Low-Density Parity-Check Codes: Construction and Implementation

by

Gabofetswe Alafang Malema

B.S Computer Engineering

M.S Electrical Engineering and Computer Science

Thesis submitted for the degree of

Doctor of Philosophy

in

School of Electrical and Electronic Engineering,

Faculty of Engineering,

Computer and Mathematical Sciences

The University of Adelaide, Australia

November 2007

Typeset in LaTeX 2_ε
Gabofetswe Alafang Malema

Contents

Abstract

Low-Density Parity-Check Codes: Construction and Implementation

by

Gabofetswe A. Malema

Low-density parity-check (LDPC) codes have been shown to have good error correcting performance approaching Shannon's limit. Good error correcting performance enables efficient and reliable communication. However, a LDPC code decoding algorithm needs to be executed efficiently to meet cost, time, power and bandwidth requirements of target applications. The constructed codes should also meet error rate performance requirements of those applications. Since their rediscovery, there has been much research work on LDPC code construction and implementation. LDPC codes can be designed over a wide space with parameters such as girth, rate and length. There is no unique method of constructing LDPC codes. Existing construction methods are limited in some way in producing good error correcting performing and easily implementable codes for a given rate and length. There is a need to develop methods of constructing codes over a wide range of rates and lengths with good performance and ease of hardware implementability. LDPC code hardware design and implementation depend on the structure of target LDPC code and is also as varied as LDPC matrix designs and constructions. There are several factors to be considered including decoding algorithm computations,processing nodes interconnection network, number of processing nodes, amount of memory, number of quantization bits and decoding delay. All of these issues can be handled in several different ways.

This thesis is about construction of LDPC codes and their hardware implementation. LDPC code construction and implementation issues mentioned above are too many to be addressed in one thesis. The main contribution of this thesis is the development of LDPC code construction methods for some classes of structured LDPC codes and techniques for reducing decoding time. We introduce two main methods for constructing structured codes. In the first method, column-weight two LDPC codes are derived from distance graphs. A wide range of girths, rates and lengths are obtained compared to existing

methods. The performance and implementation complexity of obtained codes depends on the structure of their corresponding distance graphs. In the second method, a search algorithm based on bit-filing and progressive-edge growth algorithms is introduced for constructing quasi-cyclic LDPC codes. The algorithm can be used to form a distance or Tanner graph of a code. This method could also obtain codes over a wide range of parameters. Cycles of length four are avoided by observing the row-column constraint. Row-column connections observing this condition are searched sequentially or randomly. Although the girth conditions are not sufficient beyond six, larger girths codes were easily obtained especially at low rates. The advantage of this algorithm compared to other methods is its flexibility. It could be used to construct codes for a given rate and length with girths of at least six for any sub-matrix configuration or rearrangement. The code size is also easily varied by increasing or decreasing sub-matrix size. Codes obtained using a sequential search criteria show poor performance at low girths (6 and 8) while random searches result in good performing codes.

Quasi-cyclic codes could be implemented in a variety of decoder architectures. One of the many options is the choice of processing nodes interconnect. We show how quasi-cyclic codes processing could be scheduled through a multistage network. Although these networks have more delay than other modes of communication, they offer more flexibility at a reasonable cost. Banyan and Benes networks are suggested as the most suitable networks.

Decoding delay is also one of several issues considered in decoder design and implementation. In this thesis, we overlap check and variable node computations to reduce decoding time. Three techniques are discussed, two of which are introduced in this thesis. The techniques are code matrix permutation, matrix space restriction and sub-matrix row-column scheduling. Matrix permutation rearranges the parity-check matrix such that rows and columns that do not have connections in common are separated. This techniques can be applied to any matrix. Its effectiveness largely depends on the structure of the code. We show that its success also depends on the size of row and column weights. Matrix space restriction is another technique that can be applied to any code and has fixed reduction in time or amount of overlap. Its success depends on the amount of restriction and may be traded with performance loss. The third technique already suggested in literature relies on the internal cyclic structure of sub-matrices to achieve overlapping. The technique is limited to LDPC code matrices in which the number of sub-matrices is equal to row and column weights. We show that it can be applied to other codes with a lager number of

sub-matrices than code weights. However, in this case maximum overlap is not guaranteed. We calculate the lower bound on the amount of overlapping. Overlapping could be applied to any sub-matrix configuration of quasi-cyclic codes by arbitrarily choosing the starting rows for processing. Overlapping decoding time depends on inter-iteration waiting times. We show that there are upper bounds on waiting times which depend on the code weights. Waiting times could be further reduced by restricting shifts in identity sub-matrices or using smaller sub-matrices. This overlapping technique can reduce the decoding time by up to 50% compared to conventional message and computation scheduling.

Techniques of matrix permutation and space restriction results in decoder architectures that are flexible in LDPC code design in terms of code weights and size. This is due to the fact that with these techniques, rows and columns are processed in sequential order to achieve overlapping. However, in the existing technique, all sub-matrices have to be processed in parallel to achieve overlapping. Parallel processing of all code sub-matrices requires the architecture to have the number of processing units at least equal to the number sub-matrices. Processing units and memory space should therefore be distributed among the sub-matrices according to the sub-matrices arrangement. This leads to high complexity or inflexibility in the decoder architecture. We propose a simple, programmable and high throughput decoder architecture based on matrix permutation and space restriction techniques.

Publications

1. G. Malema and M. Liebelt,"Quasi-Cyclic LDPC Codes of Column-Weight Two using a Search Algorithm", *European Journal of Advances in Signal Processing*,Volume 2007,Article ID 45768, 8 pages,2007.

2. G. Malema and M. Liebelt,"High Girth Column-Weight Two LDPC Codes based on Distance Graphs", *European Journal of Wireless Communications and Networking*,Volume 2007,Article ID 48158, 5 pages,2007.

3. G. Malema and M. Liebelt,"Very Large Girth Column-weight two Quasi-Cyclic LDPC Codes", *International Conference on Signal Processing*,pp.1776-1779,Guilin, China,Nov,2006.

4. G. Malema and M. Liebelt,"Interconnection network for strcutured Low-Density Parity-Check Decoders", *Asia Pacific Communications Conference*",pp.537-540,Perth, Australia,October,2005.

5. G.Malema, M. Liebelt and C.C. Lim," Reduced routing in fully-parallel LDPC decoders", *SPIE: International Symposium on Microelectronics, MEMS, and Nanotechnology*,Brisbane, Australia,December 2005.

6. G.Malema and M. Liebelt," Low-Complexity Regular LDPC codes for Magnetic Storage Devices", *Enformatika : International Conference on Signal Processing*,vol.7,pp. 269-271,August,2005.

7. G. Malema and M. Liebelt,"Programmable Low-Density Parity-Check Decoder", *Intelligent Signal Processing and Communications Systems,(ISPACS'04)*,pp.801-804, Nov,2004.

This page is blank

Statement of Originality

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

I give consent to this copy of the thesis, when deposited in the University Library, being available for loan and photocopying.

Signed

Date

This page is blank

Acknowledgements

This work is dedicated to my mother, Tshireletso Alafang.

I would like to thank my supervisors, Assoc. Prof. M. Liebelt and Assoc. Prof. C.C. Lim, for the guidance they have provided throughout my research. Thank you, Prof. M. Liebelt for also reading my thesis. Your comments have been of a great help. I am also grateful to the staff of School of Electrical and Electronic Engineering for their collective support. My office mates and colleagues, Tony, Adam, Zhu and Bobby you have helped me a lot.

I also thank my mother again for giving me a second chance in education. Without your vision and patience I wouldn't be where I am. And thanks Dad, Alafang Malema, for listening.

Gabofetswe Alafang Malema

This page is blank

List of Figures

This page is blank

List of Tables

This page is blank

Chapter 1

Introduction

1.1 Overview

Communications systems transmit data from source to destination through a channel or medium such as air, wire lines and optical fibres. The reliability of the received data depends on the channel and external noise that could interfere or distort the signal representing the data. The noise introduces errors in the transmitted data. Shannon[1] showed through his coding theorem that reliable transmission could be achieved if the data rate is less than the channel capacity. The theorem shows that a sequence of codes of rate less than the channel capacity have the capability of correcting all errors as the code length goes to infinity [1]. Error detection and correction is achieved by adding redundant symbols to the original data. This realization has lead to the development of error correction codes(ECCs) and the subject of information theory to meet Shannon's conditions. Without ECCs data will need to be retransmitted if it could be detected that there is an error in the received data. Retransmission adds delay, cost and wastes system throughput. Alternatively the number of errors could be reduced by using a stronger signal to dominate the noise. However, this approach increases power consumption of the system as more power is needed to drive a signal[2]. ECCs could be used to detect and correct errors in the received data thereby increasing the system throughput, speed and reducing power consumption. They are especially suitable for long distance one-way communication channels such as satellite to satellite and deep space communication. They are also used in wireless communications and storage devices. Figure 1.1 shows a basic

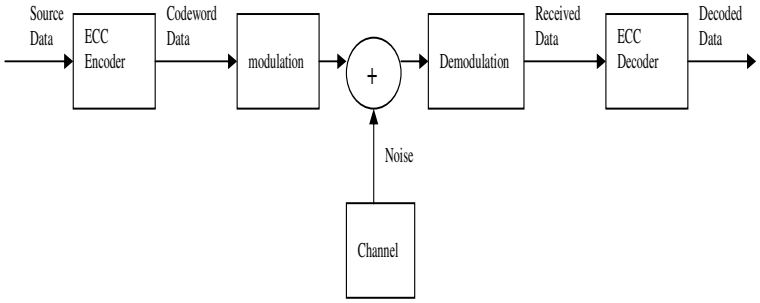

Figure 1.1. A basic communication system block diagram

communication system diagram showing data movement from source to destination. Data from the source is encoded using an encoder based on an error detection and correcting algorithm before it is modulated and sent through a channel. Encoding adds redundant symbols to the data to be transmitted. The channel noise or interference might affect the transmitted data, changing some symbols. At the destination received data (source data plus noise) is demodulated and estimated using a predefined method defined by the decoder algorithm.

Several error correction codes have been developed over time to encode and decode sent and received data respectively. They differ in correcting performance, computation and implementation complexity. ECCs include Viterbi, convolution, Bose-Chaudhuri-Hocquenghen (BCH), Reed-Solomons[2][3], turbo [4] and low-density parity-check codes (LDPC)[5].

1.2 LDPC Codes

In 1992 Turbo codes developed by Berrou et al in [4] were the first codes to be shown to perform close to the Shannon limit or channel capacity. They iteratively estimate received bit probabilities using Pearl's belief propagation algorithm[6]. The success of Turbo codes led to the rediscovery of low-density parity-check codes by MacKay and Neal in [7]. They were originally developed by Gallager in the 1960s[5][8]. They were largely ignored for a long time because their computational complexity was high for the hardware technology at the time.

LDPC codes also use iterative updating of bit probabilities based on belief algorithm. Richardson et al also proved the results obtained by MacKay and Neal in [9].

LDPC codes match Turbo codes in decoding performance[10]. However, they have several advantages over Turbo codes including parallelism in decoding and simple computation operations.

LDPC decoding computations are divided into two sets of nodes, check and variable nodes. Nodes on each side do computations independently of each other. A node is only connected to nodes on the other side. This allows computations in each side to be done in parallel. In Turbo codes, decoding operations in a block or window are dependent on each other in both ascending and descending order. This forces decoding calculations to be serialized within a block or window [11]. Although some LDPC code computations involve complex operations such as tangent and inverse tangent there are complexity reducing techniques for approximating these operations without a significant loss of performance. Low computational complexity combined with parallelism and good error correcting performance are some of the reasons LDPC codes have since received much attention and are being recommended for some communications systems such as digital video broadcasting (DVB-2) [12] and considered for many others[13][14].

Despite these advantages good LDPC code construction methods and efficient hardware (encoder and decoder) implementations are still a challenge. Meeting or coming close to the channel capacity performance assumes that an infinitely long code is used. Richardson and Urbanke [15] used a one million block length to come within 0.13dB of the channel capacity at 10^{-6} bit error probability. Different application systems have different decoding performance, latency, power and cost requirements. These requirements put constraints on code size and hardware implementations. As a result, different code sizes are recommended for different applications to meet both performance and hardware requirements. For example, LDPC codes can be applied to wireless, wired and optical communication systems, storage applications such magnetic discs and compact discs. Wireless applications require low power implementations with few rates at several Mbps. Storage applications require about 1Gbps and high rates codes [16], while optical communication throughput can be above 10Gbps[14].

These applications tolerate different delays, hardware cost and throughput. In addition they have different error probability expectations. The challenge is to find ways of constructing good performing LDPC codes given a fixed length and rate.

Construction of LDPC codes is not unique. It is varied and could be designed with several parameters including length and rate. These parameters also vary widely. For LDPC

codes to be successfully applied to many applications systems, methods for constructing good codes given a limited block length and other parameters are needed. Constructed codes must also satisfy hardware constraints such as latency, cost, power consumption flexibility and scalability depending on the application. These are the issues that will be addressed in this thesis.

1.3 Thesis Contribution

The main subject of this thesis is the construction of LDPC codes and their hardware implementations, in particular structured codes. There are a number of contributions in the literature on LDPC codes construction and implementation. Our aim is to add to that knowledge by developing possible solutions to some of the pressing issues on LDPC codes construction and implementation. Existing LDPC codes construction methods for structured (defined interconnection pattern) codes have limitations in using arbitrary lengths or rates. We introduce methods for constructing structured codes over a wide range of rates, lengths and girths. In hardware implementation, a case is made for using multi-stage networks as a communication network for quasi-cyclic LDPC decoders. Techniques for reducing decoding delay by overlapping decoding computations are also introduced and discussed. A decoder architecture based on developed overlapping techniques is also proposed.
The contributions of this thesis are summarized in the following points:

- A method for deriving column-weight two LDPC codes from distance graphs is introduced. A wide range of codes are obtained in terms of girth and rate. (Chapter 3, section 3.3)

- A structured search method for constructing structured LDPC codes of a desired girth is also introduced. Rows and columns are divided into equal groups to obtain a block or sub-matrix structure in the parity-check matrix. Connections within groups are made if they do not violate a desired girth. Searching for such connections is done sequentially within groups. Although codes of a desired girth are obtained such codes show poor bit-error probability. (Chapter 3, section 3.4)

- Another search algorithm is introduced to obtain quasi-cyclic LDPC codes. Connections within row and column groups are in consecutive order such that sub-matrices are cyclically shifted. Connections are made between rows and columns if they do not violate a desired girth. Searching for such connections could be done sequentially or randomly within a group. Randomly searched codes outperform sequential searched ones. Although the algorithm guarantee only a girth of six, higher girths are also easily obtained. A major advantage of this algorithm compared to other methods is its flexibility (code size,rate, regular, irregular and sub-matrix configurations). The algorithm could be used with any number of sub-matrices and sub-matrices arrangement or configuration. (Chapter 4)

- To reduce wiring congestion and complexity in fully parallel decoders, the parity-check matrix is reordered using matrix reordering algorithms. Average connection ranges are reduced resulting in smaller cut-sizes. (Chapter 5, Section 5.3)

- Decoder check and variable node interconnections are discussed. A case is made for banyan and Benes multistage networks as a means of communication between check and variable processing nodes in quasi-cyclic LDPC decoders. Multistage networks are efficient than hardwired or memory banks interconnections when sub-matrix configurations in a quasi-cyclic are random. For multistage networks to be efficient vector processing should be used. (Chapter 6, Section 6.2)

- Techniques for reducing decoding time for quasi-cyclic LDPC decoders are suggested. Permutation of the parity-check matrix, matrix connections restrictions and quasi-cyclic computation overlapping methods are introduced and discussed. Matrix permutation can be applied to any matrix but is limited by structure and row and column weights of the code. Matrix-space restriction gives the same amount of overlapping regardless of the code. Its performance depends on the extent of row-column connections restriction which needs to be weighed against possible performance loss. Careful calculation of starting rows and columns in a quasi-cyclic code can lead to a decoding time of up to 50%. Worst cases of overlapping are calculated based on row and column weights of target code. Techniques of matrix permutation and space restriction allow sequential processing of code sub-matrices. This property leads to simpler and flexible decoder architectures compared to the existing technique. In the existing technique sub-matrices are processed in parallel

to achieve overlapping. A simple and programmable LDPC decoder architecture is proposed based on matrix permutation and matrix space restriction overlapping techniques. The architecture can run codes of any length, rate and both regular and irregular codes. The throughput can be increased by simple increasing the number of processing elements. Although the proposed decoder is designed based on quasi-cyclic LDPC codees, it can be used for random and other structured codes. To our knowledge this is the most flexible decoder to date. (Chapter 6, section 6.3)

1.4 Thesis Outline

This thesis is organized as follows.

Chapter 2 Presents an overview of LDPC codes and the message passing decoding algorithm.

Chapter 3 An overview of LDPC code construction methods is presented. A method for constructing codes with a column-weight of two from distance graphs is proposed. A search method is introduced for constructing structured codes. Codes of different rates and girths are obtained using the method.

Chapter 4 The proposed search algorithm in Chapter 3 is modified to obtain quasi-cyclic codes. Quasi-cyclic codes over a wide range of rates and lengths are obtained with a girth of at least six. The algorithm could also be used with an arbitrary arrangement of sub-matrices for both regular and irregular codes. Decoding performance of these codes is also evaluated.

Chapter 5 Previews LDPC decoder architectures. Methods for reducing routing complexity and average wire-length in fully parallel decoders are also discussed. We use sparse matrix reordering algorithms to reduce the overall wire-length and routing complexity.

Chapter 6 Decoder communication implementation using multistage interconnection networks is discussed. Banyan and Benes networks are suggested for quasi-cyclic codes. Techniques for overlapping decoder computations are introduced and evaluated. A decoder architecture based on proposed overlapping techniques is developed.

Chapter 7 Summarizes the thesis and proposals for further research in relation to some presented ideas.

This page is blank

Chapter 2

LDPC Codes

2.1 Linear Block Codes

Error correcting codes attach extra bits (or in general symbols) to the transmitted data. The extra bits are the redundancy which are then used to detect and correct errors on the received data. In block coding, the transmitted data is segmented into blocks of fixed length of K bits. Linear block codes are a special class of block codes where each bit or symbol can be expressed as a linear combination of other bits or symbols in the transmitted data. The encoder then based on certain rules transforms the input segment into an output block of length N. $N > K$, thus providing the redundancy needed for error correction and detection. The rate of the code is expressed as $R = \frac{K}{N}$.

With K bits in an input message, there are 2^K distinct input entries possible. Each output message of N bits associated with each input message is called a *codeword*. With input messages of K bits, an arbitrary encoding would require the encoder to store a table of 2^K entries each of length N. This approach is not practical for large K. Linear block codes reduce the complexity of encoding by using a linear generator matrix to transform inputs to codewords.

A code is a linear code if and only if the modulo-2 (modulo-q in general) sum of two codewords is also a codeword. This property of the code allows the encoder designer to find a generator matrix(G) that defines the code. The generator matrix is made up of K

linearly independent row vectors of size N,$g_1 ... g_K$, such that it can be as expressed as

$$G = \begin{bmatrix} g_1 \\ . \\ . \\ . \\ g_K \end{bmatrix} \tag{2.1}$$

The encoder generates a codeword by multiplying the input vector with the generator matrix, $c = uG$, where c is the codeword and u is the input vector of bits. The encoder has to store only G, thus reducing the space complexity from $2^K \times N$ (arbitrary encoding) to $K \times N$ (size of G). From the generator matrix G, a parity-check matrix H can be derived. The matrices are related by $GH^T = 0$. When a decoder receives a word,y, it checks using H if the word is indeed a codeword. The received word is a codeword if

$$yH^T = 0 \tag{2.2}$$

since $uGH^T = 0$. The decoder uses this expression to detect and correct errors. G can be put in systematic form as $G = [I_K \mid P]$ from which the derived H is

$$H = [-P^T \mid I_{N-K}], \tag{2.3}$$

where P is an $(N - K) \times K$ sub-matrix, and I_{N-K} is the (N-K) identity matrix and P^T is the transpose matrix of P [17]. For binary codewords, $-P^T = P^T$. If H is given, a corresponding G could also be derived from H similarly to the way it is derived from G.

2.2 Low-Density Parity-Check Codes

Low-density parity-check codes are a class of linear block code defined by a sparse MxN parity-check matrix, H [5],where $N > M$ and $M = N - K$. Although LDPC codes can be generalized to non-binary symbols, we consider only binary codes. The parity-check matrix has a small number of '1' entries compared to '0' entries, making it sparse. The number of '1's in a parity-check matrix row is called the *row-weight*, k, and the number of '1's in a column is the *column-weight*, j. A *regular* LDPC code is one in which both row and column weights are constant, otherwise, the parity check matrix is *irregular*.

Row and column weights are much smaller than the matrix dimensions, with row weights greater than column weights. The rate of the parity check or code matrix is the fraction of information bits in the codeword. It is given by $\frac{K}{N} = \frac{N-M}{N} = 1 - \frac{M}{N}$. The number of '1' entries in the parity-check matrix is given by Mk or Nj. From $Mk = Nj$, we get $\frac{M}{N} = \frac{j}{k}$. Hence, the rate of the matrix could also be expressed as $1 - \frac{j}{k}$. We briefly describe LDPC code representation, encoding and decoding and LDPC characteristics and evaluation in the following sections. Detailed introductions and tutorials of LDPC codes are found in [18][19][20].

2.3 LDPC Representation

Although a LDPC code is defined by a sparse matrix, a bipartite graph, also known as a Tanner graph[21], can be used to represent the code. A bipartite graph is a graph whose nodes can be divided into two sets such that each node is connected to a node in the other set. The two sets of nodes in a Tanner graph are called *check nodes* and *variable nodes* representing rows and columns respectively. Figure 2.1 shows a parity check matrix with a corresponding Tanner graph. The a^{th} check node is connected to the b^{th} variable node if and only if $H_{a,b} = 1$. Check nodes $f_0 ... f_5$ represent the six rows of the matrix, whereas $v_0 ... v_{11}$ are the columns. The number of edges in each check node is equal to the row weight and the number of edges in each variable node is equal to the column weight. The row and column weights are four and two respectively in this example.

A cycle in a parity check matrix is formed by a complete path through '1' entries with alternating moves between rows and columns. In a Tanner graph a cycle is formed by a path starting from a node and ending at the same node. The length of the cycle is given by the number of edges in the path. A cycle of six is shown in bold in the graph of Figure 2.1 (b). The smallest cycle in a Tanner graph or parity check matrix is called its *girth*. The smallest possible girth is four. A bipartite graph has a minimum cycle of length four and has even cycle lengths.

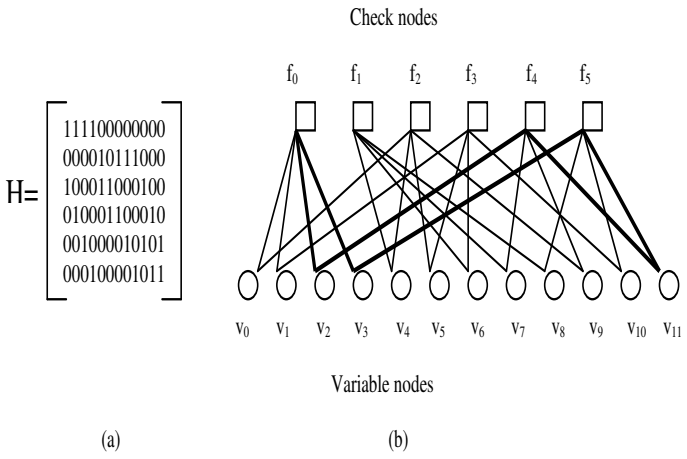

Figure 2.1. LDPC code (a) matrix representation (b) Tanner graph representation.

2.4 LDPC Encoding

LPDC codes encoding is performed in a similar way as in linear codes briefly discussed above. From a given parity-check matrix, H, a generator matrix, G, is derived. Data, $u = u_1...u_N$, is encoded by multiplying it with the generator matrix, $c = uG$, where u is a string of information bits. It has to be noted that putting H in systematic form, $H = [P^T \mid I_M]$, no longer has fixed column or row weights and P is very likely to be dense. The denseness of P determines the encoder computational complexity. A dense generator matrix requires a large number of operations when doing the matrix multiplication with the data to be sent. The encoding process complexity is $O(N^2)$ or more precisely $N^2 \frac{R(1-R)}{2}$ operations where R is the code rate[22]. Encoding complexity could be reduced for some codes by parity-check matrix preprocessing. An efficient encoding technique has been developed to reduce encoding complexity to $O(N)$ by rearranging the parity-check matrix before encoding[22]. The encoding complexity also depends on the structure (row-column interconnections) of the code. Quasi-cyclic codes are codes in which a cyclic shift of one codeword results in another codeword. Their encoding has been shown to be linear with code length thanks to the cyclic row-column connections[23][24].

2.5 LDPC Decoding

LDPC code decoding tries to reconstruct the transmitted codeword, c, from the possibly corrupted received word, y. It is achieved by using the parity-check matrix, H. The condition that $cH^T = 0$ defines the set of parity-check constraints or equations that must be satisfied for the received codeword to be the same as the transmitted codeword. Using the parity-check matrix of Figure 2.1, the parity-check constraints are as follows:

$$\begin{aligned} v_0 + v_1 + v_2 + v_3 &= f_0 \\ v_4 + v_6 + v_7 + v_8 &= f_1 \\ v_0 + v_4 + v_5 + v_9 &= f_2 \\ v_1 + v_7 + v_8 + v_{10} &= f_3 \\ v_2 + v_7 + v_9 + v_{11} &= f_4 \\ v_3 + v_8 + v_{10} + v_{11} &= f_5 \end{aligned} \tag{2.4}$$

If the values assigned to the set of variable nodes represent a valid code then each constraint equation is equal to zero. The equations can be generalized in the form

$$f_a = \oplus_{H_{ab}=1} v_b \qquad a = 1...M, b = 1...N \tag{2.5}$$

where f_a is the a^{th} row of H and v_b is the b^{th} column. The parity check equations are formed from each row of the matrix.

2.5.1 Decoding Algorithm

LDPC code decoding is achieved through iterative processing based on the Tanner graph, to satisfy the parity check conditions. A message passing algorithm (MPA) based on Pearl's belief algorithm [6] describes the decoding iterative steps. The passed messages are probability estimations.

The M check nodes of a Tanner graph correspond to the parity constraints and the N variable nodes represent the data bits of the codeword. The algorithm estimates codewords by iteratively updating and exchanging messages between connected variable and check nodes on the Tanner graph. Check nodes estimate the probability that a given parity check equation is satisfied given the messages or estimates from connected variable

nodes. That is, check nodes probabilities measure the reliability of the bit (data) probability estimations using estimations from adjacent variable nodes. The variable nodes estimate the probability that a given bit is 0 or 1 based on the received bit (codeword) and messages or estimates from connected check nodes. The codeword estimations and message computations are based on the initial values (received codeword) and received messages from adjacent nodes. Each variable node sends a message to each check node it is connected to, and vice versa.
The decoding algorithm is usually implemented in the log domain to simplify computations. Multiplication operations are converted to additions and divisions to subtractions in the log domain. There are other approximations which are used to reduce computational complexity. A summary of the steps and computations of the decoding algorithm in the log domain is presented in the next subsection.

Message Passing Algorithm

The MPA algorithm estimates the bit probabilities using *intrinsic*(knowledge before an event) and *extrinsic* (knowledge after an event) information. For a variable u, there are different types of probabilities to express its relation to an event E. The *a priori* probability of u with respect to the event E is the probability that u is equal to a, and is denoted by

$$P_E^{priori}(u = a) = P(u = a). \tag{2.6}$$

This probability is called a priori because it refers to what was known about the variable u before observing the outcome of the event E. On the other hand, the *a posteriori* probability of u with respect to the event E is the conditional probability of u given the outcome of the event E, and is denoted by

$$P_E^{post}(u = a) = P(u = a \mid E) \tag{2.7}$$

This probability represents what is known about the variable u after observing the outcome of the event E. Using Bayes' theorem [6], the *a posteriori* probability can be written as

$$P(u = a \mid E) = \frac{1}{P(E)} P(E \mid u = a) P(u = a) \tag{2.8}$$

The term $P(E \mid u = a)$ is proportional to what is called the *extrinsic* probability, which describes the new information for u that has been obtained from the event E. The extrinsic probability is denoted as

$$P_E^{ext}(u = a) = dP(E \mid u = a), \tag{2.9}$$

where d is a normalization constant to make the extrinsic probability sum to 1. Therefore, the relationship between a priori, extrinsic and a posteriori probabilities can be written as

$$P_E^{post}(u = a) = P_E^{priori}(u = a)P_E^{ext}(u = a). \tag{2.10}$$

In the binary case, $u = [0, 1]$, it is convenient to express the probability of a binary variable u in terms of a real number called the *log-likelihood ratio* (LLR). Assuming $P(u = 1) = p$, the log-likelihood ratio of u is defined as

$$LLR(u) = log\frac{P(u = 1)}{P(u = 0)} = log\frac{p}{1 - p} \tag{2.11}$$

LLR(u) is positive if $p \geq 0.5$ and negative if $p < 0.5$. Equation (2.8) can be rewritten in terms of log-likelihood ratios as

$$LLR_E^{post}(u) = LLR_E^{priori}(u) + LLR_E^{ext}(u) \tag{2.12}$$

The extrinsic information reflects the incremental gain in knowledge of a posteriori information over a priori information.

The message passing algorithm is based on a priori, extrinsic and a posteriori probabilities. The a priori information is obtained from the channel whereas the extrinsic information is obtained from other nodes using the decoding algorithm. Below are steps and equations for calculating the probabilities in the log domain. For a full explanation of the MPA algorithm including convergence issues refer to [2][15][25].

MPA iterative processes:

1.Initialization: Initialize each variable node with the received information, y, from the source. Each variable node,n, calculates the initial log likelihood ratio (LLR), given by

$$L(u_n) = ln\left\{\frac{P(u_n = 1 \mid y_n)}{P(u_n = 0 \mid y_n)}\right\} \tag{2.13}$$

In the case of an additive white Gaussian noise (AWGN) channel,

$$L(u_n) = 2y_n/\sigma^2, \tag{2.14}$$

where σ^2 is the noise variance and y_n is the received data [26]. $L(u_n)$ is the probability that the sent bit u_n is 1 or 0 given the received bit y_n. For every variable node the initial LLR is given by $L(u_n)$ and messages along edges (to check nodes) are initialized to zero. Check node LLR and messages (to variable nodes) are both initialized to zero. Figure 2.2 shows the initialization and calculation of LLRs and direction of incoming and outgoing messages. Incoming and outgoing messages are exchanged among connected nodes only.

2.Check-node update: For each check node,m, calculate LLR and check-to-variable node messages based on the incoming messages from variable nodes. The check node LLR is given by

$$\lambda_m = \sum_{all-msgs} ln\left\{tanh(\frac{abs(\Omega_{n,m})}{2})\right\}, \tag{2.15}$$

where $\Omega_{n,m}$ represents messages from variable nodes to a given check node. The outgoing check-to-variable messages are given by

$$\Lambda_{m,n} = 2 * atanh\left\{exp\{ln(\lambda_m) - ln(tanh(\frac{\Omega_{n,m}}{2}))\}\right\} \tag{2.16}$$

The sign of λ_m is given by the exclusive-or (XOR) of all the incoming messages and (ANDed with) the sign of $\Lambda_{m,n}$ is given by the sign of λ_m and the sign of the

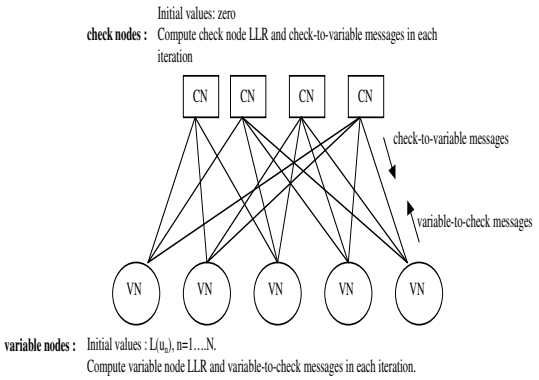

Figure 2.2. MPA calculations on a Tanner graph.

corresponding incoming message($\Omega_{n,m}$)[27].

3.Variable-node update: For each variable node,n, calculate LLR and outgoing messages along its edges to check nodes. The LLR is given by

$$\lambda_n = L(u_n) + \sum_{all-msgs} \Lambda_{m,n}, \tag{2.17}$$

where $\Lambda_{m,n}$ represents a check-to-variable node message. LLR is the sum of all incoming messages plus the initial value of the variable node (equation 2.14). The outgoing messages to check nodes are given by

$$\Omega_{n,m} = \lambda_n - \Lambda_{m,n} \tag{2.18}$$

The outgoing message for each edge is given by the check node LLR minus the message received on that edge.

4.Decision: Quantize the LLR of variable nodes such that $LLR_n = 0$ if $\lambda_n < 0$, and $LLR_n = 1$ if $\lambda_n \geq 0$. If $LLR \times H^T = 0$, then halt the algorithm with LLR at the decoder output. LLR gives the estimation of the codeword, c_n. Otherwise go to step(ii). If the algorithm does not halt within some maximum number of iterations, then declare a decoder failure.

2.6 LDPC Code Design

LDPC code design is determining the basic parameters of a code such as rate and size. These parameters are often determined in consideration of the target application. Below are brief descriptions of some of these parameters and how they affect performance and implementation.

Code size The code size specifies the dimensions of the parity check matrix ($M \times N$). Sometimes the term *code length* is used referring to N. Generally a code is specified using its length and row-column weights in the form (N, j, k). M can be deduced from the code parameters N,j and k. It has been shown that very long codes perform better than shorter ones[7][9]. Long codes are therefore desirable to have good performance. However, their hardware implementation requires more resources(memory plus processing nodes).

Code Weights and Rate The rate of a code, R, is the number of information bits over the total number of bits transmitted. It is expressed as $\frac{N-M}{N}$ or $1-\frac{j}{k}$. Higher row and column weights result in more computations at each node because of many incoming messages. However, if many nodes contribute in estimating the probability of a bit the node reaches a consensus faster. Higher rates mean fewer redundancy bits. That is, more information data is transmitted per block resulting in high throughput. However, low redundancy means less protection of bits and therefore less decoding performance or higher error rate[28]. Low rate codes have more redundancy with less throughput. More redundancy results in more decoding performance. However, very low rates may have poor performance with a small number of connections.
LDPC codes with column-weight of two have their minimum distance (see below) increasing logarithmically with code size as compared to a linear increase for codes with column weight of three or higher[5]. As a result column-weight two codes perform poorly compared to higher column-weight codes. Column weights higher than two are usually used. Although regular codes are commonly used, carefully constructed irregular codes could have better error correcting performance [29][30].

Code Structure The structure of a code is determined by the pattern of connections between rows and columns. The connection pattern determines the complexity of the communication interconnect between check and variable processing nodes in

an encoder and decoder hardware implementations. Random codes do not follow any predefined or known pattern in row-column connections. Structured codes on the other hand have a known interconnection pattern. Many methods have been developed for constructing those types of codes, some of which are described in Chapter 3. New construction methods are introduced in Chapters 3 and 4.

Number of iterations The number of iterations is the number of times the received bits are estimated before a hard decision is made by the decoding algorithm. A large number of iterations may ensure decoding algorithm convergence but will increase decoder delay and power consumption. The number of corrected errors generally decreases with an increasing number of iterations. In performance simulations a large number of iterations, (about 100 to 200), can be used. For practical applications 20 to 30 iterations are commonly used[31][32][33].

2.7 LDPC Optimization and Evaluation

There are several ways of improving the decoding performance of a LDPC code including improving girth, and minimum distance. The improvement in performance also depends on the technique used. There are also performance measures to determine how good a code is in correcting errors. Besides bit error rate simulations, other parameters could be used to predict the performance of a code. Below we describe some of the common techniques used in the literature.

2.7.1 LDPC Code Performance Optimization Techniques

There are several parameters of a LDPC code that could be changed to improve its performance. These parameters include girth, average girth and minimum distance. Here were mention some of the common techniques.

Minimum distance The *Hamming weight* of a codeword is the number of 1's of the codeword. The *Hamming distance* between any two codewords is the number of bits with which the words differ from each other, and the *minimum distance* of a code is the smallest Hamming distance between two codewords. The larger the

distance the better the performance of a code. Very long and large girth LDPC codes tend to have larger minimum distances[7]. A better code could be determined by using minimum distance as a measure. However, for randomly constructed codes, no algorithm is known to efficiently and accurately calculate its minimum distances. This problem was proved to be NP-hard[34]. Algebraic software products such as MAGMA[35] have been used by some researchers to calculate minimum distances of structured codes.

Girth and Average Girth Both girth and average girth affect the decoding performance of a code. Large girth and average girth tend to improve code performance whereas small ones especially of length four degrade performance. The average girth is the sum of smallest cycles passing through nodes divided by the number of nodes. With small cycles a node gets a probability estimate including its own contribution after a few iterations. When the girth is large the estimates are less dependent on the node's contribution for a larger number of iterations, which is the assumption of the MPA decoding algorithm. Sullivan[36] showed using bit error rate simulations that large girth codes perform better than those with lower girths. Mao[37] showed that girth distribution matters more than girth. A code with a larger average girth is likely to outperform a code with a lower average of the same girth. LDPC construction algorithms are used to deliberately look for row-column connections resulting in lager girth codes. In this thesis we develop construction algorithms for obtaining large girth codes in Chapters 3 and 4.

Stopping Sets A stopping set S is a subset of V, the set of variable nodes, such that all neighbors of the variable nodes in S are connected to S at least twice. The size of a stopping set s is defined as the cardinality of S. It has been shown that prevention of small stopping sets improves minimum distance of a code. Prevention of small stopping sets has been used to improve code performance in [38].

Density Evolution Density evolution[15] is an algorithm that tracks the probability density function of the messages through the graph nodes under the assumption that the cycle free hypothesis is verified. It is a kind of belief propagation algorithm with probability density function messages instead of log likelihood ratios messages. As the code length tends to infinity, the bit error probability can be made arbitrarily

small if the noise level is smaller than some constant threshold. By observing the density of messages between nodes the performance of a code is estimated.

2.7.2 Error Rate

Although parameters such as girth could be used as a metric to measure decoding performance, they do not show how much error correction the code can do. Also a code with a better girth or average girth does not guarantee better performance than one with a lower girth or average girth. LDPC codes are often evaluated using bit-error rate (BER) performance over a specified channel and type of modulation. In this thesis all LDPC code performance simulations were performed on an Additive White Gaussian Noise (AWGN) channel with Binary Phase Shift Key (BPSK) modulation. Log domain MPA was used in all simulations.

Channels are described by a mathematical model making it easy to design suitable modulations and coding schemes. The AWGN channel is one of the simplest channel models. It subjects a vector of transmitted bits, u, to some noise in the form of random peaks of energy. The amount of noise at any time instant can be described by a random normally distributed variable, n, such that the channel bits are $y_i = u_i + n_i$, where i is the position of a bit in the signal, noise and received bit vectors. The randomness of the Gaussian noise has a one-sided power spectral density N_o which depends on the noise level or variance,σ^2, by the expression $N_o = 2\sigma^2$.

The BER measures the number of errors ($y_i \neq c_i$) found per iteration over the code length at a given signal-to-noise ratio (SNR). It is expressed as

$$BER = \frac{number\_of\_errors}{number\_of\_bits} \tag{2.19}$$

Errors are bits that are not equal to the sent bits of the transmitted data. The SNR is the power ratio of the signal (transmitted data) and the background or channel noise. A high SNR means the signal is much stronger than noise whereas a low SNR means the noise is significantly close to the signal. In low SNRs the signal can be badly distorted by the noise.The number of errors generally decreases with increasing signal to noise ratio. SNR is defined as $SNR = 10log\frac{E_s}{N_o}$, where E_s is the signal energy. To achieve a sufficient degree of statistical confidence in the BER, the simulation is repeated many times for a given SNR, and the average BER is reported. A plot is then generated with average BER versus SNR. The BER curves show the probability that a bit, after decoding, will

be in error at a particular SNR. Another performance measure related to BER is frame error rate (FER) or Word error rate(WER). FER is the number of decoded words (length of code) that contain errors as a fraction of the total number of words decoded. In applications were it is essential that all of a word is correctly received, FER is preferred over BER.

2.8 LDPC Implementation

LDPC encoding and decoding are mostly done in hardware to meet high throughput required by most applications. Encoding complexity is quadratic with respect to the code length. There are several methods suggested for reducing encoding complexity by preprocessing the parity check matrix. The complexity of the encoder also depends on the structure of the parity check matrix as already stated earlier. Large codes require more hardware in terms of memory communication network and processing nodes.
LDPC decoder memory required depends on the structure of the code and implementation architecture. Interconnection complexity between nodes and large memory are the main difficulties in hardware decoders implementation especially for fully parallel, random and irregular codes. Semi-parallel decoder architectures based on structured codes are often implemented. They offer a better tradeoff between throughput and hardware cost and complexity. Several issues including type of MPA, numeric precision, decoding delay, power consumption, scalability and programmability are considered in designing and implementation of LDPC decoders to suit a particular application. A brief overview of these factors is presented in Chapter 5. In Chapter 6, we discuss use of multistage networks in quasi-cyclic LDPC decoders. Methods of overlapping node computations to reduce decoding delay are introduced and discussed.

2.9 LDPC Applications

LDPC codes have been applied to applications in communication and storage systems. Code design, construction and implementation is dictated by the target applications. Storage systems require very high rate ($\frac{8}{9}$ and higher), low SNRs (7 to 12dB), and very high data rates in the Giga-bits per second range (Gbps) and faster[16]. The codes are

expected to have BER of 10^{-12} to 10^{-15}. In [16][39][40] column-weight two codes are investigated for disk storage because of their low complexity (few edge connections). Low complexity codes such as column-weight two codes or quasi-cyclic codes are important as disk storages are also sensitive to VLSI cost[41]. Communications standards such as digital video broadcasting version 2 (DVB-2) recommend a variety of rates, $\frac{1}{4}$ to $\frac{8}{9}$, and very long code lengths of 3200 and 64800 [12]. LDPC codes are also recommended for other communication environments such as Gigabit Ethernet, wireless broadband and optical communications[13][42][43][44].

This page is blank

Chapter 3

Constructing LDPC Codes

LDPC code construction requires the definition of the pattern of connection between rows and columns of a parity-check matrix or between check and variable nodes of a corresponding Tanner graph. The construction process considers LDPC code parameters such as row and column weights, rate, girth and code length. A code is first designed from which the rate and length are determined before it is constructed. The main objectives in code construction are good decoding performance and easier hardware implementation. A LDPC code could be constructed such that it has low hardware complexity and cost. This is mostly achieved by having row-column connections that have a regular pattern. Good error correcting performance and low complexity hardware characteristics could also be optimized at the same time. However, putting constraints on construction methods to obtain hardware aware codes may degrade or limit decoding performance.

The challenge in LPDC code construction is to obtain a wide range of codes in length and rate that have good decoding performance and are also easy to implement in hardware. For practical purposes the length of the code is constrained. There are numerous ways a code can be realized for a given length and rate. However, developed methods often have limitations in meeting the flexibility in code design and ease of implementation.

Construction methods can be either random(unstructured row-column connections) or structured(row-column connections predefined in some way). Random constructions have flexibility in design and construction but lack row-column connections regularity, which increases decoder interconnection complexity. Structured constructions may have regular interconnection patterns but often produce a class of codes limited in rate, length and girth. There is still need to develop methods that can produce a wide range (rate, length

and girth) of LDPC codes with consideration of performance and implementation factors. In this chapter we review some of the existing constructions methods and suggest two new methods of constructing structured LDPC codes.

3.1 Random Constructions

Random constructions connect rows and columns of a LDPC code matrix without any structure or predefined connection pattern. They are actually pseudo-random connections done by computer searches. Constructions could be done in the Tanner graph by connecting check to variable nodes with edges or in the parity-check matrix by connecting rows to columns with '1' entries where all other entries are '0's. Randomly adding edges to a Tanner graph or '1' entries in the parity-check matrix will not produce a desired rate and will probably have cycles of four. However, the resulting code could be optimized by either post processing or by putting constraints on the random choices as the code is built. Post processing exchanges or deletes some connections in order to get a desired girth and rate. Random construction with constraints add a connection in the code if it does not violate the desired girth or row and column weights.
Random codes have good performance especially at long code lengths compared to structured codes [7][9]. Random construction methods could be used to maximize performance (e.g by girth) and rate for a given size as demonstrated by Campello et al in [45]. In [37] a heuristic algorithm is developed to search for good LDPC codes based on average girth distribution. While random codes show better performance compared to structured or constrained codes at code lengths of several thousands, there is no assurance that a particular code chosen at random will have good performance. Below we look at some random construction methods relevant to this thesis. Detailed examples of random construction methods or algorithms are found in [5][7][29][45][46].

3.1.1 MacKay Constructions

Mackay [7] showed that random LDPC codes have good performance approaching Shannon's limit. He also developed some random construction methods for developing codes some of which are listed below[47].

1. Matrix H is generated by starting with an all zero matrix and then randomly flipping bits in the matrix. Flipped bits are not necessarily distinct.

2. Matrix H is generated by randomly creating weight j columns.

3. Matrix H is generated with weight j per column and uniform weight per row and no two columns are connected to the same row more than once (avoiding four-cycles).

4. Matrix H is generated as in 3 with the girth condition further constrained so that the girth is larger than six.

Mackay's algorithms were used to find good performing codes with a variety of rates and length. Some of the obtained codes are listed in a World Wide Web database in [48].

3.1.2 Bit-Filling Algorithm

A bit-filling (BF) algorithm introduced in [45] constructs a LDPC code by connecting rows and columns of a code one at a time provided a desired girth is not violated. The number of connections to rows and columns are kept almost balanced (about same number of connections) by connecting rows or columns with the least number of connections first. The algorithm obtains irregular codes with either a fixed row or column weight. The algorithm is as follows according to [45]:

Bit-Filling Algorithm

c is check node,

M is the number of check nodes

$\aleph_c$ is a set of check nodes sharing a variable node and

U_1 is a set of check nodes connected to the new or current variable node.

F_0 is a set of check node that can be connected

to U_1 without violating the girth or check-degree constraint.

Set $N = 0$, $A =\mid M \mid$, and $U_1 = \emptyset$

for $c \in \mid M \mid$, set weight(c)=0 and $\aleph_c = \emptyset$

do {

$\forall c \in U_1$, set $H_{c,N} = 1$ and increment deg(c) by 1

set $i = 0, U_1 = \emptyset$, and $U = \emptyset$

do {

compute $F_0 = A \setminus U$

if $(F_0 \neq \emptyset)$ {

choose $c^\star$ from F_0 according to some heuristic

$\forall c \in U_1$, update $\aleph_c = \aleph_c \cup \{c^\star\}$ and update $\aleph_{c^\star} = \aleph_{c^\star} \cup U_1$ update

$U_1 = U_1 \cup \{c^\star\}$, $U = U \cup V_{g/2-1}(c^\star)$, and A }

$i = i + 1$

} while $((i < a)$ and $(F_0 \neq \emptyset))$

n=n+1

} while $((i = a)$ and $(F_0 \neq \emptyset))$

The algorithm can obtain very-high rate and high-girth codes with a computational complexity of $O(kM^3)$. It is extended in [49] to get better girths and rates for a given code length. Although the algorithm produces high-rate and high-girth codes given a particular code size, resulting codes are not easily implementable in hardware. This is because the structure of row-column connections is not consistent enough to be an advantage in hardware implementation. The objective of the algorithm is to optimize girth or rate for a given code size.

3.1.3 Progressive Edge-Growth Algorithm

The progressive edge-growth (PEG) algorithm is another simple non-algebraic algorithm that can be used to construct codes of arbitrary length and rate. It is similar to the bit-filling algorithm. In PEG node degrees are distributed according to some performance criteria (e.g density evolution) before edges are added. The algorithm builds a Tanner graph by connecting the graph's nodes edge by edge provided the added edge has minimal impact on the girth of the graph. With this algorithm regular and irregular codes can be obtained with optimized performance. Codes obtained using this method are one of the best known performing codes at short lengths[46][50]. The algorithm as described in [51] and is as follows:

Progressive Edge-Growth Algorithm

v_l is a variable node l and f_l is check node l.

E_{v_l} is a set of edges connected to variable node v_l

d_{v_l} is the edge degree of variable node l

$\aleph_{v_l}^g$ is a set of check nodes reached by a subgraph spreading from variable node v_l with depth g.

for $l = 0$ to $N-1$ do {

for $t = 0$ to d_{v_l}-1 do {

if t =0 {

$E_{v_l}^0 \leftarrow edge\ (f_i, v_l)$, where $E_{v_l}^0$ is the first edge incident to variable node v_l, and f_i is one check node such that it has the lowest check-node degree under the current graph setting $E_{v_0} \cup E_{v_1} ... \cup E_{v_{l-1}}$

}

else { expand a subgraph from symbol node v_l up to depth g under the current graph setting such that the cardinality of $\aleph_{v_l}^g$ stops increasing but is less than M, or $\overline{\aleph}_{v_l}^g \neq \emptyset$ but $\overline{\aleph}_{v_j}^{g+1} = \emptyset$, then $E_{v_j}^t \leftarrow$ edge (f_i, v_l), where $E_{v_l}^t$ is the k^{th} edge incident to v_l, and f_i is one check node picked from the set $\overline{\aleph}_{v_l}^g$ having the lowest check node degree.

} } }

There are other variations of the algorithm that slightly improve performance of the obtained codes[38][52].
Just as with bit-filling code, a major disadvantage of PEG codes is their high hardware complexity making them impractical at very large lengths. The unstructured interconnection results in routing complexity and congestion in decoder implementations[27]. When row-column connections are done at random, there would be no general rule to define how a set of rows or columns are connected. Therefore the random connections need to be defined by a table (addressing) for each individual row or column or hardwired. Since codes are generally a few or more thousands in size, the tables would also be very large or the number of interconnections will be high and unstructured leading to long wire-lengths, large decoders with slow clock rates.

3.2 Structured Constructions

Structured construction methods put constraints on row-column connections to get a desired or predefined pattern. The main objectives are to achieve good performance and at the same time have a connection pattern that is easier to implement in hardware. There are many structured methods already developed producing codes differing in structure (row-column interconnection pattern), performance and hardware implementation complexity. Developed methods include those based on graphs, combinatorial designs, algebra and heuristic searching techniques. Below we briefly describe some existing structured construction methods relevant to this thesis.

3.2.1 Combinatorial Designs

A combinatorial design is an arrangement of sets of v points into b subsets called blocks. The inclusion or placement of points in blocks is according to some defined constraints. The basic constraints are

1. A pair of points can appear together in λ blocks for a defined value of λ.

2. The number of points in each block is given by γ and the number of blocks in which a point appears is denoted as ρ [53].

A design is balanced if the covalency , λ, is the same for all pairs of points. To form a LDPC code from a combinatorial design, points and blocks are interpreted as rows and columns respectively(or columns and rows) to form an incidence matrix. The dimensions of the code are $v \times b$ (or $b \times v$) which is given by $M \times N$ in our notation. γ and ρ are column weight (j) and row weight (k) respectively. The design is regular if points appear the same number of times and all blocks have the same number of points.

The design ensures that obtained codes do not have a four-cycles by using a covalency of 1 in the first constraint. From a matrix point of view, a four-cycle is formed by having a pair of rows connected to the same columns more than once or having a pair of columns connected to the same rows more than once. The first constraint breaks this condition by requiring that two points appear in the same block only once. That is, $\lambda = 1$. This constraint is known as a row-column (RC) constraint[3]. It is used in different forms in many constructions methods to avoid four-cycles. The example in Figure 3.1 shows a combinatorial design with column weight of two and row weight of three. Part (a) shows the division of points in subsets. An equivalent graph is also shown with points as vertices and edges as blocks. The incidence matrix forms a LDPC code in which rows are connected to a column if they do not belong to the same block. Part(b) of Figure 3.1 shows the derived matrix from the graph. It is an adjacency matrix showing graph vertex connections by a '1' entry in matrix form.

This construction method was used in [54] to construct column weight two codes with girth of eight. The size of the codes is given by $2k \times k^2$, where k is equal to ρ. The same combinatorial design could be applied to blocks with three or more points to obtain codes with column weight three and higher. Examples of such constructions include those based on Kirkman and Steiner triple systems described in [28]and [53]. Codes obtained from these systems have a girth of six which is a limitation in large codes lengths. With large codes decoding performance can be improved by increasing the girth of the code. Although a wide range of rates and lengths are obtained, the code lengths are determined by row and column weights in combinational designs.

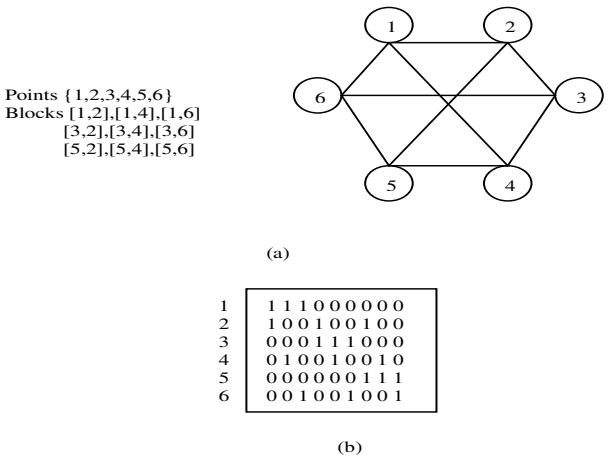

Figure 3.1. Combinational design (a) points and subset arrangement with corresponding graph form (b) incidence matrix.

3.2.2 Finite Geometry

Finite geometries are another approach similar to combinatorics that can be used to design structured LDPC codes. A finite geometry is defined by n points and J lines with the following properties[55]

1. Every line consists of ρ points.
2. Any two points are connected by one and only one line.
3. Every point lies on γ lines.
4. Two lines are either parallel or they intersect at only one point.

Figure 3.2 shows an example of a finite geometry with $n = 4$, $J = 6$, $\gamma = 3$ and $\rho = 2$. An incidence matrix is also shown derived from the geometry. Lines are represented by rows and points by columns. The intersection of a line and a point is represented by a '1' entry in the matrix. When $\rho << n$ and $\gamma << J$ the incidence matrix can be regarded as a low density parity check matrix as the number of '1' entries is very small compared to the number of '0' entries. The example matrix with n as the block length is referred to as type-I geometry LDPC codes (Euclidean geometry or EG-LDPC codes). The transpose of the matrix is a matrix with length of J which is referred to as type-II geometry LDPC code (projective geometry or PG-LDPC codes). γ and ρ are column and row weights respectively in type-I codes and they are row and column weights in type-II codes.

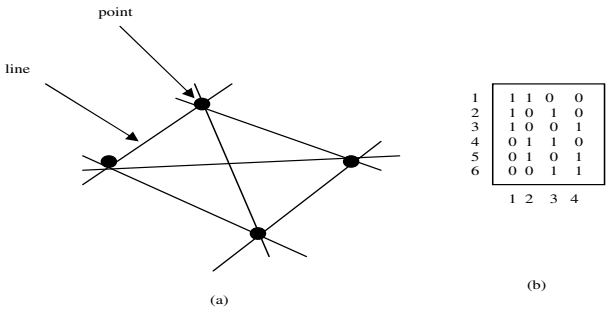

Figure 3.2. (a) Finite geometry with $\rho = 2$ and $\gamma = 3$ (b) corresponding type-I incidence matrix.

Properties 1) and 3) of finite geometry make the codes regular as lines have the same number of points and points are crossed by the same number of lines. Property 2) ensures that obtained codes have a girth of at least six. That is, four-cycles are avoided. As was explained above in combinational designs, a cycle of four is formed if two rows share more than one column or when two columns share more than one row. Property 2) breaks four-cycles since no two points will appear in more than one line at the same time. It was shown however, that these codes will always have a girth of six [55] which is a limitation in terms of improving performance by larger girths. In [55] and [56] finite geometry of the two types were constructed and analyzed. They can be obtained over a wide range of lengths and rates and have good minimum distances. Type-I codes have a minimum distance of at least $\gamma + 1$ and type-II codes of at least $\rho + 1$[55]. Although a wide range of code lengths and rates could be obtained, they could not be chosen arbitrarily. For example the length of a code is determined by its row weight. The length of EG-LDPC codes is $a(p^{ms} - 1)$ with a row-weight of ap^s, where p is a prime and a, m, s are positive integers. The length of PG-LDPC codes is $\frac{a(p^{(m+1)s}-1}{p^s-1}$, and a row-weight of $a(p^s + 1)$[56]. This clearly limits the flexibility of code design and construction.

3.2.3 Algebraic Methods

Parity check matrix connections could be constrained algebraically to obtain codes with a particular structure. Constraints could also be used to get a desired girth, rate or length. Fossorier [57] presents algebraic constraints to obtain quasi-cyclic codes of a given girth. The code matrix is divided into shifted identity sub-matrices of equal sizes. The structure of the matrix is given by,

$$H = \begin{bmatrix} I_{1,1} & I_{1,2} & ... & I_{1,k} \\ . & . & . & . \\ I_{j,1} & I_{j,2} & ... & I_{j,k} \end{bmatrix} \tag{3.1}$$

Fossorier presents algebraic conditions under which cycles are present in a code. The conditions are then used to find codes with girths of six to twelve. A necessary condition to have a girth of at least six is that $h_{j_1,k_1} \neq h_{j_2,k_1}$ for $j_1 \neq j_2$ and $h_{j_1,k_1} \neq h_{j_1,k_2}$ for $k_1 \neq k_2$. This condition basically breaks any four-cycles in the matrix. A four-cycle consists of '1' entries forming a square or rectangle in a matrix. The author also shows that girth-six codes have a length,N,of at least k^2 if k is odd and $k(k+1)$ if k is even. Conditions breaking cycles of six to obtain at least girth eight are also presented. The author also proves that quasi-cyclic codes have a maximum girth of twelve when the number of sub-matrices per row and column is equal to k and j respectively.

The given algebraic conditions are meant to break or avoid cycles of a given length in a matrix. However, the conditions do not determine values of shifts in sub-matrices or size of sub-matrices that will work for a given N,j and k. In the paper, two construction methods are presented to find codes of a particular girth given N,j and k. Random constructions randomly generate shift values that satisfy the girth conditions. As reported in the paper, it takes a long time to find the right combination of shifts that results in minimum (smallest code size) codes. It took long computer searches to get some codes for girths six and eight using an algebraic software program called MAGMA [35] developed at the University of Sydney. A structured construction method is suggested to reduce code search. The method puts more constraints on the girth conditions such that there is less space to search. However, obtained codes are slightly larger than the minimums (theoretical smallest code size).

The advantage of this algorithm compared to other algebraic methods in [36][58] is that it does not put many restriction on the size of sub-matrix group.

There are many other methods in literature for constructing structured LDPC codes. These codes also apply the row-column constraint in some way to avoid four-cycles. Most of them are as good as other codes from the examples given here. They also have limitations in some way.

3.2.4 Advantages of Structured Codes

Structured connections generally reduce hardware complexity and the cost of encoders and decoders[14][59][60]. For example, a code matrix divided into sub-matrices can be mapped onto a decoder with the number of processing nodes equal to the number of row and column sub-matrices. Since there are fewer sub-matrices compared to individual rows or columns, the number of processing nodes and interconnections is also reduced. In most structured codes, several rows or columns have a common connection rule. Hardware implementations of such codes can be simplified in the sense that a group of messages between check and variable nodes can be routed using a single connection rule. If messages are stored in memory blocks, the source and destination addresses are calculated using the connection rule.

Another advantage of structured codes besides ease of implementation is that they could be characterized by their performance profile. Hence, their general performance is known. Construction constraints often produce a family of codes of a similar structure. Code characteristics such as girth, rate and minimum distance may be applied to all codes in the family. A major drawback of structured codes is that performance is degraded compared to random codes for codes longer than 10 000[9]. Constraints may limit the girth, rate and minimum distance of a code. If there is a weakness in the code graph it is replicated because of the regularity in the connections. Another disadvantage is that, they sometimes exist for only a few parameter combinations (rate,girth,length) that may not fit or satisfy some applications. The 'structure' and performance of structured codes depends on the constraints used. However, compared to random codes, structured codes offer the best performance and hardware implementation tradeoffs. One major contribution of this thesis is to construct structured over a wide range of rates, lengths and girths.

In the following sections we introduce two new methods for constructing structured codes. In the first method, we derive column-weight two LDPC codes from existing distance graphs. The second methods constructs codes with a higher column-weight than two using a search algorithm. In both methods a wide range of code rates, lengths and girths are obtained. The structure of the codes also has regular patterns leading to less complex implementations compared to random codes.

3.3 Column-Weight Two Codes Based on Distance Graphs

Gallager[8] has shown that column-weight two codes have minimum distance increasing only logarithmically with code length compared to a linear increase when the column-weight is at least three. Despite the slow increase in minimum distance, column-weight two codes have shown potential in magnetic storage applications[16][39][54]. Their performance has been shown to be good enough for partial response channels and inter-symbol interference (ISI) signals. They also have low computational complexity since there are only two column connections per row.
Construction methods for column-weight two codes are found in [16][39][54][61]. Girth-eight column-weight two codes can be constructed from combinatorial designs[54] as was described in the previous section. Cyclic codes with a girth of twelve are constructed in [39] and applied to partial response channels. These codes are of size $(k^2 - k + 1)$ where $(k - 1)$ is a prime number. Moura et al[62] constructed codes with girths of 16 and 18 based on graphical models for rates $\frac{1}{2}$ and $\frac{1}{3}$ respectively with lengths of 4000. The method also has limited code rates and lengths.

In this section we show how column-weight two codes can be derived from distance graphs. A distance graph is a connected graph of a given number of vertices (m) and vertex degree (k) in which the minimum cycle length between vertices or edges is g. To derive column-weight two codes, vertices of a distance graph are defined to represent rows whereas edges represent columns of a parity-check matrix. Using this representation LDPC codes of various girths and rates are obtained from already known distance graphs. A wide range of codes are obtained using this method compared to previous methods.

3.3.1 LDPC Code Graph Representation

A LDPC matrix is conventionally represented by a Tanner graph. The matrix can also be represented by a non-bipartite or distance graph in which vertices are rows and edges represent columns. With this representation a column is a set of edges forming a complete graph between vertices involved in the column. That is, a column in the matrix is connected to connected vertices(rows) in the graph. In the case of two rows per column (column-weight of two), a complete graph between two vertices is formed by a single

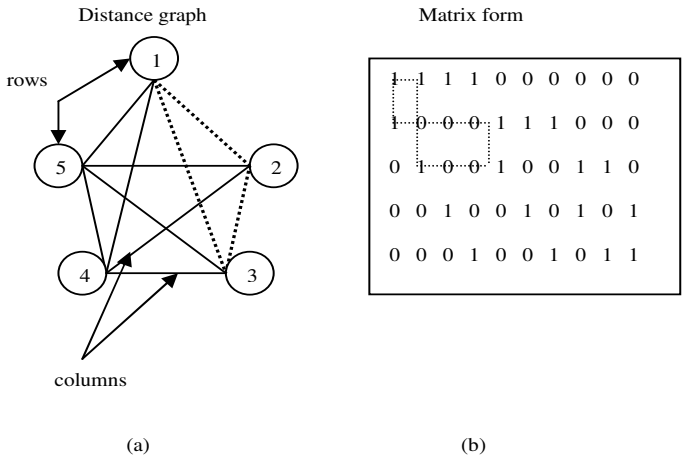

Figure 3.3. A LDPC code matrix derived from a distance graph.

edge. Figure 3.3 (a) shows a distance graph of five vertices and a minimum cycle length of three. Taking each vertex as a row and each edge as a column a corresponding matrix is formed as in part (b) of the figure. A column is connected to two rows that are connected in the graph. The connections are represented by '1' entries in the matrix. That is, if vertices, v_x and v_y, are connected in the graph, then $H_{x,z} = H_{y,z} = 1$ for some column z. The number of rows is equal to the number of vertices in the graph whereas the number of columns is equal to the number of edges. In general the size of the code is given by $m \times \frac{mk}{2}$, where $\frac{mk}{2}$ is the number of edges. mk is divided by two because each edge is shared by two vertices. The rate of a code is given by $1 - \frac{j}{k}$, hence for these codes the rate is $1 - \frac{2}{k}$.

A cycle in a distance graph is formed by a path of edges or vertices starting from a vertex, v_x and ending at v_x. No more than two vertices forming the cycle belong to the same column. The girth, g, is the smallest cycle in the graph. A cycle of length of g in the graph corresponds to a cycle of length $2g$ in matrix form. In the graph we calculate the length using either vertices or edges only. In matrix form a cycle alternates between rows and columns. Therefore, the graph cycle represents half of the matrix cycle. A cycle of three in the example graph is shown in dotted lines between vertices 1,2 and 3. It forms a cycle of six between rows 1,2,3 and columns 1,2,5 in matrix form as shown.
Using distance graphs of different sizes, vertex degrees and girths we can form column-weight two codes of different lengths, rates and girths. The following subsections show sizes (number of vertices) of some known distance graphs from the literature and sizes of derived codes.

3.3.2 Cages

Distance graphs could be regular with vertices of the same edge degree or irregular with vertices of different edge degrees. Both regular and irregular graphs can be used in a similar way to construct LDPC codes. There are several methods described in [63][64] that construct distance graphs of girth g, given an arbitrary number of vertices m and the average number of vertex edges k. These algorithms achieve girths in the order of $log_{k-1}(m)$. There are also known distance graphs called cages that have higher girths with a lower number of vertices compared to other graphs. A (k, g)-*cage* is a k-regular (uniform vertex degree of k) graph of girth g with the fewest possible or known number of vertices (m). The lower bound (Moore bound) on the number of vertices for a given k and g depends on whether g is odd or even[65][66].
If g is odd then

$$m(k, g) = 1 + k + k(k-1) + ... + k(k-1)^{\frac{g-3}{2}} \tag{3.2}$$

and if g is even, then

$$m(k, g) = 1 + k + k(k-1) + ... + k(k-1)^{\frac{g}{2}} - 2 + (k-1)^{\frac{g}{2}} - 1 \tag{3.3}$$

However, these bounds are met very infrequently[67]. Although there is no uniform approach to constructing arbitrary cages, many cages have been constructed for some vertex degrees and girths. A discussion on algebraic methods for constructing cage graphs is beyond the scope of this thesis. They are described elsewhere in cited references and references within. Meringer[68] presented ways of generating regular graphs and cages. There is also an associated software by the same author at [69] that generates some cages.

Cubic Cages

Cages with a vertex degree of three are called cubic cages. Table 3.1 shows the number of vertices for some of the known cubic cages found in [70]. It also shows corresponding derived LDPC code sizes and their girths. Cubic cage construction methods could be found in [65][71][66] and [70]. These graphs produce a parity-check matrix with girth twice the corresponding graph girth, column weight of 2 and rate $\frac{1}{3}$. As shown in the table, obtained girths are very large.

Table 3.1. Sizes of some known cubic cages with corresponding code sizes, girths and rates.

(k, g)	$m(k, g)$	code size$(m \times \frac{mk}{2})$	girth $(2g)$	rate $(1 - \frac{2}{k})$
(3,8)	30	30×45	16	$\frac{1}{3}$
(3,9)	58	58×87	18	$\frac{1}{3}$
(3,10)	70	70×105	20	$\frac{1}{3}$
(3,11)	112	112×168	22	$\frac{1}{3}$
(3,12)	126	126×189	24	$\frac{1}{3}$
(3,13)	272	272×408	26	$\frac{1}{3}$
(3,14)	384	384×576	28	$\frac{1}{3}$
(3,15)	620	620×930	30	$\frac{1}{3}$
(3,16)	960	960×1440	32	$\frac{1}{3}$
(3,17)	2176	2176×3264	34	$\frac{1}{3}$
(3,18)	2640	2640×3960	36	$\frac{1}{3}$
(3,19)	4324	4324×6486	38	$\frac{1}{3}$
(3,20)	6048	6048×9072	40	$\frac{1}{3}$
(3,21)	16028	16028×24042	42	$\frac{1}{3}$
(3,22)	16206	16206×24309	44	$\frac{1}{3}$

Cages of Higher Vertex Degrees

Cages of higher degrees are harder to construct[67]. However, there are many examples of these cages and some construction algorithms[68][72] in the literature. Table 3.2 shows the number of vertices for some of the known high vertex degree cages found in [70][72]. Derived code sizes, girths and rates are also shown. Corresponding LDPC code matrices from these graphs have higher rates but lower girths compared to cubic cages. The higher rates are due to higher vertex degrees.

3.3.3 Code Expansion and Hardware Implementation

Some cage graphs are small in size resulting in practically small dimension LDPC codes. Small codes have low decoding performance as already stated. One method of obtaining larger dimension codes is to expand the code obtained from cage graphs. The codes can be

Table 3.2. Some of known cages graphs with vertex degree higher than three.

(k, g)	$m(k, g)$	code size($m \times \frac{mk}{2}$)	girth $(2g)$	rate($1 - \frac{2}{k}$)
(4,9)	275	275×550	18	$\frac{1}{2}$
(4,10)	384	384×768	20	$\frac{1}{2}$
(5,7)	152	152×380	14	$\frac{3}{5}$
(5,8)	170	170×425	16	$\frac{3}{5}$
(6,7)	294	294×882	14	$\frac{2}{3}$
(6,8)	312	312×936	16	$\frac{2}{3}$
(7,5)	50	50×175	10	$\frac{5}{7}$
(7,6)	90	90×315	12	$\frac{5}{7}$
(8,8)	800	800×3200	16	$\frac{3}{4}$
(9,8)	1170	1170×5265	16	$\frac{7}{9}$
(10,8)	1640	1640×8200	16	$\frac{4}{5}$
(12,8)	2928	2928×17568	16	$\frac{5}{6}$
(14,8)	4760	4760×33320	16	$\frac{6}{7}$

expanded by replacing each '1' entry in the matrix by a shifted $p \times p$ identity sub-matrix and each '0' by a $p \times p$ zero sub-matrix. The resulting matrix size is p times the original matrix. The girth is at least that of the original matrix.

The codes obtained from cage graphs are structured in that their connections can be defined by some defined rules (algebraic construction methods). However, their structures vary from graph to graph and so does the complexity of their hardware implementation. Figures 3.4 shows a (6,4) cage with a corresponding matrix respectively. In this graph an odd vertex is connected to all even vertices and an even vertex to all odd vertices. In fact, all $(k, 4)$ cages are formed this way.

The matrix could be arranged such that rows are separated into two groups and columns into k groups. The connections could also be arranged such that they form cyclically shifted identity sub-matrices as shown in Figure 3.5. The matrix comprises of 6×6 shifted twelve identity sub-matrices. The structure of this code is easier to implement compared to a random structure. Rows and columns of this code could be divided into groups of equal sizes (6) with two row groups and six column groups. The groups can be mapped to an architecture with the same number of processing nodes. With two row

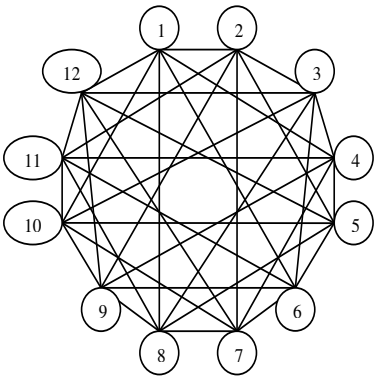

Figure 3.4. A (6,4) cage graph.

100000	100000	100000	100000	100000	100000
010000	010000	010000	010000	010000	010000
001000	001000	001000	001000	001000	001000
000100	000100	000100	000100	000100	000100
000010	000010	000010	000010	000010	000010
000001	000001	000001	000001	000001	000001

100000	000001	000010	000100	001000	010000
010000	100000	000001	000010	000100	001000
001000	010000	100000	000001	000010	000100
000100	001000	010000	100000	000001	000010
000010	000100	001000	010000	100000	000001
000001	000010	000100	001000	010000	100000

Figure 3.5. A matrix representation of (6,4) cage graph.

groups and six column groups, the interconnection complexity is reduced compared to when individual rows and columns are mapped as in random codes. Connections within row and column groups are also cyclic leading to simple linear addressing of messages between row and column processing nodes. However, other cage structures are not so uniform in their connections. Figure 3.6 shows a (4,5) cage graph which does not have uniform connections between vertices. Therefore, the ease of implementation varies from graph to graph.

Column weight two codes also have low computational and memory requirements in variable nodes. Variable nodes calculate LLR by summing the intrinsic LLR and the incoming messages by equation 2.13. Outgoing messages are calculated by subtracting incoming message. Since there are only two incoming messages outgoing messages are calculated by swapping incoming messages in equation 2.14.

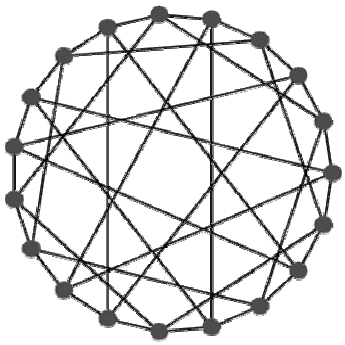

Figure 3.6. (4,5) cage graph with 19 vertices.

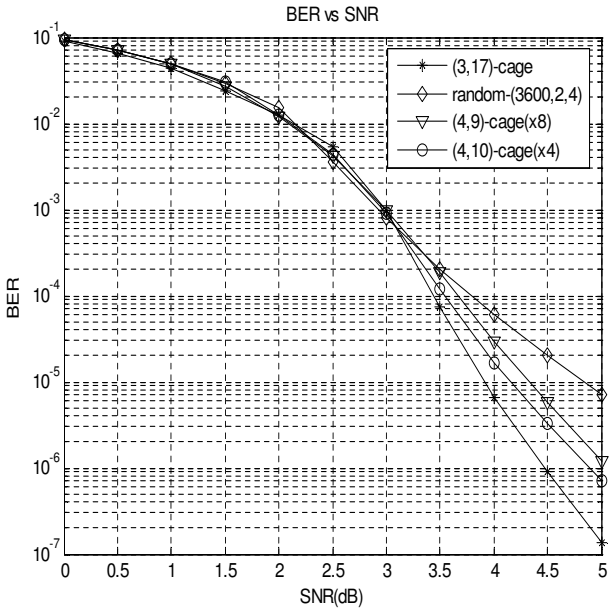

Figure 3.7. BER performances of very high girth LDPC codes constructed from cage graphs (25 iterations).

3.3.4 Performance Simulations

Decoding performances of codes obtained from cages was measured by BER simulations on AWGN channel with BPSK modulation. The codes show BER performances approaching 10^{-6} between 5 and 6dB for some codes. Figure 3.7 shows performance curves for codes derived from (3,17), (4,9) and (4,10) cage graphs, where the (k, g) format is used. k is the vertex degree and g is the graph girth. The codes from (4,9) and (4,10) cages are expanded by 8 and 4 respectively as explained in Section 3.4.3. The codes perform better than a random code free of four-cycles. Random codes have been found to generally perform better than structured codes. The random code, constructed using MacKay's

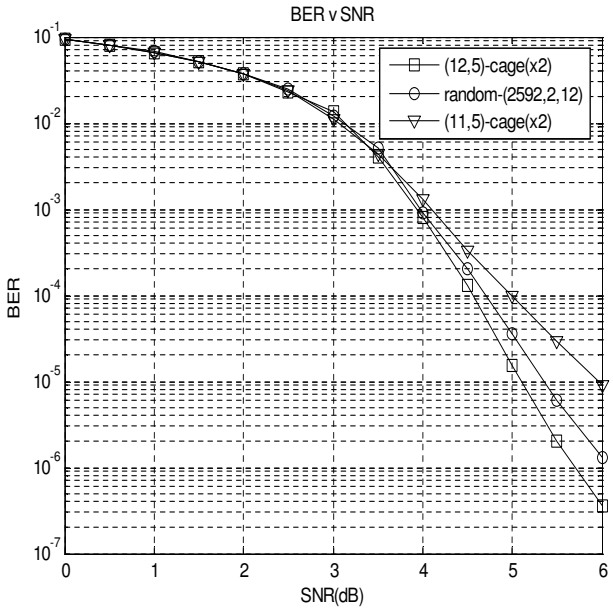

Figure 3.8. BER performances of some $(k, 5)$-cages LDPC codes (25 iterations).

algorithm, has a girth of eight. The performances of the structured codes is according to their girths with higher girths codes performing better. The (3,17)-cage LDPC code has the best performance which could be attributed to its large girth of 34 compared to girths of 20 and 18 for the (4,10) and (4,9) cage graphs respectively.

Figure 3.8 shows performances of two codes with higher rates derived from a family of $(k, 5)$ cages. Both codes are expanded by a factor of 2. The code from the (12,5) cage performs better than that from the (11,5) cage and a random code of about the same size free of four-cycles. Although the $(11, 5)$ cage code has lower rate it performs poorly compared to the $(12, 5)$ cage code. Performance differences between (12,5) and (11,5) cages may be attributed to structural differences of the graphs.

3.4 Code Construction Using Search Algorithms

In the previous section distance graphs were used to construct column-weight two codes. To construct codes of column weight higher than two, we need distance graphs in which a column is formed by a complete graph with more than two vertices. We could not find many such graph constructions in the literature.

Random search algorithms BF and PEG (section 3.1) have flexibility in designing codes

of a desired girth, rate and length. We adopt their strategy in order to have flexibility in constructing structured codes. In this section we construct structured codes with higher column weights than two using a search algorithm.

3.4.1 Proposed Search Algorithm for Structured Codes

We add more constraints to the BF and PEG algorithms to introduce some structure in the obtained codes. Rows are divided into equal-sized groups, $RG_1 ... RG_j$. A column is formed by j rows, $r_1 ... r_j$, one from each row group. $r_1 ... r_j$ rows are a distance of at least g from each other and each row has k connections. The distance between any two rows is given by the minimum number of edges between the rows. It can also be defined as the shortest path between the rows. Rows that meet the distance of g from each other are searched sequentially in each group. With these constraints, the code connections are structured in that for each column we know in which row groups its connections are and vice versa. Rows from different row groups are connected together forming a column. The algorithm forms a distance graph in which vertices are rows and edges form columns. The algorithm is described below.

Sequential Search Algorithm

r_x is row x. $\cup_{r_x}$ is a set of rows within a distance of g from r_x.

1. Divide rows into j (column weight) groups $(RG_1, ...RG_j)$ of equal sizes of p. It is assumed $\frac{M}{j}$ is an integer. If the group size is not given start with a theoretical minimum number of rows if known otherwise start with a group size of k (row-weight). A pair of rows is connected at most once to avoid four-cycles, therefore at least k rows are required in each group.

2. Connections can be done in two ways.

   (a) *Type I connections:*
   $\forall r_i \in RG_1$ do {

   $r_i \in RG_1$
   for $z =$ to k {
   find $(r_a...r_d) \in (RG_2...RG_j)$ respectively where $r_x \notin \sum_{y \neq x}(\cup_{r_i}, \cup_{r_a}...\cup_{r_d})$
   else the algorithm fails.
   connect r_i to $(r_a...r_d)$. } }

   (b) *Type II connections:*
   For $i = 1$ to k {

   $r_i \in RG_1$
   find $(r_a...r_d) \in (RG_2...RG_j)$ respectively where $r_x \notin \sum_{y \neq x}(\cup_{r_i}, \cup_{r_a}...\cup_{r_d})$
   else algorithm fails.
   connect r_i to $(r_a...r_d)$ }

3. Construct the code matrix using the row connections made. The number of columns or code length is equal to the number of connections made

The proposed algorithm attempts to find a code with a given girth, g, and rate (defined by j, k) given the code size,M and N. The algorithm may fail to find the code, if it does not find any rows that satisfy the girth or rate condition. The girth condition is sufficient to obtain a desired girth. That is, if rows are found satisfying the desired distance from each other, then the resulting code will have the desired girth. In *type I connections* (step 2 (a) of algorithm) the algorithm makes k connections to each row in row-group 1 before moving to the next row. In *type II connections* (step 2(b) of algorithm), the

algorithm makes a single connection to each row in row-group 1 each time. The two types of connections result in different code structures.

The complexity of the algorithm is analyzed as follows. Given M rows, there are j row groups of size $\frac{M}{j}$. Each connection at a row in group 1 searches other groups once. Taking each row search as one operation, there are $(j-1)\frac{M}{j}$ searches that can be made. Before searching, neighbors of each row at least a distance of g in non-reference groups $(RG_2...RG_j)$ are updated. Updating row neighbors takes g operations and updating all rows takes $g(j-1)\frac{M}{j}$ operations. Making k connections for a row requires $kg(j-1)\frac{M}{j}$ operations. The total number of operations for all the rows in group 1 (reference row) is $kg(j-1)\frac{M^2}{j^2}$. Therefore the complexity of the algorithm is O(M^2).

The proposed algorithm was used to obtain codes with girth of at least six. From experiments we obtained regular girth six and eight codes for various rates and lengths. Regular codes were obtained with column weights of 3 and 4 for girth six codes. With girth eight codes only a column weight of 3 produced regular codes. Girth-ten and higher codes were all irregular. Below we describe regular girth six and eight codes that were obtained.

3.4.2 Girth-Six Codes

To obtain girth-six codes no two rows should be connected to the same column more than once. As the algorithm forms a distance graph, it searches for rows that have not been connected together before. Figure 3.9 shows row connections for two girth-six LDPC codes obtained using type I connections (see step 3 of algorithm). Each triplet shows three (j) rows that are connected to the same column. The number of rows connected is j and each row has k connections. The number of rows is equal to M and the number of row connection triplets is equal to the number of columns (N). In part(a) of Figure 3.9, a (16,3,4) code is formed with a group size of 4 (4 rows in each group). In part (b) a group size of 5 is used which result in a code size of (20,3,4). By increasing group sizes larger codes could be obtained. Figure 3.10 shows construction of girth-six codes using type II connections. Part (a) is a (20,3,4) code with row group size of 5. A larger code is obtained in part (b) with a group size of 6. With type II connections, cyclic codes are formed with groups shifted relative to each other. Larger dimension codes could also be obtained by increasing the group size. Table 3.3 shows minimum group sizes obtained with column-weights of three and four. For column-weight three codes the observed minimum group

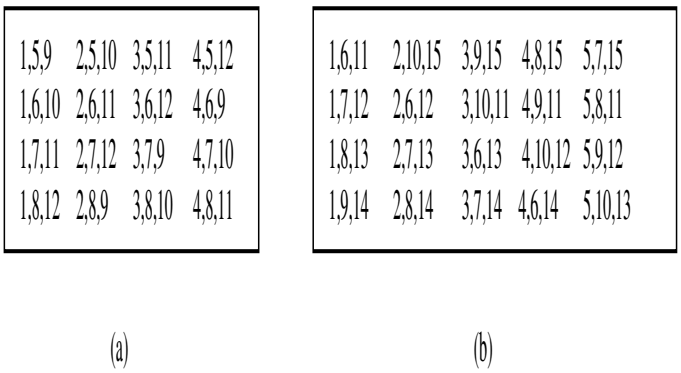

Figure 3.9. Column formations for column-weight three girth six codes using type I connections (a) (16,3,4) code (b) (20,3,4) code.

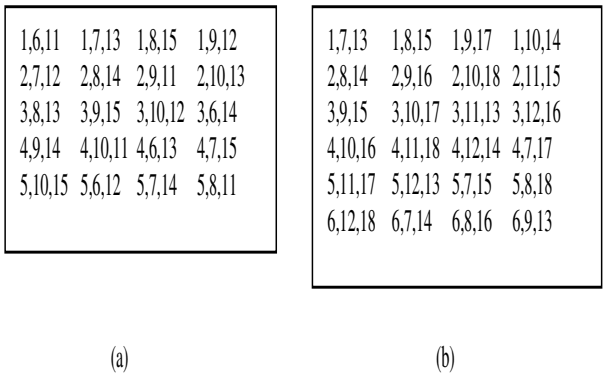

Figure 3.10. Column formations for column-weight three girth six codes using type II connections (a) (20,3,4)code (b) (24,3,4) code.

size is k for type I connections. In type II the group size is k and $k+1$ for odd and even k respectively. No known general group size was observed for column-weight four codes.

Column-weight four codes were also obtained using the two types of connections. Figure 3.11 (a) shows row connections obtained using the first connection type for a (45,4,5) LDPC code. Part (b) is a (25,4,5) code obtained using the second type of connections. Table 3.3 shows minimum group sizes obtained for some row weights for both cases. Larger dimension codes can also be obtained by using larger group sizes in both connections.

1,10,19,28	2,15,24,33	3,11,21,31	4,16,26,36	5,12,23,34
1,11,20,29	2,16,25,34	3,12,22,32	4,17,27,28	5,13,24,35
1,12,21,30	2,17,26,35	3,13,23,33	4,18,19,29	5,14,25,36
1,13,22 ,31	2,18,27,36	3,14,24,34	4,10,21,32	5,15,26,28
1,14,23,32	2,10,20,30	3,15,25,35	4,11,22,33	5,16,27,29
6,17,19,30	7,13,25,28	8,18,21,33	9,14,27,31	
6,18,20,31	7,14,26,29	8,10,23,36	9,15,19,32	
6,10,22,34	7,15,27,30	8,11,24,28	9,16,20,23	
6,11,23,35	7,16,19,31	8,12,25,29	9,17,21,34	
6,12,24,36	7,17,20,32	8,13,26,30	9,18,22,35	

(a)

1,6,11,16	1,7,13,19	1,8,15,17
2,7,12,17	2,8,14,20	2,9,11,18
3,8,13,18	3,9,15 ,16	3,10,12,19
4,9,14,19	4,10,11,17	4,6,13,20
5,10,15,20	5,6,12,18	5,7,14,16
1,9,12,20	1,10,14,18	
2,10,13,16	2,6,15,19	
3,6,14,17	3,7,11,20	
4,7,15,18	4,8,12,16	
5,8,11,19	5,9,13,17	

(b)

Figure 3.11. Column formations for column-weight four girth six codes (a) type I connections (b) type II connections.

3.4.3 Girth-Eight Codes

To obtain a code with a girth of eight, rows in the same column must have a distance of at least four in the distance graph. Figure 3.12 shows column formations for $k = 4$ using type I connections. The graph structure of the code is shown in Figure 3.13. Each row triplet is represented by three vertices connected by two edges. The third edge is not shown, for clarity. According to the non-bipartite graph representation (distance graph), a column is formed by completely connecting three vertices (rows). From the figure we observed that row groups 2 and 3 form a bipartite graph with each group divided into 4 sub-groups of size 4. Each row from group 1 is connected to all rows in one of the sub-groups in groups 2 and 3. The graph structure could be extended to any k by changing the number of sub-groups to k each with k rows. Then the general code size can be expressed as $3k \times k^3$ for all row weights of k. Larger dimension regular codes are obtained by extending each group by sub-groups of k rows. That is, larger dimension codes are obtained with row group sizes larger than k^2 and are multiples of k. With type II connections minimum code sizes obtained are as shown in Table 3.4.

The proposed algorithm was modified such that the search for rows starts from the beginning of the group each time in step 2. With this modification we obtained girth-six codes of the same structure and size as before. Girth-eight codes are also of the same

1,17,33	2,21,37	3,25,41	4,29,45	5,17,37	6,21,41	7,25,45	8,29,33
1,18,34	2,22,38	3,26,42	4,30,46	5,18,38	6,22,42	7,26,46	8,30,34
1,19,35	2,23,39	3,27,43	4,31,47	5,19,39	6,23,43	7,27,47	8,31,35
1,20,36	2,24,40	3,28,44	4,32,48	5,20,40	6,24,44	7,28,48	8,32,36
9,17,41	10,21,45	11,25,33	12,29,37	13,17,45	14,21,33	15,25,37	16,29,41
9,18,42	10,22,46	11,26,34	12,30,38	13,18,46	14,22,34	15,26,38	16,30,42
9,19,43	10,23,47	11,27,35	12,31,39	13,19,47	14,23,35	15,27,39	16,31,43
9,20,44	10,24,48	11,28,36	12,32,40	13,20,48	14,24,36	15,28,40	16,32,44

Figure 3.12. Column formations for a girth eight (64,3,4) code.

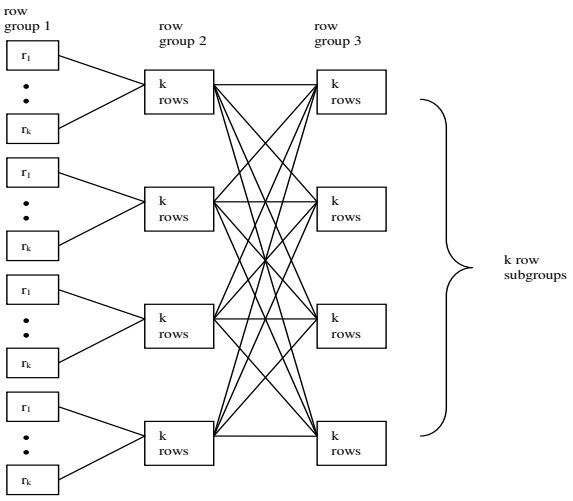

Figure 3.13. Column formations graph structure for girth eight codes for (N,3,k) codes.

structure and size but with an increment of k^2 for larger dimension codes when $k = 3$. For column-weight four, regular codes were obtained for $k = 4, 8$ and 32 only, for $k \leq 40$. Larger regular codes were obtained with multiples of k^2. Girth-ten and twelve codes are of different structure than the original algorithm but still we could not obtain regular codes.

Another modification made was using two groups for column weight three codes. In this case, a group 1 size of $\frac{k^2+k}{2}$ and a group 2 of size $k^2 + k$ produced regular codes of girth eight with row weight of k. Larger groups of multiples of $\frac{k^2+k}{2}$ also produced regular codes. The groups sizes were observed by experiment. Other modifications could be made to the algorithm to obtain codes of different connection structures. For example, random searches could be made with each connection as was done in the original bit-filling and progressive-edge growth algorithms.

Table 3.3. Column-weight four girth-six minimum group sizes.

k j	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3(connection I)	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3(connection II)	5	5	7	7	9	9	11	11	13	13	15	15	17	17
4(connection I)	9	11	13	15	17	19	21	23	25	27	29	31	33	35
4(connection II)	5	7	7	10	11	13	13	13	13	16	17	17	17	19

Table 3.4. Column-weight three girth-eight minimum group sizes using type II connections.

k j	4	5	6	7	8	9	10	11	12
3(connection II)	9	19	21	25	27	55	61	63	67

3.4.4 Performance Simulations

Codes obtained using the search algorithm were simulated for BER performance on an AWGN and BPSK modulation. Figure 3.14 the shows the performance of regular girth-six and eight codes. The codes have very poor decoding performances despite the lack of four-cycles. The poor performance may be due to their connection structure. Gallager[5] has shown than random connections do better than structured ones. The sequential search results in very poor performing codes. A random code of the same size and girth six developed using Mackay's algorithm performs far better than the obtained codes. Obtained codes are outperformed by 2dB at 10^{-4} BER by a random matrix with a girth of six. This results show that the structure of a code also plays an important role in decoding performance. As already stated, it has been shown that random connections result in better codes. In the proposed construction method, connections are made sequentially to the next available row. Despite the high girths of six and eight obtained code perform very poorly because of their connections pattern.

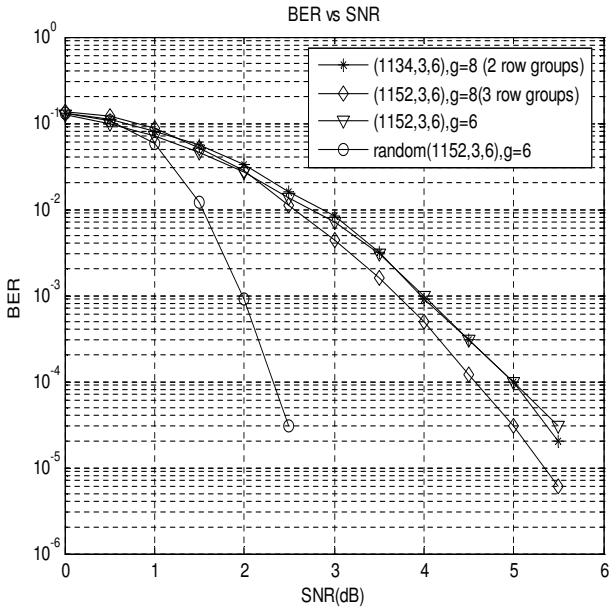

Figure 3.14. BER performances of obtained codes with 25 iterations.

3.5 Summary

Since the rediscovery of LDPC codes, many researchers have been searching for ways of constructing good performing codes. However, construction of LDPC codes should also take hardware complexity and cost into consideration to meet application system hardware requirements. Random constructions generally result in good performing codes compared to structured codes. Methods such as BF and PEG could be used to maximize code performance, rate and girth. They are also flexible in satisfying parameters such as girth, rate and length. However, their unstructured interconnections have high hardware complexity and cost. Structured methods on the other hand constrain parity-check matrix row-column connections such that obtained codes have a defined connection pattern. These methods produce a family of codes which can be characterized by performance, rates, length or girths. They could also be used to construct codes which are easily implementable in hardware. Such methods include combinatorics, finite geometry and algebraic methods. Four-cycles are avoided by observing the row-column constraint. However, construction methods can not obtain codes of arbitrary lengths and rates.

Two methods were developed in this chapter for constructing structured codes. In the first method, column-weight two codes were derived from known distance graphs. Codes

with a wide range of girth, rate and length were derived from known distance graphs. BER performance of the obtained codes depends on the structure of the corresponding distance graph. The ease of hardware implementation also depends on the structure of the distance graph. In some cases graph connections could be rearranged to produce a more hardware aware structure.

We then presented a search algorithm based on BF and PEG algorithms to obtained codes of higher column weights. Rows of a code are divided into groups of equal sizes. Regular codes were obtained for girths of six and eight. Girth-ten and twelve codes were all irregular codes. Obtained codes have some structure in that rows and columns are divided into groups and rows or columns in a group are connected to the same groups. Though obtained codes are free of four-cycles they show very poor BER performance. The poor performance may be due to the sequential search connections.

Chapter 4

Constructing Quasi-Cyclic LDPC Codes

Structured LDPC codes differ in performance and implementation complexity. Check and variable node interconnection patterns or code structure affects decoder communication interconnection complexity. Some structured codes obtained in the previous chapter do not have the same pattern of connections for all rows or columns. Codes obtained from proposed methods in the previous chapter have some defined structure. Distance graph connections are determined algebraically whereas in the search algorithm rows and columns are divided into sub-matrices. However, the connection pattern of individual rows or columns in codes derived from distance graphs and in sub-matrices of searched codes is not the same for all rows or columns. Hence the row-column interconnections are defined by several or many patterns. The more row-column interconnection patterns or criteria the more data the corresponding decoder has to store and manage. Hence many interconnections patterns generally increases complexity of a decoder.

Quasi-cyclic (QC) LDPC codes are codes in which rows or columns in a sub-matrix have similar and cyclic connections. Due to the quasi-cyclic structure, QC-LDPC codes can be encoded efficiently with shift registers[23][24][73] and their decoder architectures require simple address generation mechanisms, less memory and localized memory accesses[74]. Row-column connections in QC-LDPC codes are constructed by shifting identity sub-matrices. Knowing the location of one row or column in a decoder memory one can deduce locations of the remaining rows or columns in the same sub-matrix. Quasi-cyclic decoder implementations will be discussed in more detail in Chapter 6.

A QC-LDPC code can be simply represented by shift values of all of its sub-matrices. This provides a compact representation of the matrix and easy construction. Although row-column connections are constrained QC-LDPC codes can perform close to the capacity limit as demonstrated in [75].
There are several exisiting methods for constructing QC-LDPC codes. In this chapter we use a search algorithm similar to BF and PEG algorithms to obtain quasi-cyclic LDPC codes. We search for connections with large girth to improve decoding performance. The proposed algorithm is flexible in that it can be used to construct codes over a wide range of rates, girths and lengths comparable to random constructions. Another major advantage of the proposed algorithm is that, it can be applied to different sub-matrix configurations or arrangements. This property is very important when codes are constructed such that they have a reduced decoder complexity and delay as shown later in Chapter 6. Performance simulations show that constructed codes can perform as well as random codes at short lengths. Our algorithm could be used to construct codes optimized for decoding performance, hardware implementation or both. It offers the flexibility offered by random codes at the same time producing hardware aware codes.

4.1 Quasi-Cyclic LDPC Codes

A QC-LDPC code can be formed by a concatenation of circularly shifted sub-matrices with or without zero sub-matrices. The codes have a general structure as

$$H = [A_1, A_2..., A_k], \tag{4.1}$$

where A_i is a circulant matrix. Such structures can be obtained with combinatorial construction and finite geometry methods[56][55] which were described in the previous chapter. Other construction methods result in matrices consisting of isolated shifted identity sub-matrices as in Figure 4.1 or equation 3.1. In Figure 4.1 $\mathbf{I}_{xy}$ is a $p \times p$ shifted identity sub-matrix whereas $\mathbf{O}$ is a $p \times p$ zero sub-matrix, where p is a positive integer. A shifted identity sub-matrix is obtained by shifting each row of an identity sub-matrix to the right or left by some amount. There are p such shifted matrices for a $p \times p$ identity matrix. Figure 4.1 shows arrangement of sub-matrices for a code of rate $\frac{1}{4}$. In part (a) all sub-matrices are shifted identity sub-matrices whereas in part(b) the code is made up of shifted identity and zero sub-matrices. In (a) the number of sub-matrices in a row is equal

$\mathbf{I}_{11}$	$\mathbf{I}_{12}$	$\mathbf{I}_{13}$	$\mathbf{I}_{14}$
$\mathbf{I}_{21}$	$\mathbf{I}_{22}$	$\mathbf{I}_{23}$	$\mathbf{I}_{24}$
$\mathbf{I}_{31}$	$\mathbf{I}_{32}$	$\mathbf{I}_{33}$	$\mathbf{I}_{34}$

(a)

$\mathbf{I}_{11}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{I}_{12}$	$\mathbf{I}_{13}$	$\mathbf{O}$	$\mathbf{I}_{14}$
$\mathbf{O}$	$\mathbf{I}_{21}$	$\mathbf{I}_{22}$	$\mathbf{I}_{23}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{I}_{24}$	$\mathbf{O}$
$\mathbf{I}_{31}$	$\mathbf{I}_{32}$	$\mathbf{O}$	$\mathbf{I}_{33}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{I}_{34}$
$\mathbf{O}$	$\mathbf{I}_{41}$	$\mathbf{I}_{42}$	$\mathbf{O}$	$\mathbf{I}_{43}$	$\mathbf{I}_{44}$	$\mathbf{O}$	$\mathbf{O}$
$\mathbf{O}$	$\mathbf{O}$	$\mathbf{I}_{51}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{I}_{52}$	$\mathbf{I}_{53}$	$\mathbf{I}_{54}$
$\mathbf{I}_{61}$	$\mathbf{O}$	$\mathbf{O}$	$\mathbf{I}_{62}$	$\mathbf{I}_{63}$	$\mathbf{O}$	$\mathbf{I}_{64}$	$\mathbf{O}$

(b)

Figure 4.1. Quasi-cyclic code sub-matrices arrangement (a) with all non-zero sub-matrices (b) with zero sub-matrices.

to the matrix row-weight (4) and equal to column weight (3) in a column. Such structures have been shown to have a maximum girth of twelve[57]. In part (b) the number of sub-matrices is greater than row and column weights. This structure could be the result of irregular code designs or decoder architecture. It has also been shown that matrix sub-divisions larger than row and column weights results in better performing codes than those with sub-divisions equal to row and column weights[37].

Several methods have been suggested for constructing QC-LDPC codes. The structure of the code depends on the arrangement of sub-matrices and their shift values. Random shifting of identity sub-matrices may result in codes with properties that reduce performance such as four-cycles. Some construction methods impose constraints to avoid four-cycles and they include finite-geometry[56],combinatorial designs [76][77], algebraic[57], difference sets[53] and search algorithms[78][57]. As was stated in the previous chapter, construction methods avoid four-cycles by enforcing the row-column constraint.
Finite-geometry and combinatorial designs ensure that there are no four-cycles by adding the condition that no two points are in the same line more than once or no two points are in the same block more than once. Kou et al [56] present a geometric approach to the design of LDPC codes based on the lines and points of Euclidean and projective geometries over finite fields. Codes constructed from these finite geometries have girth of six and have a wide range of lengths and code rates.

Construction methods such as combinatorics, finite geometry and algebra produce codes that are limited in all or one of rate, length or girth parameters. Other methods such as the recursive approach developed in [79] could be used to obtain a wide range of lengths rates and girths. This method constructs a basis QC-LDPC code using one of the developed methods, geometric or algebraic construction. The basis matrix is then expanded by replacing '1' entries by $p \times p$ randomly shifted identity sub-matrices and '0' entries by $p \times p$ zero sub-matrices. The expanded matrix has girth and minimum distance at least that of the basis matrix. Although the method can be used to obtain a wide range of rates and lengths it divides the code into many sub-matrices which eventually increases data management in a decoder. A lager number of sub-divisions are required to obtain a large girth and high rates. In the next section we introduce a construction method with which a wide range of girths, rates and lengths could be obtained with as few sub-matrices as row and column weights. In this algorithm sub-matrices could be arranged arbitrarily.

4.2 Proposed Search Algorithm for QC-LDPC Codes

Random search algorithms such as BF and PEG have flexibility in constructing both regular and irregular codes over a wide range of girths, rates and lengths as previously discussed. In the proposed method we add more constraints to random searches such that the obtained codes are quasi-cyclic. The proposed algorithm produces regular and irregular quasi-cyclic codes with much more flexibility in girths, rates and lengths compared to other algorithms.

The algorithm is similar to the search algorithm developed in the previous chapter. Rows are divided into groups as before to obtain a block structure in the form of sub-matrices. To obtain cyclic connections in sub-matrices, rows in a group are connected in consecutive order according to their position. That is, if $\{r_a, r_b, ...r_c\}$ are connected rows, so are rows $\{r_{a+1}, r_{b+1}, ...r_{c+1}\}$. A non-bipartite or distance graph representation described in the previous chapter is assumed. Rows are used to form a distance graph which is then transformed to a parity-check matrix. To obtain a given girth, rows that are at a desired distance from each other are searched sequentially or randomly in each group and connected. Row connections are between j rows from j different groups. According to the distance graph representation, a set of j connected rows (vertices) are the rows that

are connected to the same column in matrix form. So each connection of j rows forms a column of the designed code. The algorithm is described as follows.

QC-LDPC Search Algorithm

1. Divide rows into j' equal groups of size p, $(RG_1...RG_{j'})$. If the number of rows is unknown or not given, start with a theoretical minimum number of rows if known otherwise start with a group size of k (row-weight).
   r_x is row x. $\cup_{r_x}$ is a set of rows within a distance of g from r_x. Distance is the shortest path between any two vertices. g determines the desired or target girth.

2. Make sub-groups with j distinct row-groups with each row-group appearing k times. The number of sub-groups is $\frac{kj'}{j}$ which is assumed to be an integer. Row-group sub-groups are $(RGP_1...RGP_{\frac{kj'}{j}})$.

3. For $t = 1$ to $\frac{kj'}{j}$ {

   select $RG_{ref} \in RGP_t$, where $1 \leq ref \geq j$
   select $r_i \in RG_{ref}$, where $1 \leq i \geq p$
   sequentially or randomly search for $(r_a...r_d) \in RGP_t(z = 1...j, z \neq ref)$ respectively where $r_x \notin \sum_{y \neq x}(\cup_{r_i}, \cup_{r_a}...\cup_{r_d})$, else the algorithm fails.
   For $z = 1$ to p {

   r_{i+z} is connected to $(r_{a+z}...r_{d+z})$ if $r_x \notin \sum_{y \neq x}(\cup_{r_{i+z}}, \cup_{r_{a+z}}...\cup_{r_{d+z}})$
   else the algorithm fails. } }

4. Use obtained distance graph to form a LDPC parity-check matrix.

The construction of the code graph is done in step 3 of the algorithm. First a reference row group (RG_{ref}) is selected from a sub-group (RGP_t) of row-groups. Selection of RG_{ref} could is arbitrary. A reference row (r_i) is then also arbitrarily selected from the reference row-group. The algorithm then searches for rows $(r_a...r_d)$ from the remaining row-groups of RGP_t, where rows $(r_a...r_d)$ are at a distance of at least g from r_i and from each other. If such rows are not found in their respectively row-groups, the algorithms fails and exits. In the second part of step 3, the algorithm connects the remaining rows of RG_{ref} to the remaining rows in other row-groups if the first part is was successful. The connection of rows r_{i+z} and $(r_{a+z}...r_{d+z})$ is made if it does not violate the girth condition.

Figure 4.2 shows row connections for a (16,2,4) QC-LDPC code with girth of eight, constructed using the proposed algorithm. The number row-groups, j', is 2, each group with 4,(p), rows. We refer to the groups as group 1 and group 2. Group 1 has rows 1 to 4 and group 2 rows 5 to 8. The number of sub-groups, $\frac{kj'}{j}$, is 4, where $j = 2$ and $k = 4$. The 4 sub-groups are [1 2],[1 2],[1 2] and [1 2] with each group appearing four times (desired row weight,k). A sequential search for a row satisfying the distance is used in this case. Group 1 and row 1 are always picked as the reference group and row respectively. In the first connection row 5 is found to satisfy the distance of four (desired girth) from row 1. The rest of group 1 rows, rows 2 to 4, are then connected to rows 6 to 8. In the second connection, row 6 is the first to satisfy the distance. It is connected to row 1 with the rest of group 1 connected to the rest of group 2. The process is repeated in connections three and four as shown in the figure. The row connections form a distance graph with the number of vertices equal to eight, a vertex degree equal to four and a girth of four. Figure 4.3 shows a matrix representation of the code. Each set of connections forms a column group with each row group as a 4×4 shifted identity sub-matrix. Since the first group is not searched or shifted, rows in this group are connected in their natural order in each sub-matrix. The top four rows contain four unshifted identity sub-matrices corresponding to the four connections for group 1 rows. Group 2 rows are connected in their natural order only in the first connection. The bottom 4×4 sub-matrices represent group 2 connections. Figure 4.4 shows the general structure of the codes obtained when group 1 and its first row are always the reference group and row respectively. If the reference row is chosen randomly, sub-matrices in the first column and row will also be shifted.

The complexity of the algorithm is analyzed in terms of the number of rows,M, and the number of row groups as follows.

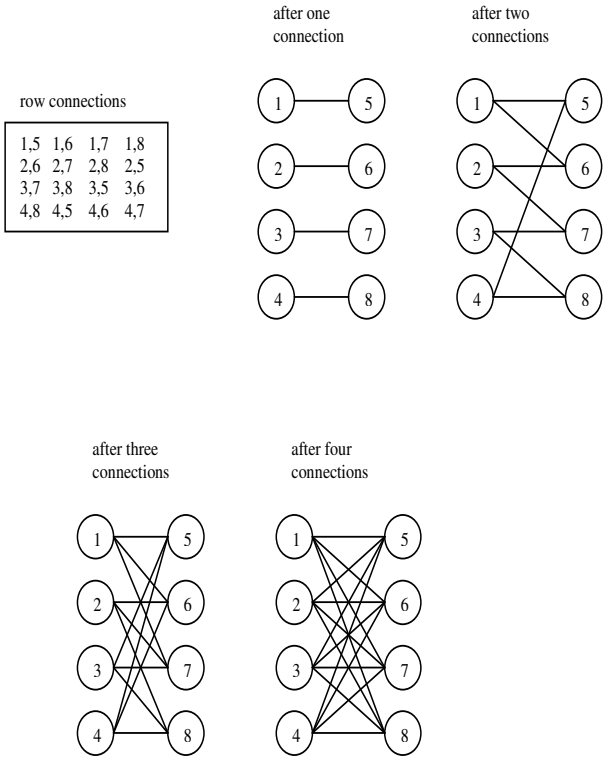

Figure 4.2. Graph representation of a (16,2,4) code with girth eight.

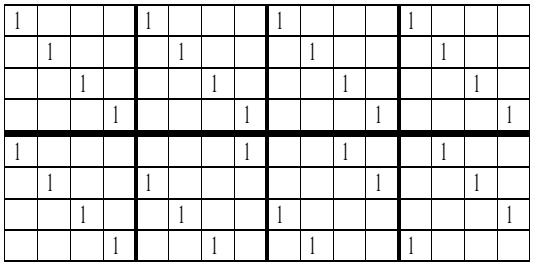

Figure 4.3. Matrix representation of a (16,2,4) code with girth eight.

I	I	I	I
I	I_{22}	I_{23}	I_{24}

Figure 4.4. General structure of QC-LDPC codes using sequential search.

- If the number of row groups is j', each group is of size $p = \frac{M}{j'}$.
- For each row-group set (sub-group), rows in non-reference row-groups and at a distance of at least g from the reference row, r, are updated before searching for a row satisfying the distance g from reference row, r_i. Updating neighbors of a row at a distance of g takes g cycles (or operations). For a single row-group it takes $\frac{gM}{j'}$ cycles. Since there are $(j-1)$ non-reference groups rows, it takes $\frac{gM}{j'}(j-1)$ cycles to update rows within distance g from reference row, r_i.
- For each row-group set, rows from the j row-groups are connected according to the reference row connections. There are $\frac{M}{j'}$ connections in each group set and in each connection the girth condition must not be violated. Checking for girth violation with j rows takes $\frac{j!}{(j-2)!2!}$ comparisons. Each row-group set search and connect process takes $\frac{Mg(j-1)}{j'} + \frac{M\alpha}{j'}$ cycles, where α is $\frac{j!}{(j-2)!2!}$.
- The connection process is repeated for each row-group set. There are $\frac{kj'}{j}$ row-group sets for regular codes with row weight of k. Therefore it takes $\frac{kM(g(j-1)+\alpha)}{j}$ cycles to complete all connections. This is assuming that the group size is large enough for the algorithm to form all connections and not counting the number of extra tries in case the connections failed the girth condition. The complexity of this algorithm is therefore $\mathrm{O}(M)$.

The complexity of the algorithm may also be determined by how it is implemented. Straight forward group searches may take a long time especially when group sizes are very large. We adopt the set algebra approach used in the bit-filling algorithm [45] to reduce searching time. Rows that are not far enough (in terms of the girth condition) from reference row r_i are updated before searching for rows to connect to in other groups. To find a set of rows that satisfy the desired distance in other groups we perform set operations with the searched groups and neighbors of the reference row. Rows satisfying the girth condition are then either sequentially or randomly picked from the resulting set. If the set of rows satisfying the girth condition is empty the algorithm fails. Since all rows in a group are connected the same way, the algorithm has a linear complexity with respect to the number of rows. In bit-filling the complexity of connecting individual rows is cubic with respect to the number of rows[45].

The proposed algorithm constructs a code of a given girth by searching and connecting rows that are at some specified distance from a reference row. However, connecting the remaining rows in the same group as the reference row, as stated in step 3 of algorithm, does not guarantee that the target girth or distance will not be violated. The search and connect process is therefore not sufficient to obtain a desired girth. Step 3 of algorithm checks whether the girth is violated or not, when the rest of group rows are connected. The algorithm guarantees a girth of six by not connecting any two rows more than once. Since any two row groups form a bipartite graph between themselves, if the reference row avoids connecting to another row more than once then other rows in the same group cannot be connected to any row more than once. That is, if r_i and r_x are connected once then r_{i+a} and r_{x+a} are also connected only once. Therefore the girth-six condition is sufficient. For girth-six codes, the algorithm does not need to check if the rest of the group connections do not violate the girth condition (in step 3 of algorithm). With girths larger than six, the girth condition can be violated when connecting the rest of the group rows especially with column weights larger than three and small group sizes. Figure 4.5 shows how a cycle of three could be formed when the target cycle length is four in distance graphs. After the first two successful connections rows, 22 and 42 are found to satisfy the distance of at least four from the reference row 1 for the third connection. If this connection is made, rows 9,16 and 36 will be connected. Then rows 1 and 16 which are already connected will both be connected to row 36 in separate connections forming a cycle of length of three. Figure 4.6 shows how a target girth of twelve could be violated when the rest of the group rows are connected. The solid and dotted lines represent the first and second connections respectively. In the second connection attempt row 20 is found to satisfy the length of at least six from reference row 1. However, connecting the rest of group 1 rows creates cycles of length four. An example of such a cycle is between rows 1,15,6,20.

Although the proposed algorithm does not guarantee girths larger than six, codes with girths larger than six have been easily obtained for a wide range of lengths and rates. In the constructions presented below LDPC codes were constructed targeting a specific girth given the code size and weights. For codes with large group sizes, the algorithm could be greedy such that it maximizes the girth or average girth. Instead of choosing a row that

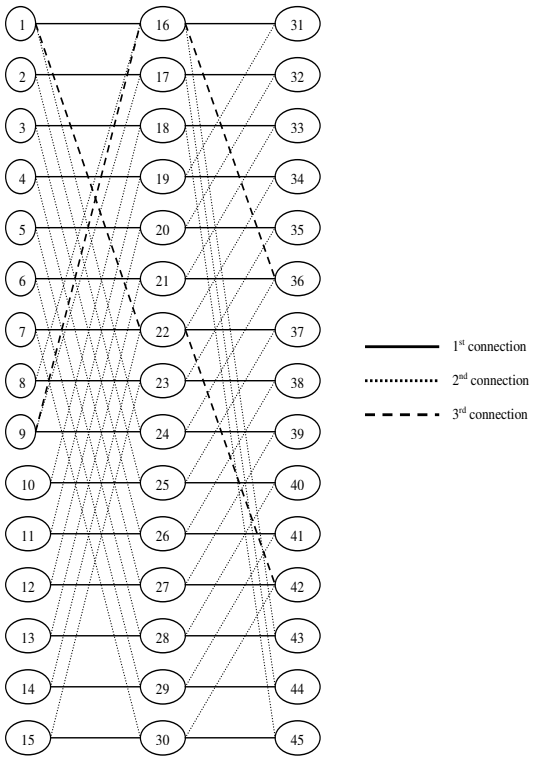

Figure 4.5. LDPC graph three-cycle formations with three groups.

satisfies the distance at random, the algorithm could choose a row with the maximum distance from the reference row similar to progress-edge growth[46] (a greedy approach). Mao[37] showed girth distribution matters more than the minimum girth in decoding performance. Codes with a larger average girth perform better compared to codes with the same or higher girth. Below we look at codes with column weight of two and then those with higher column weights obtained with the proposed algorithm.

4.3 Column-Weight Two Quasi-Cyclic LDPC Codes

The proposed algorithm obtains column-weight two codes with a minimum girth of eight if two row groups are used. Row connections form a bipartite graph between the two groups. The minimum cycle length in a bipartite graph is four, which translates to a cycle length of eight in matrix form. Since bipartite graphs have even length cycles, only girths of eight and twelve are obtained. Girths higher than twelve could be obtained by

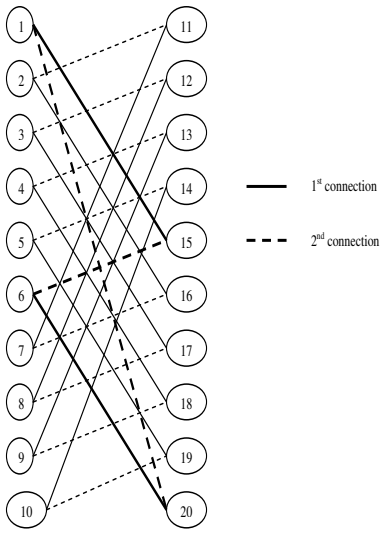

Figure 4.6. Formation of smaller cycles than the target girth.

using a number of row groups larger than two. Some obtained codes are presented below for some girths.

4.3.1 Girth-Eight Codes

Girth-eight codes are formed by a bipartite graph with $2k$ (k is row-weight) vertices when two row-groups are used. A smallest cycle length of four is formed between vertices (rows) of the two row-groups. The minimum group size,p, of each group was found to be k resulting in a code size of $2k \times k^2$ for all row weights(k). The length of the code is calculated using the relation $N = \frac{Mk}{j}$. The obtained sizes of the codes are the minimum dimensions possible for a quasi-cyclic or any code. k is the minimum size of each group as each row has to be connected to k different rows to avoid cycles of four. The graph size also is the minimum possible as it corresponds to the size of a cage with girth of four for a given vertex-degree ($(k, 4) - cages$)[67][68]. These codes could also be obtained using combinatorial designs[54].

For girth-eight codes, the first row found satisfying the girth condition from reference row will have a successful connection in the construction algorithm. The row will have distance of at least four if it has not been connected to the reference row before. If the two rows (reference row of group one and found row in group two) are not yet connected, then the rest of the rows will not be connected since the connections are relative to each other.

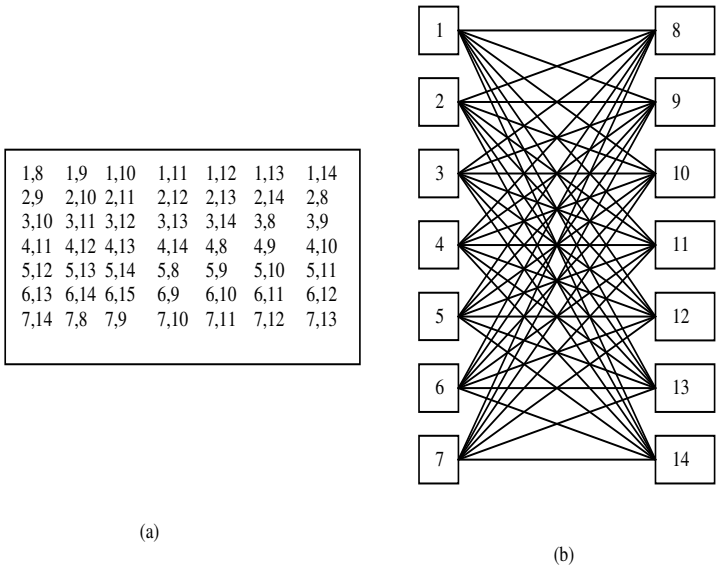

Figure 4.7. (49,2,7) girth-eight code (a) row connections (b) distance graph connections, (7,4) cage.

1,11	1,12	1,13	1,14	1,15	1,16	1,17
2,12	2,13	2,14	2,15	2,16	2,17	2,18
3,13	3,14	3,15	3,16	3,17	3,18	3,19
4,14	4,15	4,16	4,17	4,18	4,19	4,20
5,15	5,16	5,17	5,18	5,19	5,20	5,11
6,16	6,17	6,18	6,19	6,20	6,11	6,12
7,17	7,18	7,19	7,20	7,11	7,12	7,13
8,18	8,19	8,20	8,11	8,12	8,13	8,14
9,19	9,20	9,11	9,12	9,13	9,14	9,15
10,20	10,11	10,12	10,13	10,14	10,15	10,16

Figure 4.8. Row connections for a (70,2,7) code with girth eight.

1,8	5,9	6,14	3,9	6,8	4,14	7,12
2,9	6,10	7,8	4,10	7,9	5,8	1,13
3,10	7,11	1,9	5,11	1,10	6,9	2,14
4,11	1,12	2,10	6,12	2,11	7,10	3,8
5,12	2,13	3,11	7,13	3,12	1,11	4,9
6,13	3,14	4,12	8,14	4,13	2,12	5,10
7,14	4,8	5,13	2,8	5,14	3,13	6,11

Figure 4.9. Girth-eight (49,2,7) code using random search.

Therefore, girth-eight is guaranteed for column-weight two codes when two row-groups and a row-group size of at least k are used.

Figure 4.7 shows row connections for a (49,2,7) code resulting from a sequential search with two groups. The two row groups have rows 1-7 and 8-14. The reference group and reference row are always chosen as group 1 and row 1 respectively. Figure 4.7 (a) shows the pairs of rows that are connected after construction. Connecting the rest of group 1 rows relative to the row 1 produces a shift connection of rows. In the seven sets of connections, group 2 is shifted by 0,1,2,3,4,5 and 6 respectively relative to group 1. The connections are presented in a graphical form in part (b) of the figure. The graph is a bipartite graph with distance of four and vertex degree of seven. It is actually a (7,4) cage graph[72]. Large codes can be obtained by using larger group sizes. With larger groups, the same number and size of shifts are performed. Figure 4.8 shows row connections with 20 rows in each group for a (70,2,7) LDPC code. The code is constructed in a similar way to the code in Figure 4.7.

Generally a quasi-cyclic column-weight two LDPC code with girth of eight can be constructed by simply starting with two row groups of size at least k and shifting one or both groups k times using different shift values. The shifting could also be random. Sequential searches result in the same code every time for a given rate and length whereas random searches result in different codes in terms of shifts some of which may give better BER performances as demonstrated in the next sub-section. Random shifts may also result in better average girth in some cases. Figure 4.9 shows row connections for a (49,2,7) girth-eight code obtained using random searches (step 3 of proposed construction algorithm). Reference rows (r_i) are also chosen randomly for each connection.

4.3.2 Girth-Twelve Codes

Girth-twelve codes are formed by a bipartite graph with a smallest cycle length of six when two row groups are used. As with girth-eight codes, girth-twelve codes could be constructed from cage graphs. For codes with $(k-1)$ as a prime, derived quasi-cyclic girth-twelve codes from cages are of length $k(k^2-k+1)$[39][80]. Construction of codes based on cages could only be done for those values of k where cages are known.

Using the proposed algorithm, girth-twelve LDPC codes with row weights ranging from 3 to 18 were constructed. Table 4.1 shows code sizes obtained when a sequential search

Table 4.1. Girth-twelve $(N, 2, k)$ code sizes using sequential searches and two row groups.

k	min. group size	code size
3	7	14×21
4	15	30×60
5	25	50×125
6	35	70×210
7	61	122×427
8	77	154×616
9	119	238×1071
10	134	268×1340
11	174	348×1914
12	216	432×2592
13	251	502×3263
14	304	608×4256
15	390	780×5850
16	509	1018×8144
17	615	1230×10455
18	663	1326×11934

is used with two groups. The codes are about twice the size of codes derived from cage graphs. Codes with row-weights larger than 18 could also be obtained using the construction algorithm. Figure 4.10 shows row connections for two girth-twelve codes using sequential searches. Part (a) is a (60,2,4) code with a row-group size of 15. In part (b) a larger group size of 20 is used to construct an (80,2,4) code. The same shifts are obtained in both cases in the second row-group with sequential searches. Smaller code sizes closer to those from cages could be obtained using random searches as shown in Table 4.2. However it may take many tries (executing the algorithm many times) to get the right shift combinations resulting in a smaller code especially for large row weights. For both types of searches (sequential and random) larger dimension codes can be obtained by using larger group sizes. This is an advantage compared to a single dimension obtained from cage graphs. However, there is no guarantee that all larger group sizes will always produce a girth of twelve.

1,16	1,17	1,19	1,23
2,17	2,18	2,20	2,24
3,18	3,19	3,21	3,25
4,19	4,20	4,22	4,26
5,20	5,21	5,23	5,27
6,21	6,22	6,24	6,28
7,22	7,23	7,25	7,29
8,23	8,24	8,26	8,30
9,24	9,25	9,27	9,16
10,25	10,26	10,28	10,17
11,26	11,27	11,29	11,18
12,27	12,28	12,30	12,19
13,28	13,29	13,16	13,20
14,29	14,30	14,17	14,21
15,30	15,16	15,18	15,22

1,21	1,22	1,24	1,28
2,22	2,23	2,25	2,29
3,23	3,24	3,26	3,30
4,24	4,25	4,27	4,31
5,25	5,26	5,28	5,32
6,26	6,27	6,29	6,33
7,27	7,28	7,30	7,34
8,28	8,29	8,31	8,35
9,29	9,30	9,32	9,36
10,30	10,31	10,33	10,37
11,31	11,32	11,34	11,38
12,32	12,33	12,35	12,39
13,33	13,34	13,36	13,40
14,34	14,35	14,37	14,21
15,35	15,36	15,38	15,22
16,36	16,37	16,39	16,23
17,37	17,38	17,40	17,24
18,38	18,39	18, 21	18,25
19,39	19,40	19, 22	19,26
20,40	20,21	20, 23	20,27

Figure 4.10. Row connections for girth-twelve LDPC codes (a) (60,2,4) code (b) (80,2,4) code.

Group Pairs →	[1 2]	[1 2]	[1 3]	[1 3]	[2 3]	[2 3]
	1,55	1,56	1,109	1,112	55,119	55,134
	2,56	2,57	2,110	2,113	56,120	56,135
Row →	3,57	3,58	3,111	3,114	57,121	57,136
connections	4,58	4,59	4,112	4,115	58,122	58,137
	. .	. .	. .	. .	. .	. .
	. .	. .	. .	. .	. .	. .
	. .	. .	. .	. .	. .	.
	54,108	54,55	54,162	54,111	108,117	108,133

Figure 4.11. Group row connections forming girth-sixteen LDPC code with row weight of 4.

4.3.3 Girths Higher than Twelve

Codes with higher girths were obtained by using a number of row groups larger than two. Figure 4.11 shows row division and row-group pairing for a girth-sixteen code. There are 162 rows in three row groups, 1-54, 55-108, 109-162. The three groups are paired as [1 2],[1 2],[1 3],[1 3],[2 3] and [2 3] to construct group sets or sub-groups with each group appearing four times. Table 4.3 shows code sizes for some of the codes obtained with girth higher than twelve using a sequential search. The sizes and girths of codes obtained may differ depending on the number and combination of groups. The number of row groups and group combinations used here were chosen arbitrarily.

Table 4.2. Girth-twelve $(N, 2, k)$ codes sizes using random searches and two row groups.

k	min. group size	code size
3	7	14×21
4	13	26×52
5	21	42×105
6	31	62×186
7	53	106×371
8	67	134×536
9	105	210×945
10	125	250×1250

4.3.4 Performance Simulations

Bit error rate (BER) performances of constructed codes were simulated on an AWGN channel with BPSK modulation. Performance curves are shown in Figures 4.12 and 4.13. Simulated codes are all of size (2556,2,4) in Figure 4.12.

Four points are noted from performance curves in Figure 4.12. Firstly, the randomly shifted codes perform better than the sequentially shifted codes. The two seq-(2556,2,4) and ran-(2556,2,4) codes in the figure have sequential and random shifts respectively from two row groups. They have a girth and average girth of twelve. However, the randomly shifted code outperforms the sequentially shifted code by about 0.4dB at 10^{-5} BER.

Secondly, multi-level or multi-division codes perform better than those with two groups. The multi-level code used here was constructed with six row groups (instead of two groups). It has a girth and average girth of twelve as does the other sequentially shifted code. It however, performs better by about 0.4dB at 10^{-5} BER. This confirms results obtained in [81] showing that multi-division could improve error correcting performance. Codes in [81] are quasi-cyclic but of a different structure.

Thirdly, performance curves show larger girth codes performing better. A girth-twenty code also from six row groups with sequential shifts outperforms the girth-twelve code by 1dB at 10^{-5} BER.

Lastly, performance curves also show a random code outperforming girth-twelve QC-LDPC codes by about 0.7 dB at 10^{-5} BER. The random code was constructed using a

Table 4.3. Code sizes with girth higher than twelve using sequential searches.

k	number of groups	min. group size	code size	girth
3	4	9	36×54	14
3	4	16	64×96	16
3	4	17	68×102	18
3	4	18	72×108	20
4	3	35	105×210	14
4	3	54	162×324	16
4	3	69	207×414	18
4	6	213	1278×2556	20
4	8	390	3120×6240	24
5	4	65	260×650	14
5	4	112	448×1120	16
6	6	121	726×2178	14
6	12	108	1286×3888	16

slightly modified bit-filling algorithm to obtain a regular code. It has a girth of ten and average girth of 14.6. However, the random code is outperformed by the girth-twenty quasi-cyclic code.

Figure 4.13 shows larger codes compared to codes obtained using graphical models in [61] and a random code with a high girth of 14. The row-weight three codes have similar performances. The obtained QC-LDPC code with row-weight of four outperforms the graphical code by about 0.6dB at 10^{-5} BER and outperforms the random code by about 0.1dB at 10^{-5}BER.

Performance simulations show that codes obtained using the proposed algorithm perform similar or higher than random codes when their girths are high (at least girth of 16). Performance of the codes is also improved by using row-groups larger than the column-weight,j. Obtained codes also perform as well as other structured column-weight two codes such as those from graphical models in [61].

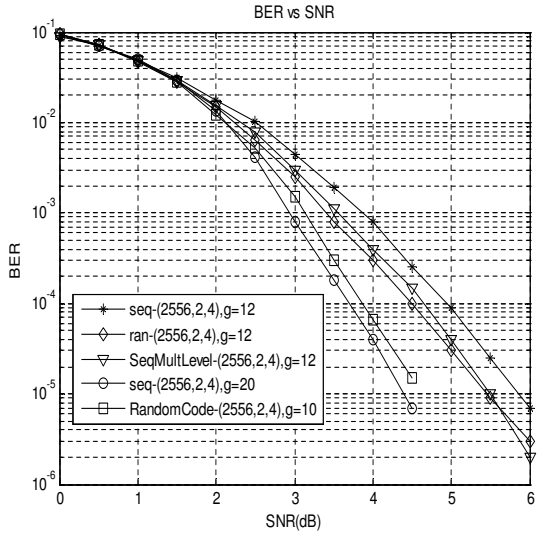

Figure 4.12. BER performance of obtained codes with 35 iterations.

4.4 Quasi-Cyclic Codes of Higher Column-Weights

The proposed search algorithm could also be used to construct codes with column weights higher than two. In this case a column is formed by a complete graph of more than two vertices. Codes with girths of six, eight, ten and twelve were obtained and are discussed below.

4.4.1 Girth-Six Codes

The proposed search algorithm was used to obtain column-weights of 3,4 and 5 using the number of row groups equal to the column weight. Figure 4.14 (a) shows row-column connections for a (42,3,6) girth-six code with a group size of seven. The number of row connections is 42, which is the length of the code or number of columns. The distance between rows is three. There are three groups (1-7, 8-14 and 15-21). Groups 2 and 3 are shifted by the algorithm in each connection to avoid four-cycles using a sequential search starting from the last row used in the previous connection. That is, if the reference row was connected to row x in the first connection, in the second connection searching for a row satisfying the girth condition starts at row $x + 1$. These are the search criteria that gave the smallest code for column-weight three codes but other variations of the search criteria could be used. Shift values for each row group are also shown at the top

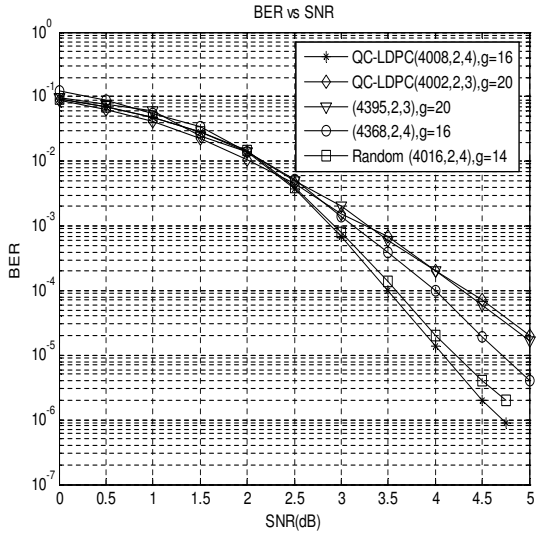

Figure 4.13. BER performance of larger dimension codes compared to graphical codes with 35 iterations.

of the figure. These connections form a distance graph in which a column is formed by a connection of three rows (vertices). Each triplet is a set of rows connected to the same column in the matrix form. The matrix form has the same structure as that of Figure 4.4 (a). Larger dimension codes can be obtained by using larger group sizes. Increasing the group size does not reduce the girth as the algorithm will still have group rows shifted such that no two rows are connected more than once as shown in Figure 4.14 (b) with a group size of ten. Figures 4.15 (a) and 4.15 (b) show row-column connections for column-weight 4 and 5 codes respectively. The first row of group one is always used as a reference row in these examples with sequential searching criteria.

Table 4.4. girth-six minimum group sizes with a sequential search.

k / j	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	5	5	7	7	9	9	11	11	13	13	15	15	17	17	19
4		5	7	7	10	10	11	11	13	13	16	17	17	17	19
5			7	7	11	11	11	11	13	13	17	17	17	17	19

In the example codes given above non-reference groups are shifted by a different increment each time. The reference group is shifted by an increment of zero each time, group 2 by one, group 3 by two and so on. Girth-six codes are obtained by simply using

Shift values	0 0 0	0 1 2	0 2 4	0 3 6	0 4 8	0 5 10
	1,8,15	1,9,17	1,10,19	1,11,21	1,12,16	1,13,18
	2,9,16	2,10,18	2,11,20	2,12,15	2,13,17	2,14,19
	3,10,17	3,11,19	3,12,21	3,13,16	3,14,18	3,8,20
	4,11,18	4,12,20	4,13,15	4,14,17	4,8,19	4,9,21
	5,12,19	5,13,21	5,14,16	5,8,18	5,9,20	5,10,15
	6,13,20	6,14,15	6,8,17	6,9,19	6,10,21	6,11,16
	7,14,21	7,8,16	7,9,18	7,10,20	7,11,15	7,12,17

(a)

0 0 0	0 1 2	0 2 4	0 3 6	0 4 8	0 5 10
1,11,21	1,12,23	1,13,25	1,14,27	1,15,29	1,16,22
2,12,22	2,13,24	2,14,26	2,15,28	2,16,30	2,17,23
3,13,23	3,14,25	3,15,27	3,16,29	3,17,21	3,18,24
4,14,24	4,15,26	4,16,28	4,17,30	4,18,22	4,19,25
5,15,25	5,16,27	5,17,29	5,18,21	5,19,23	5,20,26
6,16,26	6,17,28	6,18,30	6,19,22	6,20,24	6,11,27
7,17,27	7,18,29	7,19,21	7,20,23	7,11,25	7,12,28
8,18,28	8,19,30	8,20,22	8,11,24	8,12,26	8,13,29
9,19,29	9,20,21	9,11,23	9,12,25	9,13,27	9,14,30
10,20,30	10,11,22	10,12,24	10,13,26	10,14,28	10,15,21

(b)

Figure 4.14. Girth-six (42,3,6) code using sequential searches.

different shift increments for each row group. However, this does not work for all group sizes. We observed that for $j = 3$, the minimum p that produces a girth of six is k and $k + 1$ for odd and even row-weights respectively.

Figure 4.16 shows why these are the minimum group sizes. With three groups, groups 2 and 3 are shifted such that no two groups are shifted by the same increment more than once. Part (a) of Figure 4.16 shows that with group size of six (p), only five different shifts are obtained with increments of 1 and 2 for row groups 2 and 3 respectively. The sixth shift results in a four-cycle because its the same shift as the first one. However, when p is odd, seven in part (b) of figure, seven different shifts are obtained.

A shift increment of 2 in group 3 does not produce k consecutive increments when p is even and equal to k. The first half of the increments pick odd positions and ends where we started. The second half of the increments is then started at the first position available which is the first even position. Even positions are then picked until the last one which is k for even p and $p = k$, which is the same position as the increment of one in group two. Hence in the last connections we have groups 2 and 3 aligned the same way again. The first and last connections are the same for groups 2 and 3 producing a four cycle as shown in the example of Figure 4.16 (c). We can generalize these observations by saying that for column-weight three codes, the minimum group size is k and $k + 1$ for odd and even k respectively. These minimums were proved by Fossorier in [57] using algebraic constraints. Minimum group sizes for column-weights higher than three will be at least that of column-weight three. Table 4.4 shows minimum group sizes obtained with

Shift values	0 0 0 0	0 1 2 3	0 2 4 6	0 3 6 9	0 4 8 12	0 5 10 15
	1,8,15,22	1,9,17,25	1,10,19,28	1,11,21,24	1,12,16,27	1,13,18,23
	2,9,16,23	2,10,18,26	2,11,20,22	2,12,15,25	2,13,17,28	2,14,19,24
	3,10,17,24	3,11,19,27	3,12,21,23	3,13,16,26	3,14,18,22	3,8,20,25
	4,11,18,25	4,12,20,28	4,13,15,24	4,14,17,27	4,8,19,23	4,9,21,26
	5,12,19,26	5,13,21,22	5,14,16,25	5,8,18,28	5,9,20,24	5,10,15,27
	6,13,20,27	6,14,15,23	6,8,17,26	6,9,19,22	6,10,21,25	6,11,16,28
	7,14,21,28	7,8,16,24	7,9,18,27	7,10,20,23	7,11,15,26	7,12,17,22

(a)

Shift values	0 0 0 0 0	0 1 2 3 4	0 2 4 6 8	0 3 6 9 12	0 4 8 12 16	0 5 10 15 20
	1,8,15,22,29	1,9,17,25,33	1,10,19,28,30	1,11,21,24,34	1,12,16,27,31	1,13,18,23,35
	2,9,16,23,30	2,10,18,26,34	2,11,20,22,31	2,12,15,25,35	2,13,17,28,32	2,14,19,24,29
	3,10,17,24,31	3,11,19,27,35	3,12,21,23,32	3,13,16,26,29	3,14,18,22,33	3,8,20,25,30
	4,11,18,25,32	4,12,20,28,29	4,13,15,24,33	4,14,17,27,30	4,8,19,23,34	4,9,21,26,31
	5,12,19,26,33	5,13,21,22,30	5,14,16,25,34	5,8,18,28,31	5,9,20,24,35	5,10,15,27,32
	6,13,20,27,34	6,14,15,23,31	6,8,17,26,35	6,9,19,22,32	6,10,21,25,29	6,11,16,28,33
	7,14,21,28,35	7,8,16,24,32	7,9,18,27,29	7,10,20,23,33	7,11,15,26,30	7,12,17,22,34

(b)

Figure 4.15. Row-column connections for (a) (42,4,6) and (b) (42,5,6) girth-six quasi-cyclic codes.

a sequential search. The sizes obtained are the same sizes as those found by algebraic methods in [36][57].

Girth-six codes can also be obtained by using random searches. Random searches are likely to take longer to find the right shift combination that gives the minimum code size. Since practical codes will be larger than minimum sizes for most row weights, a random search is recommended. Random searches result in a variety of codes some of which may have better performance as was shown with column-weight two codes in the previous section. Figure 4.17 shows a girth-six code constructed using random searches. Random searches could quickly obtain minimum codes in some cases. Algebraic constructions in [36][57] used the MAGMA algebraic software to find minimum groups sizes and shift values that satisfy the girth of six. These papers report that, it takes a long time to find the minimum group sizes using these methods. However, with the proposed method, girth-six codes are almost instantly formed. For example, a girth-six code with $k = 3$, $p = 6$ takes 0.4s in MATLAB on standard PC (Pentium IV processor, 0.5GB RAM). The construction complexity is linear with respect to the number of rows as was shown in the previous section.

Increment	p	group positions
1	6	0,1,2,3,4,5
2	6	0,2,4,1,3,5

(a)

Increment	p	group positions
1	7	0,1,2,3,4,5,6
2	7	0,2,4,6,1,3,5

(b)

Shift values	0 0 0	0 1 2	0 2 4	0 3 7	0 4 9	0 5 11
	1,7,13	1,8,15	1,9,17	1,10,14	1,11,16	1,12,18
	2,8,14	2,9,16	2,10,18	2,11,15	2,12,17	2,7,13
	3,9,15	3,10,17	3,11,13	3,12,16	3,7,18	3,8,14
	4,10,16	4,11,18	4,12,14	4,7,17	4,8,13	4,9,15
	5,11,17	5,12,13	5,7,15	5,8,18	5,9,14	5,10,16
	6,12,18	6,7,14	6,8,16	6,9,13	6,10,15	6,11,17

(c)

Figure 4.16. Group shifts increments for girth-six code.

Shift values	0 6 1	0 3 3	0 5 6	0 1 0	0 4 2	0 2 5
	1,14,16	1,11,18	1,13,21	1,9,15	1,12,17	1,10,20
	2,8,17	2,12,19	2,14,15	2,10,16	2,13,18	2,11,21
	3,9,18	3,13,20	3,8,16	3,11,17	3,14,19	3,12,15
	4,10,19	4,14,21	4,9,17	4,12,18	4,8,20	4,13,16
	5,11,20	5,8,15	5,10,18	5,13,19	5,9,21	5,14,17
	6,12,21	6,9,16	6,11,19	6,14,20	6,10,16	6,8,18
	7,13,15	7,10,17	7,12,20	7,8,21	7,11,15	7,9,19

Figure 4.17. Row-column connections for a (42,3,6) quasi-cyclic girth-six code using a random search.

4.4.2 Girth-Eight Codes

To construct a code with girth of eight, connected rows must have a distance of at least four. Cycles of length three between rows must be avoided. To avoid six-cycles (3-cycle in row connection), if rows r_a and r_b are connected to r_c, then r_a and r_b cannot be connected. Tables 4.5 and 4.6 show obtained minimum group sizes of girth-eight codes obtained with sequential and random searches. The codes obtained are of the same size as those obtained by algebraic methods in [57] and [82].

With random searches smaller codes can be obtained even though it may take longer in some cases (because of large row weights). Table 4.6 shows code sizes for some $(N, 3, k)$ codes using random searches. As with girth-six codes the algorithm does not take long

to find a code even for the minimum code sizes shown in Tables 4.5 and 4.6. Larger dimension codes can be obtained by using larger group sizes as in column weight-two codes. The proposed algorithm can therefore quickly construct a variety of girth-eight codes over a wide range of rates and lengths. However, also in this case, there is no guarantee that all larger group sizes would hold the girth of eight.

Table 4.5. (N,3,k) and (N,4,k) girth-eight codes minimum group sizes using sequential search.

k j	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	9	19	21	25	27	55	57	61	63	73	75	79	81	163
4		31	38	44	56	73	84	108	121	134	149	180	210	230

Table 4.6. Obtained $(N, 3, k)$ girth-eight LDPC codes sizes using random searches

k	group size	code size
4	9	27×36
5	13	39×65
6	18	54×108
7	22	66×154
8	27	81×216
9	38	114×342
10	41	123×410
11	56	168×616
12	58	174×696

4.4.3 Girth-Ten and Twelve Codes

With girths of ten and twelve, the proposed algorithm often has failing row connections. That is, even though we find rows meeting the distance required from the reference row, the rest of the reference group row connections often result in smaller cycles than the targeted one. Most connections collapse to form smaller cycles of four as was shown in the

examples of Figures 4.5 and 4.6. In such cases the algorithm fails and is restarted. From our experiments it took too many (a few thousands) tries to get the right shifts resulting in girth-ten and twelve codes. The proposed algorithm is not time efficient in constructing distance graphs for codes with girths of ten and twelve. To reduce this problem the algorithm was used to construct a Tanner graph instead of a distance graph. Rows and columns are divided in to equal groups as before and connected to form a Tanner graph. Each row group is connected to k column groups and each column group to j row groups. Although the two constructions (Tanner and distance graph) are equivalent they behave differently in some cases. When constructing a Tanner graph the proposed algorithm was found to be more successful compared to constructing a distance graph for girth-ten and twelve codes. The algorithm could add connections (edges) between nodes from either side (rows or columns). For clarity the algorithm is repeated below for constructing Tanner graphs with connections made from row side.

Tanner Graph Search Algorithm

1. Divide rows into j' equal groups of size p, $(RG_1...RG_{j'})$.
   Columns are also divided into k' groups of size p, $(CG_1...CG_{k'})$.
   If the number of rows or columns is unknown or not given, start with a theoretical minimum number if known otherwise start with a group size of k (row-weight).
   r_x is row x, and c_x is column x.
   $\cup_{r_x}$ is a set of columns within a distance of g from r_x.
   $\cup_{c_x}$ is a set of rows within a distance of g from c_x.

2. Pair row and column groups such that each row-group appears k times and each column-group j times. We denote these row-column group pairs as RCG.
   RCG=$\{(RG_1, CG_1)...(RG_{j'}, CG_{k'})\}$. There are kj' or jk' row-column group pairs.

3. For t = 1 to kj' {

   $RG \in RCG_t$, $CG \in RCG_t$
   select $r_i \in RG$
   sequentially or randomly search for some $c_x \in CG$ such that $c_x \notin \cup_{r_i}$ else algorithm fails.
   For $z = 1$ to p {

   r_{i+z} is connected to c_{x+z} if $c_{x+z} \notin \cup_{r_{i+z}}$ else algorithm fails.

   } }

4. Use obtained Tanner graph to form a LDPC parity-check matrix.

The algorithm constructs the Tanner graph in step 3. A reference row, r_i, is arbitrarily selected from a row-group,RG, from the row-column group pair RCG_t. A column,c_x that is at least a distance of g from r_i is sequentially or randomly searched from CG of RCG_t. If c_x is not found the algorithm fails and exits. In the second part of step 3, RG rows are connected to CG columns relative to the positions of r_i and c_x provided the girth condition is not violated. If one of the connections violated the girth condition the algorithm fails and exits.

The complexity of the algorithm is analyzed in terms of the number of columns or the length of the code,N, and the number of column groups as follows. It is assumed that

the algorithm adds edges to the graph from the row side. The complexity is the same if edges are added from column side.

- If the number of row groups is j', each group is of size $p = \frac{M}{j'}$.
- For each row-column groups pair neighbors of columns at a distance within g are updated before searching for a column satisfying the distance g from reference row, r_i. Updating neighbors of a column at a distance of g takes g cycles (or operations). For a single column-group it takes $\frac{gM}{j'}$ cycles.
- For each row-column pairing, rows and columns are connected according to the reference row connections. These connections add an extra $\frac{M}{j'}$ cycles. Each row-column group pair search and connect process takes $\frac{Mg}{j'} + \frac{M}{j'}$ cycles.
- The connection process is repeated for each row-column group pair. There are kj' pairs for a regular code with row weight of k. Therefore it takes $kM(g+1)$ cycles to complete all connections. This is assuming that the group size is large enough for the algorithm to form all connections and not counting the number of extra attempts in case the connections failed the girth condition. The complexity of this algorithm is therefore O(M). When connections are done from column groups the complexity is $jN(g+1)$ which is the same as $kM(g+1)$ since $kN = jN$.

The complexity of the algorithm when constructing a Tanner graph is also linear with respect to the number of rows. However, the complexity of constructing a Tanner graph is j times more than constructing a distance graph. j connections to a column in a Tanner graph correspond to a single connection between rows in a distance graph. Although the Tanner graph has more operations, our construction experiments show that it is more successful in making connections compared to a distance graph for girth-ten and twelve codes. Table 4.8 shows some of the code sizes obtained for girth-ten and twelve codes using a random search. Although searches were not exhaustive, the code sizes obtained are comparable in size to those obtained in [36] using algebraic constraints.

Table 4.7. (N,3,k) LDPC codes sizes with girth ten and twelve.

k	group size	code size	girth
4	47	141×188	10
4	110	330×440	12
5	134	402×670	10
5	341	1023×1705	12
6	161	483×966	10
6	513	1539×3072	12
7	231	693×1617	10
7	851	2553×5957	12
8	356	1068×2848	10
9	515	1545×4635	10

4.4.4 Performance Simulations

Codes obtained using the proposed algorithm were evaluated using BER performance. Figure 4.18 shows performance curves for (1200,3,6) regular codes. As in column-weight two codes, higher column-weight codes using sequential search show poor performances compared to those obtained using random searches. The difference in performance is particularly pronounced in low girth codes. Girth-six and eight sequential codes perform very poorly compared to randomly shifted codes of the same girth. Girth-ten codes have the smallest difference up to 10^{-6} BER. Random codes pass 10^{-6} BER at about 3dB at a length of 1200 and half rate.

The codes obtained could vary in their performance depending on the combinations of shifts when random searches are used. Fossorier in [57] showed that some shift combinations result in better minimum distances than others even with the same size and rate. A good performing code may be found by constructing many codes and then determining the best one by minimum distance measure. The performance shown may also vary depending on code size. In [36], girth-six and eight quasi-cyclic code showed a much higher difference in performance at 1K length compared to code length of 282.

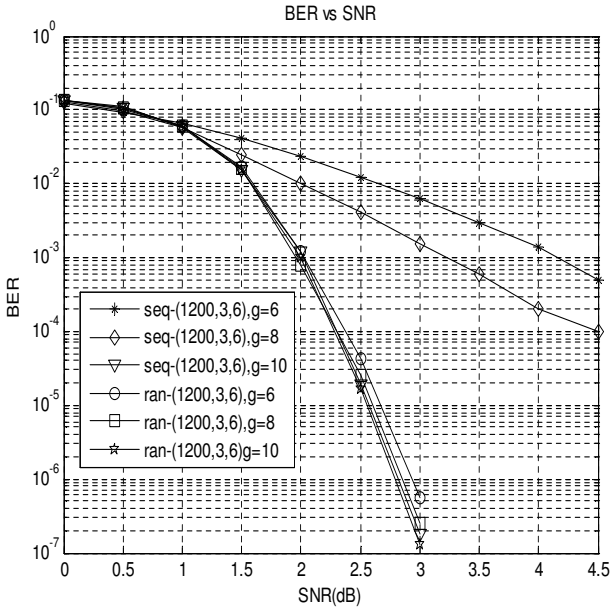

Figure 4.18. BER performance curves for (3,6) regular codes with 25 iterations.

Matrices with Zero Sub-Matrices

Matrices with zero sub-matrices are designed by using row and column groups larger than row and column weights. The number and interconnection of row-column groups may be dictated by decoding performance, hardware architecture or just be random. Codes with zero sub-matrices could be regular or irregular. In [60][83][84] protographs are used to design codes that match the structure of a semi-parallel architecture. A protograph is a small bipartite graph from which a larger graph can be obtained by a copy-and-permute procedure. The protograph is copied a number of times and then edges of individual replicas are permuted to obtain a single, large graph. Figure 4.19 shows an example of a graph derived from a protograph by connecting two copies of a protograph. In [60] the protograph is designed to resemble a decoder architecture. The decoder architecture may first be chosen based on a protograph decoding performance. Random or simulated annealing techniques were used to find the best performing protograph. Performance of the code also depends on the size of sub-matrices and their shift values. The matrix is expanded with $p \times p$ shifted identity sub-matrices. We could use the proposed algorithm to improve performance of obtained codes by improving the girth. The advantage of the proposed algorithm is that it can be used to construct codes with any sub-matrix or

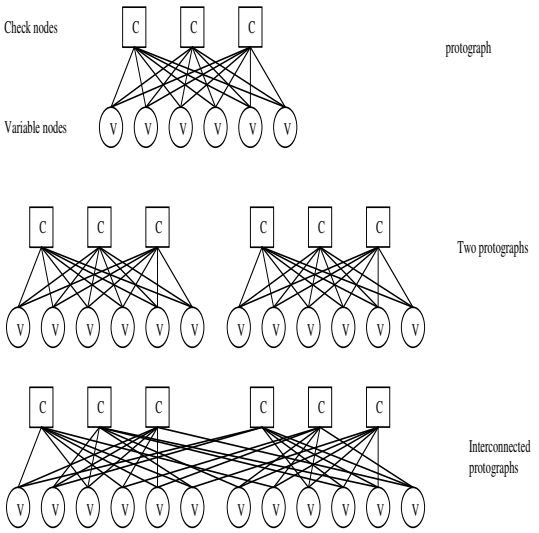

Figure 4.19. Simple protograph with derived code graph structure.

protograph configuration when the number of sub-divisions are at least equal to the row weights.

Irregular Codes

Carefully constructed irregular codes could have better performance compared to regular codes[29]. It was also shown in [50] that the degree of a variable node plays an important role in determining its error correcting performance. Variable nodes with a high degree tend to be decoded correctly compared to others. By targeting message bits to have higher degrees compared to parity bits we can improve the performance of a parity-check code. In [85] irregular quasi-cyclic codes from difference sets were found to have slightly better performance compared to regular ones. As stated in our algorithm, if the number of row or column groups is not uniform we obtain irregular codes. We can therefore construct irregular codes by having different numbers of appearances for row or column groups or both. A group weight is equal to the number of times it appears in connections. Figure 4.20 shows regular and irregular QC-LDPC code structures. The matrices show connected row-column groups by '1' entries. The average column and row weights are 3 and 6 respectively for both matrices. In the irregular structure extra connections are deliberately made in information groups compared to parity groups. Information groups have four connections whereas parity groups have only two connections as shown in Figure 4.20 (a). The regular structure has uniform connections for all groups with connections

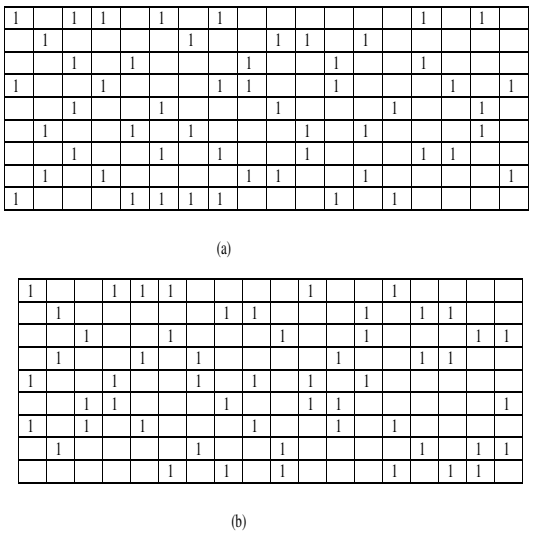

Figure 4.20. QC-LDPC code structures (a) irregular structure (b) regular structure.

randomly placed. The number of sub-divisions was chosen arbitrarily. BER simulations show one irregular code performing better than one regular code by 0.2dB at 10^{-4} BER as shown in Figure 4.21. The codes were constructed using the proposed algorithm with the structure of Figure 4.20.

Performance Comparison

We compare some of the codes obtained by our algorithm to other well known best performing codes. Short block codes obtained by Mackay[48] and by Hu using the PEG algorithm[46] are used for comparison. The codes used are regular and irregular (504,3,6) and (1008,3,6). These codes can be obtained from [48] and their performance is discussed in [46].

We constructed our codes based on the two matrix structures in Figure 4.20. The structures are not optimized in any way. The number of sub-divisions is also arbitrary.

For regular codes Mackay's algorithm obtains a girth of 6 and average girth of 6.74 for the (504,3,6) codes and girth of 8 and average girth of 9 for the (1008,3,6) code. In the PEG code, the girth is 8 with an average girth of 8.01 for the (504,3,6) code and girth of 8 and average girth 9.66 for the (1008,3,6) code. Using our proposed algorithm we obtain girth and average girth of 8 for the first code and girth and average girth of 10 for the second code. Figure 4.22 shows BER performances of the (504,3,6) codes with 80 iterations. Our code performs as well as Mackay's and almost as well as PEG codes at this size despite

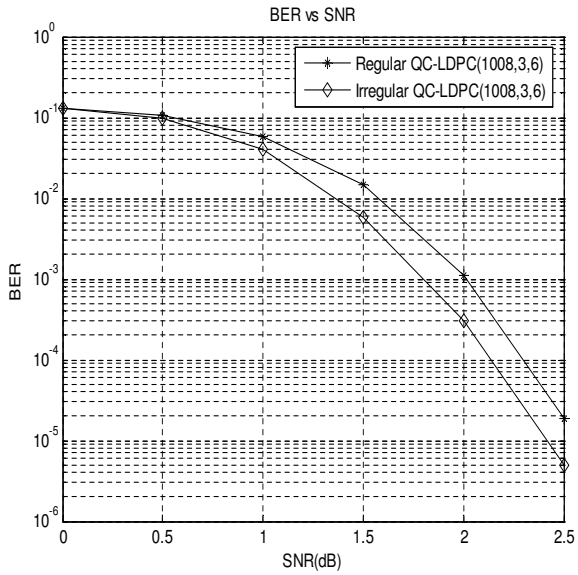

Figure 4.21. BER performances of irregular compared to regular codes.

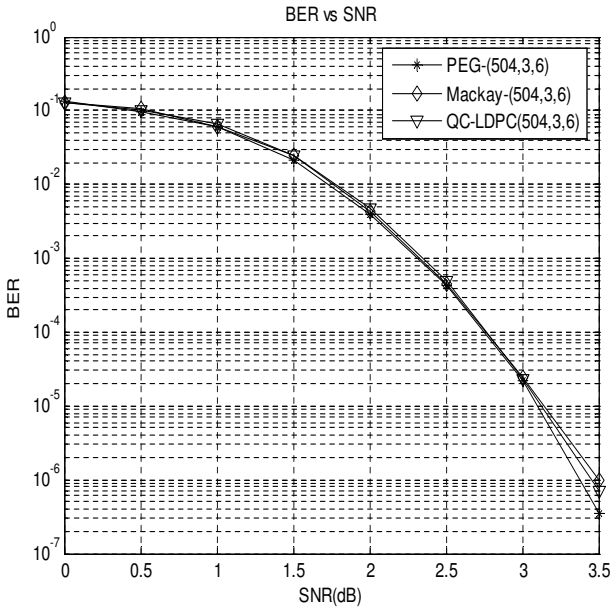

Figure 4.22. BER performances regular (504,3,6) qc-ldpc code compared to Mackay and PEG codes of the same size.

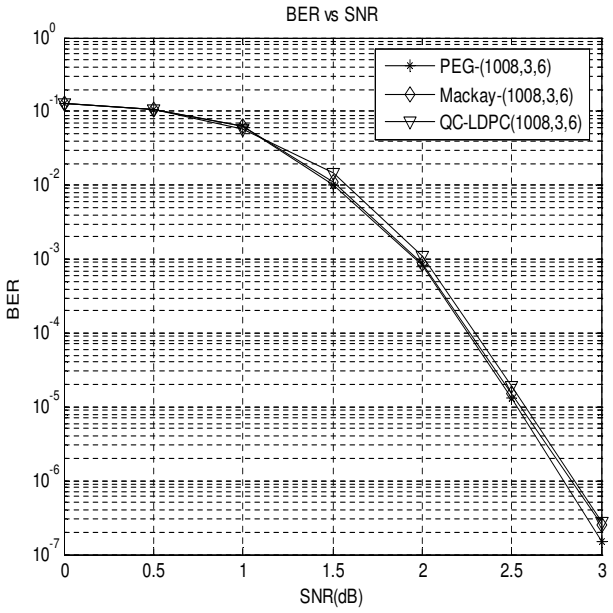

Figure 4.23. BER performances regular (1008,3,6) qc-ldpc codes compared to Mackay and PEG codes of the same size.

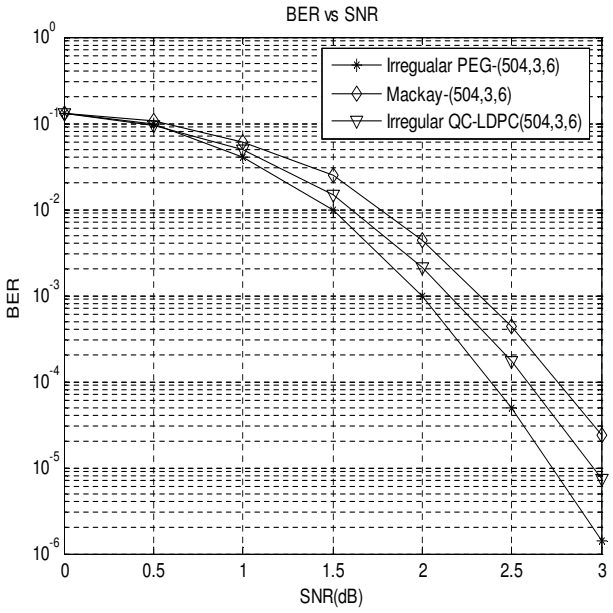

Figure 4.24. BER performances irregular (504,3,6) qc-ldpc code compared to Mackay and irregular PEG codes of the same size.

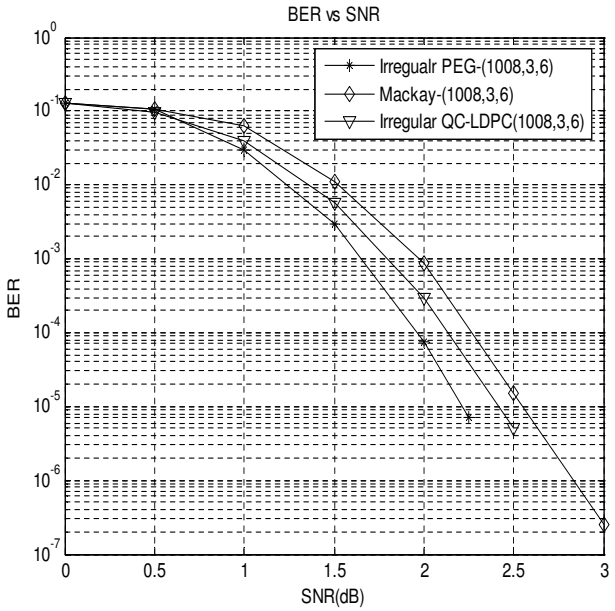

Figure 4.25. BER performances irregular (1008,3,6) qc-ldpc code compared to Mackay and irregular PEG codes of the same size.

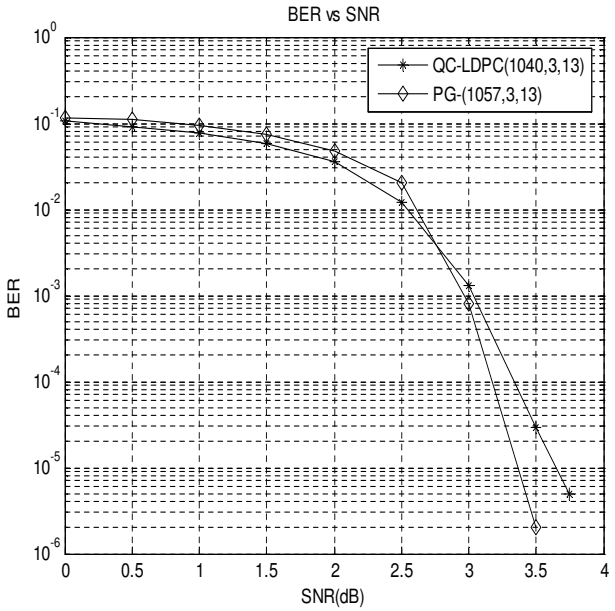

Figure 4.26. BER performances high-rate qc-ldpc code compared to a finite geometry code.

its constraints in structure (quasi-cyclic). It is outperformed by the PEG code at 10^{-6} BER by about 0.1dB. Larger codes of length 1008 show similar performance as shown in Figure 4.23. Our code still performs as well as PEG and Mackay's codes at this size. The PEG code still out performs our code by 0.1dB.

The same experiment was repeated with irregular codes. Figures 4.24 and 4.25 show performance comparisons of the two code sizes. In this case our code performs much worse than the PEG code. In the first instance the performance difference is about 0.25dB at 10^{-5} BER and about 0.2dB in the larger code. It however out performs Mackay's code. As already stated application of optimization techniques for performance on the basis graph may improve performance. Techniques such as density evolution[46] and simulated annealing[60] could be used to optimized or improve performance of the basis graph. It was shown in [75] that quasi-cyclic codes can perform as well as random codes. The result obtained here confirm that and show that the performance is also close for very short codes.

Figure 4.26 shows a highrate obtained code compared to a finite geometry code. The finite geometry code outperforms our code by about 0.4dB at 10^{-5}BER.

4.5 Summary

In this chapter a construction algorithm for quasi-cyclic codes based on BF and PEG search algorithms was presented. The algorithm divides rows and columns into equal groups sizes to obtain the sub-matrix or block structure. Connections in groups are in numerical order to produce the cyclic structure in sub-matrices. Although the proposed algorithm guarantees at least girth of six, higher girths are easily obtained as well. The algorithm could be used to construct a distance or Tanner graph. The complexity of the algorithm is linear in code size when constructing both distance and Tanner graphs.

Compared to other search methods the proposed algorithm is much more efficient in constructing both regular and irregular quasi-cyclic LDPC codes with no four-cycles. One major advantage of this method is that it can easily be used to construct codes over a wide range of girths, lengths and rates. It can also be used with any sub-matrices arrangement with the number of groups at least equal to the row and column weights. It is generally easy to generate QC-LDPC codes using this algorithm. Random searches result in better performing codes compared to sequential codes. Random searches may

also result in smaller codes even though it may take more tries to get them. Obtained codes also show comparable BER performances when compared to random codes. Regular codes perform as well as some of the best known short codes in literature. Irregular codes however, fall short compared to random irregular codes but have the advantage of ease of implementability. Using more row and column groups larger than row or column weights results in better performing codes in column-weight two codes.

This page is blank

Chapter 5

LDPC Hardware Implementation

For most applications LDPC encoding and decoding is done using hardware to speed up processing. Successful application of LDPC codes to various systems depends mainly on encoders and decoders meeting cost, power, complexity and speed requirements of those systems. Applications differ on these requirements and LDPC performance expectations. The large size of LDPC codes, wide range of rates and unstructured interconnection patterns are some of the characteristics that make hardware implementation a challenge.
Although decoder implementations are often targeted for a particular application, their architectures are often required to be scalable, programmable and have low chip area. Some applications require a wide range on some parameters such as rate and length. Also, application requirements may change from time to time. In such cases, it is desirable to have a flexible decoder that could easily be adapted to new requirements. The adaptability of hardware to new requirements may depend on its architecture.
We will briefly discuss some of hardware implementation issues in the following sections. We start with a broad classification of decoder architectures. A matrix rearranging method is then suggested for reducing routing congestion and complexity for random and fully parallel architectures.

5.1 LDPC Decoder Architecture Overview

LDPC decoder architectures differ mainly in the arrangement of check and variable processing nodes and their interconnections, message passing or scheduling, node implementations and number of nodes. The interconnection between nodes can be done using a variety of components such as memory blocks, buses and crossbar switches.
Message scheduling can also take various forms. The most common or natural scheduling criteria is called flooding. In flooding all check node messages are sent to all variable nodes after computation and vice versa. Other scheduling methods include staggering in which only a fraction of nodes send on demand messages across[14]. Staggering tries to reduce memory conflicts and improve computation units utilization. Depending on the interconnection network and storage of messages there might be memory access conflicts. Because the network is very likely not to accommodate all messages at the same time, scheduling of messages is needed. During message transmission no computations are done which reduces computation nodes utilization. Decoding equations can be implemented in different forms, including approximations and look up tables. LDPC decoders are often classified according to the number of processing nodes in comparison to the size of the code. Below are the three classifications based on this criteria.

5.1.1 Number of Processing Nodes

Fully Parallel architectures

Fully parallel architectures resemble the bipartite graph of the parity check matrix of a code as shown in Figure 5.1. There are mostly implemented for random codes. Each node of the Tanner graph is mapped onto a processing unit. The nodes are connected by wires or buses wherever there is a check-variable node connection in the graph. Hence the number of check processing nodes is equal to the number of rows (M) and the number of variable processing nodes is equal to the number of columns (N). The number of connections is equal to the number of edges in the corresponding bipartite graph which is kM or jN. If connections are not bidirectional the number of connections is doubled to $2kM$ or $2jN$. In each iteration each set of nodes computes probability ratios and messages. Messages are then passed onto the opposite or adjacent nodes via the interconnection network.
The architecture is fast as it provides the maximum number of computation nodes and

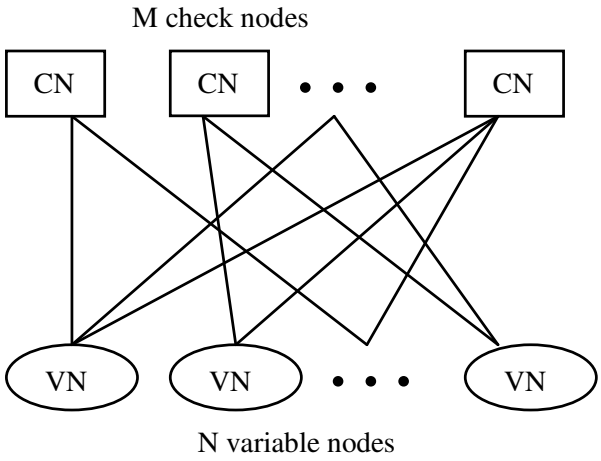

Figure 5.1. Fully parallel LDPC decoder architecture

communication wires. There is no queuing or scheduling of messages for computation or communication. The number of decoding cycles is equal to the number of iterations. With this architecture the throughput (number of decoded bits per unit time) depends on the node and interconnection implementations. However, for large codes the interconnection could be very complex and costly. The number of wires is very large and combined with unstructured check-variable node connections and limited layers in a chip it can be difficult to successfully do routing.The complex routing is due to the random connection of rows and columns in the code matrix. Each '1' or row-column connection in the code matrix represents a wire/bus connection between processing elements (variable and check nodes). The connections are random and over a large range which makes it almost impossible to route for large codes. Blanksby and Howland in[27] developed a 1024-b LDPC fully parallel irregular LDPC decoder. Although the decoder operated at a low frequency of 64Mz with a throughput of 1Gbps, the interconnect occupied half of the chip area. As reported in the article, the routing proved to be the main challenge of the implementation. Fully parallel architectures are also not flexible and not scalable. There are unable to execute codes with different parameters such as rate. As the size of the code increases the complexity of the architecture also increases.

Serial architectures

Fully parallel architectures instantiate each node and edge of the Tanner graph. The architecture requires maximum hardware with respect to the size of the code. The other

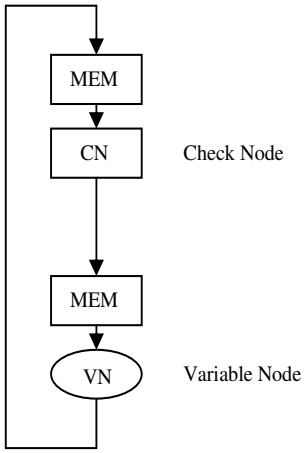

Figure 5.2. Serial LDPC decoder architecture with unidirectional connections.

extreme case would be to provide minimum hardware. Serial architectures as shown in Figure 5.2, have one check and one variable node computation units. The check node unit processes one row at a time and the variable node does the same with columns. As in fully parallel architectures the interconnection network could be bidirectional or unidirectional. Figure 5.2 shows two unidirectional connections. Serial architectures are very flexible and less costly compared to fully parallel ones. Any code structure could be executed on this architecture. However, serial implementations would not meet time constraints of most applications[14] which require high throughputs. Examples of serial decoder implementations can be found in [86][87].

Semi-parallel architectures

To improve throughput of serial architectures and reduce the complexity and cost of fully parallel architectures, semi-parallel architectures have been developed. Several processing nodes are implemented with an interconnection network between the two set of nodes as shown in Figure 5.3. The number of nodes, F and T, are much smaller than M and N and are greater than one. Hence, several columns or rows are mapped onto a single node. Semi-parallel architectures are often based on structured codes. The choice of interconnect network and scheduling of messages depends on the structure of the target code. Semi-parallel architectures have reduced cost and complexity compared to fully parallel architectures as there are fewer nodes. They are also easily scalable. However, they have less throughput. The throughput depends on the number of processing elements,F and

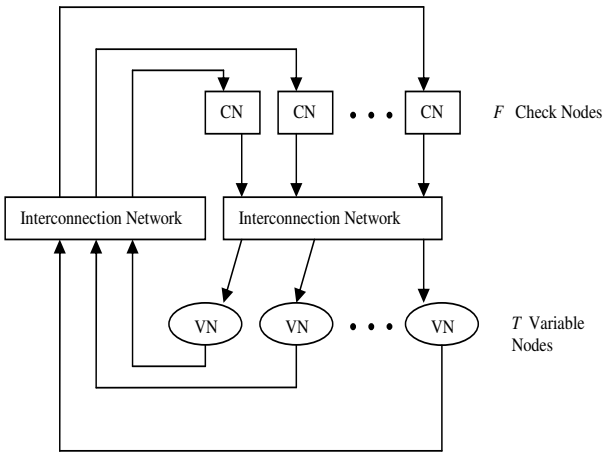

Figure 5.3. Semi-parallel LDPC decoder architecture with unidirectional connections.

T. Throughput can be traded with hardware cost to suit target application. Examples of semi-parallel architectures are found in [14][88][89][90] and many other references within.

5.1.2 Reduced Hardware Complexity

Besides the number of nodes there are other issues to be considered when designing LDPC hardware architecture. Although serial and semi-parallel decoders instantiate a small number of nodes they could still be large in area due to large memory required to store messages prior to transmission to adjacent nodes. There are several techniques which have been suggested to reduce decoder memory. Most techniques use variations or approximations of the MPA algorithm commonly referred to as reduced complexity algorithms. These algorithms reduce the number of messages to be transmitted across and in some cases the amount of intermediate values to be stored during calculations. The min-sum algorithm reduces complexity by simplifying the check node update. Issues related to its implementation and its modifications are explored in [91]. In [88] various log-likelihood-ratio-based belief-propagation decoding algorithms and their reduced complexity derivatives are presented. Performance and complexity of various reduced complexity algorithms are compared in the paper. Other reduced memory implementations are suggested in [92][93] and [94].

Besides hardware size and complexity, decoding latency is one of the critical factors for most applications. One major drawback of LDPC codes compared to Turbo codes is their

low convergence rate. Turbo codes take on average 8-10 iterations to converge [4] while LDPC codes typically need about 25-30 iterations to match the performance. A large number of iterations means longer decoding time. The overall decoding time could be reduced by faster convergence of the decoding algorithm and faster or simpler computations. Some of the reduced complexity algorithms mentioned above have faster convergence and simplified computations reducing decoding time as well.
In [95] an early detection method is used to reduce the computational complexity and delay. In this method nodes with a high log-likelihood ratio are considered reliable and not likely to change in the next iteration. Hence only a fraction of nodes do their updates in the next iteration. These nodes are not updated. They are instead passed as they are saving computation time.
It has also been shown that message scheduling can affect the convergence rate and performance of a code. Hocevar in [96] showed by simulations that the convergence rate of LDPC codes can be accelerated by using 'turbo scheduling' applied on variable node messages. In [97] it is shown analytically that the convergence rate for this scheduling is increased by about twice for regular codes. Another scheduling algorithm based on density evolution was suggested in [98] which also doubles the convergence rate and improves performance. Decoding time could also be reduced by overlapping computations[74][99]. Overlapping is discussed in detail in the next chapter.

The interconnection network is also a major consideration in decoder implementation. The interconnection could dominate the decoder both in area and decoding time[27][100]. Different forms of interconnection networks include memory banks, crossbar switches, multiplexors and buses. The choice of the interconnect depends on code structure and scheduling of messages. Interconnection networks will also be covered in more details in the next chapter.

5.1.3 Numeric Precision

Implementation of the MPA algorithm involves some complex operations, hyperbolic and inverse tangents as described by equations 2.5 and 2.6. The MPA algorithm could be simplified using approximations which reduce implementation complexity, but incurs decoding performance degradation. There are many complexity-reduced variants of the

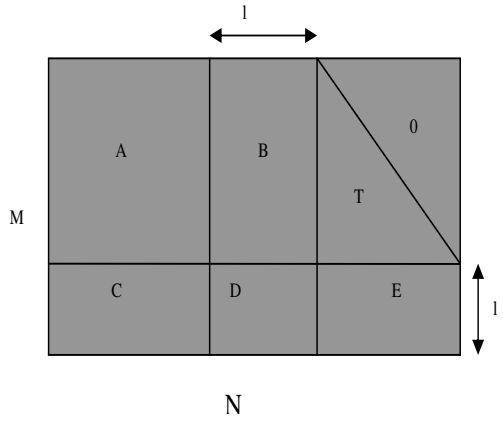

Figure 5.4. Rearranged LDPC matrix for reduced encoding.

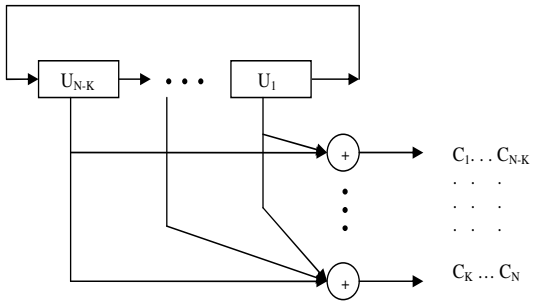

Figure 5.5. Shift encoder for quasi-cyclic LDPC codes.

MPA which have near optimum peformance[32][33][75][101]. The original MPA and its variations are sensitive to finite quantization. A large number of quantization bits gives more precision and better decoding performance. However, precision bits also affect hardware cost and decoding delay. Large precision numbers need more memory for storage and more time to read and process. Efficient hardware implementation of MPA requires a tradeoff between algorithm performance and hardware cost. The effects of quantization on performance have been measured in several articles in which a minimum quantization of 4 to 6 bits is found to be close to the un-quantized MPA[33][101]. However, the precise number of bits may vary depending on the algorithm used and target application. Quantization could be varied or uniform. In varied quantization the LLR and intermediate values use different number of bits. Uniform quantization has the same number of bits for all computations.

5.2 Encoder Implementation

Although most of research in LDPC codes is concentrated in decoder designs and implementations, encoder implementation also has significant complexity. The encoding complexity involves a large amount of data and a large number of computations. A straight forward encoding (uG) has $O(N^2)$ computational complexity[9]. The generator matrix is dense, increasing the number of multiplication operations with the incoming word,u. In [9], Richardson and Urbanke (RU), showed that linear time encoding can be achievable through manipulation of some LDPC codes.They present methods for preprocessing the parity check matrix and a set of matrix operations to perform the encoding. The preprocessing could be done in software since it is done once for a code.
The parity-check matrix can be rearranged into sub-matrices of the form

$$H = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \tag{5.1}$$

as shown in Figure 5.4, where A is of size $(M-l) \times (N-M)$, B is $(M-l) \times l$, C is $l \times (N-M)$, D is $l \times l$ and E is $l \times (M-l)$. T is a lower-triangular sub-matrix and l is called the "gap". If a codeword $c = (u, p_1, p_2)$ then multiplying $Hc^T = 0$ on the left by

$$\begin{bmatrix} I & 0 \\ -ET^{-1} & I \end{bmatrix} \tag{5.2}$$

we get

$$\begin{bmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & 0 \end{bmatrix} \begin{bmatrix} u \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5.3}$$

This gives two equations and two unknowns p_1 and p_2 since u is the known information bit vector. It is then shown in [102], that the complexity of solving for the unknowns is linear with respect to the code length, N, except solving for p_1 which has the complexity of O(l^2). Hence to reduce the overall complexity of encoding the parity-check matrix is permuted such that l is reduced or eliminated. Algorithms for permuting H such that l is reduced are presented in [102].

Cyclic codes such as quasi-cyclic could be encoded using shift registers taking advantage of the cyclic structure as was done in [3] with cyclic codes. The cyclic parity-check matrix produces a cyclic generator matrix. To multiply the input vector u with G, a shift register could be used to hold u as in Figure 5.5. A network of connections and XORs are then used to obtain the product. The interconnections of the register and XOR could be hardwired based on a particular code or reconfigurable. The interconnection circuitry executes the $N-K$ multiplications to the u. After u is loaded, the shift register is shifted $N-K$ times. Each time the interconnection circuitry calculates the corresponding output to each input bit. Examples of fast and efficient QC-LDPC codes encoding architectures based on this technique are presented in [24][73][103][104].

5.3 Fully Parallel and Random LDPC Decoders

Some applications require low-power,area and high throughput hardware. They also require large size codes. Applications such as magnetic recording systems, are sensitive in area and power[16]. For such systems a fully parallel decoder will be more suitable than a serial or semi-parallel decoder. Parallel decoders may be operated at low frequencies reducing power consumption. The large number of processing units increases throughput and the unconstrained matrix design allows us to have better decoding performance. However, for fully parallel decoders to be incorporated in these applications we need to overcome the routing complexity. As already stated fully parallel and random decoders have a high routing complexity and congestion of wires. The example decoder implemented by Blanksby and Howland in [27] was only 1024 bit long but proved hard to route. The large number of processing nodes and their unstructured interconnection increase the complexity of routing. We propose the use of reordering algorithms to reduce congestion and average wire length in fully parallel random decoders.

5.3.1 Structuring Random Codes for Hardware Implementation

There are techniques which have been suggested to reduce routing complexity in fully parallel LDPC decoder architectures. In [105] the routing congestion and average wire length are reduced by restricting the range of row-column connections as shown in Figure 5.6. Row-column connections are only in the shaded region. This produces localized

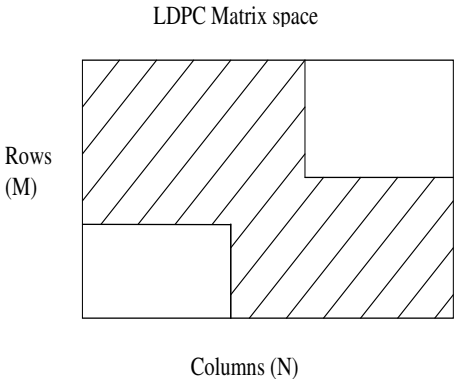

Figure 5.6. Restricted space for random code matrix.

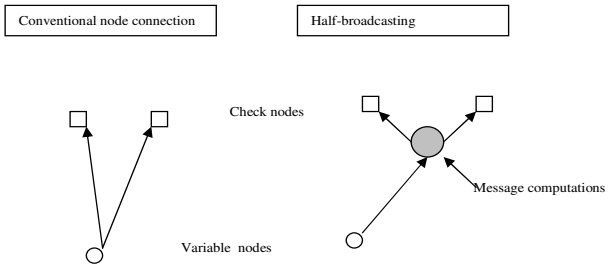

Figure 5.7. Conventional and half-broadcasting node connections.

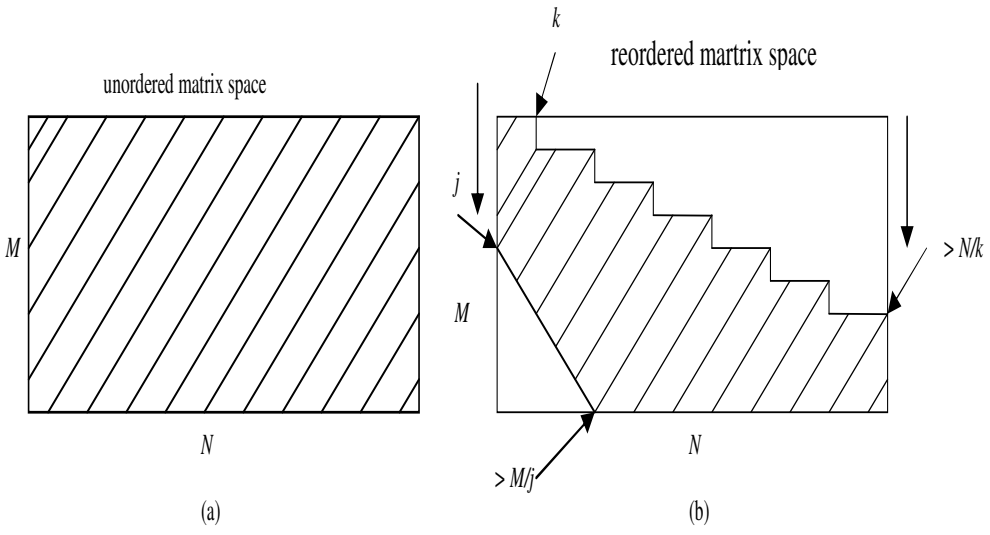

Figure 5.8. Unordered and ordered random code matrix space.

connections between check and variable nodes within restricted ranges. The reduction in congestion depends on the amount of restriction. Performance simulations need to be performed to analyze how much performance if any is lost due to the constraints on the formation of the matrix. In [106] a similar method is used to minimize the maximum number of wires (maximum cut-size) across the decoder chip area. The reduced number of wires eases routing. However, constraining the parity check matrix may reduce decoding performance of a code. The constraints may restrict parameters such as maximum girth and rate that could be obtained for a given code size.
In [107] half-broadcasting of messages is used to reduce routing congestion and complexity. In half-broadcasting computation and transmission of messages is delayed until near the destination. As shown in Figure 5.7, conventionally node messages are computed and broadcasted to their destinations using separate wires. In half-broadcasting message computation units are moved closer to the destination nodes. A single wire is used to carry needed values from one node to its destinations. Individual messages are computed closer to the destinations. The scheme is reported to reduce average wire length by 40%[107]. A heuristic method is used to reorder columns of the parity-check matrix such that the maximum wire length is reduced. For a 2048-bit code a reduction of 30% in maximum wire-length was achieved. Half-broadcasting assumes destination nodes (nodes with a common neighbor) can be placed close enough such that these nodes are not far from the source. The reordering algorithm reduced the maximum distance between connected rows and columns of the matrix.

We propose the use of sparse matrix reordering algorithms to reduce average wire length and routing congestion in fully parallel LDPC decoders by first designing the code matrix and then restructuring it for implementation. Sparse matrix reordering algorithms reorder columns such that row-column connections move closer to the diagonal of the matrix. Check nodes and their corresponding variable nodes are grouped closer together. The reordering does not change the matrix graph hence its decoding performance remains the same.

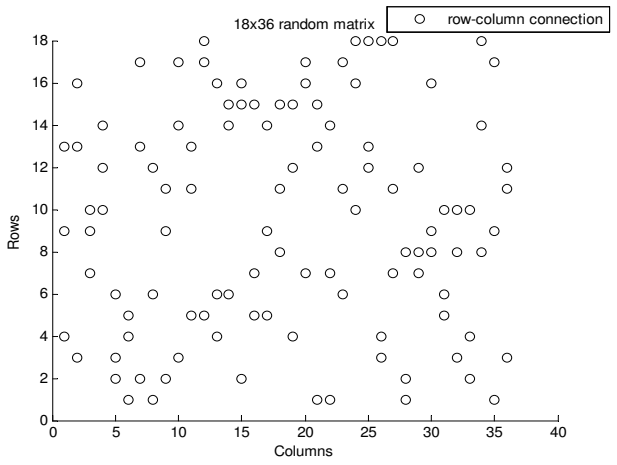

Figure 5.9. Row-column connections for an 18×36 random code.

Reverse Cuthill-Mackee algorithm

There are several algorithms for reducing bandwidth and profile of a sparse matrix. The bandwidth is the connection range of each row and profile is the area covered by row-column connections. The effect of these algorithms is to shift connections closer to the matrix diagonal thereby reducing the profile or area and bandwidth of connections. They are used in many fields such as networking, parallel and distributed processing, profile identification and layout problems in VLSI. Common algorithms include reverse Cuthill-McKee (RCM), King's and Gibbs-Poole-Stockmeyer (GPS) algorithms[108][109][110]. We used the RCM algorithm to reduce the bandwidth (range of row connections) of random LDPC connections. We did not get any significant difference with GPS or King's algorithms compared to RCM.

Reverse Cuthill-MacKee algorithm rearranges a matrix as follows. The code matrix is converted into a corresponding bipartite graph with one set of vertices representing rows and the other set columns. Edges are made between the vertices if there is a '1' entry in the corresponding row-column matrix location. The vertices are numbered from 1 to $M + N$. The RCM algorithm [108] proceeds as described below.

1. Choose a vertex (v) with a lowest edge degree, breaking ties arbitrarily. Number the vertex as level 1.

2. Number all the unvisited neighbors of v in increasing order by increasing degree.

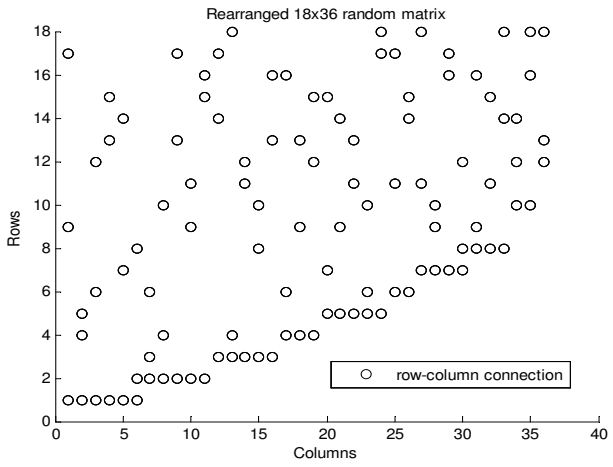

Figure 5.10. Permuted 18×36 random code.

3. Repeat the process for the neighbors of the visited vertices until all the vertices of the graph are visited.

4. Starting with vertices with the highest level (numbering), renumber the unvisited neighbors with decreasing degree until all vertices are renumbered.

The original Cuthill-Mackee algorithm does not have the last (reverse) step. In [109] it was shown that the reverse numbering of the Cuthill-Mackee algorithm achieves bandwidth equal to or less than of the original algorithm. Hence reverse Cuthill-Mackee (RCM) is more commonly used. The algorithm does not guarantee a minimum bandwidth and the results may depend on the starting vertex.

Although reordering matrix algorithms are mostly applied to symmetric matrices they have the same effect on nonsymmetric matrices. A LPDC code matrix is rectangular and non-symmetric. Random code connections may cover the whole matrix space as shown in Figure 5.8 (a). After reordering the matrix using the RCM algorithm the connection space takes a shape similar to that shown in part (b) of Figure 5.8. The algorithm takes each row and connects all its neighbors (columns) and vice versa. Each row has a maximum of k neighbors and each column a maximum of j neighbors. After a row has visited all its neighbors the algorithm moves to another row producing a 'stairs' shape as shown in the figure. Since there are N columns, a maximum number of $\frac{N}{k}$ rows can have no overlaps in their connections. Therefore the last column cannot be connected to a row less than $\frac{N}{k}$. Also a maximum number of $\frac{M}{j}$ columns can have no overlaps in their connections as marked in the figure. Any given matrix with row and column weights k and j will

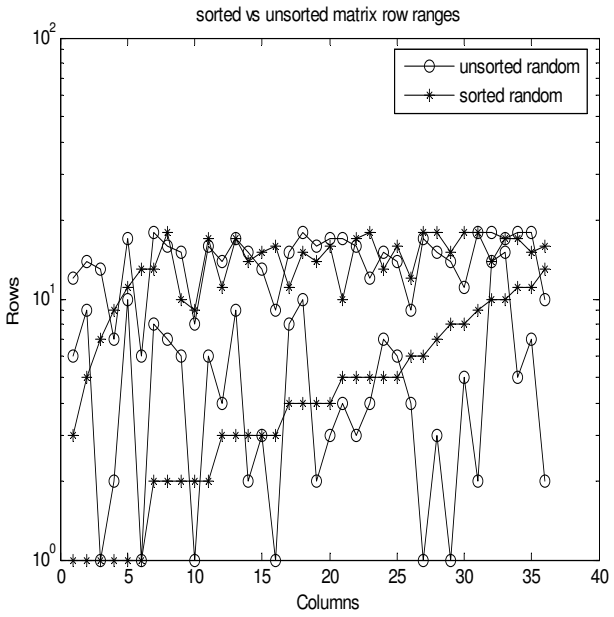

Figure 5.11. Column-row ranges for a random (36,3,6) LDPC matrix.

produce the 'stairs' shape with points k,j and $\frac{N}{k}$ and $\frac{M}{j}$ as marked in figure.
From this general shape of the rearranged matrix we can approximate the reduction in area. Since $k << N$ and $j << M$ we assume the lengths or heights of the two unshaded triangles of the rearranged matrix space are N and M. Then the area of the two triangles is given by

$$Unshaded\_Area = \frac{M^2}{2j} + \frac{N^2}{2k} \tag{5.4}$$

The total area is MN, hence the relative area reduction is $\frac{Unshaded\_Area}{MN}$.

$$Area\_Reduction = \frac{M}{2Nj} + \frac{N}{2Mk} \tag{5.5}$$

From the relationship between code weights and matrix size given by

$$\frac{N-M}{N} = 1 - \frac{j}{k} \tag{5.6}$$

we get

$$\frac{M}{N} = \frac{j}{k} \tag{5.7}$$

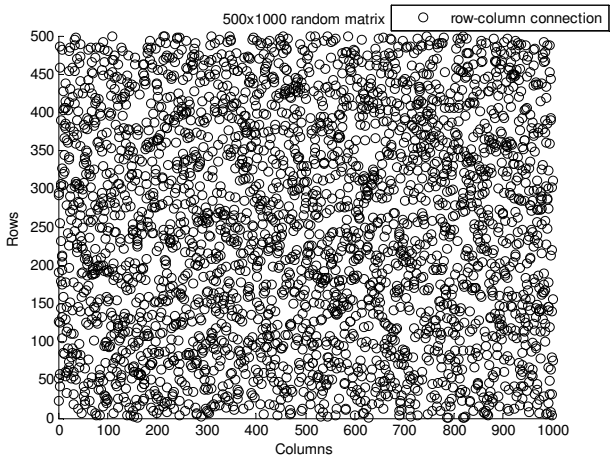

Figure 5.12. Unordered random matrix space, with average wire length of 500.

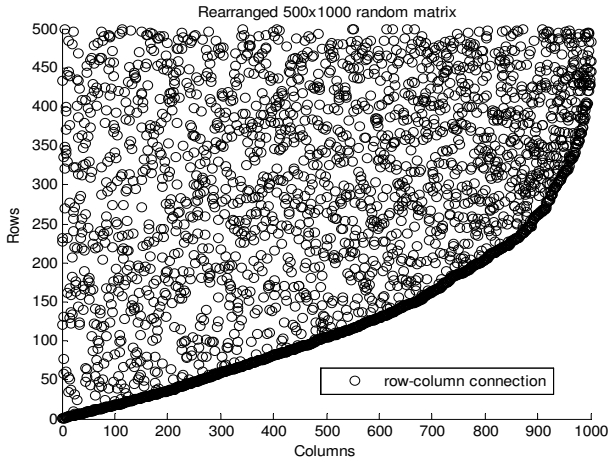

Figure 5.13. Rearranged random matrix space with average wire length reduced by 13%.

Substituting (5.7) in (5.5) we get area reduction of

$$Area\_Reduction = \frac{1}{2j} + \frac{1}{2k} \tag{5.8}$$

Small weights will have larger reduction in area. For a $(N, 3, 6)$ code the area is reduced by at least a quarter.

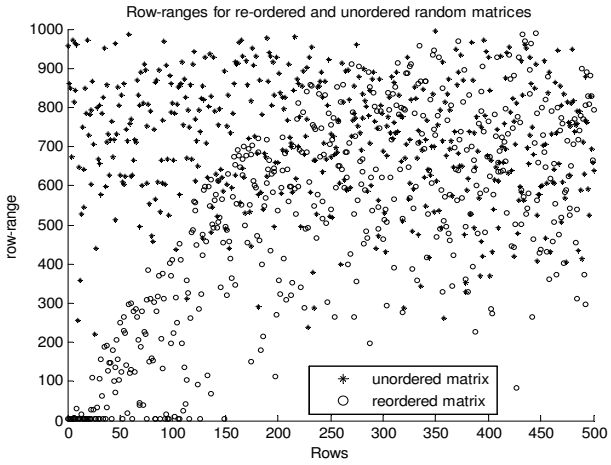

Figure 5.14. Row ranges or bandwidths for the original and rearranged matrices.

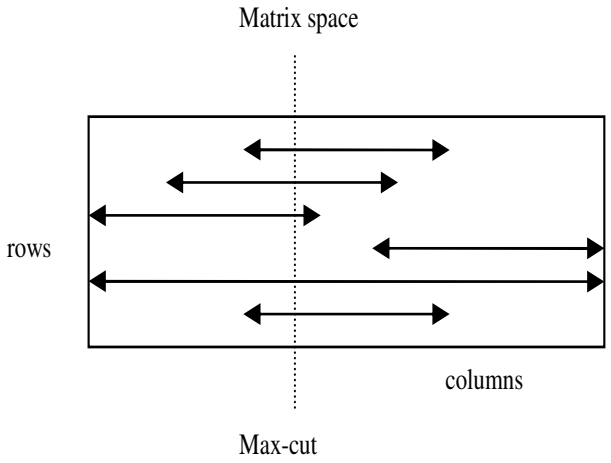

Figure 5.15. Maximum cut is the number of row-ranges crossing a column.

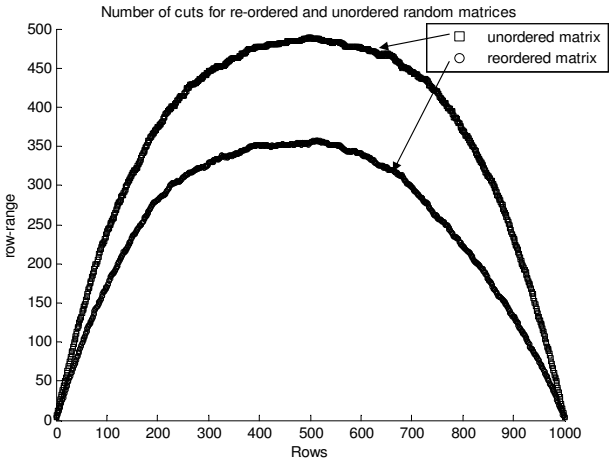

Figure 5.16. Number of vertical row-range cuts for columns.

Simulation Results

A (36,3,6) random code with girth of six was used to show how row connections and ranges can vary before and after applying the RCM algorithm. Figure 5.9 shows row-column connections for this code. The connections cover the whole matrix space. After reordering the matrix, row-column connections are moved closer to the diagonal as shown in Figure 5.10. The reordered matrix connections are similar in shape to the general shape of Figure 5.6 except that the other connection-free triangle is very small in this case. Figure 5.11 shows the row-column connections of both permuted and the original random codes. The plots show the range of each row, that is, the minimum and maximum column connections for each row. The original matrix connections vary widely from row to row whereas in the reordered code connections vary slightly. The lower variation would reduce routing complexity in that consecutive processing nodes would have wire length of approximately the same size. The ranges for rows also increase or decrease slightly. The area covered by the ordered matrix is also reduced.

Figures 5.12, 5.13 ,5.14 and 5.15 are results for a larger random code of size 500x1000. After reordering the connection space is reduced as shown by graphs of Figure 5.12 and 5.13. Figure 5.14 shows the maximum range for all the rows. The maximum row connection does not change but the average for all the rows is reduced by 30%.

The number of cuts for each column is the number of row connection ranges that cross the column as shown in Figure 5.15. It measures the number of wires to cross a particular area of a chip assuming the decoder chip follows the matrix layout. Figure 5.16 shows the number of row connections ranges crossing each column for a 500×1000 random code. As expected the middle columns have more cuts compared to extreme end columns. The unordered matrix (top curve) has a large maximum cut. The maximum number of cuts or cut-size is reduced by 24% whereas the average falls by 30% after reordering. Similar results were obtained for different code sizes as shown in Table 5.1. The first values are for rearranged matrix and the second for the original matrix.

Reordering algorithms could be used as part of the node placement algorithm to reduce the average wire length. Unlike the algorithm suggested in [107], in our experiments, the maximum wire length is not reduced. The overall wire length is reduced by reducing the average distance between connected rows and columns.

Table 5.1. Results for different parity-check matrix sizes. (original/reordered matrix)

Code Size	Ave. row-range	Max row-range	Ave. col-range	Max cut-size	Ave. cut-size
350x808	314/357(12%)	800/798	428/602(29%)	256/345(26%)	185/264(30%)
500x1154	243/512(15%)	1133/1137	604/866(30%)	373/492(24%)	266/375(29%)
800x1848	703/804(13%)	1829/1827	965/1379(30%)	598/790(24%)	421/596(30%)
1200x2773	1037/1189(13%)	2766/2761	1455/2046(29%)	878/1168(25%)	618/881(30%)

5.4 Summary

LDPC code design depends on many parameters. They could be obtained by a variety of construction methods. Their hardware implementations are also as varied as their construction. There are many issues to be considered when designing LDPC encoders and decoders. Decoder architectures are broadly classified according to the number of processing nodes compared to the size of the code. Beyond the number of processing nodes, the list of other factors to be considered include the type of MPA algorithm, node architecture, scheduling, quantization, interconnection network and corresponding encoder. Most of these factors overlap in determining the size and processing time of the hardware and code performance. There are several methods already suggested for tackling each issue, the appropriate choice will depend on the target application and tradeoffs between performance, size and processing time.

Hardware LDPC decoder architectures are based on the structure of LDPC codes. Fully parallel decoders have high interconnection complexity. Reordering the code matrix can reduce the area and average wire-length of interconnections. Reordering algorithms move row-column connections closer to the diagonal. Although the matrix space does not accurately represent an actual chip floor plan, the results obtained show that rearranging rows and columns of a LDPC code could reduced the routing complexity. On average using RCM, wire-length reduction of 15% was achieved with average maximum length and maximum cut-size at 25% and 30% respectively for a code a 1K. The gradual increase/decrease in connection range of consecutive rows would reduce placement complexity. The reordering also makes the connections more predictable. We get the same

shape for all the matrix, therefore optimized placement and routing algorithms for this type of shape or profile could be developed.

This page is blank

Chapter 6

Quasi-Cyclic LDPC Decoder Architectures

Partly parallel architectures are often designed for structured codes. Structured codes have characteristics that could be exploited to reduce hardware implementation cost and complexity. In particular the structure of a code affects the processing node interconnection network choice and message scheduling. Quasi-cyclic codes are one example of structured codes that have reduced hardware complexity and cost in both the encoder and the decoder. Quasi-cyclic codes are constructed by division of the code matrix into sub-matrices which divides rows and columns into groups that can be mapped onto a single processing node. The cyclic shifting of identity sub-matrices simplifies routing and addressing of messages within processing nodes. The cyclic structure could also be exploited to reduce encoding hardware and delay.

LDPC decoders for quasi-cyclic codes can be realized in many different architectures examples of which are found in [9][74][84][111] and references within. Decoder architectures mainly differ in processing node interconnection, communication scheduling and node implementations. As was stated in the previous chapter, several issues need to be addressed in some way to design and implement an efficient decoder for a particular application.

In this chapter we look at inter-connection networks and message overlapping techniques for quasi-cyclic LDPC codes. In the first half of the chapter we explore the use of multi-stage interconnection networks as a means of communication between check and variable processing nodes. Although multi-stage networks have large delays compared to other communication implementations they are more flexible in accommodating a wide range

code designs.
In the second half of the chapter we discuss message overlapping techniques for quasi-cyclic codes. Overlapping could reduce decoding time by up to 50%. We adopt the technique of matrix permutation to quasi-cyclic codes, introduce matrix space restriction and discuss modifications to the existing technique of overlapping quasi-cyclic codes by aligning sub-matrices internally. The amount of overlapping that could be achieved by each techniques is analyzed. A quasi-cyclic structure obtained by configuration of sub-matrices as was done with proposed construction algorithm in Chapter 4 is assumed throughout the chapter.Lastly we propose a flexible and high-throughput LDPC decoder based on developed computation overlapping techniques and our proposed QC-LDPC construction algorithm.

6.1 Interconnection Networks for QC-LDPC Decoders

As with other LDPC codes decoders, interconnection of the nodes in QC-LDPC decoders could be implemented by memory banks, hardwired, crossbar switches or multistage interconnection networks as was described in the previous chapter. Most suggested and implemented decoders use memory banks and hardwired interconnect[74][84]. The advantage of QC-LDPC codes is that rows and columns in a group (sub-matrix) are connected to the same group. Group messages or data could therefore be stored and transmitted as a group. This reduces the number of addresses and transmissions compared to when rows and columns are accessed and transmitted individually.

6.1.1 Hardwired Interconnect

A hardwired interconnection network could be used in a QC-LPDC decoder as shown in Figure 6.1. In such an architecture, one or more sub-matrices or group of rows or columns are mapped on to a processing node. Each processing node computes results for each row or column group. The processing nodes could have one or more processing units for serial and vector processing respectively. Messages are exchanged directly between processing nodes along the connecting wires. The communication could also be serial or in vector form. Although the direct communication between nodes provides fast communication, it does not match all matrix configurations hence restricting the type of codes that could

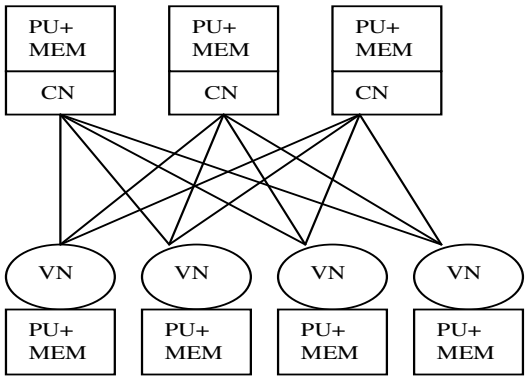

Figure 6.1. Block diagram of LDPC decoder direct interconnection nodes.

be run on the decoder.

The interconnection graph between processing nodes determines sub-matrix configuration of the parity-check matrix. As was also pointed out in the previous chapter, the interconnection architecture requires multiple ports which are expensive in hardware. Large row and column weights will require a large number of ports. Mapping several row or column groups to a single processing node could reduce the number of ports needed, however, it further restricts the flexibility of the decoder. Codes with a large number of sub-matrices per row or column compared to row or column weights can be mapped onto the architecture if the sub-matrix configuration is a replication of the decoder interconnect.

Figure 6.2 (a) shows a matrix in which a (3,4) structure is replicated three times. The matrix can be mapped onto the architecture in Figure 6.1 as it appears with three rows to each check processing node and three columns to each variable processing node. However, if the sub-matrix configuration is random as in part(b) of the figure, then the configuration does not fit the decoder interconnect of Figure 6.1. For such configurations a more flexible interconnect is needed.

6.1.2 Memory Banks

A memory bank interconnect such as the one in Figure 6.3 could be used for executing QC-LDPC codes. In this architecture, code sub-matrices are mapped onto memory banks. Processing nodes communicate by storing and reading data to or from memory. The processing could also be in serial or in vector form in each sub-matrix. One advantage of this architecture is that it uses half the memory compared to other interconnections as

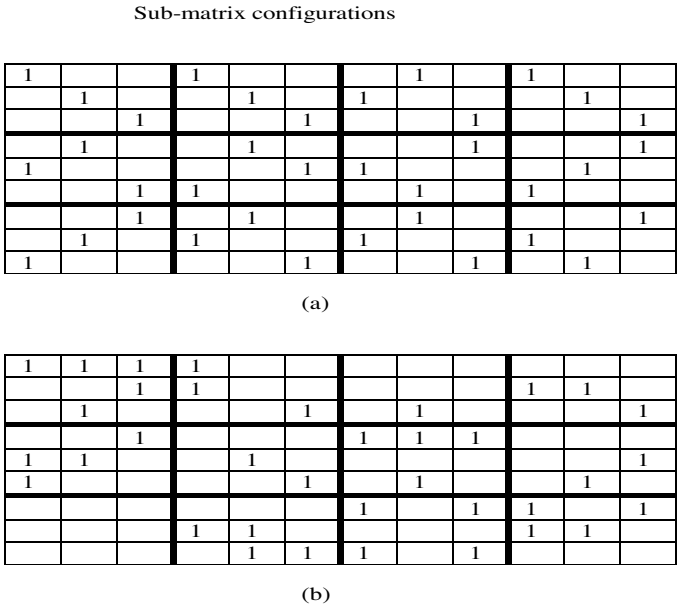

Figure 6.2. Sub-matrix configuration for a parity-check matrix.

both check and variable processing nodes share the same memory banks. As in hardwired interconnect, multiple ports are required.

Larger codes in terms of sub-matrices are executed by mapping more than one sub-matrix onto a memory bank. However, not all sub-matrix configurations can be efficiently mapped onto the architecture. Like the hardwired interconnect, the architecture is well suited for codes which are an exact replication of the decoder interconnect. For example, the configuration of Figure 6.2 (a) can be mapped onto the architecture with each memory bank holding exactly three sub-matrices. However, if sub-matrices are randomly configured as in part (b) of the figure, some memory banks will have more or less than three sub-matrices. For example, the first row is connected to the first three columns which are in the same memory bank. To access messages for the first row, the check node has to read the same memory bank three times. Memory banks with more sub-matrices will create an access bottleneck increasing communication delay. This architecture is not flexible enough to execute the randomized sub-matrix configuration efficiently.

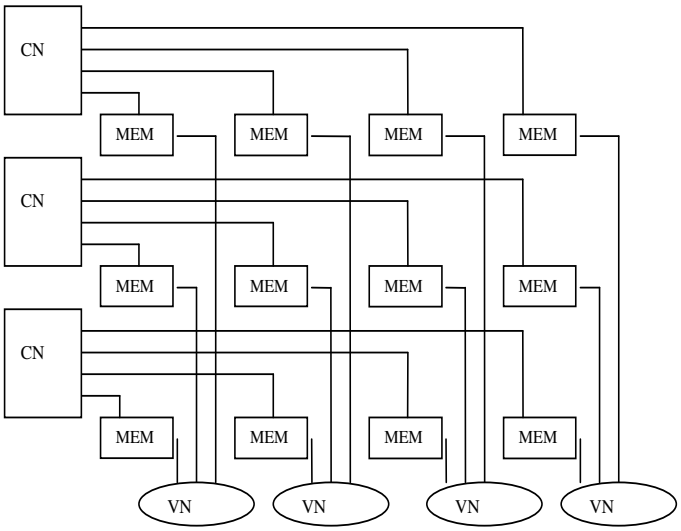

Figure 6.3. Block diagram of LDPC decoder using memory blocks for communication.

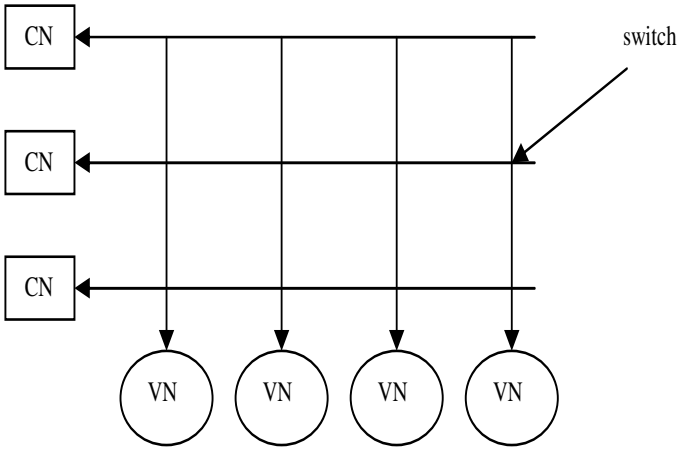

Figure 6.4. Crossbar communication network.

6.2 LDPC Communication through Multistage Networks

Hardwired interconnection and memory banks require multi-ports for processing nodes and are generally not flexible. Crossbar switches could be used for QC-LDPC decoder interconnect to offer flexibility as shown in Figure 6.4. Each check processing node has a switched connection to each variable processing node. Any sub-matrix configuration could be mapped onto the interconnect. The challenge would be to have a mapping that has the lowest communication scheduling time.

Although a crossbar switch offers maximum flexibility it is expensive to implement for a large number of nodes. The number of switches required is $m \times n$, where m and n

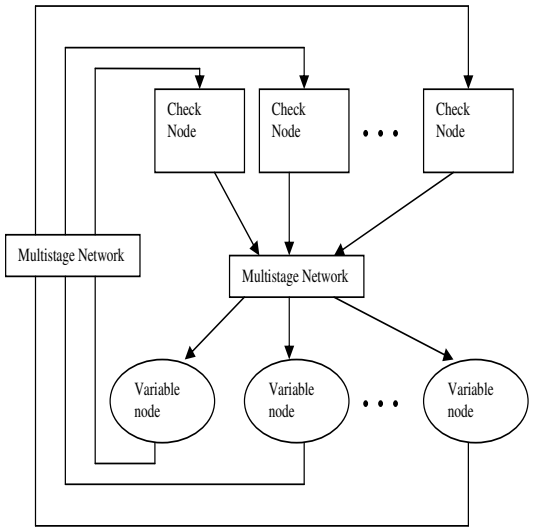

Figure 6.5. Block diagram of a LDPC decoder using multistage networks.

are the number of nodes. Multistage networks have been used in areas where crossbar switches are prohibitively expensive. They could also be used in connecting check and variable processing nodes in LDPC decoders as shown in Figure 6.5. Two unidirectional networks are used for communication. A single bidirectional network could also be used. For message overlapping, which is covered later in the chapter, two networks are required. Multistage networks offer flexibility at a reduced cost. They have a reduced number of switches compared to crossbar switches. However, the number of switches is reduced at the expense of increased delay because of the increased number of stages. These networks also have higher scheduling and routing complexity as data has to be routed through the network using a defined scheduler and routing algorithm.

Multistage stage networks were suggested as a means of communication in a LDPC decoder in [22]. The decoder was implemented by Flarion Technologies Inc.[112] for a wide range of codes the details of which are withheld. The design does not specify the type of multistage network used or if it is used at all. Olcer [111] uses banyan multistage networks for decoder architecture based on array QC-LDPC. Array codes have the number of sub-matrices in rows and columns equal to row and column weights respectively with the sub-matrix size as a prime number. Messages for sub-matrices in array codes can be sent in ascending or descending order through a banyan network without data collisions (data sent to the same output at the same time). Quasi-cyclic codes in which more than row or column-weights of sub-matrices are used and randomized in configuration, scheduling

data in ascending or descending order may increase the number of transmission cycles. For such matrices, we suggest a Benes multistage network.

6.2.1 LDPC Communication

LDPC code computations produce many messages at the same time. Each check node needs k messages for each computation producing k outgoing messages to k variable nodes. Each variable node receives and sends j messages from and to j check nodes. In hardwired and memory banks communication networks, k and j paths are required.
With multistage networks, there is only one connection port between a processing node and the network. Therefore, messages need to be scheduled in and out of the processing node. Also there are no dedicated paths through the network. There is a possibility of data collisions as data is routed through the network. Handling data collisions requires extra hardware and will increase communication delay. To efficiently send data across the network without data collisions scheduling and routing of messages are needed.
Data collisions could be avoided by scheduling communication using edge coloring. The Tanner graph is edge colored such that no vertex (nodes) have the same color more than once. Scheduling data sequentially by color ensures that each node sends or receives one distinct data (color) at a time to and from a distinct node. A bipartite graph could be edge-colored using the number of colors equal to the highest edge degree, which is the highest check node degree in LDPC codes [113]. Since the row/column group connections do not change through the lifetime of a code, a scheduling timetable can be computed and stored. The scheduling timetable is determined by edge coloring. The advantage of a multistage network compared to hardwired and memory banks is that more flexibility is allowed in sub-matrix configuration. Sub-matrix configuration does not necessarily have to match the interconnection structure. This is possible because every check node has a connection to every variable processing node and vice versa. The random sub-matrix configuration example of Figure 6.2 (b) can be mapped onto the decoder with a multistage network in such a way that the communication delay is reduced. Scheduling and routing through banyan and Benes networks are discussed below.

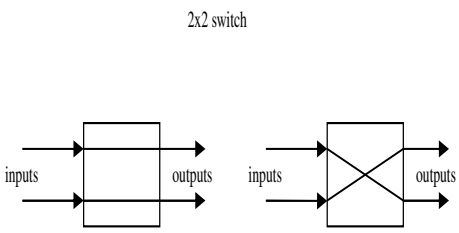

Figure 6.6. 2×2 switch passes input data to lower or upper output port.

6.2.2 Multistage Networks

Multistage networks are built by small switches arranged in stages. In our case we consider only binary switches. A binary switch has two inputs and outputs as shown in Figure 6.6. We also assume that routing is determined by addresses attached to input data and that each switch senses and then removes one bit address from the input data. The switch routes data to the upper output port if input address tag is '0' and to the lower output port if its address tag is '1'.

There are a wide variety of multistage interconnection networks varying in topology (switch interconnection), routing, switching strategies, flow control, delay and cost. Based on the LDPC communication pattern discussed above, a non-blocking (no internal data collisions at ports), self-routing and cheap network is most desirable. Blocking networks would increase the number of communication cycles and decoding delay. Self-routing networks determine the path of data based on address tags or bits attached to the data. Since routing is pre-computed, self routing offers a low routing complexity. The cost and delay of networks is determined by the number of switches. Based on these characteristics we determined that banyan and Benes networks are most appropriate for structured and semi-parallel LDPC decoder interconnect.

6.2.3 Banyan Network

Banyan networks are a family of multistage interconnection networks that have a single path between inputs and output. They are blocking and self-routing networks with $slog_2s$ switches and log_2s stages, where s is the number of inputs or outputs. They have the fewest number of switches and stages compared to other types of networks[114]. Banyan networks are non-blocking if the data is sorted in decreasing or increasing destination address order[111]. When the number of row and column groups is equal to the row and

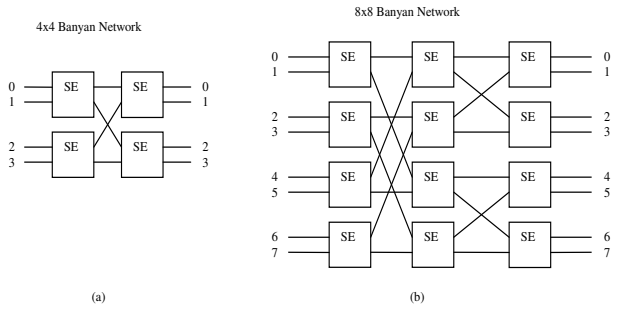

Figure 6.7. 4x4 and 8x8 banyan networks.

column weights, group scheduling could be done in k steps in ascending or descending order. Hence for codes with j and k number of groups, banyan type networks could be used in the architecture of Figure 6.5. Examples of banyan networks include Omega and Delta networks[114]. Figure 6.7 shows 4×4 and 8×8 banyan networks. Banyan networks are built recursively using binary switches. The last two stages of the 8×8 network is made up of two 4×4 networks.

Tables 6.1 and 6.2 show scheduling and routing information for an $(N, 3, 4)$ code through a 4×4 banyan network. In variable-to-check node communication (Table 6.1) there are more source nodes than destination nodes. Three nodes (variable nodes,VN) send data at a time to match with the three destination nodes (check nodes,CN). The table shows the source variable nodes and destination check nodes together with the address tag for the three data transmissions. The notation $X - Y(ZZZZ)$ is used to show the source, destination and address, where X is the source, Y is the destination and $(ZZZZ)$ is the routing address. The last three columns of the table show variable nodes from which check nodes are receiving their data. The information could also be found from columns 2 to 5. The destinations always appear in sorted or cyclic order in each schedule. The ith stages of the network uses the ith bits of the address tags to determine the direction of the inputs data. In Table 6.2 three check nodes send data to four variable nodes. All check nodes send data at the same time to different and ordered variable nodes. Four schedules are also required to send all data across the network.

Banyan networks could be used with quasi-cyclic codes with zero sub-matrices (number of row or columns groups larger than k or j) if the parity-check matrix is obtained by replication of a basis matrix as in Figure 6.2 (a). Exactly three rows or columns are

Table 6.1. Variable to check nodes communication

schedule	$VN_0 - CN$	$VN_1 - CN$	$VN_2 - CN$	$VN_3 - CN$	CN_0	CN_1	CN_2
1	0-0(0000)	1-1(0101)	2-2(0010)		0	1	2
2	0-1(0001)	1-2(0110)		3-0(0100)	3	0	1
3	0-2(0010)		2-0(0000)	3-1(0101)	2	3	0
4		1-0(0000)	2-1(0101)	3-2(0010)	1	2	3

Table 6.2. Check to variable nodes communication

schedule	$CN_0 - VN$	$CN_1 - VN$	$CN_2 - VN$	VN_0	VN_1	VN_2	VN_3
1	0-0(0000)	1-1(0101)	2-2(0010)		0	1	2
2	0-1(0001)	1-2(0110)	2-3(0011)		3	0	1
3	0-2(0010)	1-3(0111)	2-0(0000)		2	3	0
4	0-3(0111)	1-0(0000)	2-1(0101)		1	2	3

mapped onto each check or variable processing node respectively. Taking the groups in each processing nodes as one large group, the groups can be scheduled as in Tables 6.1 and 6.2. The difference is that in the expanded structure there are three sub-matrices in one schedule. With the basis matrix it takes k cycles to schedule messages across the network as shown in Tables 6.1 and 6.2. When the basis graph is replicated three times, then the number of communication cycles is $3k$.

If the sub-matrix configurations are randomized as in Figure 6.2 (b) sorting the input does not guarantee the same communication delay. The configuration in Figure 6.2 (b) could be scheduled through a 4×4 banyan network with sorted destinations. Since the sub-matrices are not evenly distributed, sorted scheduling will likely be more than $3k$ cycles.

The communication delay will also depend on how data is used. If data is produced and consumed immediately scheduling in sorted criteria may not produce the shortest delay. When destination addresses are not ordered there is a possibility of data collisions in the network. There are various techniques used to overcome this limitation [115][116]. The techniques include sorting the data before inputting it to the banyan network, providing alternative or extra paths, buffering etc.

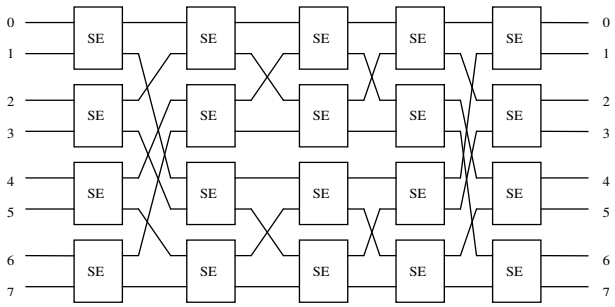

Figure 6.8. A 8x8 Benes network.

The drawback of these techniques is that they either add more hardware or delay. Some techniques such as addition of extra paths, do not completely eliminate the possibility of a collision. For codes with a larger number of sub-matrices than row or column weights scheduling by sorting according to destination may result in more communication delay. The scheduling may not be done in $3k$ cycles. For such configurations rearrangeable networks such as a Benes network would be more appropriate compared to a banyan network.

6.2.4 Benes network

A Benes network is the least costly rearrangeable network with $O(slogs)$ switches[117][118]. Figure 6.8 shows an 8×8 Benes network with five stages. Rearrangeable networks are non-blocking if the data is rerouted for each input-output combination. The path of data at a particular input depends on the destination of other inputs to the network. Since all LDPC code input-output permutations are known before transmission, all the routing addresses can be pre-computed. Addresses are used at each stage to direct the data to the correct destination. Routing in a Benes network could be done using a looping algorithm[117]. To realize a given input-output permutation $(input_i, output_i, i = 1...s)$ in an $s \times s$ Benes network, for each connection $(input_i, output_i)$ the algorithm determines whether the connection goes through the upper or lower $\frac{s}{2}$ subnetwork. The process is repeated recursively for each of the subnetworks. The upper subnetwork is given a 0 address and lower subnetwork a 1.

Randomly distributed or arranged sub-matrices as in Figure 6.2 (b) can be routed through a Benes network more efficiently than on a banyan network. The Benes network has more paths and does not require that inputs destinations be sorted.

6.2.5 Vector Processing

With the communication through multistage networks described above, serial or vector processing of sub-matrices in the check and variable nodes could be used. If serial processing is used the decoder would be very inefficient. For each processing check node, k messages are to be read or received one at time. k messages are also sent out one at a time since there is only one port in or out of the processing node. The same applies to variable processing nodes with j messages. With serial communication it takes kp cycles for communicating, where p is the size of sub-matrices. The single-instruction multi-data (SIMD) processing of LDPC decoding suits vector processing well. In semi-parallel architectures the number of processing elements is less than the size of the code. Processing nodes could read and process many rows or columns per memory access. Processing one row requires k reads and writes and j for a column. When several nodes are read and processed at the same time k or j read/write will serve several rows or columns at the same time. With vector processing and communication it takes $\frac{kp}{f}$ cycles for transmission of messages, where f is the vector size. Vector processing reduces decoding delay and power consumption by reducing the number of memory reads and writes. However, vectorization comes at a cost with more processing nodes, large ports, buses and network switches.

6.3 Message Overlapping

Besides MPA algorithm modifications for faster convergence and computations, decoding time could be reduced by overlapping variable and check node computations and communications. Traditionally variable node computations start after the entire cycle of check node processing is completed and vice versa. Generally it cannot be guaranteed that all messages for a particular row or column would be available when needed if overlapping of computations is done arbitrarily. Figure 6.9 (a) shows conventional node computation scheduling (Communication time is assumed to be included in check and variable processing times). In the first half of each iteration check nodes are processed and variable nodes are processed in the second half. Without overlapping, one iteration is the time it takes to process all check nodes plus the time to process all variable nodes.

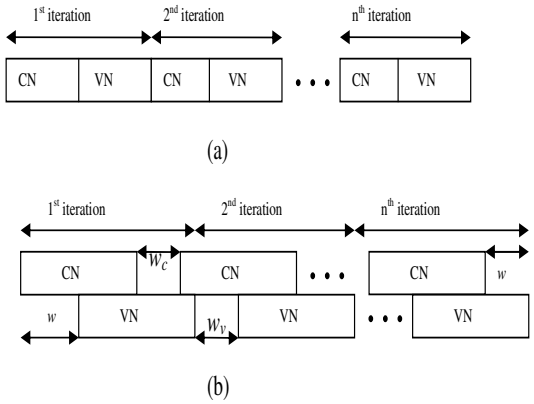

Figure 6.9. Computation scheduling of check and variable nodes with and without overlapping.

There are more variable nodes than check nodes in a Tanner graph. However, check nodes have more connections and higher computational complexity compared to variable nodes. For simplicity, we assume it takes q clock cycles to process one set of nodes (this assumption does not affect decoding time analysis). Then conventional decoding time is $2nq$ cycles, where n is the number of iterations. With overlapping of node processing, variable nodes start processing while the check nodes have not completed their processing. Check nodes also start processing while variable nodes have not processed all of the columns.

In the first iteration variable nodes wait a number of cycles,w, for some columns to have all their messages (check to variable messages) updated for computations. Part (b) of Figure 6.9 shows overlapped node processing for n iterations. With overlapping, the number of decoding cycles could be less than $2nq$. The decoding time is equal to the number of cycles for processing check nodes plus the number of cycles check node processing is stalled. Nodes are stalled when they have to wait for their messages to be updated before computations. It could also be calculated as the number of variable nodes processing plus stalls. Overlapping decoding time is hence expressed as $nq + (n-1)w_c + w$ or $nq + (n-1)w_v + w$ using check and variable nodes processing times respectively. Inter-iteration waiting times, w_c and w_v are the number of cycles that check and variable nodes are stalled between iterations. The number of overlapping decoding cycles approaches nq as w,w_c and w_v approach zero, which is half the conventional or non-overlapping time. Overlapping decoding gain in time is given by

$$Gain = \frac{2nq}{nq + (n-1)w_v + w} \tag{6.1}$$

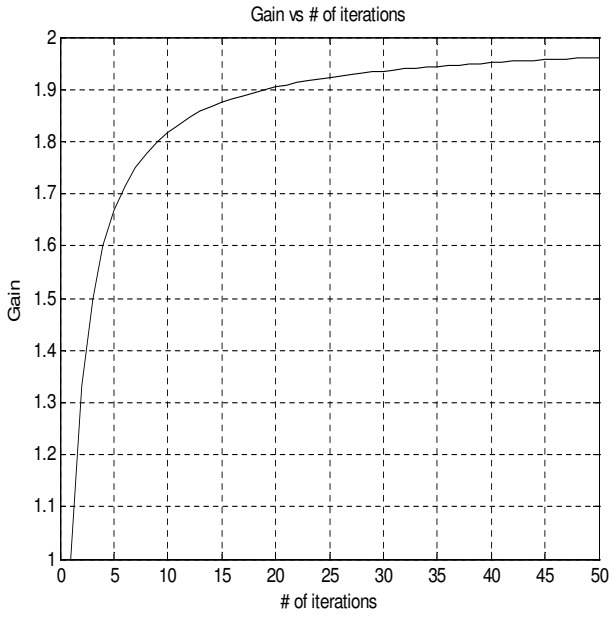

Figure 6.10. Plot of gain with respect to the number of iterations when inter-iteration waiting time is zero.

If w_v is zero the gain is given by

$$Gain = \frac{2nq}{nq + w} \tag{6.2}$$

As w tends to zero, gain tends to 2. Also, as w tends to q gain can be expressed as

$$Gain = \frac{2n}{n + 1} \tag{6.3}$$

which depends on the number of iterations and approaches 2 for large n.
According to equation 6.3 if the inter-iteration stalls are eliminated and the waiting, w, is as large as q, the gain largely depends on the number of iterations only. Figure 6.10 shows a plot of gain versus the number of iterations using equation 6.3. The rate of increase in gain decreases considerably beyond twenty iterations with at least a gain of 1.9. The graph shows that if inter-iteration waiting times are eliminated the gain is still good even if the waiting time,w, is not short (compared to q) with a sizeable number of iterations. The gain is reduced when the inter-iteration waiting times are non-zero. In this case the gain depends on how much w_v is compared to q. Larger inter-iteration times would adversely reduce gain as they are multiplied by the number of iterations. Therefore

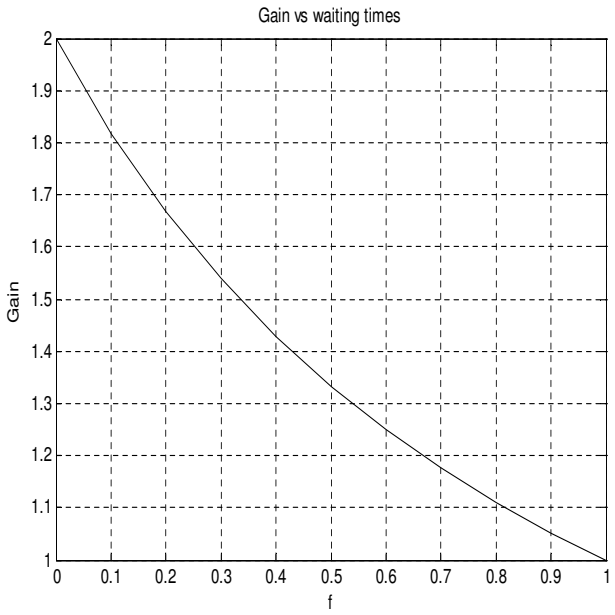

Figure 6.11. Plot of gain with respect to waiting times compared to sub-matrix size,p.

keeping w_v as small as possible is important in maximizing the gain. Assuming w_v is the same as w the gain equation can be written as

$$Gain = \frac{2nq}{nq + n(wv)} = \frac{2nq}{nq + nqf} = \frac{2}{1+f}, \tag{6.4}$$

where $f = \frac{w_v}{q}$. With this equation the gain depends on how much the waiting times are compared to the size of check or variable nodes,q. If there are almost zero then the gain is 2, and if there are as large as the q there is no gain. Figure 6.11 shows the gain plot with varying f. For gain to be above 1.9, the waiting times have to be less than $0.05q$. The assumption that $w_v = w$ above is valid in some cases. It depends on the techniques used for overlapping as we will see in the following sub-sections.

For overlapping to work all messages for some columns must be available while some rows are still being processed. The same is true with rows. In [99] overlapping is achieved by matrix permutation for any matrix while in [119] the choice of starting rows and columns allows overlapping in quasi-cyclic codes. Overlapping does not only increase throughput by reducing processing time, it also improves hardware utilization efficiency. Hardware utilization efficiency is the amount of time processing units are used over the overall processing time. The increase in throughput is directly proportional to hardware utilization efficiency. Without overlapping, processing nodes are unused half of the time.

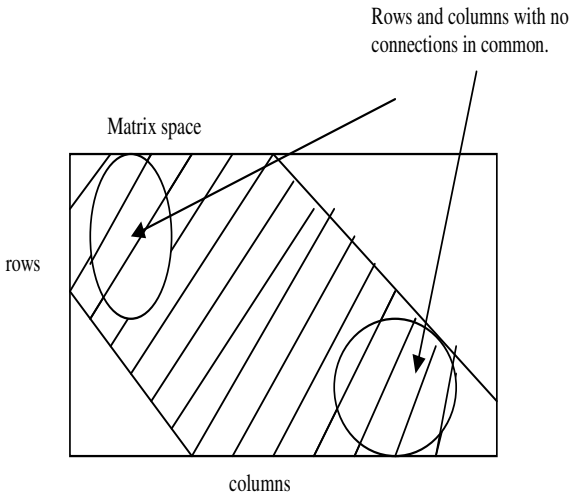

Figure 6.12. Row-column connections space.

When check nodes processing units are computing, variable nodes processing units are idle. Below we suggest new methods and modify existing methods of overlapping check and variable node computations for quasi-cyclic codes.

6.3.1 Matrix Permutation

A LDPC matrix could be rearranged such that rows and columns that do not have connections are separated and are at extreme ends of the matrix space. With such a separation, rows and columns with no connections in common can be processed in parallel. Figure 6.12 shows a matrix space in which some rows and columns do not have connections in common. Row-column connections are in the shaded area. Since the top-left rows do not have connections to extreme right columns, these rows and columns could be processed at the same time as they do not have data dependencies.

An algorithm was suggested in [99] to rearrange a given parity-check matrix such that row-column connections with no connections in common are separated. The algorithm effectively moves row-column connections closer to the diagonal such that empty spaces are left at the left-bottom and top-right corners. Since top-left rows do not have connections at top-right corner, their processing could be overlapped with that of columns at bottom-right corner. The permutation algorithm as suggested in [99] is as described below.

Matrix Permutation Algorithm

1. Initialize row and column sequences to null.

2. Select a row r_k randomly.
   Order row indices randomly and pick the first one. Go to step 4.

3. If all the rows contained in a column are selected, such a column is called a completed column. If a column contained in a row is also in the column sequence, it is called a common column of the row. For each unselected row, count the number of common columns (CC) and the number of columns to be completed (CTC) when the row is selected. Select a row from unselected rows, r_i, that has the maximal CTC. If there are more than two rows associated with the maximal CTC, select the one that has the largest CC.
   The row sequence is appended by r_i, and the column sequence is enlarged by appending the columns of r_i that are not in the current column sequence.

4. Find completed columns, and move them to the front of the first uncompleted column in the column sequence.

5. If there are unselected rows, to go step 3.

6. Permute the parity-check matrix according to row and column sequences.

Steps 3 to 5 of the algorithm does the permutation of the matrix. The algorithm tries to move connections between rows and columns closer together. This is done by connecting rows and columns that are likely to form a cluster. This is achieved by selecting that have maximal CTC in step 3 of algorithm. Moving row-column connections together separates row and columns that do not have connections. Rows and columns that do not have connections in common do not have data dependencies and could therefore be processed in parallel.
The matrix permutation algorithm can be applied to any matrix (random, structured, any size etc). Its effectiveness depends on the structure of the code (row-column interconnections and row and column weights). The algorithm is similar to sparse matrix reordering algorithms such as RCM, King's and GPS, which were used in the previous

chapter (section 5.2.1) to reduce matrix space and overall connection ranges. It is in particular similar to the King's algorithm in that it moves rows or columns which are likely to form a cluster first. We can thus use known sparse matrix permutation algorithms such as the ones mentioned here to reorder LDPC matrices.
As was shown in the previous chapter the reduction in row-column connection area depends on row and column weights, with smaller weights having larger reductions (equation 5.5). Hence high rate codes will offer smaller overlaps. In [99] a time reduction of 25% was obtained for a $(N, 3, 6)$ example matrix.
Figure 6.13 (a) shows an example of a $(12, 3, 4)$ LDPC matrix. The matrix could be rearranged such that connections are closer to the diagonal as in part (b) of Figure 6.13. As a result some row and column computations could be overlapped. For example if there are three check node processing units and four variable node units, scheduling could be done as shown in Figure 6.14 with two iterations. After processing rows 1 to 6, columns 1 to 4 will have all their messages updated. When check nodes are processing the last group of rows, variable nodes can start processing the first group of columns. In the second iteration rows from group one have all their messages after two column groups are processed so check nodes also have to wait one group cycle to start processing. The first and second iteration scheduling is repeated in the subsequent iterations. With this overlap the decoding time is reduced by a third.

We propose to apply matrix permutation matrix to quasi-cyclic codes by using the permuted matrix as a basis matrix. To construct quasi-cyclic codes we design a small code such as the example in Figure 6.13 to be the basis matrix. An expanded matrix is then obtained from the basis matrix by replacing each '1' by a $p \times p$ shifted identity sub-matrix and each '0' entry by a $p \times p$ zero sub-matrix. The identity matrices are shifted according to the construction algorithm developed in chapter 4 to get a girth of at least six. The expanded code has the same overlap and reduction in processing time as the basis matrix with the number of processing units remaining the same.

6.3.2 Matrix Space Restriction

Reducing decoding time by matrix permutation is limited especially when column and rows weights are high. With high weights it is harder to have all connections localized without having connections covering the whole matrix range. We propose another method

1	1			1			1		1			
2		1						1			1	1
3	1		1		1				1			
4			1						1	1	1	
5	1			1		1		1				
6		1		1	1		1					
7		1				1				1		1
8			1				1	1		1		
9					1	1					1	1
	1	2	3	4	5	6	7	8	9	10	11	12

(a)

1	1	1	1	1								
2	1			1	1	1						
3	1	1					1	1				
4		1	1			1			1			
5			1		1			1		1		
6				1	1					1	1	
7						1	1				1	1
8							1		1	1		1
9								1	1		1	1
	1	2	3	4	5	6	7	8	9	10	11	12

(b)

Figure 6.13. Scheduling by rearranging the matrix (a) original constructed LDPC matrix (b) rearranged LDPC matrix.

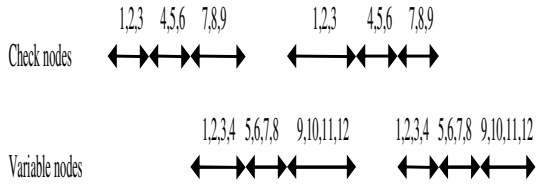

Figure 6.14. Overlapped processing of the rearranged matrix.

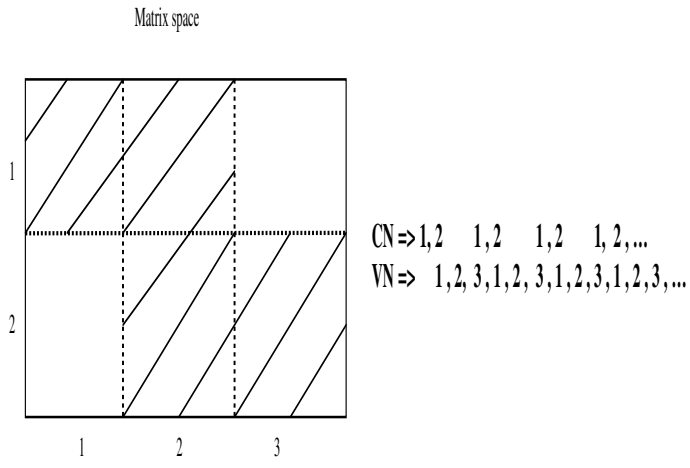

Figure 6.15. Overlapping by matrix space restriction.

1	1		1		1			1		1		1
2		1		1		1	1		1		1	
3	1			1	1	1		1	1			
4		1	1		1		1			1		1
5	1	1		1			1	1			1	
6			1			1			1	1	1	1
	1	2	3	4	5	6	7	8	9	10	11	12

(a)

1	1	1	1	1		1		1				
2	1	1	1		1	1		1				
3	1	1	1	1			1		1			
4					1		1		1	1	1	1
5				1	1			1		1	1	1
6						1	1		1	1	1	1
	1	2	3	4	5	6	7	8	9	10	11	12

(b)

Figure 6.16. Quasi-cyclic basis matrix (a) without space restriction (b) with space restriction.

that achieves check and variable node computations by deliberately isolating nodes without connections in common. We call this method or technique, matrix space restriction. The row-column connections of a code matrix are constrained such that there are blocks of rows and columns that could be processed in parallel without data dependencies.
Figure 6.15 shows matrix space row-column connection restrictions divided into two row and three column groups respectively. Connections are made in the shaded region. The space is divided such that some check and variable node computations can be overlapped. The top rows do not have connections to far-right columns. Specifically, group 1 rows and group 3 columns are not connected. Assuming row and column groups take the same amount of time we can schedule the processing of groups as in the figure. One row or column group is processed at a time. Without overlapping the total processing time is $5n$ for the five groups, where n is the number of iterations. With overlapping the processing time is $3n + 1$. Variable node processing waits for one row-group processing in the first iteration. The gain in time for this scheduling is $\frac{5n}{2n-1}$ which is approximately a reduction in time of 40% as n gets large.

The advantage of this technique is that all matrices will have the same time reduction for a particular restriction basis matrix. However, restricting matrix connections may reduce decoding performance. As with rearranged matrices, a basis matrix could be designed and then expanded with $p \times p$ sub-matrices to obtain quasi-cyclic LDPC codes.

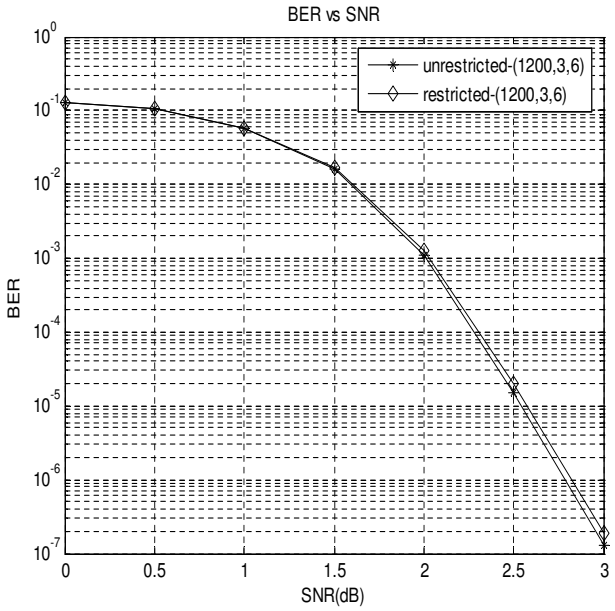

Figure 6.17. BER performance of restricted and unrestricted qc-ldpc codes.

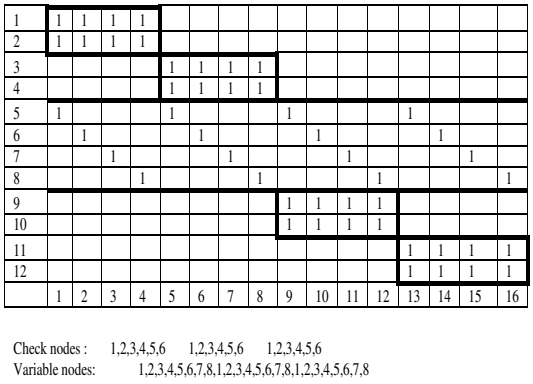

Figure 6.18. Another overlapping by matrix space restriction.

Figure 6.16 shows two $(N, 3, 6)$ basis matrices codes with and without space restrictions. There are 6 row groups and 12 column groups. The first code, Figure 6.16 (a), is designed by pairing column and row groups randomly with the condition that row and column groups appear 3 and 6 times respectively. In the second code, part (b) of the figure, group pairings are made only in the restricted connection space according to Figure 6.15. With a group size of 100, (1200,3,6) codes were constructed according to the basis matrix in Figure 6.16. The restricted code shows a BER performance degradation of about 0.1dB at 10^{-5} BER as shown in Figure 6.17 compared to the unrestricted code. The codes have the same girth of 6 and average girth of 7.333 using the quasi-cyclic LCPC code construction algorithm proposed in chapter 4. In the restricted code connections are

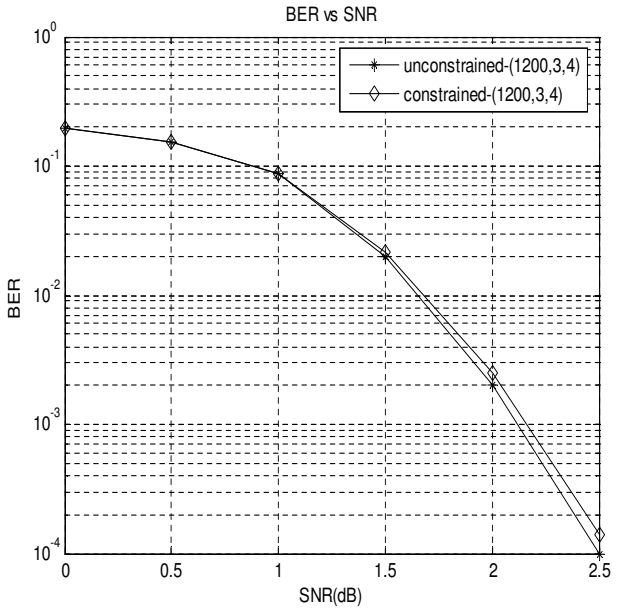

Figure 6.19. BER performance of restricted and unrestricted qc-ldpc codes using second space restriction (25 iterations).

divided into two sub-graphs with fewer messages crossing through the few connections in the middle.

Figure 6.18 shows another space restriction in which short row-column connection spaces are connected to one long connection space. When two rows and columns are processed at a time the processing can be overlapped as shown in the figure. Without overlapping, the total decoding time is $14n$. With overlapping the time is reduced to $8n + 3$, which is a reduction of 42%. Performance could be traded with processing delay depending on the demands of the target application. Figure 6.19 shows decoding performance of this structure compared to a randomized row-column connection space. The unconstrained code shows slightly better performance compared to the constrained code.

6.3.3 Sub-Matrix Row-Column Scheduling

Overlapping in quasi-cyclic LDPC codes could also be achieved by calculated scheduling of rows and columns of the sub-matrices[74][119]. When sub-matrices belonging to the same column group are processed one row at a time in consecutive order in parallel, some columns will have all their messages updated at some point. Columns which have all their messages updated could be processed while rows are still being processed. The time

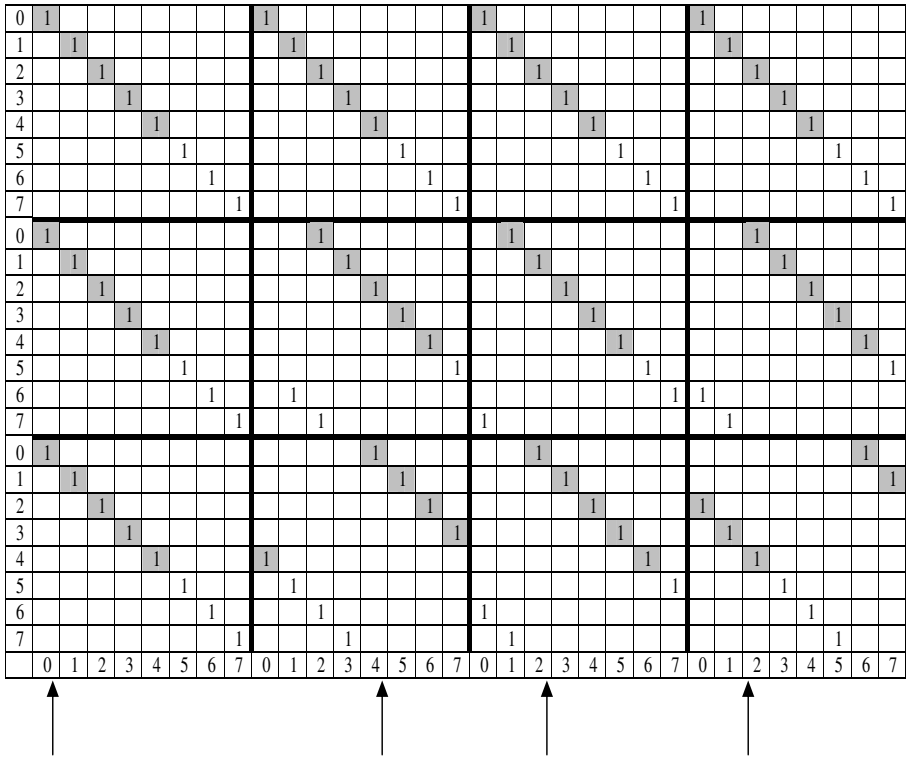

Figure 6.20. quasi-cyclic code.

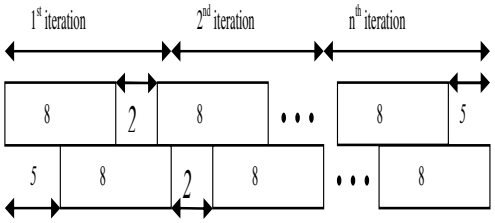

Figure 6.21. Scheduling example of check and variable nodes with overlapping.

it takes, for one column to have all its messages updated is the waiting time, w, which depends on the shift values on the sub-matrices and where row processing was started. The shift value for a sub-matrix is the column number the first row of the sub-matrix is connected to. Columns with all of their messages ready could be processed before row processing is completed. Similarly, some rows will have all their messages ready while columns are still processed. Columns or rows with all updated messages are in consecutive order because of the cyclic structure of QC-LDPC codes. The consecutive order allows us to know which columns or rows will have their messages updated next.

Figure 6.20 shows an example of a quasi-cyclic matrix with sub-matrices of size 8×8. If all

row processing in each row group starts at row 0, the waiting time for column processing is 5. That is, after processing five rows in each row group some columns in each column group will have all their messages updated and will be ready for computation. Processed rows are shaded. The arrows at the bottom of the figure show which columns will have all their messages updated after 5 rows are processed. In column group 1, column 1 has all its values after 1 row operations. In column group 2 it takes 5 row operations to have one column with all of its messages updated. In column group 3 and 4, 3 and 5 row operations are needed respectively. Therefore after processing 5 rows, we could start processing columns in each column group starting at the indicated columns. In the 6^{th} cycle the next columns to the starting columns will have all their messages updated. Columns are processed consecutively from the starting columns.

The overlap between variable and check nodes processing may also have a stall. That is, by the time check nodes finish processing none of the rows have all their messages updated from columns. In the example of Figure 6.20, zero rows are all updated after 5 columns are processed. The waiting time for rows in the second time is 2 cycles since three columns were processed in the overlap. Figure 6.21 shows overlap scheduling for our example in Figure 6.20. The inter-iteration waiting times w_c and w_v are equal to 2 with w equal to 5. For 20 iterations, non-overlapping scheduling takes 320 cycles whereas overlapping takes 241 cycles. The processing time is reduced by 25% with overlapping.

Waiting Time Minimization

As already stated, processing starting points for columns and waiting times for a code depend on where row processing was started in each row-group (sub-matrix in that row) and the shift values of each sub-matrix. When a quasi-cyclic LDPC code is constructed sub-matrices in a column group are shifted by some values. The minimum waiting time in each column group is obtained if starting rows (first processed rows in each sub-matrix) have the same difference between themselves as sub-matrix shift values. That is, if processing is started from rows x_1 and x_2 from two sub-matrices with shift values s_1 and s_2, then the column minimum waiting time is obtained if the differences $\mid x_1 - x_2 \mid$ and $\mid s_1 - s_2 \mid$ are equal.
For the example in Figure 6.20, sub-matrices in the first column group are all shifted by zero. Any row processing of sub-matrices with the difference between the starting rows

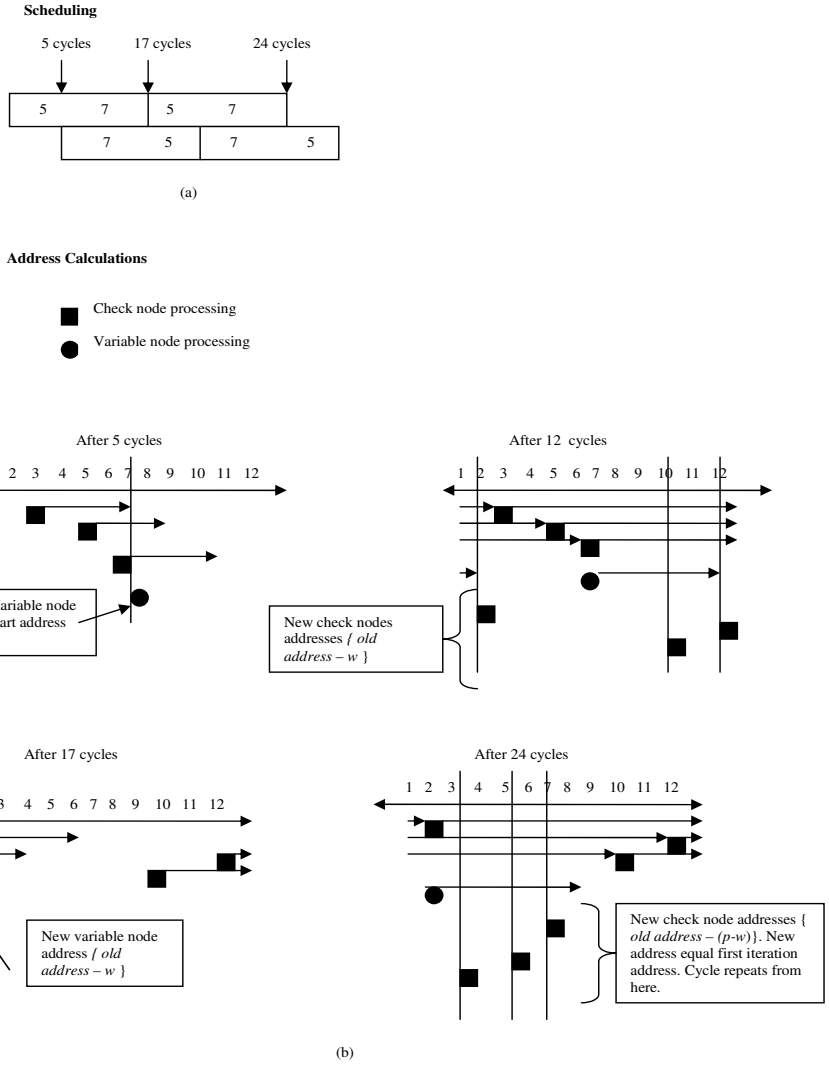

Figure 6.22. Calculation of starting addresses for check and variable nodes with overlapping.

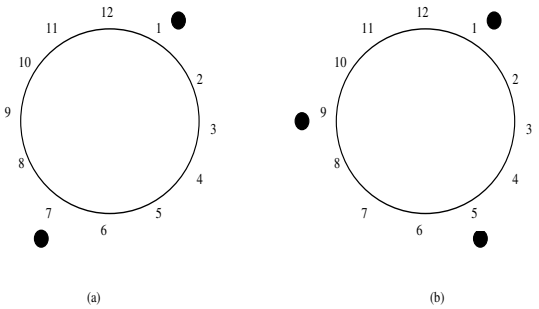

Figure 6.23. Maximum distance covering all points on a circle (a) with two points (b) with three points.

as zero will have the minimum waiting time of one. To minimize waiting time for a code, the difference between starting rows must be as close as possible to the difference in sub-matrix shift values of the code.

In the example code of Figure 6.20, the four column groups are shifted by {0,0,0},{0,2,4} ,{0,1,2}, and {0,2,6} where each triplet is the shift values of the sub-matrices from top to bottom. If processing starting rows have the same differences between themselves as shift values, the waiting time is minimized. Column group 1 shifts have a difference of zero. Starting at rows with a difference of zero, such as {4,4,4} gives the minimum waiting time of one. However, starting rows apply to all sub-matrices which do not have the same shift differences. Therefore, finding the minimum waiting time becomes an optimization problem.

In [74][119], a heuristic algorithm is suggested to minimize the waiting time. The heuristic method calculates the waiting time by taking one row block or group as the reference (ie. reads from 0 to $p-1$). Then the minimum waiting time is calculated for each remaining $(j-1)$ row-groups against the reference group. The process is then repeated with the remaining $j-1$ row groups as reference groups. The combination resulting with the smallest waiting time wins.

Finding the minimum w does not guarantee that w_v and w_c are minimum. The delay of w only ensures that starting columns will have updated messages for the first iteration. For subsequent iterations, there could be stalls. Stalls are eliminated in [74] by subtracting w from check and variable node starting addresses for even iterations. For odd iterations starting or next addresses are determined by subtracting $p-w$ from the current addresses. With this technique the inter-iteration gaps (w_c, w_v in Figure 6.9) are reduced to zero or one cycle.

Figure 6.22 shows calculation of addresses for check and variable nodes in a group size of 12. Part(a) of the figure shows scheduling of check and variable node computations. Part (b) shows starting addresses for check and variable nodes at some intervals. Only one variable or column group starting address is shown. Check node starting addresses are 3,5 and 7 as shown in Figure 6.22 (a). The covering distance or waiting time is equal to 5, starting from 3 through 5 to 7. After 5 cycles variable nodes start processing at point 7. At the end of check node processing, variable nodes have processed $p-w$ (12-5=7) points. For check nodes to start processing, we must access those points that are already processed by variable nodes. Starting at check node positions and counting w backwards

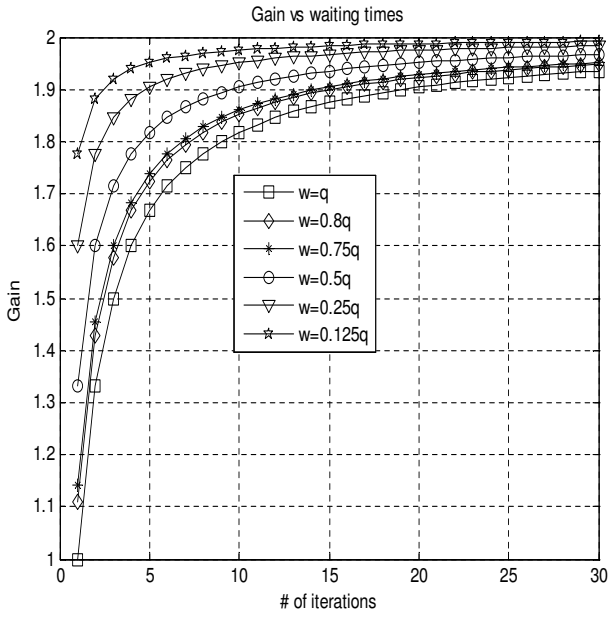

Figure 6.24. Gain with varying waiting time and zero or constant inter-iteration waiting time .

lands us in the variable node processed region. After 12 cycles we look for rows that already have all their messages. Rows 3,5, and 7 have not yet been covered by column processing. Recalculating starting addresses using the $(p-w)$ technique changes starting points from 3,5,7 to 10,0,2. All the new starting points are already processed by variable nodes by the end of check node processing in iteration one.

Part (b) shows the second iteration. By the end of the first iteration for variable nodes, starting point 7 does not have all its messages. The next starting point is calculated by counting w backwards which lands us at point 2. You will notice, point 2 has just been covered by all the check nodes starting points. After $p-w$ cycles of variable nodes, check nodes are at the end of iteration two processing. The next starting points are calculated by counting $p-w$ backwards which lands us at points 3,5,7 where we started. By the time check nodes finish iteration two, point 7 will be covered. The process is repeated for the number of iterations given.

Maximum Waiting Time

The algorithm described above for minimizing the waiting time assumes the number of row and column groups is equal to the row and column weights respectively. For codes with a larger number of groups than row and column weights such as the examples in

9x18 matrix structure shift values

1	34			32	39	23					53			1				
2		52						17	13				47		42	41		
3			19			44				45			1				52	18
4		48			56		26					17			51	42		
5	20			56			25		9		28		13					
6			41	26				56			52	21						37
7	33		16		31				46			4		2				
8		32					26			33					4		25	48
9						28		6		56				26		19	55	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Waiting times for columns when row processing starts at 1.

15	21	26	21	26	22	2	18	24	24	26	18	23	26	19	24	30	31
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Figure 6.25. Waiting times for a quasi-cyclic (1008,3,6) code of example in Figure 4.24.

Figure 6.2 and irregular codes, row groups do not have all of their sub-matrices aligned. For such codes overlapping starting rows and columns could be still be determined by starting processing at arbitrary points. This would not guarantee the shortest overlap delay. In this sub-subsection we show that the worst waiting time depends on row and column weights.

Figure 6.23 shows that the maximum distance that covers all starting points on a circumference of a cycle depends on the number of points. Since the distance between rows or columns in a code matrix is modulo p (sub-matrix size), it can be represented by a circle as in the figure. The figure shows a circumference of 12 which is equivalent to p. If there are only two points, part (a), the maximum distance between them is $\frac{1}{2}p$. Moving one of the points either way reduces the arc length on one side and increases it on the other side. The distance is now given by the smaller arc which is less than $\frac{1}{2}p$ or 6 in this case. Part (b) shows division of the circumference with three points. The three points are equidistance of 4 from each other. If one of them is moved the smaller arc covering all three points will be less than 8 or $\frac{2}{3}p$. Hence generally given a group size of p and j starting rows it takes a maximum distance (minimum arc length) of $(1 - \frac{1}{j})p$ to cover all starting rows. Similarly with k columns, a maximum distance of $(1 - \frac{1}{k})p$ is needed to cover all starting columns. The time to cover all starting points is the same as the waiting time in the quasi-cyclic code. When all points are covered we start to have a row or column processed by all the sub-matrices. This row or column is ready for processing

because it will have all its messages updated. That is, if j starting columns are chosen randomly in j sub-matrices of a column group, the maximum time (number of operations) it will take to have the one row processed in all of the j sub-matrices is given by $(1-\frac{1}{j})p$. Also with any k starting rows in k sub-matrices of the same group, the maximum time it will take to have one column processed by all sub-matrices is given by $(1-\frac{1}{k})p$. However, with overlapping all messages must be available. Thus, the covering distance must include the starting point. Hence the actual delay is $(1-\frac{1}{j})p+1$ or $(1-\frac{1}{k})p+1$.

Codes with small j and k do not have a large worst case shift or starting point differences. Codes would rarely have shifts dividing the group size in equal distances. We can generally expect the differences between shifts to be less than the maximum possible difference. The minimum difference between starting points is equivalent to the waiting time w on the overlapping scheduling scheme.

Since check nodes are processed first we can use the worst case of $(1-\frac{1}{j})p+1$, with random starting points. Random starting points could generally work if inter-iteration delays are reduced to minimum. Inter-iteration waiting times in quasi-cyclic codes can be eliminated or reduced to one by re-calculating starting addresses as was described in the method above. Figure 6.24 shows that the gain is close to two even if w is significantly large compared to size of sub-matrix.

Most LDPC codes have column weights of 2 to 4. Column weights of 2 to 4 have a worst waiting time of $\frac{1}{2}p$ to $\frac{3}{4}p$. These worst waiting periods have a gain of at least 1.9 above 20 iterations as shown in Figure 6.24 assuming inter-iteration delays are eliminated. Hence, we could generally start row processing at any position or at zero addresses and calculate the corresponding column addresses without loosing much decoding time gain. This applies to all regular and irregular quasi-cyclic codes created from shifted identity sub-matrices.

Figure 6.25 shows a matrix structure for a (1008,3,6) code used in Chapter 4 (Figure 4.12). The code was constructed by our proposed quasi-cyclic algorithm in chapter 4. It has a girth and average girth of 10. Figure 6.25 shows shift values of each sub-matrix. The shift values are the columns connected to the first row in each sub-matrix. If rows in each sub-matrix are processed from row 1 the waiting times for columns in each sub-matrix are as shown in the figure. Since the column weight is 3, the maximum waiting time is $\frac{2p}{3}$ which is 37 ($p=56$)in this case. In the example matrix code the maximum waiting time calculated is 31. Addresses in the following iterations are calculated as in [74] as described earlier.

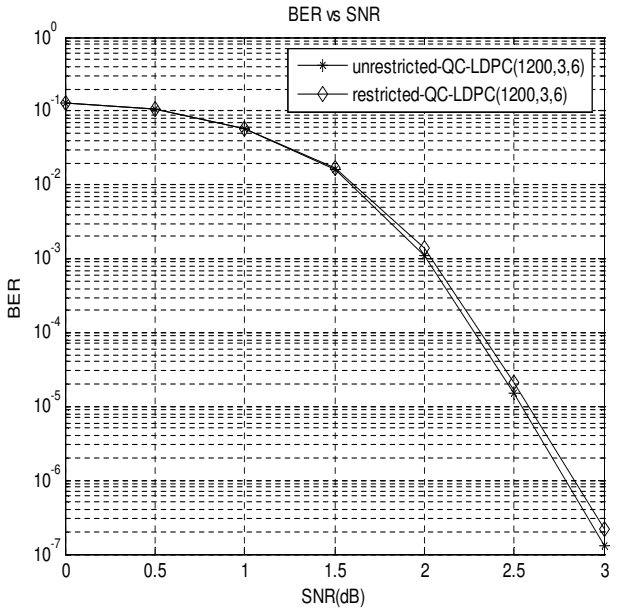

Figure 6.26. BER performance of code with constrained shifts compared to code with unconstrained shifts (25 iterations).

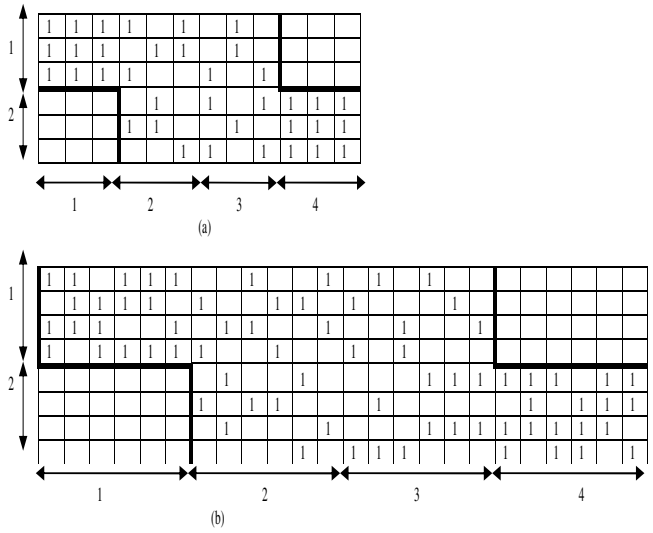

Figure 6.27. Matrix configuration with matrix space restriction.

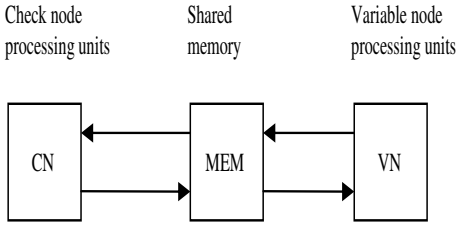

Figure 6.28. Overlapping Decoder architecture based on matrix permutation and space restriction.

Reducing Waiting Time by Restriction

As already stated waiting time depends on how much the difference between starting points vary from differences between sub-matrices shift values in the same column or row. For very high time critical applications, we can push the decoding time gain much closer to two by restricting the identity sub-matrix shift values.

Above, overlapping was achieved by restricting matrix space connections. Here, shift values and starting points are restricted in each sub-matrix. For example a quasi-cyclic code could be designed with $p = 100$ (sub-matrix size) and random shift values restricted to a maximum difference of 50. The processing starting row could then be calculated using a heuristic algorithm, or randomly selected. In both cases the worst waiting time will be less than or equal to 50 ($\frac{1}{2}p$). However, the matrix constraints have to be weighed against decoding performance losses if any. Restricting shift values could limit the size of girth that could be obtained thus limiting the decoding performane of the code.

Figure 6.26 shows BER performance of a shift restricted code compared to an unrestricted one. The restricted code shows a performance degradation of less than 0.1dB with 25 iterations. The codes are constructed from a $(N, 3, 6)$ structure with a sub-matrix size of 100. The restricted code has shifts restricted to the first 50 columns or rows. It has a girth of 8 and average girth of 8.6667. The unrestricted code has a girth of 10 and average girth of 10. The codes were constructed using our proposed quasi-cyclic algorithm which once again proves its flexibility in accommodating a variety of code designs.

Reducing Waiting Time by Matrix Sub-division

Waiting time could also be reduced by constraining the size of sub-matrices. For example instead of using sub-matrixes of 1000, the sub-matrix could be divided into four smaller sub-matrices of 250. The waiting time with smaller sub-matrices is smaller compared to the original sub-matrix size. Dividing the matrix into smaller sub-matrices may result in better decoding performance as was shown in chapter 4. However, sub-dividing the matrix would reduce the overall waiting time if the new sub-matrices are processed in parallel. If they are processed serially their waiting times accumulate.

This technique would work if there are many processing nodes per sub-matrix. That is, the architecture uses vector processing and sub-vectors are also possible. Even then, the overall reduction may depend on the structure of the code. Complexity with many

sub-matrices also needs to be weighed in. Matrix sub-division increases the number of sub-matrices to be managed and memory requirements. Each sub-matrix has shift values to be stored along with starting addresses.

6.4 Proposed Decoder Architecture

As was described at the beginning this chapter, there are several architectures for QC-LDPC codes decoders. However, mapping QC-LPDC codes to hardware is as varied as their construction. Reported decoder implementations are limited in flexibility and scalability. They are for example, often targeted for a single rate. If other rates are executed extra hardware is needed as in [120]. In [121] a multi-rate and size architecture is designed only for irregular codes.

We propose a low-complexity, programmable, high-throughput and scalable QC-LDPC decoder architecture based on matrix permutation and matrix space restriction overlapping techniques suggested in the previous section. The discussed overlapping techniques have implications on the design of the decoder architecture. Although overlapping by careful scheduling of rows and columns of sub-matrices can achieve a time reduction of 50%, it leads to a complex decoder architecture. With this technique, overlapping is achieved by processing all the sub-matrices in parallel. Therefore, the decoder should have as many processing units as sub-matrices of the code and also have the capability of mapping the processing units to the sub-matrices. In cases where there are many sub-matrices, the decoder must be flexible enough to distribute its resources to all sub-matrices. This complicates design and implementation of the decoder. Also, sub-matrices need to be stored in different memory banks so that their messages can be read in parallel. These characteristics require a complex design and will not cater for all matrix designs (any number of sub-matrices and configuration).

Although matrix permutation and matrix space restriction techniques are limited in overlapping, they require a relatively less complex architecture regardless of the code design (code length and weights). In these techniques, overlapping is achieved by processing sub-matrices with no connections in parallel. Overlapping here relies on outer relationships or dependencies of sub-matrices rather than their inner dependencies. To have

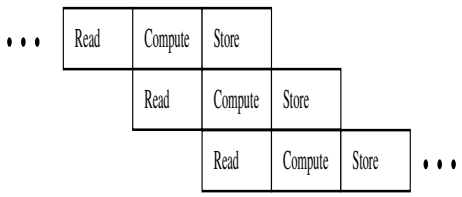

Figure 6.29. Pipelining of reading, processing and writing stages of decoding computations.

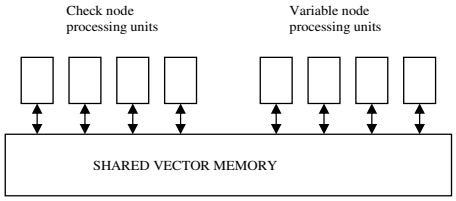

Figure 6.30. Overlapping Decoder architecture based on matrix permutation and space restriction.

overlapping, sub-matrices are scheduled and processed sequentially in both check and variable nodes. This property allows us to have more compact decoder design and flexibility in the code design. Using the example of matrix space restriction in Figure 6.7, we can design codes with this shape as in the examples of Figure 6.27. The example codes are scheduled the same as in as in Figure 6.7 producing a 33% reduction in decoding time. In Figure 6.27 (a), we have (3,6) column-row weights in a 6×12 basis matrix. Part (b) is a (3,9) matrix on an 8×24 structure. The codes could be mapped onto a simple architecture as that of Figure 6.28. Check and variable node processing units are connected by a shared memory. Processing on each size could be serial or in vector form. With this architecture, any overlapped code using matrix permutation or space restriction can be efficiently mapped and executed. Using large basis matrices or larger row-column weights does not affect the decoder structure. Processing is performed in a similar way with the same time reduction as with any other matrix fitting the restricted space. Overlapping is achieved when check and variable processing units read and process rows and columns with no connections in common in parallel. To further reduce decoding time, reading, processing and storage of rows and columns could be pipelined as shown in Figure 6.29. Reading, processing and storage stages are pipelined so that data is continually fed to processing units improving both throughput and hardware utilization.

Throughput of the proposed architecture can also be improved by increasing the number of processing elements in check and variable nodes. However, the number of processing

elements is limited by how larger the vector registers can be. This limitation can be overcome by implementing several processing units as shown in Figure 2.30. Four check and variable processing units are shown. The processing units have many processing elements and do processing in vector form. The units can read and write any vector in the shared memory bank. The overall architecture is similar to the Imagine Processor[122][123] designed for multimedia processing. The Imagine processor has eight ALU clusters connected to stream register file which provides a bandwidth of several GB/s. With this amount of memory bandwidth the decoder architecture can have high throughput and still have flexibility and scalability. Although this architecture is developed for quasi-cyclic LDPC codes, it can be used for random or any other structured codes. The major requirement is that matrices must be permuted or have connection restricted in order to have overlapping. To our knowledge this is most flexible LPDC decoder to date.

6.5 Summary

Critical implementation issues for LDPC decoders include processing node interconnection and decoding delay. In this chapter, multistage interconnection networks have been suggested as a possible means of communication between processing nodes. They are more flexible compared to hardwired or memory bank interconnections. Banyan networks have flexibility, however, they require the inputs to be sorted in ascending or descending order. The ordering of inputs may increase the number of transmission operations. Benes networks could be used to reduce the number of transmission operations as they do not require inputs to be ordered. To efficiently use multistage networks, vector processing must be used. One major drawback of the decoder with multi-stage networks is the separation of check and variable node memories which reduces opportunities for minimizing memory usage.

Decoding delay could be reduced by overlapping check and variable node computations. Overlapping could be achieved by matrix permutations, matrix space restrictions or careful calculation of starting rows and columns in quasi-cyclic codes. Matrix permutation can be applied to any matrix. However, the reduction in decoding time depends on the code structure and is limited.

Matrix space restriction is another technique that can be applied to any matrix structure. It gives a fixed reduction in time with decoding performance degradation. Reduction in

time depends on the shape or extent of restriction. In quasi-cyclic codes the reduction in decoding time depends on sub-matrix shift values and where processing was started. We derived the worst waiting time and showed that arbitrary starting points could be used assuming inter-iteration stalls are eliminated. The gain can be further improved by constraining the shift values differences or using small sub-matrices. These options have to be weighed against the complexity of the decoder and code performance. Although regular codes we used as examples in this paper, irregular codes could be overlapped in a similar way. We developed a decoder architecture based on developed computation overlapping techniques. The developed technique has high flexibility as its overlap is achieved by processing sub-matrices in serial. Pipelining of memory accesses and computation operations can further improve decoder throughput.

This page is blank

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The construction and hardware implementation of LDPC codes were the main subjects of this thesis. The objective was to construct good performing codes which are also easy to implement in hardware. We achieved some of our objectives by developing new LDPC code construction methods and ways of improving decoder efficiency.

Two main LDPC construction methods were proposed based on distance graphs and a search criteria. The first method based on distance graphs constructs column-weight two LDPC codes. Using a distance-graph representation a wide range of codes were obtained from already known distance graphs. Some codes show good error correcting performance compared to random codes whereas some codes perform poorly. The performance of the codes depends on the structure of the graph used. Although a wide range of codes in terms of rate, girths and size were obtained, the lengths of the codes are not flexible unless by the use of other expansion methods. Also, some graph interconnections are inconsistent with efficient decoder implementation.

The second method was proposed to construct quasi-cyclic LDPC codes. Although there are many proposed methods for doing this, they lack flexibility in design. The proposed algorithm is based on search algorithms such as bit-filling and progressive-edge growth to have flexibility in code construction. Rows and columns are divided into groups to form sub-matrices. Connections in groups are in consecutive order to obtain shifted identity sub-matrices in the parity check matrix. Although the algorithm guarantees only a girth of

six, it was successfully used to construct codes with higher girths. Given row and column weights, the size of sub-matrix and girth as inputs, the algorithm tries to find a code with the specified girth. The algorithm has a low complexity, which is linear with respect to the length of the code. From our experiments the algorithm does not take long to find most codes especially low girth and rate codes. Codes obtained from the proposed quasi-cyclic codes are flexible in rate, length and have good girth. These characteristics make them a good candidate for many applications. The disadvantage of the proposed algorithm compared to other methods is that it may not find a code, especially at high girths, even if they exist. It also does not detect whether the given dimensions can meet the girth or not.

In the second part of the thesis, we looked at some decoder implementations issues. A case was made for use of multistage interconnection networks as a means of communication between check and variable processing nodes. Banyan and Benes networks could be used depending on the design of the codes. Although these networks are flexible in executing any type of quasi-cyclic configuration, they add delays in communication.

Computation overlapping of check and variable processing was also looked at to reduce decoding delay. Matrix permutation, space restriction could be used with any matrix. We showed that matrix permutation is limited depending on row and column weights. The success of matrix space restriction depends on the shape and extent of restriction which has to be weighed against performance degradation. We also discussed overlapping in quasi-cyclic codes by careful scheduling of rows and columns. Reduction in decoding time could be reduced up to 50%. We showed that the worst overlap depends on column and row weights. We suggested improving overlapping by restricting the shifting of identity sub-matrices and matrix sub-division. However, shift-restriction will likely affect decoding performance and matrix sub-divisions further complicates the decoder architecture. Finally we proposed a decoder architecture for quasi-cyclic codes based on matrix permutation and matrix space restriction overlapping techniques. Since, these techniques process rows and columns serially, they lead to a simple decoder architecture. The decoder can also be used for random and other structured LDPC codes.

7.2 Future Work

Girth conditions given in the proposed algorithm for quasi-cyclic codes are not sufficient for girths above six for column weights higher than three. When the girth is above six there are cases where smaller cycles are formed when all row or column connections are made. Further work is needed to improve the success rate of finding codes with lager girths (larger than 8) without significantly increasing the computational complexity of the algorithm. Further analysis of the obtained codes is also needed in terms of minimum distances, bounds on girth, rate and error correcting performance at very low bit error rates. Some of of the obtained codes in Chapters 3 and 4 perform worse than random codes despite their better girths. Analyzing the obtained codes using other parameters such as minimum distance may reveal the reason for lower performances. This may lead to better ways of constructing structured codes with better error correcting performances. Our construction methods for quasi-cyclic codes in Chapter 4 does not determine bounds on girth, rate or code dimensions. Adopting algebraic methods such as those in [57] may give us bounds on these parameters to better maximize girth or rate for a given code size. BER simulations for obtained codes were performed up to 10^{-7} and for small codes (a few thousands). Most applications require very low BER performances and some use very large codes than those experimented with. Further BER or FER simulations at lower BERs will be necessary to evaluate the suitability of these codes for some applications.
Optimization of a basis matrix before applying the proposed algorithm may improve performance of obtained codes. Other decoding performance improving techniques such as density-evolution, reduction of stopping sets could be added to the algorithm. Obtained codes in this thesis were tested using a single channel model and modulation (AWGN and BPSK). These codes could be applied to other channels and modulations or other technologies.
A detailed study of hardware costs of proposed interconnect and overlapping methods is needed to have a better comparison between the proposed interconnect, architecture and existing decoder implementations. Implementing a prototype of the developed decoder architecture would give a better case to compare it to other existing architectures.

This page is blank

Bibliography

[1] C.E. Shannon. A Mathematical Theory of Communications. *Bell Systems Technology Journal*, pages 379–423,623–657, 1948.

[2] S.B. Wicker and S. Kim. *Fundamentals of Codes, Graphs and Iterative Decoding*. Kluver Intenational Series in Engineering and Computer Science, 2003.

[3] S. Lin and D. Costello. *Error-Control Coding: Fundamentals and Applications*. Prentice-Hall, 2nd edition edition, 2004.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes. *Proceedings of International Conference on Communications*, pages 1064–1070, May 1993.

[5] R. G. Gallager. Low-Density Parity-Check Codes. *IRE Transactions on Information Theory*, pages IT–8:21–28, January 1962.

[6] J. Pearl. *Probablistic reasoning in intelligent systems - Networks of plausable inference*. Morgan Kaufmann, 1988.

[7] D. MacKay and R. Neal. Near Shannon Limit Performance of Low-Density Parity-Check Codes. *Electronics Letters*, 32(18):1645–1646, August 1996.

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

[9] S. Chung, G.D. Forney, J.J. Richardson, and R. Urbanke. On the Design of Low-Density Parity-Check Codes within 0.0045dB of the Shannon Limit. *IEEE Communication Letters*, 5:58–60, February 2001.

[10] T. Richardson and R. Urbanke. Design of Provably Good Low-Density Parity-Check Codes. *IEEE Transaction on Information Theory*, 1999.

[11] Y. Wu, W. Ebel, and B. Woerner. Forward Computation of Backward path Metrics for MAP Decoder. *Proceedings of IEEE Vehicular and Technology Conference*, 3:2257–2261, May 2000.

[12] M. Eroz, F. Sun, and L. Lee. DVB-S2 Low-Density Parity-Check Codes with near Shannon Limit Performance. *International Journal of Satellite Communications and Networking*, 22:269–279, 2004.

[13] IEEE. IEEE Broadband Wireless Access Working Group. *http://ieee802.org/16*.

[14] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam. High Througput Low-Density Parity-Check Decoder Architectures. *Proceedings of GLOBECOM'2001,San Antonio,TX*, 5:3019–3024, 2001.

[15] T. Richardson and R. Urbanke. The Capacity of Low-Density Parity-Check Codes under Message-Passing Decoding. *IEEE Transactions on Information Theory*, 47:599–618, February 2001.

[16] H. Song, J. Liu, and B.V.K Vijaya Kumar. Low Complexity LDPC Codes for Partial Response Channels. *IEEE Globecom 2002*, 2:1294–1299, November 2002.

[17] S.B. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice Hall,New Jersey, 1995.

[18] W. Ryan. An Introduction to Low-Density Parity-Check Codes. *http://www.ece.arizona.edu/ryan/NewApr2001*.

[19] M.A. Shokrollahi. LDPC Codes: An Introduction. *http://www.imp.ac.ir/IMP/homepage/AMIN2.pdf*.

[20] D.J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge Press, 2003.

[21] R.M. Tanner. A Recursive Approach to Low Complexity Codes. *IEEE Transactions on Information Theory*, IT-27(5):533–547, Sep 1981.

[22] T. Richarson and N. Vladimir. Methods and Apparatus for Decoding LDPC Codes. *US Patent 6,633,856*, October 2003.

[23] Y.C. He, S.H. Sun, and X.M. Wang. Fast Decoding of LDPC Codes Using Quantization. *Electronics Letters*, 38(4):189–190, 2002.

[24] H. Fujita and K. Sakaniwa. Some Classes of Quasi-Cyclic LDPC Codes: Properties and Efficient Encoding Method. *IEICE Fundamentals*, E88-A(12):3627–3635, 2005.

[25] F.R. Kschischang, B.J. Frey, and H.A. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February 2001.

[26] B. Sklar. *Digital Communications: Fundamentals and Applications*. Prentice Hall Communications, 2nd edition, January 2001.

[27] A. Blanksby and C. Howland. A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder. *IEEE Journal of Solid-Sate Circuits*, 37(3):402–412, March 2002.

[28] D. MacKay and M.C. Davey. Evaluation of Gallager Codes for Short Block Length and High Rate Applications. *Proceedings of the IMA Workshop on Codes,Systems and Graphical Models*, 123:113–130, 1999.

[29] M. Luby, M. Mitzenmachen, M. Shokrollahi, and D. Spielman. Efficient Improved Low-Density Parity-Check Codes using Irregular Graphs. *IEEE Transactions on Information Theory*, 47:585–598, 2001.

[30] T. Richardson, M.A. Shokrollahi, and R. Urbanke. Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes. *IEEE Transactions on Information Theory*, 47(2):619–637, February 2001.

[31] X.Y. Hu, E. Elecftheriou, D.M. Arnold, and A. Dholakia. Efficient Implemetations of the Sum-Product Algorithm for Decoding LDPC Codes. *Proc. IEEE Globecom'01*, pages 1036–1036E, November 2001.

[32] E. Elefthriou, T. Mittelholzer, and A. Dholakia. Reduced-complexity Decoding Algorithm for Low-Density Parity-Check Codes. *IEE Electronics Letters*, 37:102–104, January 2001.

[33] Li Ping and W.K. Leung. Decoding Low Density Parity Check Codes with Finite Quantization Bits. *IEEE Communications Letters*, 4(2):62–64, 2000.

[34] A. Vardy. The Intractability of Computing the Minimum Distance of a Code. *IEEE Transactions of Information Theory*, 43:1757–1766, November 1997.

[35] University of Sydney. MAGMA Algebraic Software. *Department of Mathematics, http://www.magma.maths.usyd.edu.au/magma*.

[36] M. O'Sullivan. Algebraic Construction of Sparse Matrices with Large Girth. *IEEE Transactions on Information Theory*, 52(2):719–727, February 2006.

[37] Y. Mao and A.H. Banihashemi. A Heusristic Search for Good Low-Density Parity-Check Codes at short Block Lengths. *Proceedings of IEEE International Conference on Communications*, 1:41–44, June 2001.

[38] G. Richter. An Improvement of the PEG Algorithm for LDPC Codes in the Waterfall Region. *EUROCON,Belgrade*, 2:1044–1047, November 22-24 2005.

[39] H. Song, J. Liu, and B. Kumar. Large Girth Cycle Codes for Partial Response Channels. *IEEE Transactions on Magnetics*, 40(4):2362–2371, July 2004.

[40] H. Song and B. Kumar. Low-Density Parity-Check Codes for Partial Response Channels. *IEEE Signal Processing Magazine*, pages 56–66, January 2004.

[41] R. Lynch, E.M. Kurtas, A. Kuznetsov, E. Yeo, and B. Nikolic. The Search for a Practical Iterative Dectector for Magnetic Recording. *IEEE Transactions on Magnetics*, 40(1):213–218, January 2004.

[42] IEEE. IEEE 802.3 10GBase-T Study Group Meeting. *World Wide Web, http://www.ieee802.org/3/10GBT/public/jul04/rao-1-0704.pdf*, 2004.

[43] B. Vasic, I.B. Djordjevic, and R.K. Kostuk. Low-Density Parity Check Codes and Iterative Decoding for Long-Haul Optical Communication Systems. *Journal of Lightwave Technology*, 21(2):438–446, February 2003.

[44] I.B. Djordjevic and B. Vasic. High Code Rate Low-Density Parity Check Codes for Optical communication systems. *IEEE Photonoics Technology Letters*, 16(6):1600–1602, June 2004.

[45] J. Campello, D. S. Dodha, and S. Rajagopalan. Designing LDPC codes using Bit-Filling. *in Proceedings of the International Conference on Communications*, 1:55–59, June 2001.

[46] X.Y. Hu, E. Eleftheriou, and D.M. Arnold. Progressive edge-growth Tanner Graphs. *Proc. IEEE GLOBECOM,San Antonio,TX*, 2:995–1001, November 2001.

[47] D. MacKay. Good Error-Correcting Codes Based on Very Sparse Matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, March 1999.

[48] D. MacKay. Encyclopedia of Sparse Graph Codes. Database of Codes. *Department of Physics, http://www.inference.phy.cam.ac.uk/mackay/codes/data.html*.

[49] J. Campello, D. S. Dodha, and S. Rajagopalan. Extended Bit-Filling and LDPC code Design. *in Proceedings of the IEEE GLOBECOM Conference,San Antonio,TX*, 2:985–989, November 2001.

[50] D. MacKay, S.T. Wilson, and M.C. Davey. Comparison of Constructions of Gallager codes. *IEEE Transactions on Communications*, 47:1449–1454, October 1999.

[51] X.Y. Hu. Progressive edge-growth Tanner Graphs. *PhD Thesis, The University Sweden*, November 2001.

[52] H. Xiao and A.H. Banihashemi. Improved Progressive Edge-Growth (PEG) Construction of Irregular LDPC Codes. *IEEE Communications Letters*, 8(2):715–717, November 2004.

[53] S.J. Johnson and S.R. Weller. Construction of Low-Density Parity-Check codes from Kirkman triple Systems. *IEEE Globecom 2001,San Antonio,TX*, 2:970–974, Nov 2001.

[54] Jing Li and Erozan Kurtas. A Class of High-Rate, Low-Complexity, Well-Structured LDPC Codes from Combinatorial Designs and their Applications on ISI Channels. *IEEE Transactions on Communications*, 52(7):1038–1042, July 2004.

[55] Y. Kou, S. Li, and M.P.C Fossorier. Low-density Parity-Check Codes:Construction based on Finite Geometries. *IEEE Globecom 2000,San Fransisco,CA*, 2(7):825–829, Nov 2000.

[56] Y. Kou, S. Li, and M.P.C Fossorier. Low-Density Parity-Check Codes based on Finite Geometries: A Rediscovery and New Results. *IEEE Transactions Information Theory*, 47(7):2711–2736, Nov 2001.

[57] M. P.C Fossorier. Quasi-Cyclic Low-Density Parity-Check Codes From Circulant Permutation Matrices. *IEEE Transactions on Inforamtion Theory*, 50(8):1788–1793, August 2004.

[58] R.M Tanner, D. Sridhara, and T. Fuja. A Class of Group-structured LDPC Codes. *Proceedings of ISTA,Ambleside, England*, 2001.

[59] M. Karkooti and J. Cavallaro. Semi-parallel Reconfigurable Architectures for Real-time LDPC Decoding. *International Conference on Information Technology(ITCC)*, 1:579–585, April 2004.

[60] J. Kowk-San Lee, B. Lee, J. Hamkins, J. Thorpe, K. Andrews, and S. Dolinar. A Scalable Architecture for Structured LDPC Decoder. *IEEE Internatinal Symposium on Information Theory*, pages 292–299, July 2004.

[61] J. Lu, J.M.F Moura, and U. Niesen. Grouping-and-shifting Designs for Structured LDPC Codes with Large Girth. *IEEE Internatinal Symposium on Information Theory*, pages 236–236, July 2004.

[62] J.M.F. Moura, J. Lu, and Z. Zhang. Structured Low-Density Parity-Check Codes. *IEEE Signal Processing Magazine*, 21(1):42–55, January 2004.

[63] W. Imrich. Explicit Construction of Graphs without Small Cycles. *Combinatorica*, 2:53–59, 1984.

[64] F. Lazebnik and Ustimenko. Explicit Construction of Graphs with an Arbitrary Large Girth and of Large Size. *Descrete Applied Mathematics*, pages 275–284, June.

[65] N.L. Biggs. Cubic Graphs with Large Girth. *Combinatorial Mathematics:Proceedings of the Third International Conference*, pages 56–62, 1989.

[66] G. Exoo. A Simple Method for Constructing Small Cubic Graphs of Girths 14,15 and 16. *Electronic Journal of Combinatorics*, 3, 1996.

[67] P.K Wong. Cages– A Survey. *Journal of Graph Theory*, 3:1–22, 1982.

[68] M. Meringer. Fast Generation of Regular Graphs and Construction of Cages. *Journal of Graph Theory*, 30:137–146, 1999.

[69] M. Meringer. Genreg-download and Manual. http://www.mathe2.uni-bayreuth.de/markus/manual/genreg.html.

[70] G. Royle. Cages of Higher Valency. http://www.cs.uwa.edu.au/ gordon/cages/allcages.html.

[71] N.L. Biggs. Constructions for Cubic Graphs of Large Girth. *Electronic Journal of Combinatorics*, 5, 1998.

[72] E.W Weisstein. Cage Graph. From MathWorld–A Wolfram Web Resource. http://www.mathworld.wolfram.com/CageGraph.html.

[73] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong. Efficient Encoding of Quasi-Cyclic Low-Density Parity-Check Codes. *IEEE Transactions on Communications*, 54(1):71–81, January 2006.

[74] Y. Chen and K.K. Parhi. Overlapped Message Passing for Quasi-Cyclic Low-Density Parity-Check Codes. *IEEE Transactions on Circuits and Systems*, 51(6):1106–1113, June 2004.

[75] L. Chen, J. Xu, I. Djurdjevic, and S. Lin. Near Shannon Limit Qausi-Cyclic Low-Density Parity-Check Codes. *IEEE Transactions on Communications*, 52(7):1038–1042, July 2004.

[76] S.J. Johnson and S.R. Weller. Regular Low-Density Parity-Check Codes from Combinatorial Designs. *Information Theory Workshop,Cairns, Australia*, pages 90–92, September 2001.

[77] B. Vasic. Structured Iteratively Decodable Codes on Steiner Systems and their Applications in Magnetic Recording. *IEEE Globecom*, 5:2954–2960, Nov 2001.

[78] H.Tang, J. Xu, Y. Kou, S. Lin, and K.A.S. Abdel-Ghaffar. On Algebraic Construction of Gallager and Circurlant Low-Density Parity-Check Codes. *IEEE Transactions on Information Theory*, 50(6):1269–1279, June 2004.

[79] M.P.C. Fossorier. Systematic Recursive Construction of LDPC Codes. *IEEE Communications Letters*, 8(5):302–304, May 2004.

[80] G.Malema and M. Liebelt. Low Complexity LDPC codes for Magnetic Recordings. *Enformatika: International Conference on Signal Processing*, 5:269–271, August 2005.

[81] R. Bresnan. Novel Code Construction and Decoding Techniques for LDPC Codes. *M.Eng. Sc. Thesis, Department of Electrical and Electronic Engineering, University College Cork, Ireland*, 2004.

[82] M. Sullivan, J. Brevik, and R. Wolski. The Performance of LDPC Codes with Large Girth. *Proceedings 43rd Annual Allerton Conference; Communication, Control and Computing*, September 2005.

[83] K. Andrews, S. Dolinar, D. Divsalar, and J. Thorpe. Design of Low-Density Parity-Check Codes for Deep Space Applications. *IPN Progress Report*, pages 42–159, November 2004.

[84] J. Thorpe, K. Andrews, and S. Dolinar. Methodologies for Designing LPDC Codes Using Photographs and Circulants. *IEEE Internatinal Symposium on Information Theory*, pages 238–242, July 2004.

[85] S.J. Johnson and S.R. Weller. A Family of Irregular LDPC Codes With Low Encoding Complexity. *IEEE Communications Letters*, 7(2):79–81, February 2003.

[86] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam. VLSI Architectures for Iterative Decoders in Magnetic Recording Channels. *IEEE Transactions Magnetics*, 37(2):748–755, March 2001.

[87] S. Sivakumar. VLSI Implementation of Encoder and Decoder for Low Density Parity Check Codes. *Masters Thesis, Texas AM University*, December 2001.

[88] Y. Chen and D. Hocevar. A FPGA and ASIC Implementation of Rate 1/2 8088-b Irregular Low-Density Parity-Check Decoder. *IEEE Global Telecommunications Conference,GLOBECOM*, 1:113–117, December 2003.

[89] E. Yeo, B. Nikolic, and V. Anantharam. Iterative Decoder Architectures. *IEEE Communication Magazine*, pages 132–140, August 2003.

[90] H. Zhong and T. Zhang. Design of VLSI Implementation-Oriented LDPC Codes. *IEEE Semiannual Vehicular Technology Conference (VTC)*, 1:670–673, October 2003.

[91] J. Zhao, F. Zarkeshvari, and A.H. BanihaShemi. On the implementation of Min-Sum Algorithm and its modifications for Decoding Low-Density Parity-Check Codes. *IEEE Transactions on Communications*, 53(4):549–554, April 2005.

[92] C. Jones, E. Valles, M. Smith, and J. Villasenor. Approximate-Min constraint Node updating for LDPC Code Design. *IEEE Conference on Military Communication, MILCOM 2003*, pages 157–162, October 2003.

[93] M.M. Mansour and N.R. Shanbhag. Memory Efficient Turbo Decoder Architecture for LDPC Codes. *IEEE Workshop on Signal Processing Systems, (ISPS'2002)*, pages 159–164, October 2002.

[94] H. Sankar and K.R. Narayanan. Memory-Efficient Sum-Product Decoding of LDPC Codes. *IEEE Transactions on Communications*, 52(8):1225–1230, August 2004.

[95] E.A. Choi, J. Jung, N. Kim, and D. Oh. Complexity-Reduced Algorithms for LDPC Decoder for DVB-S2 Sytems. *ETRI Journal*, 27(5):639–642, October 2005.

[96] D.E Hocevar. A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes. *IEEE Workshop on Signal Processing Systems*, pages 107–112, 2004.

[97] A. de Baynast, P. Radosavljevic, J. Cavallaro, and A. Sabharwal. Turbo-Schedules for LDPC Decoding. *43th Annual Allerton Conference on Communication, Control and Computing*, 2005.

[98] E. Sharon, S. Litsyn, and J. Goldberger. An Efficient Message-Passing Schedule for LDPC Decoding. *23rd IEEE convention of Electrical and Electronic Engineers in Israel*, pages 223–226, September 2004.

[99] I. Park and S. Kang. Scheduling Algorithm for Partially Parallel Architecture of LDPC Decoder by Matrix Permutation. *IEEE International Symposium on Circuits and Systems*, pages 5778–5781, May 2005.

[100] G. Al-Rawi, J. Cioffi, R. Motwani, and M. Horowitz. Optimizing Iterative Decoding of Low-Density Parity-Check Codes on Programmable Pipelined Parallel Architectures. *Proceedings of the IEEE Information Technology: Coding and Computing, Las Vegas,NV,USA*, pages 578–586, April 2001.

[101] T. Zhang, Z. Wang, and K. Parhi. On Finite Precision Implementation of Low-Density Parity-Check Codes Decoder. *Proc. of IEEE International Symposium on Ccircuits and Systems*, 4:202–205, May 2001.

[102] T. Richardson and R. Urbanke. Efficient Encoding of Low-Density Parity-Check Codes. *IEEE Transactions on Information Theory*, 47(2):638–656, February 2001.

[103] S. Myung, K. Yang, and J. Kim. Quasi-Cyclic LDPC Codes for Fast Encoding. *IEEE Transactions on Information Theory*, 51(8):2894–2901, August 2005.

[104] K. Andrews, S. Dolinar, and J. Thorpe. Encoders for Block-Circulant LDPC Codes. *IEEE Internatinal Symposium on Information Theory*, pages 2300–2304, September 2005.

[105] E. Kim and G.S. Choi. LDPC Code for Reduced Routing Decoder. *Asia Pacific Communicaions Conference, Perth Australia*, pages 991–994, october 2005.

[106] M. Mohiyuddin, A. Aziz A. Prakash, and W. Wolf. Synthesizing Interconnect-efficient Low-Density Parity-Check Codes. *Design Automation Conference*, pages 488–491, June 2004.

[107] A. Darabiha, A. Carusone, and F. Kschischang. Multi-Gbit/sec Low-Density Parity-Check Decoders with Reduced Interconnect Complexity. *International Symposium on Circuits and Systems*, 5:5194–5197, May 2005.

[108] E. Cuthill and J. MacKee. Reducing the Bandwidth of Sparse Symmetric Martices. *Proceedings of the National Conference of ACM*, pages 157–172, 1969.

[109] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall,Inc,Englewood Cliffs,NJ, 1981.

[110] N. Gibbs, W. Poole, and P. Stockmeyer. An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix. *SIAM Journal of Numerical Analysis*, 13:236–250, 1976.

[111] S. Olcer. Decoding Architecture for Array-code-based LPDC Codes. *Proc. IEEE GLOBECOM*, pages 2046–2050, December 2003.

[112] Flarion Technolgies Inc. Vector-LDPC Decoder. *http://www.flarion.com*.

[113] D Konig. Uber Graphen und ihre Anwendung auf Determinantetheorie und Mengenlhere. *Math.Ann.77*, pages 453–465, 1916.

[114] W.J. Dally and B. Towles. *Principles and Practices of Interconenction Networks*. Morgan Kaufmann, 2003.

[115] R.Y. Awdeh and H.T. Mouftah. Survey of ATM Swicth Architectures. *Computing, Networks ISDN Sys.*, 27, 1995.

[116] S.F. Oktug and M.U. Caglayan. A Survery of Banyan Networks and Banyan-network-based cell Switches. *Bogazici University, Instanbul, Turkey*, Research paper,1996.

[117] D.C Opferman and N.T Tsao-Wu. On a Class of Rearangeable Switching Networks,Part I: Control. *Bell Systems Technology Journal*, 5:1579–1600, May-June 1971.

[118] V.E. Benes. Optimal Rearrangeable Multistage Connecting Netowrks. *Bell System Technical Journal*, 43:1641–1656, 1964.

[119] Y. Chen and K.K. Parhi. High Throughput Overlapped Message Passing for Low-Density Parity-Check Codes. *Great Lakes Symposium on VLSI,Washington DC*, pages 245–248, April 2003.

[120] L. Yang, H. Liu, and C.J.R. Shi. Code Construction and FPGA Implementation of a Low-Error-Floor Multi-Rate Low-Density Parity-Check Code Decoder. *IEEE Transactions on Circuits and Systems-I*, 53(4):892–904, April 2006.

[121] P. Radosavljevic, M. Karkooti, A. de Baynast, and J.R. Cavallaro. Tradeoff Analysis and Architecture Design of High Througput Irregular LDPC Decoders. *IEEE Transactions of Circuits and Systems-I*, 1(1):1–15, November 2006.

[122] S. Rixner, B. Khailany, W. Dally, U. Kapasi, P. Mattson, J. Namkoong, J. Owens, B. Towels, and A. Chang. Imagine: Media Processing with Streams. *IEEE Micro*, 21:25–45, March-April 2001.

[123] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany. The Imagine Processor. *IEEE International Conference on Computer Design: VLSI in Computer and Processors(ICCD'02)*, pages 282–290, 2002.