathbf{H}$) is defined, for $0 \leq j \leq N-1$, if all non-zero entries $h_{i,j} \in \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$) and the corresponding $\mathcal{A}'(j) = \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$), the check node $c_i \in \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$).

On the other hand, if for any $h_{i,j} \notin \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$), while the corresponding $\mathcal{A}'(i) = \mathbb{F}_{2^b}$ (respectively, $\mathbb{F}_{2^d}$), then the check node $c_i \in \mathbb{F}_{2^m}$.

4.5 Decoding Mixed Alphabet LDPC Codes

If 2^m-tuple probability vectors were used for all code-to-check node and check-to-code node messages, redundant computations would be performed. The MPA can be modified by providing the decoder with alphabet information of the row and column subgraphs. Prior to performing message passing, the decoder checks the alphabet over which a column or row is defined. In this way, the fact that only some of the subgraphs are defined over the extended alphabet is exploited.

Most of the changes to the MPA are in Step 0. In this set, the decoder now selects the number of bits to be considered as one symbol according to the code coordinate alphabet information and computes the intrinsic probability information.

If x_j is defined over $\mathbb{F}_{2^b}$, then

$$\mathbf{p}_j = \left[p_j^0 \, p_j^1 \, p_j^{\alpha^{\frac{(2^m-1)}{(2^b-1)}}} \cdots p_j^{\alpha^{(2^b-2)\frac{(2^m-1)}{(2^b-1)}}} \right]^T.$$

Else if x_j is defined over $\mathbb{F}_{2^d}$, then

$$\mathbf{p}_j = \left[p_j^0 \, p_j^1 \, p_j^{\alpha^{\frac{(2^m-1)}{(2^d-1)}}} \cdots p_j^{\alpha^{(2^d-2)\frac{(2^m-1)}{(2^d-1)}}} \right]^T.$$

Else if x_j is defined over $\mathbb{F}_{2^m}$, then

$$\mathbf{p}_j = \left[p_j^0 \, p_j^1 \, p_j^{\alpha} \cdots p_j^{\alpha^{2^m-2}} \right]^T.$$

Likewise, the check-to-code node messages are initialised according to the row subgraph alphabet information.

If the c_i is defined over $\mathbb{F}_{2^b}$ then set $\{r_{ji}^0\} = \frac{1}{2^b}$ and

$$\{r_{ji}^{\alpha^k}\} = \frac{1}{2^b} \text{ for } k \in \{0, \frac{2^m-1}{2^b-1}, \ldots, (2^b-2)\frac{2^m-1}{2^b-1}\}.$$

Else if c_i is defined over $\mathbb{F}_{2^d}$ then set $\{r_{ji}^0\} = \frac{1}{2^d}$ and

$$\{r_{ji}^{\alpha^k}\} = \frac{1}{2^d} \text{ for } k \in \{0, \frac{2^m-1}{2^d-1}, \ldots, (2^d-2)\frac{2^m-1}{2^d-1}\}.$$

Else if c_i is defined over $\mathbb{F}_{2^m}$ then set $\{r_{ji}^0\} = \frac{1}{2^m}$ and

$$\{r_{ji}^{\alpha^k}\} = \frac{1}{2^m} \text{ for } k \in \{0, 1, 2, \ldots, 2^m-2\}.$$

In Step 1, the $\mathbf{q}_{ji}$ are still computed using (3.11). The difference is that now, $\mathbf{q}_{ji}, \mathbf{p}_j$ and $\mathbf{r}_{ji}$ are column vectors of lengths 2^b, 2^d or 2^m, depending on the alphabet over which the corresponding column and row subgraphs are defined.

Correspondingly in Step 2, if the c_i is defined over $\mathbb{F}_{2^s}$ where $s \in \{b, d, m\}$ then $\mathbf{r}_{ji} = \frac{1}{2^s}\mathbf{H}_{2^s}^{-1}\left(\prod_{j' \in L(i)\backslash j} \mathbf{H}_{2^s}\mathbf{q}_{j'i}\right)$.

The changes needed in Steps 3 and 4 are trivial and hence not mentioned here.

4.5.1 Complexity of Decoding Mixed Alphabet LDPC Codes

The complexity of implementing a q-point IFFT message passing is $q\log_2(q)$ additions and q multiplications [4]. A q-point FFT requires the same number of additions (Note that the FFT requires no multiplications over a field of characteristic 2). For a parity-check constraint with $|L(i)|$ participating code nodes computation of the $\mathbf{r}_{ji}$ for ith parity-check constraint will require a total of $2|L(i)|q\log_2(q)$ additions and $4|L(i)|q$ multiplications [10]. If the ith check node is defined over $\mathbb{F}_{2^s}$, for $s \in \{b, d\}$, the above changes in the MPA reduces the number of additions and multiplications needed to compute $\mathbf{r}_{ji}$ by a factor of $2^{m-s}\frac{m}{s}$ and 2^{m-2}, respectively, over a check node with the same row weight defined over $\mathbb{F}_{2^m}$.

Hence, the number of computations required for the computation of all the $\mathbf{r}_{ji}$ per iteration is $\sum_{i=0}^{N-K-1} 4|L(i)|2^{s_i}$ where $s_i \in \{b, d, m\}$ and the check node c_i is defined over $\mathbb{F}_{2^{s_i}}$ while the number of additions required is $\sum_{i=0}^{N-K-1} 2s_i|L(i)|2^{s_i}$. On the other hand, the number of multiplications required for the computation of all the $\mathbf{q}_{ji}$ per iteration is $\sum_{j=0}^{N-1} 3|M(j)|2^{s_j}$ where $s_j \in \{b, d, m\}$ and the code node x_j is defined over $\mathbb{F}_{2^{s_j}}$.

A regular (N, K) mixed code where $N = N_1 + N_2$ and $K = K_1 + K_2$ will have N_1 code symbols over $\mathbb{F}_{2^b}$, K_2 code symbols over $\mathbb{F}_{2^d}$ and $N_2 - K_2$ code symbols over $\mathbb{F}_{2^m}$. Such a code requires $4|L(i)|[2^b(N_1 - K_1) + 2^m(N_2 - K_2)]$ multiplications and $2|L(i)|[b2^b(N_1 - K_1) + m2^m(N_2 - K_2)]$ additions to compute all the $\mathbf{r}_{ji}$ per iteration and $3|M(j)|[2^bN_1 + 2^dK_2 + 2^m(N_2 - K_2)]$ multiplications to compute all the $\mathbf{q}_{ji}$ per iteration. In particular, for a regular mixed (N, K) mixed code where $N = N_1 + N_2$ and $K = K_1$ (i.e., $K_2 = 0$), of equivalent binary length[3] $N_{\text{bin}}(= bN_1 + mN_2)$ of which $K_{\text{bin}}(= bK_1)$ are information bits,

$$4|L(i)|\left[\left(\frac{N_{\text{bin}} - K_{\text{bin}} - N_2m}{b}\right)2^b + N_2 2^m\right] \tag{4.3}$$

[3]By binary length of a code, we mean the length of the code measured in bits.

multiplications and

$$2|L(i)|\left[\left(\frac{N_{\text{bin}}-K_{\text{bin}}-N_2m}{b}\right)b2^b+N_2m2^m\right] \tag{4.4}$$

additions are required to compute all the $\mathbf{r}_{ji}$ per iteration. Further, the number of multiplications required to compute all the $\mathbf{q}_{ji}$ per iteration is

$$3|M(j)|\left[\left(\frac{N_{\text{bin}}-N_2m}{b}\right)2^b+N_22^m\right]. \tag{4.5}$$

On the other hand, a regular (N, K) single alphabet code over $\mathbb{F}_{2^s}$ ($N = N_1$ and $K = K_1$) of equivalent binary length $N_{\text{bin}} = sN$ and $K_{\text{bin}} = sK$ requires

$$4\left(\frac{N_{\text{bin}}-K_{\text{bin}}}{s}\right)|L(i)|2^s \tag{4.6}$$

multiplications and

$$2\left(\frac{N_{\text{bin}}-K_{\text{bin}}}{s}\right)|L(i)|s2^s \tag{4.7}$$

additions to compute all the $\mathbf{r}_{ji}$ per iteration. The number of multiplications required to compute all the $\mathbf{q}_{ji}$ per iteration is

$$3\frac{N_{\text{bin}}}{s}|M(j)|2^s \tag{4.8}$$

where $s \in \{b, d, m\}$.

To compute the increase in arithmetic operations required to compute all the $\mathbf{r}_{ji}$ (respectively, $\mathbf{q}_{ji}$) per iteration for the regular $(N = N_1 + N_2, K = K_1)$ mixed code over the regular $(N = N_1, K = K_1)$ code over $\mathbb{F}_{2^b}$ of the same binary length, rate, row weight distribution $\{|L(i)|\}_{i=0}^{N-K-1}$ and column weight distribution $\{|M(j)|\}_{j=0}^{N-1}$, we simply divide (4.3) by (4.6) (respectively, (4.5) by (4.8)) to obtain the fractional increase in the number of multiplications required to compute all the $\mathbf{r}_{ji}$ (respectively, $\mathbf{q}_{ji}$) per iteration for the former code over the latter one. Similarly we divide (4.4) by (4.7) to obtain the fractional increase in the number of additions required to all the compute $\mathbf{r}_{ji}$ per iteration of the former code over the latter code. Hence, the regular $(N = N_1 + N_2, K = K_1)$ mixed code requires

$$1+\frac{N_2}{N_{\text{bin}}-K_{\text{bin}}}(b2^{m-b}-m) \tag{4.9}$$

times more multiplications and

$$1 + \frac{N_2 m}{N_{\text{bin}} - K_{\text{bin}}}(2^{m-b} - 1) \tag{4.10}$$

times more additions per iteration to compute all the $\mathbf{r}_{ji}$ and

$$1 + \frac{N_2}{N_{\text{bin}}}(b2^{m-b} - m) \tag{4.11}$$

times more multiplications per iteration to compute all the $\mathbf{q}_{ji}$ than a regular (N, K) code over $\mathbb{F}_{2^b}$.

We can see that for a given $\{|L(i)|\}_{i=0}^{N-K-1}$, $\{|M(j)|\}_{j=0}^{N-1}$, N_{bin}, b and m, (4.9), (4.10) and (4.11) are linear in N_2. Also, we can see that the terms on the right hand side of the "+" denote the percentage increase in the respective arithmetic operation required for the mixed alphabet code due to the presence of the N_2 parity-check symbols over $\mathbb{F}_{2^m}$ over its single alphabet counterpart of the same binary length.

4.6 Simulations

In this section, we investigate the BER performances of the mixed alphabet codes via simulations. We present a code with all information symbols and most parity-check symbols defined over $\mathbb{F}_4$ with the remaining parity-check symbols defined over $\mathbb{F}_{64}$. Simulation results showed that the code outperformed an LDPC code defined over $\mathbb{F}_4$ for both binary block-lengths of 1000 and 2000 bits. Codes with all information symbols and most parity-check symbols defined over $\mathbb{F}_4$, with a varying number of remaining parity-check symbols defined over $\mathbb{F}_{16}$ are also presented here. Simulation results showed that the codes outperformed an LDPC code over $\mathbb{F}_4$ for the binary block length of 6000. We present another code with all information symbols and most parity-check symbols defined over $\mathbb{F}_2$ with the remaining parity-check symbols defined over $\mathbb{F}_8$ as well. Simulation results show that the code outperformed an LDPC code over its single alphabet counterpart over $\mathbb{F}_2$ for both block lengths of

1000 and 2000 bits as well. Note that the parity-check matrices of all codes simulated here have an average column weight of $M(j) = 2.5$ (half the columns are of weight 3, the other, of weight 2) and row weight $L(i) \approx 5$, where the weight 2 columns are arranged in a staircase function.

We then compare the decoding complexity of our proposed mixed alphabet codes over their single alphabet counterparts.

We begin with a brief description of the system model.

4.6.1 System Model

For the simulations, BPSK and QPSK modulation were used and AWGN was added to the signals. The intrinsic probability input (to the LDPC decoder) are the normalised conditional probabilities calculated from the channel output. The system model is shown in Figure 4.3. The LDPC encoder takes a length K message vector $\mathbf{m}$ to a length N codeword vector $\mathbf{x}$. The codeword is then passed through a symbol-to-signal mapper, mapping it to a signal block $\mathbf{b}$ to be transmitted. After these signals pass through the channel, the received block $\mathbf{y}$ is then decoded by a channel decoder.

Defining the channel output by $y_k = b_k + n_k$, where y_k (respectively, b_k) is the kth received bit (respectively, transmitted bit) and n_k is a zero mean normal random variable with variance $\sigma^2 = \frac{N_o}{2}$ (where N_o is the single-sided noise power spectral density), the likelihood values for the kth received bit y_k are given as

$$p(y_k|b_k = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_k - 1)^2}{2\sigma^2}\right), \tag{4.12}$$

$$p(y_k|b_k = -1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_k + 1)^2}{2\sigma^2}\right). \tag{4.13}$$

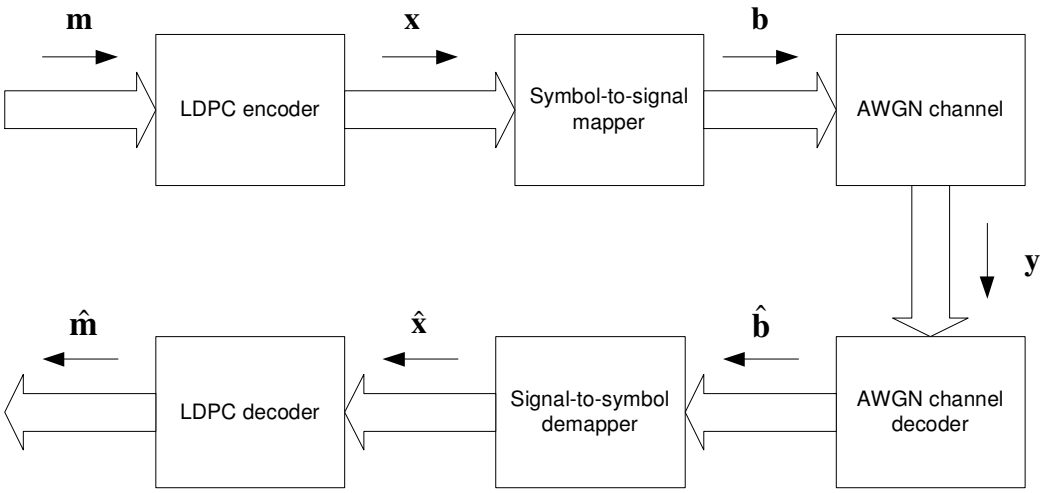

Figure 4.3: System model used for simulation.

The binary likelihood values needs to be converted to the non-binary likelihood values. Since the channel is memoryless, intrinsic input to the decoder is defined as

$$\begin{aligned} p(\mathbf{y}_i|\mathbf{x}_i = \mathbf{a}) &= p\left([y_{i_1} y_{i_2} \dots y_{i_m}]|[x_{i_1} x_{i_2} \dots x_{i_m}] = [a_1 a_2 \dots a_m]\right) \\ &= \prod_{j=1}^{m} p(y_{i_j}|x_{i_j} = a_j), \end{aligned} \tag{4.14}$$

where $\mathbf{y}_i$ is the ith received symbol and $\mathbf{a} = [a_1\ a_2\ \dots\ a_m]$ is the binary representation of the symbol.

4.6.2 Simulation Results

Performance of codes using BPSK modulation

In this section, the performance of our mixed alphabet codes is compared against their single alphabet counterparts decoded using the MPA on an AWGN channel, with BPSK modulation. Here we focus on codes of rate one-half. Thus, all the parity-check matrices $\mathbf{H}$ of all the codes simulated here have an average column weight of 2.5 (half the columns are of weight 3, the remaining half, of weight 2) and

row weight approximately 5. The weight 2 columns of all the parity-check matrices are arranged in a staircase function. We then perform column permutation of $\mathbf{H}$ to yield $\mathbf{H}'$, which is then used to obtain the systematic generator $\mathbf{G}$.

Starting with a 234×482 matrix $\mathbf{A}_1$ over $\mathbb{F}_4$ and a 6×6 matrix $\mathbf{A}_2$ over $\mathbb{F}_{64}$, a code (mixed code 1) which maps a 4-ary message block of length 248 to a codeword comprising 482 code symbols over $\mathbb{F}_4$ and 6 code (parity) symbols over $\mathbb{F}_{64}$ is generated. Similarly, starting with a 486×980 matrix $\mathbf{A}_1$ over $\mathbb{F}_4$ and a 5×5 matrix $\mathbf{A}_2$ over $\mathbb{F}_{64}$, we generate a code (mixed code 2) which maps a 4-ary message block of length 494 to a codeword comprising 980 code symbols over $\mathbb{F}_4$ and 5 code (parity) symbols over $\mathbb{F}_{64}$. Notice that for both codes, $K_2 = 0$ and so their generator matrices have the form $[\mathbf{I}_{K_1} \mid \mathbf{P}_{\mathbf{A}_1} \mid \mathbf{P}']$, i.e., each code is in fact a code over $\mathbb{F}_4$, extended by adding a small number of redundant symbols over $\mathbb{F}_{64}$; also note that in this case, the elements of $\mathbf{H}_E$ are drawn from $\mathbb{F}_4$. We thus compare their BER performance against their single-alphabet counterparts over $\mathbb{F}_4$ of similar length (in bits) and rate. As Figure 4.4 shows, both mixed codes offer an additional coding gain of about 0.07dB over their $\mathbb{F}_4$ counterparts. For completeness, the BER performance of $\mathbb{F}_8$ codes of similar binary length and rate were simulated and compared against those of the $\mathbb{F}_4$ codes. We see that the $\mathbb{F}_8$ codes offer an additional coding gain of 0.24 dB over their $\mathbb{F}_4$ counterparts for both code lengths simulated, as Figure 4.4 shows. Due to the nature of multiplications being more costly than additions in a hardware implementation, we shall only consider the number of multiplications required for decoding in the following analysis.

The $\mathbb{F}_4$ code of binary length 1000 bits requires 20000 multiplications to compute all the $\mathbf{r}_{ji}$ and 15000 multiplications to compute all the $\mathbf{q}_{ji}$ per iteration; while the $\mathbb{F}_4$ code of binary length 1990 requires 39760 multiplications to compute all the $\mathbf{r}_{ji}$ and 29850 to compute all the $\mathbf{q}_{ji}$ per iteration. Table 4.1 shows the increase in multiplications required to perform the row and column steps per iteration as well as the overall increase in multiplications for mixed codes 1 and 2 as well as the $\mathbb{F}_8$.

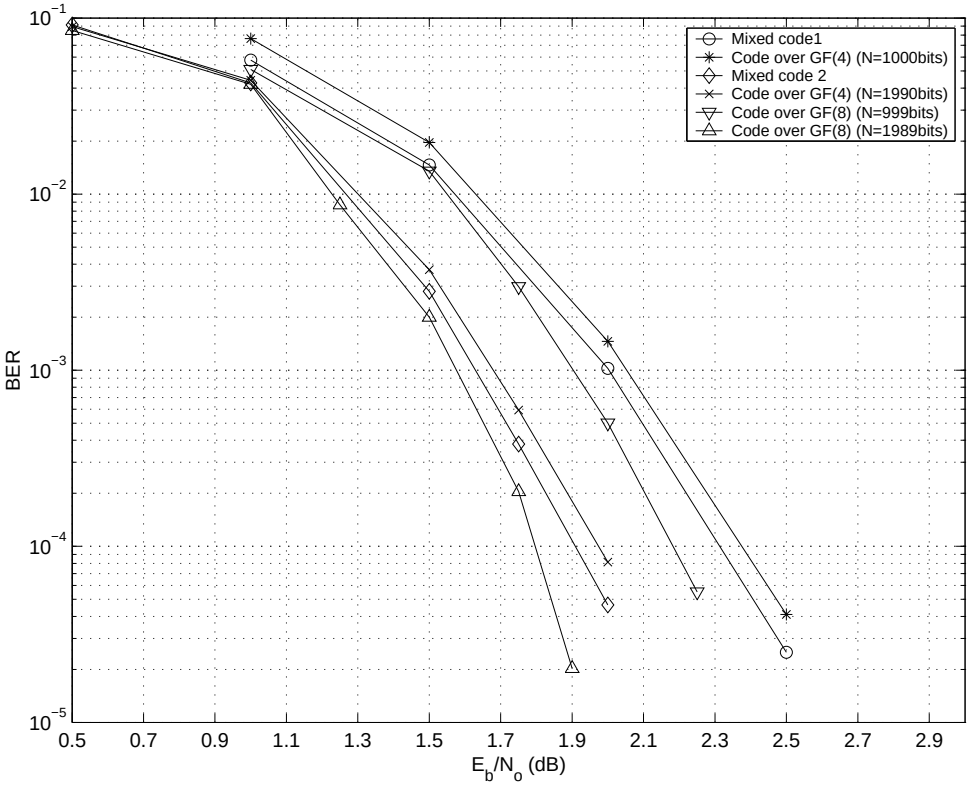

Figure 4.4: BER Performance of mixed alphabet codes and codes over GF(4) and GF(8).

These quantities may be obtained using the expressions given in Section 4.5.1.

From the last column of Table 4.1, we see that mixed code 1 and 2 required 24% and 10.2% more multiplications, respectively, than their $\mathbb{F}_4$ counterparts of the same binary length. This is the penalty incurred for 0.07 dB of coding gain. However, we see that the $\mathbb{F}_8$ codes require about 33% more multiplications than their $\mathbb{F}_4$ counterparts, whilst obtaining a coding gain of 0.24 dB. Thus, our mixed codes, particularly mixed code 2, provide a performance-complexity trade-off.

We also generate mixed codes of longer lengths ($N_{\text{bin}} = 6000$) having code symbols defined over $\mathbb{F}_4$ and $\mathbb{F}_{16}$. Starting with a 1490×2990 matrix $\mathbf{A}_1$ over $\mathbb{F}_4$ and a 5×5 matrix ($N_2 = 5$) $\mathbf{A}_2$ over $\mathbb{F}_{16}$ with elements of $\mathbf{H}_E$ drawn from $\mathbb{F}_{16}$, a code which maps a 4-ary message block of length 1500 to a codeword comprising of 2990 code symbols over $\mathbb{F}_4$ and 5 code (parity) symbols over $\mathbb{F}_{16}$ is generated. Similarly,

Code	N_{bin}	K_{bin}	b, m	N_2	increase in row multiplications	increase in column multiplications	overall increase in multiplications
mixed code 1	1000	496	$2, 6$	6	6200 (32.0%)	2340 (15.6%)	8749 (24.0%)
mixed code 2	1990	988	$2, 6$	5	5120 (12.9%)	1950 (6.0%)	7070 (10.2%)
$\mathbb{F}_8$ code	999	501	-, 3	-	6560 (32.8%)	4980 (33.2%)	11540 (33.0%)
$\mathbb{F}_8$ code	1989	996	-, 3	-	12520 (31.5%)	9930 (33.3%)	22450 (32.3%)

Table 4.1: Increase in arithmetic operations required to decode mixed codes 1 and 2 and $\mathbb{F}_8$ codes over $\mathbb{F}_4$ codes of similar N_{bin} per iteration

we generate 4 other codes for $N_2 = 10$, 15, 20 and 25; these codes map 4-ary message blocks of length 1500 to codewords comprising of 2980, 2970, 2960, 2950 code symbols over $\mathbb{F}_4$, respectively, and 10, 15, 20 and 25 code (parity) symbols over $\mathbb{F}_{16}$, respectively. We compare the BER performance against their single-alphabet counterparts of $\mathbb{F}_4$ of the same N_{bin} and rate. As Figure 4.5 shows, we see improvements in the BER performance of the mixed code as N_2 increases from 5 to 25. The mixed code with $N_2 = 25$ offers an additional coding gain of 0.065 dB over its $\mathbb{F}_4$ counterpart. Comparatively, the $\mathbb{F}_8$ code offered 0.125 dB of coding gain over the $\mathbb{F}_4$ code.

The $\mathbb{F}_4$ code of $N_{\text{bin}} = 6000$ requires 120000 multiplications to compute all the $\mathbf{r}_{ji}$ and 90000 multiplications to compute all the $\mathbf{q}_{ji}$ per iteration. Per iteration, the mixed codes with $N_2 = 5$, 10, 15, 20 and 25 require 120800, 121600, 122400, 123200, 124000 multiplications, respectively, to compute all the $\mathbf{r}_{ji}$ and 90300, 90600, 90900, 91200, 91500 multiplications, respectively, to compute all the $\mathbf{q}_{ji}$. This translates to an overall increase in complexity of 0.524%, 1.05%, 1.57%, 2.10%, 2.62%, respec-

tively, over the $\mathbb{F}_4$ counterpart. On the other hand, the $\mathbb{F}_8$ code requires 160000 multiplications to compute all the $\mathbf{r}_{ji}$ and 120000 multiplications to compute all the $\mathbf{q}_{ji}$ per iteration. This translates to an overall increase in complexity of 33.3%.

From the above, we can see that with only 25 $\mathbb{F}_{16}$ symbols, our code managed to outperform the $\mathbb{F}_4$ code by more than half the gap between the $\mathbb{F}_4$ and $\mathbb{F}_8$ codes at the expense of a negligible increase in decoding complexity.

We can attribute this drastic reduction in computational complexity to two factors. For the mixed codes of longer block lengths, we set $d = 4$ instead of $d = 6$. This greatly reduces the number of multiplications required to compute the $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$ for the rows and columns that are defined over $\mathbb{F}_{2^d}$. Secondly, with the longer block length, the multiplications required to compute the $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$ for the N_2 rows and columns over $\mathbb{F}_{2^d}$ is much smaller in proportion to the total number required, since N_1 is now much larger.

We then generate mixed codes over having code symbols defined over $\mathbb{F}_2$ and $\mathbb{F}_8$. Starting with a 490×990 matrix $\mathbf{A}_1$ over $\mathbb{F}_2$ and a 5×5 matrix $\mathbf{A}_2$ over $\mathbb{F}_8$, a code (mixed code 3) which maps a binary message block of length 500 to a codeword comprising of 990 code symbols over $\mathbb{F}_2$ and 5 code (parity) symbols over $\mathbb{F}_8$ is generated. Similarly, we generate a code (mixed code 4) which maps a binary message block of length 1000 to a codeword comprising 1985 code symbols over $\mathbb{F}_2$ and 5 code (parity) symbols over $\mathbb{F}_8$ from a 985×1985 matrix $\mathbf{A}_1$ over $\mathbb{F}_2$ and a 5×5 matrix $\mathbf{A}_2$ over $\mathbb{F}_8$. Again, $K_2 = 0$ and elements of $\mathbf{H}_E$ are drawn from $\mathbb{F}_2$ for both codes. We compare their BER performance against their single alphabet counterparts over $\mathbb{F}_2$ of the same binary length and rate (the parity-check matrices of the binary codes are obtained using the method described in Section 4.3 for obtaining the binary $\mathbf{H}$). As Figure 4.6 shows, both mixed codes offer an addition coding gain of about 0.06 dB over their $\mathbb{F}_2$ counterparts. We can also interpret the above as mixed codes 3 and 4 reducing the gap between the $\mathbb{F}_4$ codes previously

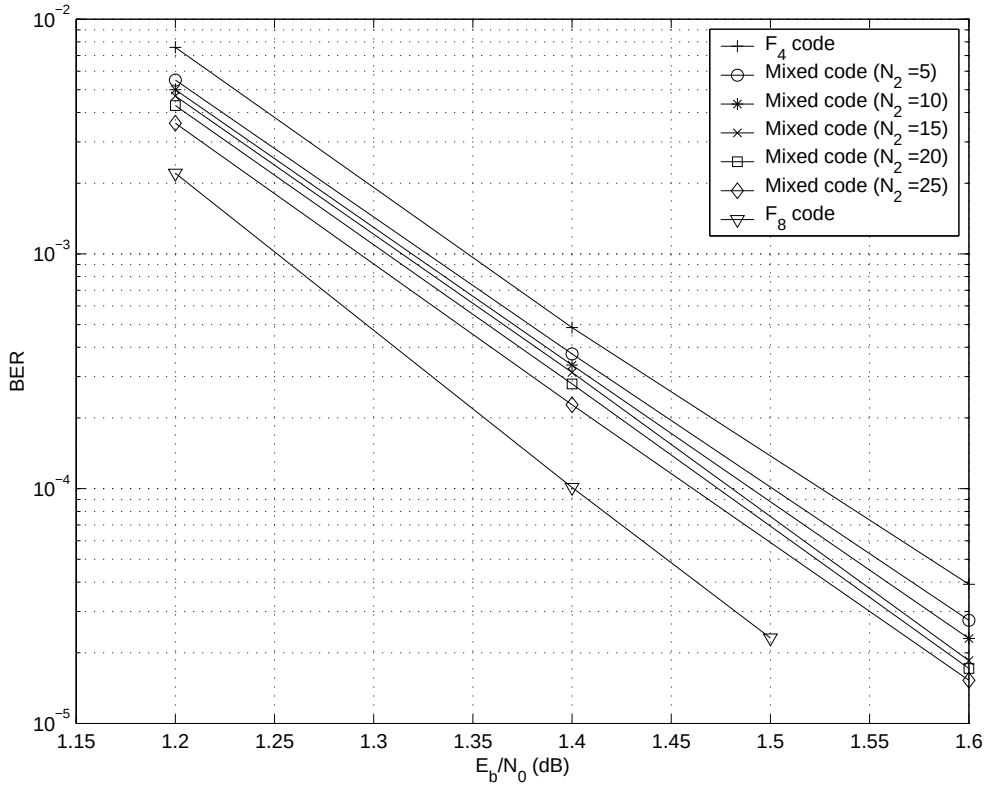

Figure 4.5: BER Performance of long length mixed alphabet codes with different N_2 and codes over GF(4) and GF(8).

considered and the $\mathbb{F}_2$ codes by 0.06 dB.

For binary LDPC codes, it is well known that decoding is performed in the logarithmic domain using log-likelihood ratios (LLR's) and involves only additions. Hence, we do not consider the increase in complexity when decoding mixed codes 3 and 4 compare to their $\mathbb{F}_2$ counterparts. Instead, we compare the complexity of these mixed codes with respect to their $\mathbb{F}_4$ counterparts.

For regular mixed codes where a code over $\mathbb{F}_2$ of length N_1 is extended with N_2 symbols over $\mathbb{F}_{2^d}$ and forms a subgroup of $\mathcal{G}_2^{N_1}\mathcal{G}_{2^d}^{N_2}$, if the row and column subgraphs defined over $\mathbb{F}_2$ are decoded in the logarithmic domain, the number of multiplications required to compute all the $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$ per iteration is $2^d N_2(4|L(i)|+3|M(j))|$. For both mixed codes 3 and 4, the total number of multiplications required to compute all the $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$ per iteration is 1100. This is just 3.14% and 1.58% of the total

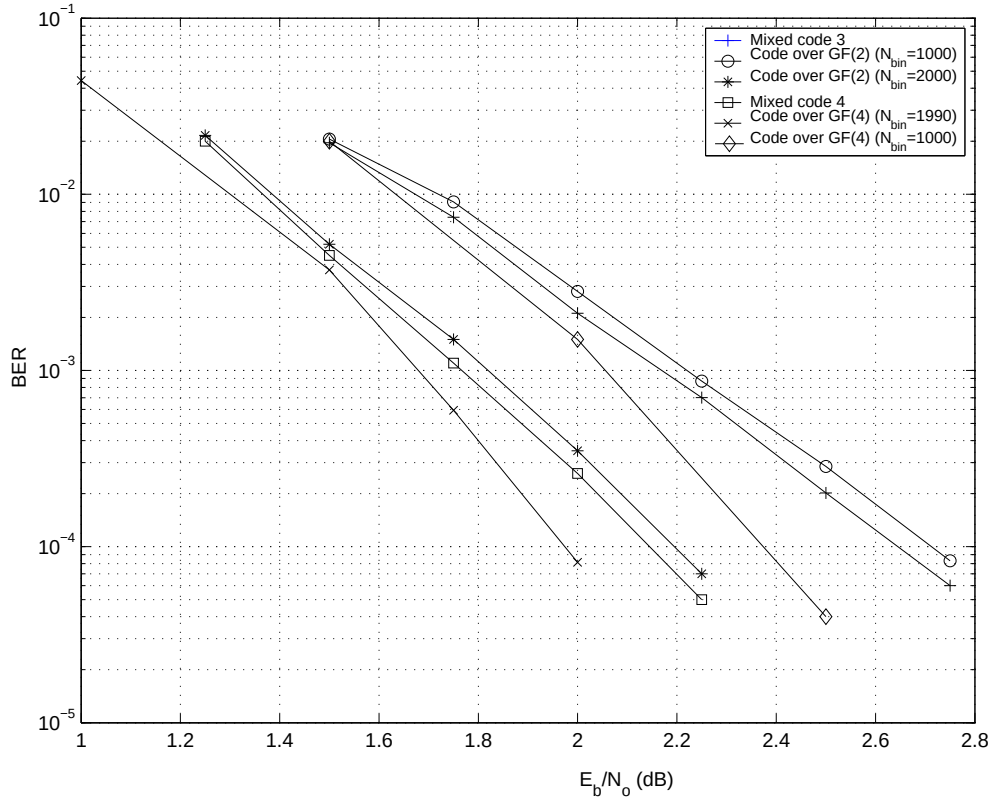

Figure 4.6: BER performance of mixed codes and binary codes and codes over GF(4).

number of multiplications required to compute all the $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$ per iteration for the $\mathbb{F}_4$ codes of binary length 1000 and 1990 respectively. This further supports our claim that our mixed codes provide a performance-complexity trade-off.

Performance of codes using QPSK modulation

In this section, we analyse the performance of the mixed codes against their single-alphabet counterparts over the AWGN channel using QPSK modulation. We compare the BER performance of the $N_{\text{bin}} = 6000$ mixed codes with $N_2 = 5$, 10, 15 and 25 against their single-alphabet counterparts over $\mathbb{F}_4$ of the same length and rate. We see from Figure 4.7 that the performance of the codes using QPSK modulation is very similar to when BPSK modulation was used. The mixed codes with $N_2 = 15$ and 25

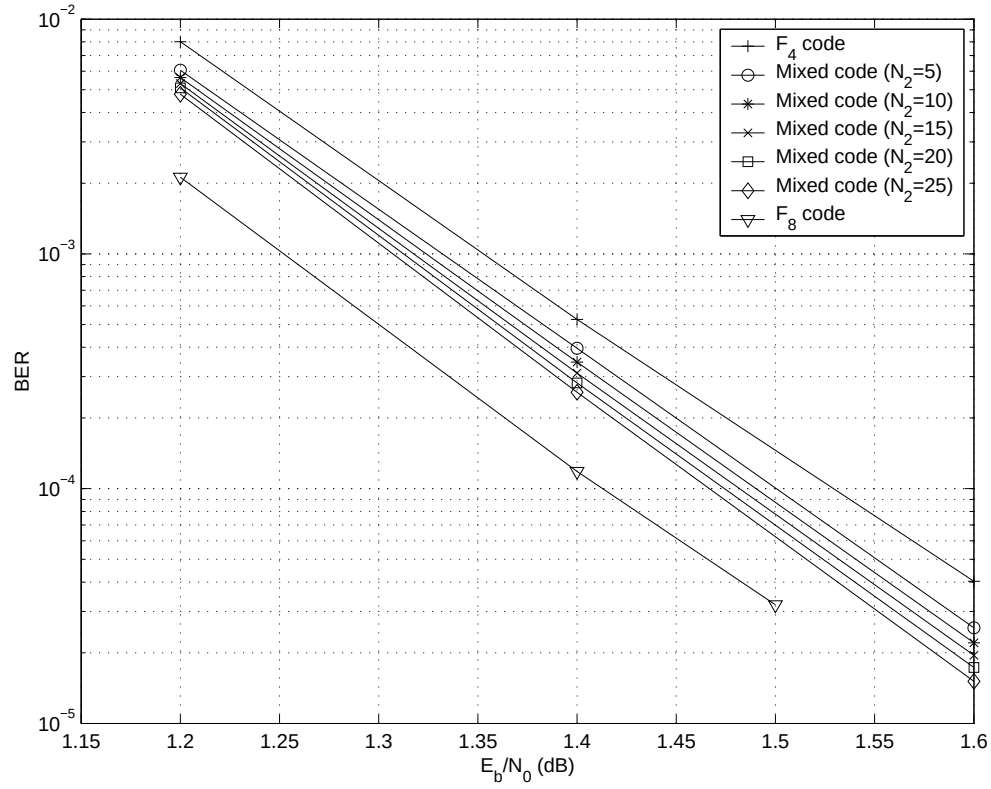

Figure 4.7: BER Performance of long length mixed alphabet codes with different N_2 and code over GF(4) with QPSK modulation.

offer additional coding gains of 0.05 and 0.064 dB over the $\mathbb{F}_4$ code, respectively.

Performance of codes under uncorrelated fast-fading Rayleigh channel

In this section, the system model illustrated by Figure 4.3 in Section 4.6.1 is modified as follows. The AWGN channel and AWGN channel decoder are changed to the uncorrelated fast-fading Rayleigh channel and uncorrelated fast-fading Rayleigh channel decoder, respectively.

In our case, we assume the fading model shown in Figure 4.8 , which is widely used in evaluating the performance of error-correcting codes [30] [48].

Hence, the received signal can be written as $y_k = a_k b_k + n_k$, where the coefficient a_k is a random variable which follows the Rayleigh distribution with probability

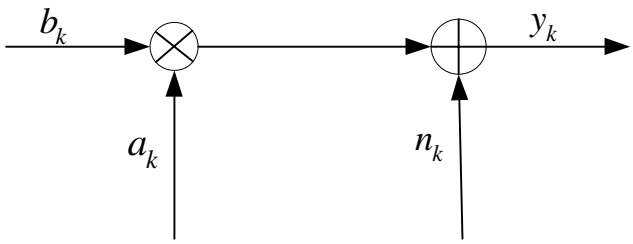

Figure 4.8: Fading channel model.

density function (pdf) $f(a) = 2a\exp(-a^2)$, with $\mathrm{E}[a] = 0.8862$ and $\mathrm{E}[a^2] = 1$; and n_k is the zero-mean AWGN with variance $\sigma^2 = \frac{N_o}{2}$. For the uncorrelated Rayleigh fading channel, the conditional pdf of the channel output is [22]

$$p(y_k|b_k, a_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_k - a_k b_k)^2}{2\sigma^2}\right). \tag{4.15}$$

We can see that (4.15) is essentially the same as (4.12) and (4.13), except that we need to account for a_k in the former.

In the case where we have exact side (or state) information (SI), we have perfect knowledge of the Rayleigh random variable a and can evaluate (4.15) directly. However, in the case where no SI is available, we follow [19] and assume that $p(y|b)$ is Gaussian distributed in the region of the most probable y and use

$$p(y_k|b_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_k - \mathrm{E}[a]b_k)^2}{2\sigma^2}\right). \tag{4.16}$$

to estimate the conditional pdf of y, where $\mathrm{E}[a]$ is used to evaluate the channel SI. We compute the non-binary likelihood values to the decoder as per (4.14).

We compare the BER performance of the $N_{\mathrm{bin}} = 6000$ mixed codes with $N_2 = 15$ and 25 against their single-alphabet counterparts over $\mathbb{F}_4$ of the same length and rate. As Figure 4.9 shows, the mixed code with $N_2 = 15$ and 25 offer additional coding gains of about 0.07 and 0.1 dB, respectively, over the $\mathbb{F}_4$ code. We also simulate the performance of a $\mathbb{F}_8$ code of similar N_{bin} and rate and compare it against that of the $\mathbb{F}_4$ code. We see from Figure 4.9 that the $\mathbb{F}_8$ code offers an additional coding gain of 0.15 dB over its $\mathbb{F}_4$ counterpart.

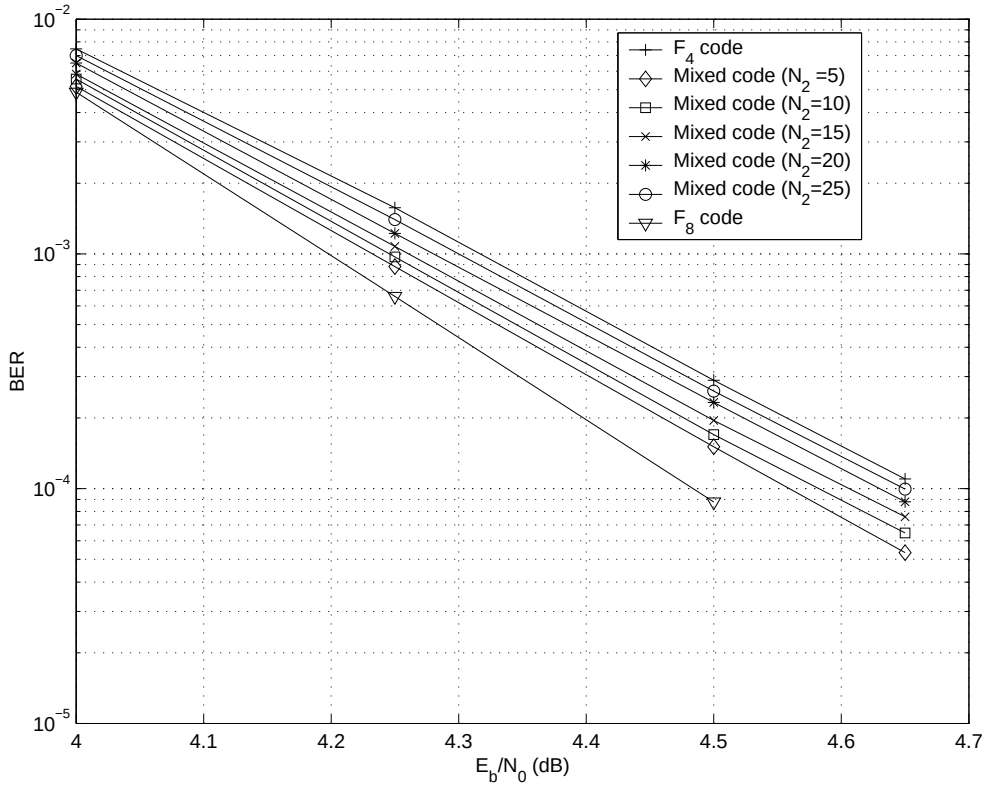

Figure 4.9: BER Performance of mixed alphabet codes and code over GF(4) and GF(8) over the Rayleigh fading channel.

Over a Rayleigh fading channel, the codes suffer an approximate performance drop of about 3 dB compared to their performance over the AWGN channel. However, codes over larger sized alphabets continue to outperform codes with smaller sized alphabets. We see this from the order of performance of the codes; the mixed codes outperform their single-alphabet counterpart over $\mathbb{F}_4$ and the $\mathbb{F}_8$ code outperforms both the mixed codes and the $\mathbb{F}_4$ code. In addition, it is interesting to note that an additional coding gain of 0.1 dB (respectively, 0.15 dB) was obtained for the mixed (respectively, $\mathbb{F}_8$) code with $N_2 = 25$ over its $\mathbb{F}_4$ counterpart, while the same code managed an additional coding gain of 0.065 dB (respectively, 0.125 dB) over the $\mathbb{F}_4$ code over the AWGN channel. Once again, the mixed code with $N_2 = 25$ $\mathbb{F}_{16}$ symbols managed to outperform the $\mathbb{F}_4$ code by more than half the gap between the $\mathbb{F}_4$ and $\mathbb{F}_8$ codes at the expense of a negligible increase in decoding complexity, this time, over the uncorrelated, fast fading Rayleigh channel.

The above results show that the coding gain obtained by our mixed codes over their single-alphabet counterparts, over the uncorrelated, fast fading Rayleigh channel is more than that obtained over the AWGN. This indicates that our the structure of the mixed codes might make them more attractive for use over the uncorrelated, fast fading Rayleigh channel than their single-alphabet counterparts, especially since the increase in decoding complexity is negligible. Furthermore, we might expect our mixed codes to perform better over their single-alphabet counterparts over the Rayleigh fading channels in the presence of SI.

4.7 Concluding Remarks

To summarise, a novel construction for LDPC codes over mixed alphabets has been presented, along with a modified MPA to decode these codes efficiently. Our computer simulations have further shown that the proposed code construction method may be used to obtain improvements in BER performance over single-alphabet LDPC codes at the expense of correspondingly slight increase in decoding complexity.

Also, we have observed that, for codes of short block length, our method of mixing code symbols from alphabets of different cardinality does not always lead to performance improvements. For example, we have found that an $\mathbb{F}_4$ (respectively, $\mathbb{F}_2$) code can outperform the same code extended with $\mathbb{F}_{16}$ (respectively, $\mathbb{F}_4$) parity-check symbols. On the other hand, an $\mathbb{F}_4$ (respectively, $\mathbb{F}_2$) code extended with $\mathbb{F}_{64}$ (respectively, $\mathbb{F}_8$) parity-check symbols can outperform the original code over $\mathbb{F}_4$ (respectively, $\mathbb{F}_2$), as our simulations show. In addition, we also observe that in the case where improvement in performance has been obtained, increasing the proportion of code symbols from the larger alphabet does not necessarily lead to further improvements in performance. This is similar to the observation made by

Davey and MacKay [11] that codes defined over larger alphabets do not necessarily outperform codes defined over smaller alphabets.

However, as we increase the block length of the mixed codes and the non-zero elements of $\mathbf{H}_E$ are drawn from the non-zero elements of $\mathbb{F}_{16}$, we noticed that our $\mathbb{F}_4$ code extended with $\mathbb{F}_{16}$ parity-check symbols can outperform its single-alphabet counterpart over $\mathbb{F}_4$, as our simulations show. We also observe that as we increased the value of N_2 from 5 to 25, there was a corresponding increase in performance, with the mixed code of $N_{\text{bin}} = 6000$ and $N_2 = 25$ giving us an additional coding gain of 0.065 dB over the $\mathbb{F}_4$ code of the same N_{bin} and rate; whilst the additional coding gain obtained by the $\mathbb{F}_8$ code over its $\mathbb{F}_4$ counterpart is 0.125 dB. This is more consistent with our intuitive predictions that as one increases the number of parity-check coordinates over the extended alphabet, one will see corresponding improvement in performance.

Chapter 5

Multistage Decoding of LDPC Codes over $\mathbb{Z}_q$

5.1 Background

In [38], Richardson and Urbanke showed that increasing the alphabet size of the LDPC code leads to a corresponding improvement in BER performance. Davey and MacKay showed via computer simulations that well-designed LDPC codes over GF(4) and GF(8) outperformed their binary counterparts over the AWGN channel. Deepak and Sridhara studied the construction of LDPC codes over $\mathbb{Z}_q$ matched to signal sets in [43]. However, their focus was more on construction of LDPC codes over groups and rings for coded modulation. Erez and Miller also made a study of LDPC codes over $\mathbb{Z}_q$ in [14]. Their main concern on the code construction and their corresponding maximum likelihood performance.

Motivated by the improved error-correcting capability of alternant codes over $\mathbb{Z}_{2^m}$ under multistage list decoding approach of [1], we develop a multistage message-passing analogue for LDPC codes over $\mathbb{Z}_q$. We achieve this by exploiting the underlying algebraic structure (natural ring epimorphism) of linear codes over $\mathbb{Z}_q$.

We proceed with a brief review of the properties of linear codes over $\mathbb{Z}_q$.

5.2 Structure of Linear Codes over Rings

An integer ring $\mathbb{Z}_q$ does not possess the properties of the finite field. The characteristics of a linear code C over $\mathbb{Z}_q$ needs to be defined properly. The following theorem by [7] helps to shed some light on the properties of such a code.

Theorem 5.2.1 *Let $\mathbb{Z}_q$ be the ring of integers modulo q. More precisely, $\mathbb{Z}_q$ is the quotient ring $\mathbb{Z}/q\mathbb{Z}$ and let $C \subset (\mathbb{Z}_q)^N$. The following statements are equivalent:*

1. *C is a subgroup of $(\mathbb{Z}_q)^N$.*
2. *There exists an integer R $(0 \leq R \leq N)$, a set of linearly independent vectors $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{R-1}\} \subset (\mathbb{Z}_q)^N$, and a set of nested ideals of $\mathbb{Z}_q$ (not necessarily distinct)*

$$\mathbb{Z}_q > a_0\mathbb{Z}_q > a_1\mathbb{Z}_q > \ldots > a_{R-1}\mathbb{Z}_q > \{0\}$$

such that C can be written as the direct sum

$$C = \bigoplus_{i=0}^{R-1} a_i \mathbb{Z}_q \mathbf{x}_i. \tag{5.1}$$

3. *There exists a unique lattice Λ, $q\mathbb{Z}^N < \Lambda < \mathbb{Z}^N$, such that $C \simeq \Lambda/q\mathbb{Z}^N$. Given any set $[\Lambda/q\mathbb{Z}^N]$ of coset representatives, C can be written as $C = [\Lambda/q\mathbb{Z}^N] mod q$.*

The set $(\mathbb{Z}_q)^N$ of N-tuples from $\mathbb{Z}_q$ is a $\mathbb{Z}_q$-module, and a linear code over $\mathbb{Z}_q$ means any $\mathbb{Z}_q$ sub-module of $(\mathbb{Z}_q)^N$. We equip $(\mathbb{Z}_q)^N$ with the inner product $v \cdot w = v_1 w_1 + \ldots + v_N w_N$ evaluated in $\mathbb{Z}_q$, and define the dual and self-dual codes in the usual way.

If we write $q = p^m$, for any prime p and positive integer m, we have the ring $\mathbb{Z}_{p^m}$, which is the quotient ring $\mathbb{Z}/p^m\mathbb{Z}$. An element $r \in \mathbb{Z}_{p^m}$ may be expressed in its p-adic form as sum $r = r^{(0)} + r^{(1)}p + r^{(2)}p^2 + \cdots + r^{(m-1)}p^{m-1} = \sum_{i=0}^{m-1} r^{(i)}p^i$, where $r^{(i)} \in \mathbb{Z}_p$.

A non-zero linear code C over $\mathbb{Z}_{p^m}$, for m finite, has a generator matrix of the form [8]:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{K_0} & \mathbf{A}_{01} & \mathbf{A}_{02} & \cdots & \mathbf{A}_{0,m-1} & \mathbf{A}_{0m} \\ \mathbf{0} & p\mathbf{I}_{K_1} & p\mathbf{A}_{12} & \cdots & p\mathbf{A}_{1,m-1} & p\mathbf{A}_{1m} \\ \mathbf{0} & \mathbf{0} & p^2\mathbf{I}_{K_2} & \cdots & p^2\mathbf{A}_{2,m-1} & p^2\mathbf{A}_{2m} \\ . & . & . & \cdots & . & . \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & p^{m-1}\mathbf{I}_{K_{m-1}} & p^{m-1}\mathbf{A}_{m-1,m} \end{bmatrix},$$

where the columns are grouped into blocks of sizes K_0, $K_1, \ldots, K_{m-1}$, K_m and the K_i are nonnegative integers summing to N, and $\mathbf{I}_{K_i}$ is the $K_i \times K_i$ identity matrix. The code C then consists of all codewords $[\mathbf{v}_0\, \mathbf{v}_1\, \mathbf{v}_2\, \ldots\, \mathbf{v}_{m-1}]\mathbf{G}$, where each $\mathbf{v}_i$ is a vector of length K_i with components from $\mathbb{Z}_{p^{m-i}}$, so that C contains p^K codewords, where $K = \sum_{i=0}^{m-1}(m-i)K_i$.

The code C with generator $\mathbf{G}$ is said to have a dual code $C^\perp$ with generator matrix of the form

$$\mathbf{G}^\perp = \begin{bmatrix} \mathbf{B}_{0m} & \mathbf{B}_{0,m-1} & \cdots & \mathbf{B}_{02} & \mathbf{B}_{01} & \mathbf{I}_{K_m} \\ p\mathbf{B}_{1m} & p\mathbf{B}_{1,m-1} & \cdots & p\mathbf{B}_{12} & p\mathbf{I}_{K_{m-1}} & \mathbf{0} \\ p^2\mathbf{B}_{2m} & p^2\mathbf{B}_{2,m-1} & \cdots & p^2\mathbf{I}_{K_{m-2}} & \mathbf{0} & \mathbf{0} \\ . & . & \cdots & . & . & . \\ p^{m-1}\mathbf{B}_{m-1,m} & p^{m-1}\mathbf{I}_{K_1} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

where the column block sizes are the same as in $\mathbf{G}$. In analogy with linear codes over fields, we define the parity-check matrix of C as $\mathbf{H} = (\mathbf{G}^\perp)^T$.

A code C over $\mathbb{Z}_{p^m}$ is a $\mathbb{Z}_{p^m}$ free-module if $K_1 = K_2 = \ldots = K_{m-1} = 0$. In this case, a free code over $\mathbb{Z}_{p^m}$ has a systematic generator matrix.

5.2.1 Epimorphism of elements in $\mathbb{Z}_q$

We let $R_l = \mathbb{Z}_{p^{l+1}}$ for $0 \leq l < m$ such that $p^m = q$. Thus, we have $R_0 = \mathrm{GF}(p)$ (or $\mathbb{F}_p$) and $R_{m-1} = \mathbb{Z}_q$. An element $r \in \mathbb{Z}_q$ has a unique p-adic expansion and we can define the ring epimorphism $\mathbb{Z}_q \mapsto \mathbb{Z}_{p^{l+1}} : r \mapsto \sum_{i=0}^{l} r^{(i)} p^i$ with kernel $p^l \mathbb{Z}_q$ for each l. Similarly, we can denote a code over $\mathbb{Z}_{p^m}$ as C_{m-1} and its parity-check matrix, $\mathbf{H}_{m-1}$.

We can then extend this notation to C_{m-1} and $\mathbf{H}_{m-1}$ in a natural way and obtain C_l and $\mathbf{H}_l$, where C_l is an LDPC code over R_l with parity-check matrix $\mathbf{H}_l$. Note that C_l has the same dimension as C_{m-1}.

For our code construction, a (N, K) LDPC code over R_{m-1} can be designed by constructing a sparse $(N-K) \times N$ parity-check matrix $\mathbf{H}_{m-1}$, with non-zero elements drawn from the units of R_{m-1}. The $(N-K)$ rows of $\mathbf{H}_{m-1}$ are linearly independent and C_{m-1} has dimension K. Since the entries of $\mathbf{H}_{m-1}$ are units of R_{m-1}, the bipartite graph of C_l is identical to that of C_{m-1} except that its edge weights are given by the corresponding edge weights of the latter graph reduced modulo p^{l+1}.

Example 4 *An* $(8, 4)$ *code* C_2 *over* R_2 *i.e.,* $\mathbb{Z}_8$*, has parity-check matrix*

$$\mathbf{H}_2 = \begin{bmatrix} 0 & 0 & 3 & 1 & 1 & 7 & 0 & 0 \\ 3 & 5 & 0 & 0 & 0 & 1 & 7 & 0 \\ 0 & 7 & 7 & 1 & 0 & 0 & 1 & 7 \\ 5 & 0 & 0 & 0 & 7 & 1 & 0 & 1 \end{bmatrix}.$$

We reduce the elements of $\mathbf{H}_2$ *modulo 4 and obtain*

$$\mathbf{H}_1 = \begin{bmatrix} 0 & 0 & 3 & 1 & 1 & 3 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 1 & 3 & 0 \\ 0 & 3 & 3 & 1 & 0 & 0 & 1 & 3 \\ 1 & 0 & 0 & 0 & 3 & 1 & 0 & 1 \end{bmatrix},$$

where $\mathbf{H}_1$ *is the parity-check matrix of the* $(8, 4)$ *image code* C_1 *over* R_1.

We further reduce the elements of $\mathbf{H}_2$ *modulo 2 and obtain*

$$\mathbf{H}_0 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

where $\mathbf{H}_0$ *is binary image of* $\mathbf{H}_2$ *and is also the parity-check matrix of the* $(8, 4)$ *image code* C_0 *defined over* R_0.

5.3 MPA for LDPC codes over $\mathbb{Z}_q$

We decode LDPC codes over $\mathbb{Z}_q$ using the MPA as well. Similar to the case of LDPC codes over $\mathbb{F}_q$ illustrated in Chapter 3, the decoder input is the set of *a priori* probabilities of the code nodes and is expressed as column vectors $\mathbf{p}_j = [p_j^0\, p_j^1\, \ldots\, p_j^{q-1}]^T$ where p_j^ξ is the probability that the jth code node is equal to $\xi \in \mathbb{Z}_q$.

Column operations now involve the computation of the code-to-check node messages $\mathbf{q}_{ji} = [q_{ji}^0\, q_{ji}^1\, \ldots\, q_{ji}^{q-1}]^T$ where q_{ji}^ξ is the probability that the jth code node, x_j, has the value ξ, given the information obtained via the i'th check node, $c_{i'}$ for all $i' \in M(j) \setminus \{i\}$ where $M(j) = \{i : h_{i,j} \neq 0\}$.

On the other hand, row operations involve the computation of check-to-code node messages, $\mathbf{r}_{ji} = [r_{ji}^0\, r_{ji}^1\, \ldots\, r_{ji}^{q-1}]^T$ where r_{ji}^ξ is the probability that the ith check node, c_i, is satisfied when the value of the x_j is fixed to be equal to ξ, i.e.,

$$\sum_{j' \in L(i') \setminus \{j\}} h_{i',j'} x_{j'} - x_j = 0$$

and the other code nodes are independent with probabilities given by $q_{j'i}^\xi$ for $j' \in L(i) \setminus \{j\}$ where $L(i) = \{j : h_{i,j} \neq 0\}$.

Prior to the computing of the $\mathbf{q}_{ji}$ (respectively, $\mathbf{r}_{ji}$), the elements of $\mathbf{r}_{ji'}$ (respectively, $\mathbf{q}_{j'i}$) are permuted according to $h_{i',j}^{-1}$ (respectively, $h_{i,j'}$) such that the resulting vector has the form $\mathbf{r}'_{ji'} = [r_{ji'}^0 \, r_{ji'}^{h_{i',j}} \ldots r_{ji'}^{h_{i',j}(q-1)}]^T$ (respectively, $\mathbf{q}'_{j'i} = [q_{j'i}^0 \, q_{j'i}^{h_{i,j'}^{-1}} \ldots q_{j'i}^{h_{i,j'}^{-1}(q-1)}]^T$). Note that if $h_{i',j}$ (respectively, $h_{i,j'}$) is equal to the multiplicative identity of $\mathbb{Z}_q$, then $\mathbf{r}'_{ji'} = \mathbf{r}_{ji'}$ (respectively, $\mathbf{q}'_{j'i} = \mathbf{q}_{j'i}$). We initialise each $\mathbf{r}_{ji} = q^{-1}[1 \, 1 \ldots 1]^T$.

5.3.1 The Column Step

Consider the column step corresponding to the code node x_j, we consider $c_{i'}$ for all $i' \in M(j) \setminus \{i\}$, which is in turn connected to the set of code nodes defined by $L(i')$. The constraint defined by $c_{i'}$ is expressed as

$$\sum_{j' \in L(i') \setminus \{j\}} h_{i',j'} x_{j'} + h_{i',j} x_j = c_{i'}, \tag{5.2}$$

where additions and multiplications are defined over $\mathbb{Z}_q$. For a check node constraint to be satisfied, $c_{i'} = 0$ and we can rewrite (5.2) as

$$\sum_{j' \in L(i') \setminus \{j\}} h_{i',j'} x_{j'} - (-h_{i',j} x_j) = 0. \tag{5.3}$$

From (5.3), we have the simple equality $\sum_{j' \in L(i') \setminus \{j\}} h_{i',j'} x_{j'} = -h_{i',j} x_j$. This means that if x_j assumes a particular value ξ, i.e., if $x_j = \xi$, then $\sum_{j' \in L(i') \setminus \{j\}} h_{i',j'} x_{j'} = -h_{i',j} \xi$. The combined information converging upon x_j from all the check nodes $c_{i'}$ for all $i' \in M(j) \setminus \{i\}$ has to be equivalent to $-h_{i',j} x_j$. Hence, updating $\mathbf{q}_{ji}$ can be accomplished by the following expression

$$q_{ji}^{\xi} = \gamma p_j^{\xi} \prod_{i' \in M(j) \setminus \{i\}} r_{ji'}^{-h_{i',j}\xi}.$$

In the case of a code defined over a field of characteristic two, i.e. $\mathbb{F}_{2^m}$ such that $\xi \in \mathbb{F}_{2^m}$, then $\xi = -\xi$ so that $\mathbf{q}_{ji}$ is updated according to the expression as given

in (3.11) in Chapter 3. For the case of codes over $\mathbb{Z}_q$, $\xi \neq -\xi$, so we update $\mathbf{q}_{ji}$ according to the relation

$$\mathbf{q}_{ji} = \gamma \mathbf{p}_j \prod_{i' \in M(j) \backslash \{i\}} \mathbf{w}_{ji'} \tag{5.4}$$

where $\mathbf{w}_{ji'} = [r_{ji'}^0 \, r_{ji'}^{h_{i',j}(q-1)} \, r_{ji'}^{h_{i',j}(q-2)} \, \ldots \, r_{ji'}^{h_{i',j}}]^T$. If $h_{i',j}$ is equal to the multiplicative identity of $\mathbb{Z}_q$, then $\mathbf{w}_{ji'} = [r_{ji'}^0 \, r_{ji'}^{(q-1)} \, r_{ji'}^{(q-2)} \, \ldots \, r_{ji'}^1]^T$.

5.3.2 The Row Step

For the row step corresponding to the ith check node, the analysis given in Section 3.2.1 of Chapter 3 is still applicable where all variables are now defined over $\mathbb{Z}_q$, i.e. in (3.2), all ξ_j and $\xi_k \in \mathbb{Z}_q$. Likewise, the Fourier transform decoding illustrated in Chapter 3 can be used to compute the check-to-code node messages for codes over $\mathbb{Z}_q$ with the use of a suitable transform operator.

For $\mathbb{Z}_q$, the appropriate transform is the $q \times q$ discrete Fourier transform operator, denoted as $\mathbf{F}_q = [f_{ab}] = \left[\exp\left(\frac{2\pi j}{q} ab\right)\right]$ for $0 \leq a, b \leq q-1$ and $j = \sqrt{-1}$. For a code over $\mathbb{Z}_2$, the Fourier transform operator is $\mathbf{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ which is the 2×2 Hadamard matrix $\mathbf{H}_2$, since $\mathbb{Z}_2 = \mathbb{F}_2$. The Fourier transform operator for $\mathbb{Z}_4$ is

$$\mathbf{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}.$$

Note however that the Fourier transform technique works only when the non-zero element of the parity-check matrix has a well-defined multiplicative inverse. This is because when the $\mathbf{r}_{ji}$ is passed through a non-unity edge $h_{i',j}$, the elements are rearranged according to $h_{i',j}^{-1}$.

Hence the transformed version $\mathbf{R}_{ji}$ of the message $\mathbf{r}_{ji}$ to be passed from c_i to x_j is written as

$$\mathbf{R}_{ji} = \prod_{j' \in L(i) \setminus \{j\}} \mathbf{F}_q \mathbf{q}'_{j'i}. \tag{5.5}$$

To obtain $\mathbf{r}_{ji}$, one simply performs the inverse transform

$$\mathbf{r}_{ji} = \mathbf{F}_q^{-1} \mathbf{R}_{ji}. \tag{5.6}$$

Finally, an estimate $\hat{x}_j$ of the jth code symbol is given by $\hat{x}_j = \arg\max_{0 \leq \xi \leq q-1} \{q_j^\xi\}$ where the q_j^ξ are the estimated *a posteriori* probabilities of the code nodes, which is computed using the following expression

$$[q_j^0 \, q_j^1 \, \ldots \, q_j^{q-1}]^T = \gamma \mathbf{p}_j \prod_{i \in M(j)} \mathbf{w}_{ji}. \tag{5.7}$$

One then checks if the codeword estimate $\hat{\mathbf{x}}$ found is in the null space of $\mathbf{H}$, i.e., $\hat{\mathbf{x}}\mathbf{H}^T = \mathbf{0}$. If so, the decoding process is terminated, otherwise, the above steps are repeated. If the algorithm completes a prescribed maximum number of iterations and no valid codeword has been found, a decoding failure is declared. Note that $\mathbf{r}'_{ji}$ replaces $\mathbf{w}_{ji}$ in (5.7) if the code alphabet is $\mathbb{F}_{2^m}$.

5.4 m-Stage Message Passing Decoding

In this section of the thesis, we present the multi-stage message passing algorithm to decode LDPC codes over $\mathbb{Z}_q$. The decoder works by decoding the code C_{m-1} over R_{m-1} in m stages, by decoding each homomorphic image of the code, C_l over R_l, for $l = 0$, $l = 1$ until $l = m - 1$. Since this decoding algorithm entails decoding at m stages, the lth stage will have its own set of check-to-code node and code-to-check node messages, denoted as $\mathbf{r}_{ji}(l) = [r_{ji}^0(l) \, r_{ji}^1(l) \, \ldots \, r_{ji}^{p^{l+1}-1}(l)]^T$ and $\mathbf{q}_{ji} = [q_{ji}^0(l) \, q_{ji}^1(l) \, \ldots \, q_{ji}^{p^{l+1}-1}(l)]^T$, respectively, where both $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$ are p^{l+1} element

column vectors. It will also have its own *a priori* probability inputs, denoted as $\mathbf{p}_j(l) = [p_j^0(l)\, p_j^1(l)\, \ldots\, p_j^{p^{l+1}-1}(l)]^T$.

At the lth stage, we initialise the values of the check-to-code node messages $\mathbf{r}_{ji}(l)$ as well as the *a priori* probability messages $\mathbf{p}_j(l)$ of the code nodes associated with C_l using $\mathbf{r}_{ji}(l-1)$, a refinement of the *a priori* probability messages $\mathbf{p}_j$ of the code nodes associated with C_{m-1} and the estimated codeword $\hat{\mathbf{x}}(l-1)$ found at the $(l-1)$th stage.

When $l = 0$, $\mathbf{r}_{ji}(0)$ is initialised as a uniform distribution, i.e $\mathbf{r}_{ji}(0) = p^{-1}[1\, 1\, \ldots\, 1]^T$ while $\mathbf{p}_j(0)$ is obtained according to the initial *a priori* probabilities $\mathbf{p}_j$ obtained from the channel output. The lth stage decoding is successful if a valid estimated codeword is obtained, i.e. $\hat{\mathbf{x}}(l)\mathbf{H}_l^T = \mathbf{0}$. We can make use of the information obtained from this stage to update the $\mathbf{p}_j$ and this is done with the use of an appropriate scaling factor β, where $0 < \beta < 1$. Assuming that the jth estimated code symbol is ξ, i.e., $\hat{x}_j(l) = \xi \in R_l$ and $\Xi \in R_{l+1}$ such that $R_{l+1} \mapsto R_l : \Xi \mapsto \sum_{i=0}^{l-1} \Xi^{(i)} p^i \neq \xi$, we update $\mathbf{p}_j$ such that $p_j^{\Xi} = \beta p_j^{\Xi}$. (Note that $\beta = 0.5$ is used in our simulations as it gave the best simulation results for the various β values used - refer to Figure 5.6.) The resulting $\mathbf{p}_j$ is then normalised and this serves as a refinement of the *a priori* probabilities of the code nodes of C_{m-1}. We can then compute the *a priori* probability input for the next stage $\mathbf{p}_j(l+1)$ from this refined $\mathbf{p}_j$. The expression below shows how $\mathbf{p}_j(l+1)$ is computed from $\mathbf{p}_j$, for each $\xi \in R_{l+1}$,

$$p_j^{\xi}(l+1) = \sum_{\Xi \in R_{m-1}, \sum_{i=0}^{l} \Xi^{(i)} p^i = \xi} p_j^{\Xi}\ ^1. \tag{5.8}$$

To initialise the $\mathbf{r}_{ji}(l)$ from the preceding stages, for $0 < l < m$ and for all $\Xi \in R_l$

[1] An underlying assumption is that $\hat{\mathbf{x}}(l)$ and the transmitted codeword are the same after reduction by p^{l+1}. Otherwise, we will be degrading rather than improving $\mathbf{p}_j$. The idea of the scaling factor β and (5.8) were contributed and derived, respectively, by Dr. M. A. Armand. The author gives his thanks and appreciation to Dr. M. A. Armand for allowing him to use his work in this thesis.

such that $\sum_{i=0}^{l-1} \Xi^{(i)} p^i = \xi \in R_{l-1}$, we set

$$r_{ji}^{\Xi}(l) = \gamma r_{ji}^{\xi}(l-1). \tag{5.9}$$

The main steps of the multistage message-passing algorithm may be summarised as follows:

- Step 0. *Initialising Stage 0.* Set $l = 0$. The intrinsic probability vector $\mathbf{p}_j$ and $\mathbf{r}_{ji}$ are initialised. The $\mathbf{p}_j$ is initialised based on the given channel output and $\mathbf{p}_j(0) = \sum_{\Xi \in R_{m-1}, \Xi^{(0)} p^0 = \xi} p_j^{\Xi}$ for $\xi \in R_0$, while $\mathbf{r}_{ji}(0)$ is initialised as a uniform distribution, i.e. $\mathbf{r}_{ji}(0) = p^{-1}[1\,1 \ldots 1]^T$. Proceed to Step 1.

- Step 1. *Decoding at Stage l.* Using the modified MPA described in Section 5.3, we compute a codeword estimate $\hat{\mathbf{x}}(l)$ of the image code C_l over R_l. If $l = m-1$, exit; else, proceed to Step 2.

- Step 2. *Initialising Stage $l+1$.* If $\hat{\mathbf{x}}(l)\mathbf{H}_l^T = \mathbf{0}$, then update $\mathbf{p}_j$, initialise $\mathbf{r}_{ji}(l+1)$ and compute $\mathbf{p}_j(l+1)$ as described above. Then set $l = l+1$ and return to Step 1. If $\hat{\mathbf{x}}(l)\mathbf{H}_l^T \neq \mathbf{0}$, then initialise $\mathbf{r}_{ji}(m-1)$ in the following fashion: for all $\Xi \in R_{m-1}$ such that $\sum_{i=0}^{l} \Xi^{(i)} p^i = \xi \in R_l$, set $r_{ji}^{\Xi}(m-1) = \gamma r_{ji}^{\xi}(l)$. Then set $l = m-1$ and return to Step 1.

For the case of p-ary modulation, we are then transmitting the coefficients of the p-adic expansion of the code symbols. Such a modulation requires m uses of the channel for the transmission of one symbol over R_{m-1}. In such an instance, it is obvious that $\mathbf{p}_j(l)$ can be computed from the first $l+1$ channel outputs corresponding to each code symbol, without the need for $\mathbf{p}_j$. (Note that we have assumed that the transmission of the code symbol begins with the zeroth coefficient of the p-adic expansion, followed by the first and so on.) In this case, the multistage approach can be simplified. The simplifications are straightforward and are thus omitted.

5.5 Complexity Analysis

In this section we consider the increase in decoding complexity of the multistage approach over the single-stage counterpart. The increase in complexity involves two components: the fixed components and the variable components.

The fixed components are the additional arithmetic operations incurred at the beginning and the end of the decoding stages, when we update the $\mathbf{p}_j$ at the end of stage l, the initialisation of $\mathbf{r}_{ji}(l)$ based on $\mathbf{r}_{ji}(l-1)$, and initialisation of $\mathbf{p}_j(l+1)$ based on $\mathbf{p}_j$.

On the other hand, the variable components deal with the arithmetic operations required to perform the row and column computations to decode the homomorphic images C_l for $l = 0, 1, \ldots, m-2$. We consider this to be a variable component since the total number of operations depends on the number of iterations required to perform the decoding.

5.5.1 Fixed Components

We begin by first considering the number of multiplications required to update $\mathbf{p}_j$ at the end of stage l, for $l = 0, 1, \ldots, m-2$ for an (N, K) code defined over $\mathbb{Z}_q$ where $q = p^m$ for p a prime. At the lth stage, we would require $Np^m\left(1 - p^{-(l+1)}\right)$ multiplications by the scaling factor, β and an additional p^m multiplications for normalisation purposes to obtain the refined $\mathbf{p}_j$. Hence, over all the relevant $m-1$ stages (starting from stage 0), we require a total of

$$Np^m \sum_{l=0}^{m-2} \left(1 - p^{-(l+1)}\right) \tag{5.10}$$

multiplications to initialise $\mathbf{p}_j$ over all the relevant stages.

Next, we consider the number of multiplications required to initialise $\mathbf{r}_{ji}(l)$ based on $\mathbf{r}_{ji}(l-1)$. At the end of the $(l-1)$th stage, the check-to-code node messages $\mathbf{r}_{ji}(l-1)$ are column vectors of length p^l, while the check-to-code node messages of the next stage $\mathbf{r}_{ji}(l)$ are column vectors of length p^{l+1}. The relationship between $\mathbf{r}_{ji}(l)$ and $\mathbf{r}_{ji}(l-1)$ has already been defined in (5.9). It is then easy to see that the number of multiplications required to initialise one $\mathbf{r}_{ji}(l)$ from its corresponding $\mathbf{r}_{ji}(l-1)$ is the p^{l+1} multiplications required for normalisation. Thus, the total number of multiplications required to initialise all the check-to-code node messages over all $m-1$ stages (starting from stage 0) is

$$\sum_{l=0}^{m-2} \sum_{i=0}^{N-K-1} p^{l+2}|L(i)|. \tag{5.11}$$

Finally, we consider the number of additions required to initialise $\mathbf{p}_j(l+1)$ from $\mathbf{p}_j$ from the 0th stage to the $(m-1)$th stage. At the lth stage, we require $p^m - p^{(l+1)}$ additions to compute one $\mathbf{p}_j(l)$. Subsequently, the total number of additions required to compute all N $\mathbf{p}_j(l)$ over $m-1$ stages (starting from stage 0) is

$$N \sum_{l=0}^{m-2} p^{(l+1)}(p^{m-(l+1)} - 1). \tag{5.12}$$

5.5.2 Variable Components

For the variable components, we first consider the complexity of implementing the q-point FFT/IFFT pair based on the $q \times q$ Fourier transform operator $\mathbf{F}_q = [f_{ab}] = \left[\exp\left(\frac{j2\pi}{q}ab\right)\right]$ where $0 \leq a \leq q-1$, $0 \leq b \leq q-1$, $j = \sqrt{-1}$ and $p = 2$. Note that since $\mathbf{F}_q$ has complex elements in it, complex arithmetic operations will be required to perform the FFT/IFFT operations. The FFT/IFFT pair will require $2q \log_2 q$ complex additions and 6 and 18 complex multiplications for $q = 4$ and $q = 8$, respectively.

Using the forward-backward algorithm to evaluate all the $\mathbf{r}_{ji}$ of a check node i with $|L(i)|$ participating code nodes requires $3|L(i)|q$ complex multiplications.

Thus, for a (N, K) code over $\mathbb{Z}_4$ (respectively, $\mathbb{Z}_8$), the total number of complex multiplications required per iteration to update all the $\mathbf{r}_{ji}$ is $18\sum_{i=0}^{N-K-1}|L(i)|$ (respectively, $42\sum_{i=0}^{N-K-1}|L(i)|$). On the other hand, the number of real multiplications required to perform the column operations remains as $4q\sum_{j=0}^{N-1}|M(j)|$.

5.6 2^m-ary Signal Space

To perform multistage message-passing algorithm with q-ary modulation, we propose a method of partitioning the signal space, specifically, we focus on 2^m-ary PSK modulation.

In a multistage decoding environment, errors from previous stages can propagate through the subsequent stages and lead to decoder errors. To reduce the occurrence of such errors to as low a level as possible, we want to maximise the probability that when a decision is made about the value of a code node over C_l for $l < m-1$, it is the same value as the transmitted code symbol of the corresponding code node of C_{m-1} modulo p^{l+1}. The signal space will be partitioned such that the signal points corresponding to $\Xi \in R_{m-1}$ mapping to the same values via the ring epimorphism $\sum_{i=0}^{l-1}\Xi^{(i)}p^i = \xi \in R_{l-1}$ are grouped together.

Consider the case of $\mathbb{Z}_4$, the constellation diagram is shown in Figure 5.1(a). We can see that the elements of 0 and 2 of $\mathbb{Z}_4$ occupy the upper half while the remaining elements, 1 and 3 of $\mathbb{Z}_4$ occupy the lower half of the constellation. This arrangement allows 0 and 2 (1 and 3) to be paired together since their binary images coincide modulo 2. From the constellation, we see that each element is represented by a 2 element binary vector, where each element of the vector represents the coefficient of the 2-adic expansion of the corresponding element of $\mathbb{Z}_4$. With the arrangement of the elements in Figure 5.1(a), we have a Gray-coded constellation. We can extend this grouping of signal sets for other alphabets $\mathbb{Z}_{2^m}$.

Figure 5.1(c) shows the proposed constellation diagram for $\mathbb{Z}_8$. We can obtain this from the constellation over $\mathbb{Z}_4$ in Figure 5.1(a) by reinterpreting the elements as elements of $\mathbb{Z}_8$. In this case, we have the elements 4 and 6 in the upper half of the constellation since 0, 2, 4 and 6 map to the binary image of 0 modulo 2, while the elements 5 and 7 are in the lower half of the constellation since 1, 3, 5 and 7 map to 1 modulo 2. 6 lies in the upper left quadrant with 2 since they map to the same element (i.e., 2) modulo 4. Likewise ,4 lies in the same region as 0 in the upper right quadrant because they coincide to the same element (i.e., 0) modulo 4. 7 and 5 lies in the same quadrants as 1 and 3 respectively, due to similar reasons.

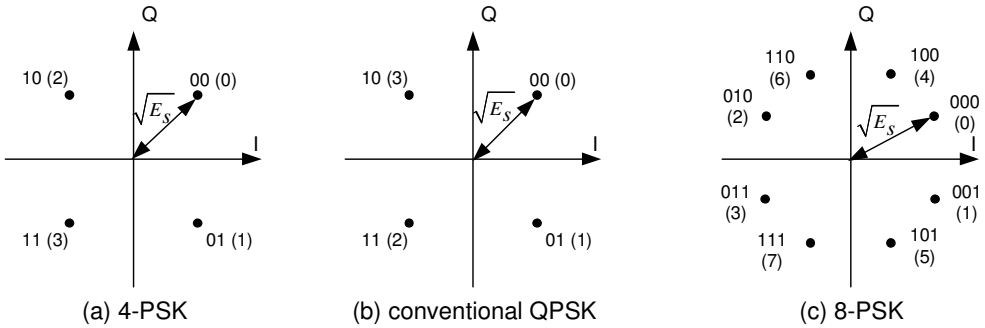

Figure 5.1: Constellation diagrams for 4-PSK and 8-PSK

For both the proposed and conventional 4PSK constellations of Figures 5.1(a) and 5.1(b), respectively, without loss of generality, if the signal $\mathbf{s}$ corresponding to the jth code symbol x_j of code C_1 is 01, the symbol transmitted is 1. The transmission here is performed over the AWGN channel with single-sided noise power spectral density N_0 and symbol energy E_s. The probability that the signal falls within the decision region of 01 is $\left[1 - Q(\sqrt{\frac{E_s}{N_0}})\right]^2$, where $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-\frac{y^2}{2}) dy$. In this case, $\mathbf{p}_j = [p_j^0\, p_j^1\, p_j^2\, p_j^3]^T$ is computed from the channel output and is reliable.

The probability of the received signal falling within the decision region of 00 and 11 is $\left[1 - Q(\sqrt{\frac{E_s}{N_0}})\right] Q(\sqrt{\frac{E_s}{N_0}})$. For the case where the signal falls into the decision region of 00, for the proposed 4-PSK constellation, we have the situation that $p_j^0(0) > p_j^1(0)$, where $p_j^0(0) = p_j^0 + p_j^2$ and $p_j^1(0) = p_j^1 + p_j^3$. In this scenario, the probability

input $\mathbf{p}_j(0) = [p_j^0(0)\, p_j^1(0)]^T$ to the zeroth stage decoder is not reliable. This is the same if we were to use the conventional 4-PSK constellation.

For the case of the received signal falling within the decision region of 11; if the proposed 4-PSK constellation is used, we have $p_j^0(0) < p_j^1(0)$ and thus, $\mathbf{p}_j(0)$ would be reliable in this case. If the conventional 4-PSK constellation were used, the situation where $p_j^0(0) > p_j^1(0)$ would still be encountered, and the $\mathbf{p}_j(0)$ obtained would remain unreliable.

Finally we consider the case where the received signal falls into the decision region of 10. The probability that the received signal falls into the decision region of 10 is $Q(\sqrt{\frac{E_s}{N_0}})^2$. Hence, $\mathbf{p}_j(0)$ will be unreliable if the proposed constellation is used. On the other hand, $\mathbf{p}_j(0)$ obtained from that of the conventional 4-PSK constellation would be reliable. It is important to note however, that the probability of the received signal falling into the decision region of 10 is much smaller than that of the received signal falling into the decision region of either 00 or 11 (since $\left[1 - Q(\sqrt{\frac{E_s}{N_0}})\right] Q(\sqrt{\frac{E_s}{N_0}}) \gg Q(\sqrt{\frac{E_s}{N_0}})^2$). Hence the received signal is more likely to fall into the decision region of either 11 or 00 than 10.

Hence, we can see that the proposed constellation provides a more reliable $\mathbf{p}_j(0)$ (compared to the conventional 4-PSK constellation) which in turn reduces the probability of decoding error at the zeroth decoding stage. This reduction in decoding error at the zeroth decoding stage in turn reduces the overall likelihood of decoding error for the two stage decoder since errors propagating from the zeroth stage to the first stage are suppressed.

This analysis can be extended to larger sized integer rings.

5.7 Worked Example

In this section, we use a worked example to illustrate the concepts and implementation of the proposed multi-stage decoding algorithm to decode LDPC codes over $\mathbb{Z}_{p^m}$.

The corresponding systematic generator matrix to the parity-check matrix $\mathbf{H}_2$ in Example 4 is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 1 & 0 & 0 & 4 & 4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 & 6 & 5 \\ 0 & 0 & 0 & 1 & 5 & 6 & 6 & 7 \end{bmatrix}.$$

Assuming that a length 4 message vector $\mathbf{m} = [7\,1\,4\,3]$ is encoded to a length 8 codeword vector $\mathbf{x} = [7\,1\,4\,3\,7\,6\,0\,6]$. Each element of the codeword is then mapped to its corresponding signal according to a $-\frac{\pi}{8}$-shifted version of the 8-PSK constellation diagram in Figure 5.1(c) and transmitted across the AWGN channel. For this example, we assume that SNR = 4dB and $E_b = 1$. Note that the code is of rate one-half. The transmitted signal is

$$\mathbf{s} = \sqrt{\frac{3}{2}} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -1 & -\frac{1}{\sqrt{2}} & 0 & 1 & 0 \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 1 & 0 & 1 \end{bmatrix}.$$

The received value from the AWGN channel is

$$\mathbf{y} = \begin{bmatrix} -0.5021 & 1.4417 & 1.3973 & -1.2336 & -1.5817 & -0.4713 & 0.8656 & 0.0979 \\ -0.5485 & -0.5677 & 0.3295 & -0.699 & -0.7512 & 1.8561 & 0.2359 & 0.8134 \end{bmatrix}.$$

The probability of each symbol being one of the eight possible states is calculated using the channel output and is tabulated in Table 5.1. (Please refer to Appendix A for subsequent tables of $\mathbf{p}_j(0)$, $\mathbf{p}_j(1)$ as well as the refined $\mathbf{p}_j$.)

Using this set of $\mathbf{p}_j$ as input to the stage 2 decoder and decoding over R_2, we find that the decoder is unable to give a correct codeword estimate after 50 iterations.

j	$\xi=0$	$\xi=1$	$\xi=2$	$\xi=3$	$\xi=4$	$\xi=5$	$\xi=6$	$\xi=7$	Hard Decision Value	Result
0	0.0003	0.0082	0.0055	0.1465	0.0001	0.1948	0.0002	0.6444	7	Correct
1	0.5289	0.4653	0.0000	0.0000	0.0033	0.0024	0.0000	0.0000	0	Error
2	0.7360	0.0141	0.0000	0.0000	0.2488	0.0000	0.0010	0.0000	0	Error
3	0.0000	0.0000	0.0651	0.8145	0.0000	0.0006	0.0003	0.1195	3	Correct
4	0.0000	0.0000	0.0009	0.3957	0.0000	0.0024	0.0000	0.6010	7	Correct
5	0.0000	0.0000	0.2143	0.0002	0.0035	0.0000	0.7820	0.0000	6	Correct
6	0.5938	0.0447	0.0002	0.0000	0.3484	0.0007	0.0123	0.0000	0	Correct
7	0.0081	0.0002	0.0992	0.0024	0.2324	0.0000	0.6576	0.0000	6	Correct

Table 5.1: Intrinsic symbol probabilities $\mathbf{p}_j$ calculated using channel output

We attempt to see if our multistage decoder will be able to perform the decoding successfully. We set $l = 0$ and compute $\mathbf{p}_j(0)$ from $\mathbf{p}_j$ based on (5.8). The $\mathbf{p}_j(0)$ shown in Table A.1 is used as input to the stage 0 decoder. Note that the $\mathbf{r}_{ji}$ used for this stage is uniformly distributed. After 1 iteration, the codeword estimate computed by the decoder is $\hat{\mathbf{x}}(0) = [1\,1\,0\,1\,1\,0\,0\,0]$. Since $\hat{\mathbf{x}}(0)\mathbf{H}_0^T = \mathbf{0}$, $\hat{\mathbf{x}}(0) \in C_0$ and decoding at stage 0 is considered to be successful and is terminated. We then proceed to refine $\mathbf{p}_j$ from $\hat{\mathbf{x}}(0)$. This refined $\mathbf{p}_j$ is shown in Table A.2. We then proceed to compute $\mathbf{p}_j(1)$ from the first refined $\mathbf{p}_j$ and set $l = 1$. The $\mathbf{p}_j(1)$ obtained is shown in Table A.3 and is used as the input to the stage 1 decoder. At the end of the first iteration, the decoder outputs the codeword estimate $\hat{\mathbf{x}}(1) = [3\,1\,0\,3\,3\,2\,0\,2]$. Again, since $\hat{\mathbf{x}}(1)\mathbf{H}_1^T = \mathbf{0}$ and $\hat{\mathbf{x}}(1) \in C_1$, decoding at stage 1 is considered to be a success and terminated. We then use $\hat{\mathbf{x}}(1)$ to refine $\mathbf{p}_j$ and this second refined $\mathbf{p}_j$ is shown in Table A.4.

We set $l = 2$ and let the second refined $\mathbf{p}_j$ be the input to the stage 2 decoder (we now decode over R_2). At the end of the second iteration, the decoder outputs the codeword estimate $\hat{\mathbf{x}}(2) = [7\,1\,4\,3\,7\,6\,0\,6]$ and $\hat{\mathbf{x}}(2)\mathbf{H}_2^T = \mathbf{0}$. Since $\hat{\mathbf{x}}(2) = \mathbf{x}$, we have successful decoding.

From this example, we have illustrated that the multi-stage approach can correct errors which were previously uncorrectable using the conventional single-stage MPA.

5.8 Simulation Results

In this section, we compare the performance of our multi-stage decoding scheme against that of the single-stage MPA on an AWGN channel under different various modulation schemes. We focus on codes over $\mathbb{Z}_4$ and $\mathbb{Z}_8$ of rate one-half and length 500 and 1000. The parity-check matrices of all the codes simulated here have an average column weight of 2.5 and row weight of approximately 5. It is shown in [35] that such matrices do not have short cycles of length 4. Further, each instance of the MPA in our simulations iterates at most 50 times.

Under BPSK modulation, the multi-stage offers 0.1 dB of coding gain over the single-stage MPA for both the $\mathbb{Z}_4$ and $\mathbb{Z}_8$ codes, as Figures 5.2 and 5.3 show. We further show that under 4-ary (respectively, 8-ary) PSK modulation corresponding to the constellation diagram shown in Figure 5.1(a), (respectively, Figure 5.1(c)), our multi-stage decoder offers a coding gain of about 0.07 dB of coding gain over the single-stage MPA for both $\mathbb{Z}_4$ (respectively, $\mathbb{Z}_8$) codes, as shown in Figure 5.4 (respectively, Figure 5.5).

We see from Figures 5.2 to 5.5 that the coding gain obtained remains the same for a $\mathbb{Z}_4$ and $\mathbb{Z}_8$ code. We can infer that, for a code C_{m-1} over R_{m-1}, the main contributor of additional coding gain is stage 0. In addition, as we increase the number of stages m, such that $m \to \infty$, the incremental coding gain contributed by each additional stage after stage 0 will diminish to zero. This is due to the property of the descending chain of ideals of R_{m-1},

$$R_{m-1} \supset pR_{m-1} \supset p^2R_{m-1} \supset \ldots \supset p^mR_{m-1} = \{0\}$$

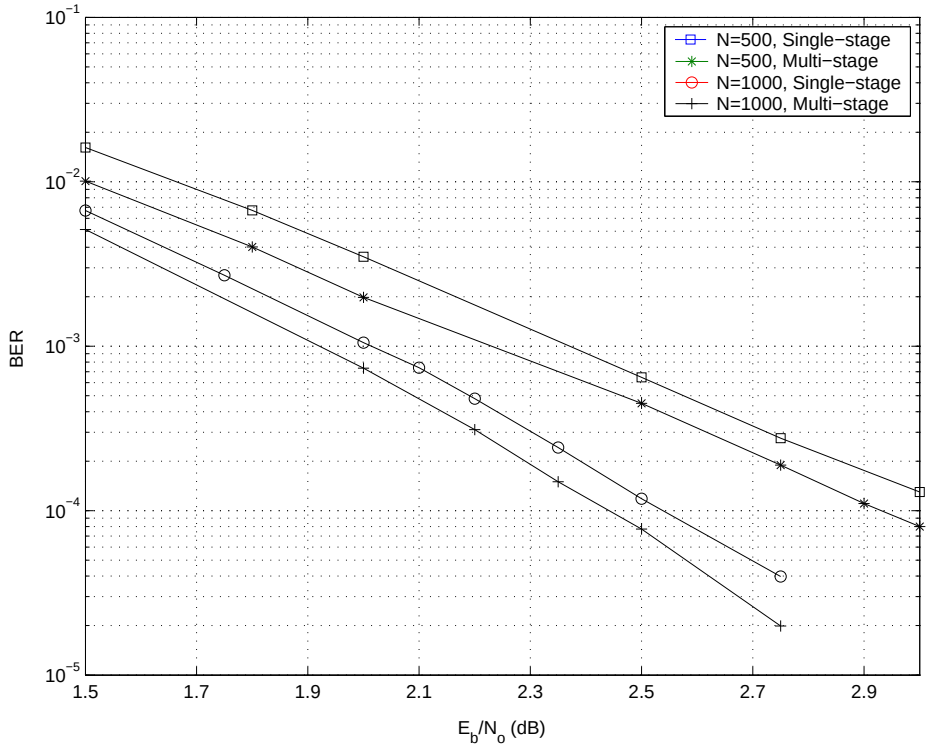

Figure 5.2: BER performance of $\mathbb{Z}_4$ codes under BPSK modulation

which comes into play at each stage of the decoding process. Stage l will attempt to correct the errors which stem from $p^l R_{m-1} \setminus p^{l+1} R_{m-1}$. Plainly put, (on average) the higher stages will have lesser errors to correct and this accounts for the diminishing additional coding gains provided by the higher stages [2].

We then consider the increase in complexity as we move from single-stage decoding to the multistage approach using the expressions obtained in Section 5.5. The increase in complexity to decode the code C_{m-1} using our multistage approach over the single-stage approach is just the number of arithmetic operations required to decode its homomorphic images C_l for $l = 0, 1, \ldots, m-2$. In our discussion, we shall only consider the increase in complexity in terms of the increase in multiplications required since multiplications are typically more costly than additions.

[2]The inferences, insights and equations of the above paragraph were contributed and derived, respectively, by Dr. M. A. Armand. The author gives his thanks and gratitude to Dr. M. A. Armand for allowing him to use his work in this thesis.

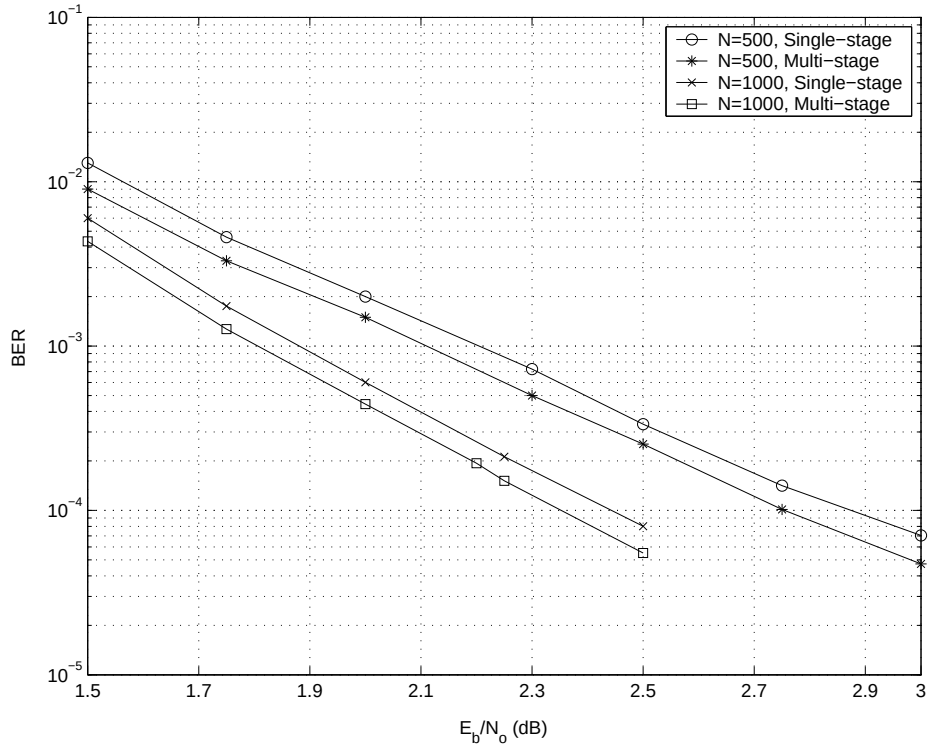

Figure 5.3: BER performance of $\mathbb{Z}_8$ codes under BPSK modulation

For the $\mathbb{Z}_4$ code of length 500 (respectively, 1000), the number of complex multiplications required to compute all the $\mathbf{r}_{ji}$ and real multiplications to compute the $\mathbf{q}_{ji}$ per iteration is 22500 (respectively, 45000) and 20000 (respectively, 40000), respectively. By similar analysis, the $\mathbb{Z}_8$ code of length 500 (respectively, 1000) requires 52500 (respectively, 105000) complex multiplications and 40000 (respectively, 80000) real multiplications to compute all the $\mathbf{r}_{ji}$ and $\mathbf{q}_{ji}$, respectively.

If we fix the maximum number of iterations at which the MPA terminates to be 50, the $\mathbb{Z}_4$ codes of length 500 and 1000 require 5.5×10^6 and 11×10^6 real multiplications[3], while the number of real multiplications required to decode the $\mathbb{Z}_8$ codes of length 500 and 1000 is 12.5×10^6 and 25×10^6, respectively.

Decoding the binary images of the codes in the logarithmic domain requires

[3]The real and complex multiplications have been summed together, with each complex multiplication replaced by 4 real multiplications.

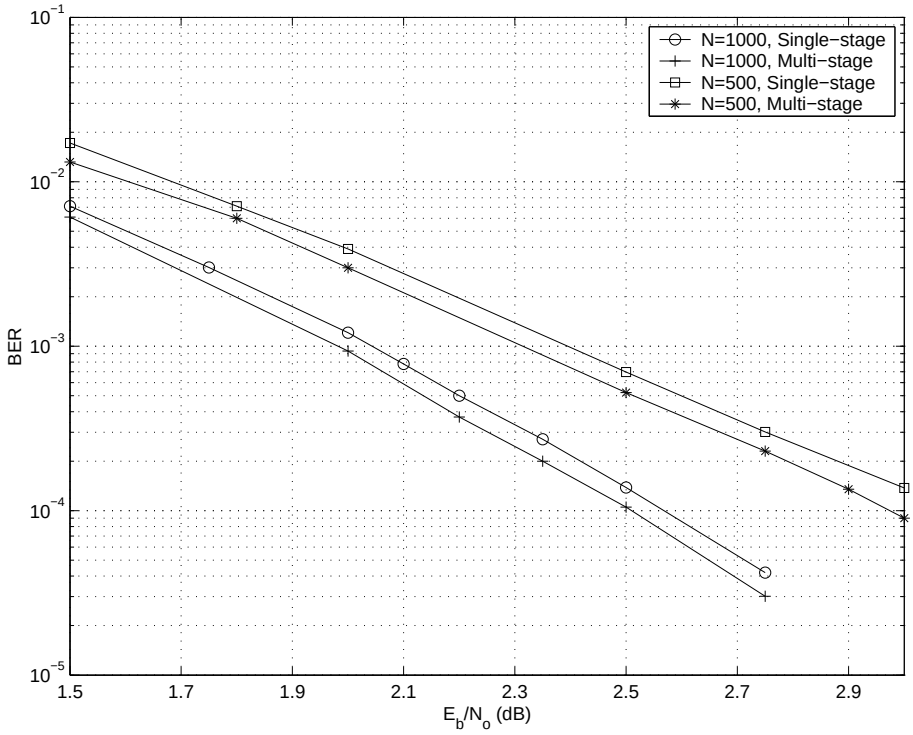

Figure 5.4: BER performance of $\mathbb{Z}_4$ codes under 4-ary PSK modulation

only additions. Hence, the increase in complexity to decode the $\mathbb{Z}_4$ codes in 2 stages is solely due to the updating $\mathbf{p}_j$ at the end of stage 0 and initialising $\mathbf{r}_{ji}(1)$ at the beginning of stage 1, i.e., due to the fixed components. From (5.10) and (5.11), the total number of real multiplications required to perform the 2 tasks are 8000 and 16000 for the $\mathbb{Z}_4$ codes of length 500 and 1000, respectively. The two tasks contribute to a marginal increase of 0.15% in decoding complexity over the single-stage decoding, in exchange for 0.07 to 0.1 dB of additional coding gain.

For the $\mathbb{Z}_8$ codes, the fixed components are now the updating of $\mathbf{p}_j$ at the end of the 0th and 1st stage, as well as initialising $\mathbf{r}_{ji}(1)$ and $\mathbf{r}_{ji}(2)$. The variable component is the decoding of the $\mathbb{Z}_4$ image of the code at stage 1. For the $\mathbb{Z}_8$ codes of length 500 and 1000, the number of real multiplications required to perform the fixed components is 28000 and 56000, respectively, while decoding of the $\mathbb{Z}_4$ images require 5.5×10^6 and 11×10^6, respectively. The increase in complexity over that of the single-stage is thus 44.22%. This represents a sharp increase in decoding

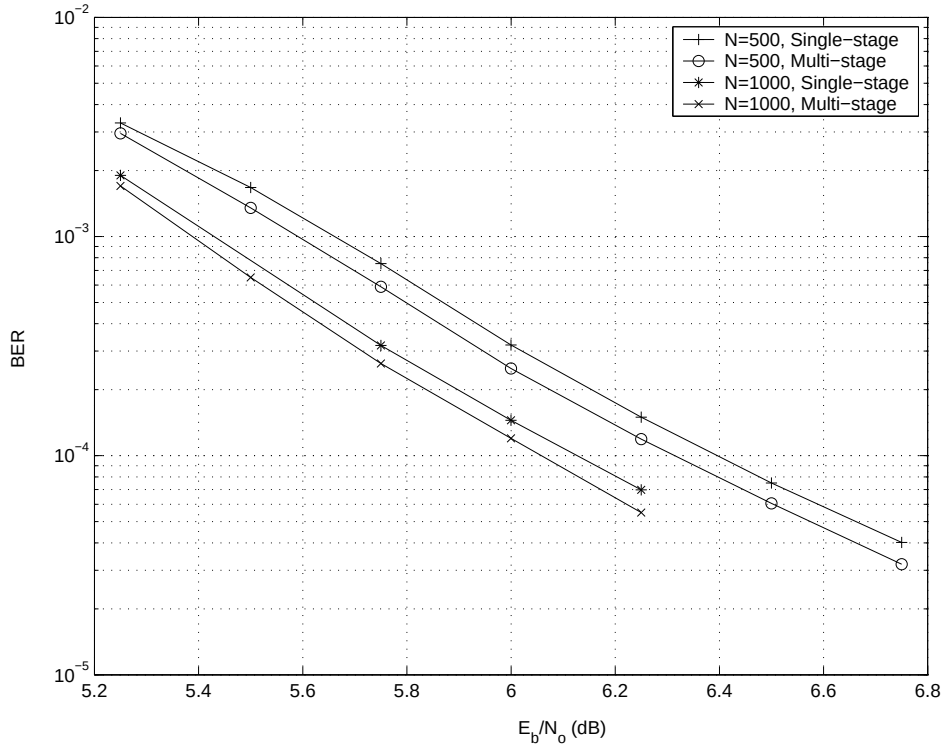

Figure 5.5: BER performance of $\mathbb{Z}_8$ codes under 8-ary PSK modulation

complexity, when compared to their $\mathbb{Z}_4$ counterparts, for similar improvements in coding gain.

Finally, Figure 5.6 shows the effect of varying β on the BER performance of the $\mathbb{Z}_8$ code of length 500. The values of β range from 0.1 to 0.9 in increments of 0.1. As Figure 5.6 indicates, the values of β corresponding to increasingly worse BER performance are: 0.5, 0.4, 0.6, 0.3, 0.7, 0.8, 0.2, 0.1 and 0.9. We note that β has a similar effect on the BER performance of the other codes considered here. (The corresponding BER graphs of the remaining 3 codes are shown in Appendix B.)

5.9 Concluding Remarks

In this chapter, we present a multistage MPA-based decoding algorithm for LDPC codes over $\mathbb{Z}_{p^m}$. We showed that our proposed algorithm outperformed its con-

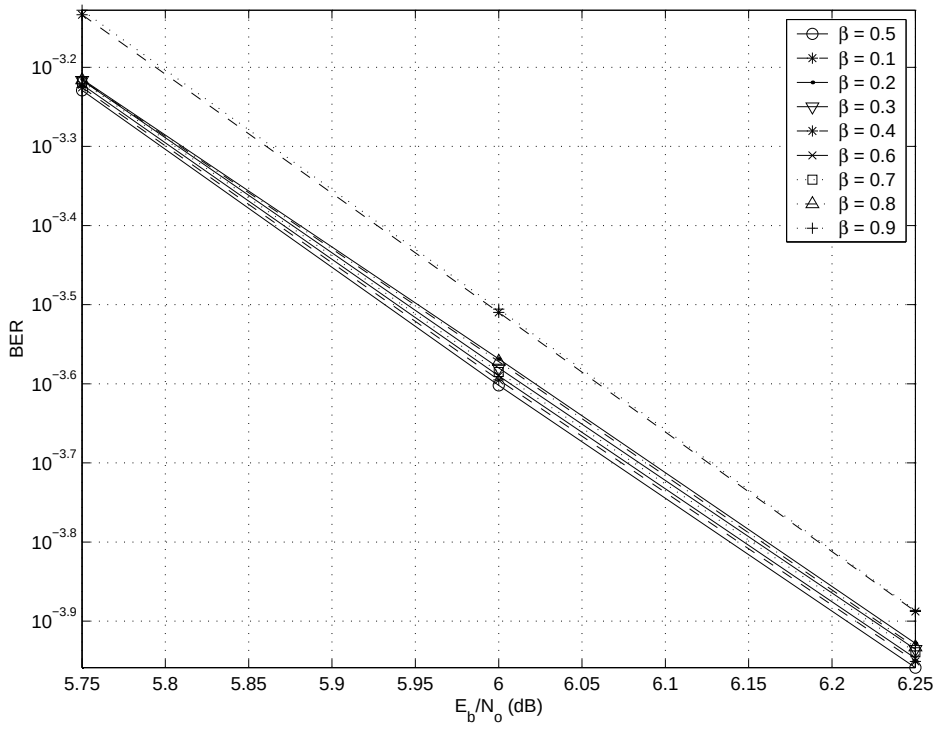

Figure 5.6: BER performance of $\mathbb{Z}_8$ code of length 500 for different values of β

ventional single-stage counterpart via computer simulations. This improvement in performance comes at the expense of increased decoding complexity. A complexity analysis shows that the increase in decoding complexity is considerable for larger values of m, i.e., for $m > 2$.

However, we also saw the diminishing additional coding gain provided by the higher stages and provided a rationale for it. Stage 0 provided the most significant amount of coding gain, compared to the subsequent stages. We thus propose a complexity-performance compromise where, for a code over $\mathbb{Z}_{p^m}$, we perform decoding using only 2 stages: first decoding at stage 0, then followed by decoding at stage $m-1$. This method allows us to skip the intermediate stages and reduces the decoding complexity by a significant amount, at only an insignificant detriment to performance.

Chapter 6

Conclusion

In this chapter, we summarise the work done as well as the discoveries made in the course of our work. We highlight some of the contributions made to the area of non-binary LDPC codes. In addition, we also include, in the final section of this chapter, recommendations for possible future research stemming from our work.

We begin with a summary of the thesis.

6.1 Thesis Summary

In Chapter 2, we gave a brief introduction as well as general terminologies used to describe LDPC codes in general. LDPC codes over various alphabets were also introduced. The factors affecting the performance of LDPC codes were also given and the code construction methods available to design good codes were presented.

In Chapter 3, a detailed description of the decoding algorithm (MPA) used to decode non-binary LDPC codes was presented. This MPA makes use of the Fourier

transform to reduce decoding complexity. A complexity analysis (in terms of the number of arithmetic operations required per iteration) was also given.

In Chapter 4, we introduced our mixed alphabet codes and gave a detailed description of the codes. We also gave a method for the construction of our codes. A modified decoding algorithm adapted from the MPA described in Chapter 3 exploiting the property that distinct code coordinates and parity-check constraints are defined over different alphabets was presented. This modified MPA reduces the number of redundant computations required to decode our mixed codes. Further we provided a complexity analysis to compare the decoding complexity of our mixed codes compared to their single alphabet counterparts. We also show via computer simulations that our mixed codes outperformed their single-alphabet counterparts and offer a performance-complexity trade-off.

In Chapter 5, we presented our work on multistage decoding of LDPC codes over $\mathbb{Z}_q$. We modified the MPA presented in Chapter 3 for codes defined over $\mathbb{Z}_q$, and adapt it for decoding in a multistage environment. A method to partition the 2^m-ary signal space to minimise the probability of an erroneous decoding decision from an earlier stage propagating through the subsequent stages was proposed so that the multistage decoder can function with 2^m-ary PSK modulation. We also showed the increase in complexity as we go from the single-stage to multistage approach. Our computer simulations showed that the multistage approach is indeed superior to the single-stage approach and offers a performance-complexity trade-off as well.

6.2 Recommendations for future work

In this section, we make some suggestions for future research that can be done based on the work done within this thesis.

For our work on mixed codes, we managed to find combinations of alphabets that combine well with each other, yielding improvement in BER performance over their single alphabet counterparts. However, we also noticed that other combinations do not yield improvements. In fact, they perform poorer than their single alphabet counterparts. Further research can be undertaken to better understand this phenomenon.

Appendix A

Tables of $\mathbf{p}_j(0)$, $\mathbf{p}_j(1)$ and refined $\mathbf{p}_j$ for worked example

j	$\xi = 0$ $\xi = 1$	Hard Decision Value	Result
0	0.0061 0.9939	1	Correct
1	0.5322 0.4678	0	Error
2	0.9858 0.0142	0	Correct
3	0.0653 0.9347	1	Correct
4	0.0009 0.9991	1	Correct
5	0.9998 0.0002	0	Correct
6	0.9546 0.0454	0	Correct
7	0.9973 0.0027	0	Correct

Table A.1: Intrinsic symbol probabilities $\mathbf{p}_j(0)$ calculated using initial $\mathbf{p}_j$

j	$\xi=0$	$\xi=1$	$\xi=2$	$\xi=3$	$\xi=4$	$\xi=5$	$\xi=6$	$\xi=7$	Hard Decision Value	Result
0	0.0002	0.0082	0.0027	0.1470	0.0000	0.1954	0.0001	0.6464	7	Correct
1	0.3603	0.6341	0.0000	0.0000	0.0023	0.0033	0.0000	0.0000	1	Correct
2	0.7412	0.0071	0.0000	0.0000	0.2506	0.0000	0.0010	0.0000	0	Error
3	0.0000	0.0000	0.0336	0.8420	0.0000	0.0007	0.0001	0.1236	3	Correct
4	0.0000	0.0000	0.0004	0.3959	0.0000	0.0024	0.0000	0.6013	7	Correct
5	0.0000	0.0000	0.2144	0.0001	0.0035	0.0000	0.7820	0.0000	6	Correct
6	0.6076	0.0229	0.0002	0.0000	0.3564	0.0003	0.0126	0.0000	0	Correct
7	0.0081	0.0001	0.0993	0.0012	0.2327	0.0000	0.6585	0.0000	6	Correct

Table A.2: Intrinsic symbol probabilities $\mathbf{p}_j$ after first refinement

j	$\xi=0$	$\xi=1$	$\xi=2$	$\xi=3$	Hard Decision Value	Result
0	0.0002	0.2036	0.0029	0.7934	3	Correct
1	0.3626	0.6374	0.0000	0.0000	0	Error
2	0.9918	0.0071	0.0010	0.0000	0	Correct
3	0.0000	0.0007	0.0338	0.9656	3	Correct
4	0.0000	0.0024	0.0004	0.9972	3	Correct
5	0.0035	0.0000	0.9964	0.0001	2	Correct
6	0.9639	0.0232	0.0128	0.0000	0	Correct
7	0.2408	0.0001	0.7579	0.0013	2	Correct

Table A.3: Intrinsic symbol probabilities $\mathbf{p}_j(1)$ calculated using refined $\mathbf{p}_j$

j	$\xi=0$	$\xi=1$	$\xi=2$	$\xi=3$	$\xi=4$	$\xi=5$	$\xi=6$	$\xi=7$	Hard Decision Value	Result
0	0.0001	0.0046	0.0015	0.1639	0.0000	0.1090	0.0001	0.7209	7	Correct
1	0.2201	0.7745	0.0000	0.0000	0.0014	0.0041	0.0000	0.0000	1	Correct
2	0.7443	0.0036	0.0000	0.0000	0.2516	0.0000	0.0005	0.0000	0	Error
3	0.0000	0.0000	0.0171	0.8567	0.0000	0.0003	0.0001	0.1257	3	Correct
4	0.0000	0.0000	0.0002	0.3964	0.0000	0.0012	0.0000	0.6021	7	Correct
5	0.0000	0.0000	0.2147	0.0000	0.0018	0.0000	0.7834	0.0000	6	Correct
6	0.6187	0.0116	0.0001	0.0000	0.3629	0.0002	0.0064	0.0000	0	Correct
7	0.0046	0.0001	0.1130	0.0007	0.1324	0.0000	0.7492	0.0000	6	Correct

Table A.4: Intrinsic symbol probabilities $\mathbf{p}_j$ after second refinement

Appendix B

BER Performance of codes for different β values

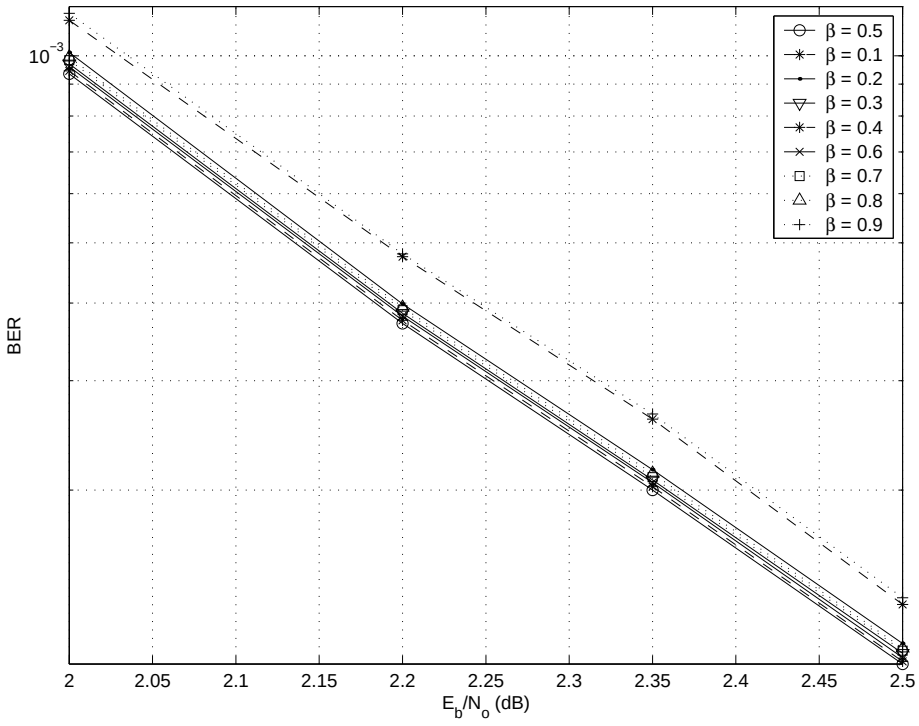

Figure B.1: BER performance of $\mathbb{Z}_4$ code of length 1000 for different values of β

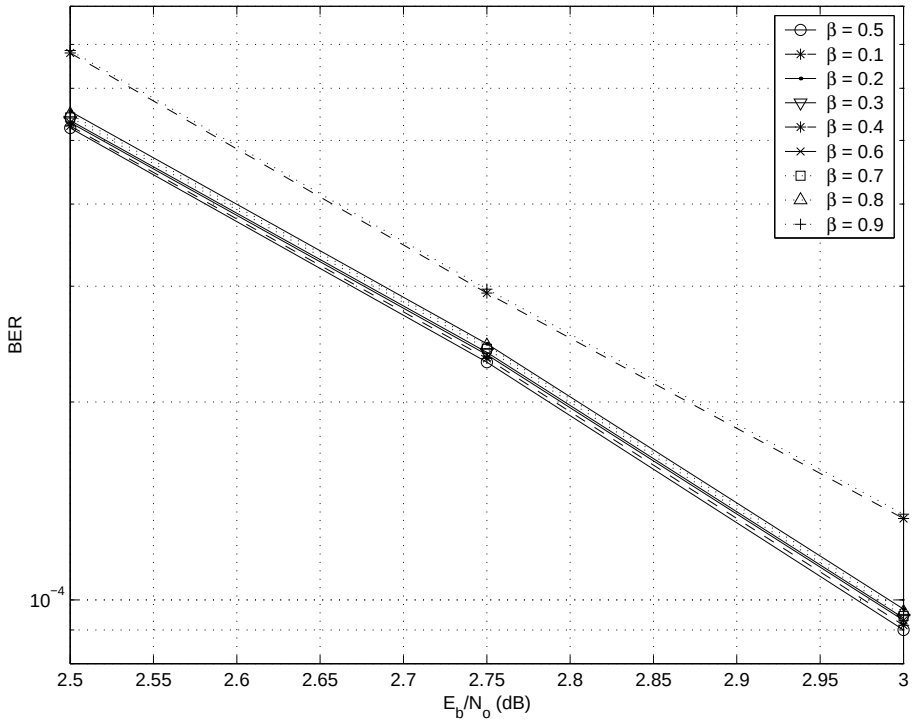

Figure B.2: BER performance of $\mathbb{Z}_4$ code of length 500 for different values of β

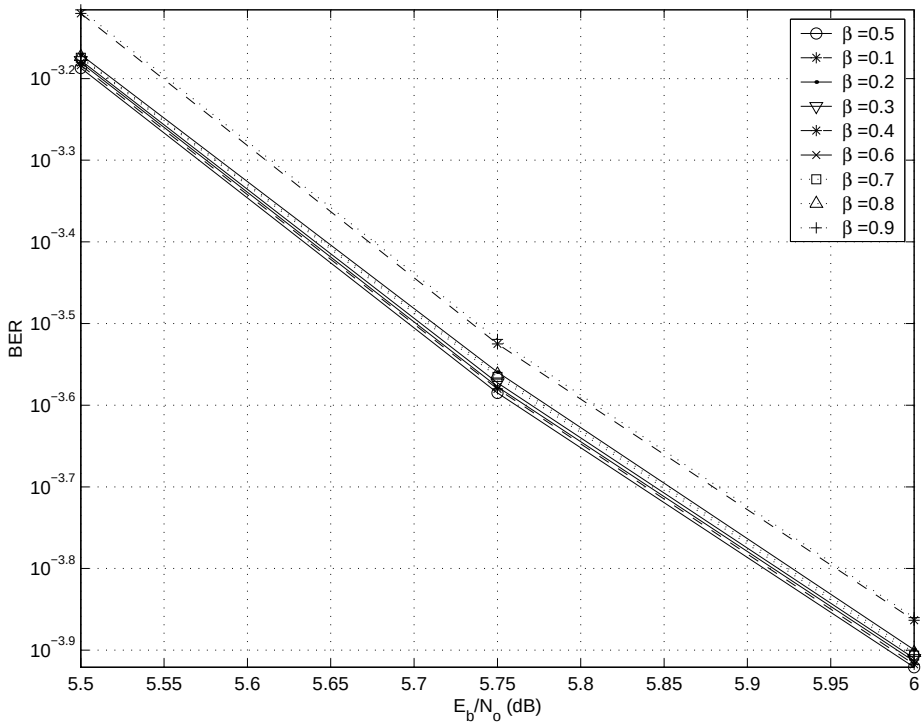

Figure B.3: BER performance of $\mathbb{Z}_8$ code of length 1000 for different values of β

Bibliography

[1] M. A. Armand and O. de Taisne, "Multistage list decoding of generalized Reed-Solomon codes over Galois rings," *IEEE Commun. Lett.*, vol. 9, pp. 625–627, July 2005.

[2] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF(2^q)," in *Proc. IEEE Inform. Theory Workshop*, March 2003, pp. 70–73.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon-limit error-correcting coding and decoding: Turbo codes," in *Proc. 1993 IEEE Int. Conf. Commun.*, May 1993, pp. 1064–1070.

[4] R. E. Blahut, *Theory and practice of error control codes.* Addison-Wesley Publishing Company, 1983.

[5] R. C. Bose and D. K. Chaudhuri, "Further results on error correcting binary group codes," *Inform. Contr.*, vol. 3, pp. 279–290, September 1960.

[6] ——, "On a class of error correcting binary group codes," *Inform. Contr.*, vol. 3, pp. 68–79, March 1960.

[7] G. Caire and E. Biglieri, "Linear block codes over cyclic groups," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1246–1256, September 1995.

[8] A. R. Calderbank and N. J. A. Sloane, "Modular and p-adic cyclic codes," *Designs, codes, and cryptography*, no. 6, pp. 21 – 35, 1995.

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

[10] M. C. Davey, "Error-correction using low-density parity-check codes," Ph.D dissertation, Univ. Cambridge, December 1999.

[11] M. C. Davey and D. J. C. MacKay, "Low-density parity-check codes over GF(q)," *IEEE Commun. Lett.*, vol. 2, pp. 165–167, June 1998.

[12] D. Divsalar, H. Jun, and R. J. McEliece, "Coding theorems for 'turbo-like' codes," in *Proc. 36th Allerton Conf. on Commun., Control, and Comp.*, 1998, pp. 201–210.

[13] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, vol. 3, pp. 37–46, June 1955.

[14] U. Erez and G. Miller, "The ML decoding performance of LDPC ensembles over $\mathbb{Z}_q$," *IEEE Trans. Inform. Theory*, vol. 51, pp. 1871–1879, May 2005.

[15] G. D. Forney, Jr., *Concatenated codes*. MIT Press, Cambridge, Mass., 1966.

[16] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, January 1962.

[17] O. Goldreich, D. Ron, and M. Sudan, "Chinese remaindering with errors," in *Electronic Colloquium on Computational Complexity*, vol. 62, 1998, pp. 1–27.

[18] F. Guo, "Low-density parity-check coding," Ph.D mini thesis, Univ. Southampton, June 2002.

[19] J. Hagenauer, "Viterbi decoding of convolutional codes for fading- and burst-channels," in *Proc. Int. Zurich Seminar Digital Commun., Zurich, Switzerland*, March 1980, pp. 1–7.

[20] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147–150, 1950.

[21] M. Herzog and J. Schonheim, "Linear and nonlinear single-error-correcting perfect mixed codes," *Inform. Contr.*, vol. 18, pp. 364–368, 1971.

[22] J. Hou, P. H. Siegel, and L. B. Milstein, "Performance analysis and code optimization of low-density parity-check codes on Rayleigh fading channels," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 924–934, May 2001.

[23] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth Tanner graphs," in *Proc. IEEE Globecom Conf.*, November 2001, pp. 995–1001.

[24] S. J. Johnson and S. R. Weller, "A family of irregular LDPC codes with low encoding complexity," *IEEE Commun. Lett.*, vol. 7, pp. 79–81, February 2003.

[25] ——, "High-rate LDPC codes from unital designs," in *Proc. IEEE Globecom Conf.*, December 2003, pp. 2036–2040.

[26] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes basd on finite geometries: A rediscovery and new results," *IEEE Trans. Inform. Theory*, vol. 47, pp. 2711–2736, November 2001.

[27] F. R. Kschichang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498–519, February 2001.

[28] S. Lin and D. J. Costello, Jr., *Error control coding: fundamentals and applications.* Prentice-Hall, Englewood Cliffs, N.J., 1983.

[29] M. Luby, M. Mitzenmacher, A. Shokrollahi, and V. Stemann, "Practical loss-resilient codes," in *Proc. 29th Annual ACM Symp. Theory of Computing*, 1997, pp. 150–159.

[30] P. Ma, D. Yuan, X. Yang, and H. Zhang, "High-rate LDPC codes in image transmission over Rayleigh fading channel," in *First IEEE Consumer Communications and Networking Conference*, January 2004, pp. 357–360.

[31] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–433, March 1999.

[32] D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Commun.*, vol. 47, pp. 1449–1454, October 1999.

[33] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*. North-Holland, Amsterdam, 1977.

[34] P. R. J. Östergård, "Constructions of mixed covering codes," Helsinki Univ. of Tech., Digital Systems Lab., report 18, December 1991.

[35] L. Ping, W. K. Leong, and M. Phamdo, "Low density parity check codes with semi-random parity check matrix," *IEE Elect. Lett.*, vol. 35, pp. 38–39, January 1999.

[36] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM J. Appl. Math.*, vol. 8, pp. 300–304, June 1960.

[37] T. J. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching low-density parity-check codes," *IEEE Trans. Inform. Theory*, February.

[38] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 559–618, February 2001.

[39] ——, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 638–656, February 2001.

[40] C. E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 1948.

[41] S. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, November 1996.

[42] H. X. Song and J. R. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," *IEEE Trans. Magn.*, vol. 39, pp. 1081–1087, March 2003.

[43] D. Sridhara and T. E. Fuja, "Low-density parity-check codes over groups and rings," in *Proc. IEEE Inform. Theory Workshop*, October 2002, pp. 163–166.

[44] H. Tang, J. Xu, Y. Kou, S. Lin, and K. Abdel-Ghaffar, "On algebraic construction of gallager and circulant low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 50, pp. 1269–1279, June 2004.

[45] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, February 1981.

[46] B. Vasic and O. Milenkovic, "Combinatorial constructions of low-density parity-check codes for iterative decoding," *IEEE Trans. Inform. Theory*, vol. 50, pp. 1156–1176, June 2004.

[47] L. Wei, "Several properties of short LDPC codes," *IEEE Trans. Commun.*, vol. 52, pp. 721–727, May 2004.

[48] X. Yang, D. Yuan, P. Ma, and H. Zhang, "Performance of LDPC codes in image transmission over Rayleigh fading channel," in *International Conference on Communication Technology*, vol. 2, April 2003, pp. 1444–1446.