PERFORMANCE OF PSEUDO-RANDOM AND QUASI-CYCLIC LOW DENSITY PARITY CHECK CODES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ONUR HÜSNÜ KAZANCI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2007

Approval of the thesis:

PERFORMANCE OF PSEUDO-RANDOM AND QUASI-CYCLIC LOW DENSITY PARITY CHECK CODES

submitted by ONUR HÜSNÜ KAZANCI in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering Dept.** _____

Assoc. Prof. Dr. Melek Diker Yücel
Supervisor, **Electrical and Electronics Engineering Dept.** _____

Examining Committee Members:

Prof. Dr. Yalçın Tanık
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Melek Diker Yücel
Electrical and Electronics Engineering Dept., METU _____

Assis. Prof. Dr. Arzu Tuncay Koç
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Elif Uysal Bıyıkoğlu
Electrical and Electronics Engineering Dept., METU _____

Semih Can
MST-TMM, ASELSAN Inc. _____

Date: December 7, 2007

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name and Surname : Onur Hüsnü KAZANCI

Signature :

ABSTRACT

PERFORMANCE OF PSEUDO-RANDOM AND QUASI-CYCLIC LOW DENSITY PARITY CHECK CODES

Kazancı, Onur Hüsnü
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Diker Yücel, Melek

December 2007, 74 pages

Low Density Parity Check (LDPC) codes are the parity check codes of long block length, whose parity check matrices have relatively few non-zero entries. To improve the performance at relatively short block lengths, LDPC codes are constructed by either pseudo-random or quasi-cyclic methods instead of random construction methods. In this thesis, pseudo-random code construction methods, the effects of closed loops and the graph connectivity on the performance of pseudo-random LDPC codes are investigated. Moreover, quasi-cyclic LDPC codes, which have encoding and storage advantages over pseudo-random LDPC codes, their construction methods and performances are reviewed. Finally, performance comparison between pseudo-random and quasi-cyclic LDPC codes is given for both regular and irregular cases.

Keywords: Low Density Parity Check codes, LDPC, pseudo-random LDPC codes, quasi-cyclic LDPC codes, Girth, EMD, ACE, Stopping Set.

ÖZ

RASTGELEMSİ VE YARI-ÇEVRİMSEL DÜŞÜK YOĞUNLUKLU EŞLİK SAĞLAMASI KODLARININ BAŞARIMI

Onur Hüsnü Kazancı
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Danışmanı: Doç. Dr. Melek Diker Yücel

Aralık 2007, 74 sayfa

Düşük Yoğunluklu Eşlik Sağlaması (DYES) kodları, uzun kod boylu ve eşlik sağlaması matrisinde sıfır dışındaki sayıların, sıfırlara göre çok az olduğu eşlik sağlaması kodlarıdır. Göreli olarak kısa kod uzunluklarında DYES kodlarının başarımını artırmak için, rastgele oluşumlar yerine rastgelemsi veya yarı-çevrimsel yöntemler kullanılmaktadır. Bu tezde, rastgelemsi DYES kodlarını oluşturma yöntemleri, kapalı döngülerin ve grafik bağlanırlığın kod başarımına etkileri incelenmiştir. Ayrıca, rastgelemsi kodlara göre kodlama ve bellek avantajları olan yarı-çevrimsel DYES kodu oluşturma yöntemleri ve başarımları irdelenerek, rastgelemsi DYES kodlarınınkiyle hem düzenli hem de düzensiz durumlar için karşılaştırılmıştır.

Anahtar Sözcükler: Düşük Yoğunluklu Eşlik Sağlaması, DYES, rastgelemsi, yarı-çevrimsel.

To My Family

ACKNOWLEDGEMENTS

I would like to thank all my teachers, especially Assoc. Prof. Dr. Melek D. Yücel for her motivating ideas and valuable guidance.

I would also like to thank my parents for encouraging and supporting me during this thesis work and my whole education.

Finally, I would like to thank my elder brother Oğuz for his support and guidance during my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACE	Approximate Cycle EMD
ACEC	ACE Check
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Ratio
BPSK	Binary Phase-Shift Keying
BSC	Binary Symmetric Channel
CCSDS	Consultative Committee for Space Data Systems
DE	Density Evolution
DVB-S2	Second Generation Satellite Digital Video Broadcasting
EMD	Extrinsic Message Degree
FEC	Forward Error Correcting
FER	Frame Error Ratio
LGDC	Local Girth Distribution Check
HDTV	High-Definition TV
JA	Joint Approach
LDPC	Low Density Parity Check
LLR	Log Likelihood Ratios
MFN	Multi-Frequency Network
PDF	Probability Distribution Functions
SDTV	Standard-Definition Television
SFN	Single-Frequency Network
SNR	Signal to Noise Ration
SSC	Stopping Set Check
VLSI	Very Large Scale Integration
WiFi	Wireless LAN
WiMax	Worldwide Interoperability for Microwave Access

CHAPTER 1

INTRODUCTION

As Shannon indicated in 1949 [Shannon-1949], the fundamental problem of communication is to reproduce a message signal, that is generated at one point, either exactly or approximately at another point. All communication channels are noisy and noise is the main cause of errors during data transmission. An analog telephone line, a radio communication link or a data recording disc drive are some examples of noisy channels. In all these cases, if a data string is transmitted over a channel, there is some probability that the received message will not be identical to the transmitted message. It is preferred to have a communication channel for which this probability of error is so close to zero that for practical purposes it is indistinguishable from zero. The solution for this problem, other than the physical solutions like increasing the reliability of the circuitry or increasing the transmission power, is channel coding. Coding theory is concerned with the creation of practical and successful encoding and decoding methods by adding an encoder and a decoder before and after the channel.

The encoder designs suitable sets of distinct codewords, and the decoder devises methods for extracting estimates of transmitted messages from the output of a noise-contaminated channel, which is a process called error correction. *Low Density Parity Check* (LDPC) codes discussed in this thesis are special examples of error correcting codes which are a class of linear block codes. Linear block codes map k-symbol messages to n-symbol codewords, all of which satisfy the $(n-k)\times 1$ matrix equation $Hc^T=0$, where H is the $(n-k)\times n$ parity check matrix and c

is the $1\times n$ transmitted codeword. Binary LDPC codes are the codes specified by a parity check matrix containing many zeros and very small number of ones, whose positions are generally chosen at random. Binary LDPC codes can be divided into two main groups as regular and irregular LDPC codes, where a regular (n, w_c, w_r) LDPC matrix is an $(n-k)\times n$ parity check matrix having exactly w_c ones in each column (column weight) and exactly w_r ones in each row (row weight), such that $w_c<w_r$ and both are very small compared to n. On the other hand, an irregular LDPC matrix is still sparse, but not all rows and columns contain the same number of ones; instead, they are defined by the weight density functions which give the probabilities of the column and row weights.

Although LDPC codes perform very well for long block length such as 10^7, the design of good codes with relatively shorter block lengths is also desired for many practical applications. The construction becomes prominent for LDPC codes of short to medium lengths because a short block length LDPC code with a randomly generated parity check matrix may typically have a poor performance.

The Tanner graph of an LDPC code [Tanner-1981] is the visualization of its parity check matrix, which defines the columns and rows as variable nodes and check nodes and the non-zero entries as the edges that connect these nodes. The performance of an LDPC code depends on both the structure of its Tanner Graph (or graph only) and its minimum distance. In other words, an LDPC code with a good minimum distance may not outperform another one with worse minimum distance but better graph structure, because the commonly used iterative message-passing decoding algorithms are suboptimum and graph dependent [Kschischang-2001]. Therefore, most of the research on *pseudo-random* LDPC code design has been concentrated on finding graphs suitable for iterative decoding by optimizing the parameters such as the shortest closed path in the graph, namely the *girth*, and *local girth distribution* that is the distribution of the shortest length closed path generated by each column [Mao-2001], or other parameters dealing with the connectivity in the graph such as the *extrinsic message degree* (EMD) [Tian-2003]

and *stopping set* (SS) [Richardson-2002]. The name pseudo-random comes from the control of these constraints during the random construction.

Most methods for designing LDPC codes are based on random construction techniques and the lack of structure implied by this randomness presents serious disadvantages in terms of storing and accessing a large parity check matrix, encoding data, and analyzing code performance (e.g., determining the distance properties). If the codes are designed with some algebraic structure, then some of these problems can be overcome. In the recent literature, several algebraic methods for constructing LDPC codes have appeared [Moura-2005], [Fossorier-2000a-2001-2004], [Tanner 1999-2000-2004-2007], [Rosenthal-2000], [Honary-2005], [Fan-2000]. Among these, the most relevant ones to our work are the *quasi-cyclic* (QC) LDPC codes designed by Tanner in 2004 and design considerations explained by Moura in 2005. The quasi-cyclic LDPC codes have sparse and elegant graph representations that make them well suited to iterative message-passing algorithms. The parity check matrix of a quasi-cyclic LDPC code is composed of blocks of circulant matrices (zero matrices, identity matrices and circularly shifted identity matrices), giving the code a quasi-cyclic property, which can potentially facilitate efficient encoder implementation. Further, the algebraic structure of the code allows an efficient VLSI implementation with simple shift registers.

1.1 History

LDPC codes (also called Gallager codes) were first proposed in 1962 [Gallager-1962] [Gallager-1963], along with an elegant iterative decoding scheme whose complexity grows only linearly with the block length of the code. Despite their advantages, LDPC codes were largely forgotten for several years primarily because the computers at the time were not powerful enough to decode them. In 1995, LDPC codes were rediscovered by MacKay and Neal [MacKay & Neal-1996], who proved that in spite of their simple construction, these codes have very

impressive performance; that is, when optimally decoded, some of them achieve information rates very close to the Shannon limit [Shannon-1949].

Today the value of LDPC codes is widely recognized and their remarkable performance ensures that they will not be forgotten again. In contrast to many codes that were invented well after 1962, LDPC codes offer both better performance and lower decoding complexity. In fact, it is an irregular LDPC code (with block length 10^7) that currently holds the distinction of being the world's best code of rate 1⁄2, outperforming all other known codes, and falling only 0.0045 dB short of the Shannon limit [Richardson-2001a].

Because of their success in approaching the channel capacity most closely, LDPC codes will definitely become the choice for next-generation communications standards. LDPC codes have already been included in the China National Standard for Digital Terrestrial TV Broadcasting standard, the European standard for the second generation of Satellite Digital Video Broadcasting (DVB-S2), satellite broadcasted, high-definition TV (HDTV) in 2004. They are currently considered for the revision of many recommendations issued by the Consultative Committee for Space Data Systems (CCSDS), and they will most probably be included in the channel coding section of many other widespread telecommunication applications, like wired and wireless digital communication networks.

The China National Standard for Digital Terrestrial TV Broadcasting standard [Zhang-2006] supports the baseband data payload from 4.813Mbit/s to 32.486Mbit/s, standard-definition television (SDTV) and high-definition television (HDTV), fixed point and mobile reception, multi-frequency network (MFN) and single-frequency network (SFN). In the system, the LDPC code is adopted as the inner code and the BCH (762, 752) is the outer code. There are 3 LDPC code modes and the numbers of information bits are 3048, 4572 and 6096, respectively. The LDPC codeword length n is the same for 3 coding modes, which is 7488 bits. There are 3 FEC coding rates, namely 0.4, 0.6, and 0.8, respectively.

- Code rate k/n = 0.4, FEC (7488, 3008);
- Code rate k/n = 0.6, FEC (7488, 4512);
- Code rate k/n = 0.8, FEC (7488, 6016).

The HDTV satellite standard, known as the DVB-S2 digital video broadcasting transmission system employs an LDPC coding technique as a channel coding scheme. The DVB-S2 satellite video broadcasting standard was designed for an exceptional error performance at very low SNR ranges (up to Frame Error Ratio (FER)$\leq 10^{-7}$ at -2.35 dB E_s/N_0). Thus the specified LDPC codes use a large block length of 64800 bits with 11 different code rates ranging from 1/4 to 9/10. This results in large storage requirements for up to 285000 messages and demands high code rate flexibility at the same time to support all specified node degrees, as shown in the first row of Table 1.1 [Brack-2007]. The performance of the LDPC code with codeword length n=64800, and information word length k=32400=$(n-k)$ is given in Figure 1.1 [Choi-2005].

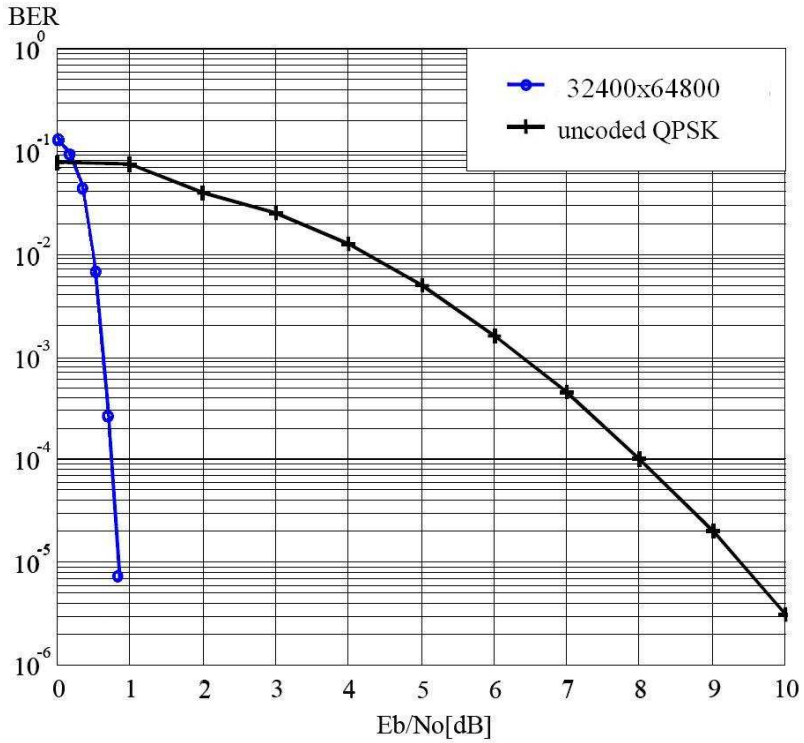


Figure 1.1 Performance of the LDPC code used in DVB-S2 Standard

The current WiMax 802.16e standard features the LDPC codes as an optional channel coding scheme. It consists of six different code classes with different row weight and column weight distributions, spanning four different code rates from 1⁄2 to 5⁄6 (see the second row of Table 1.1). All six code classes have the same general parity check matrix structure and support 19 codeword sizes, ranging from n=576 to 2304 bits [Brack-2007].

The WiFi 802.11n standard also features the LDPC codes as an optional channel coding scheme. It utilizes 12 different codes with four code rates from 1/2 to 5⁄6 for each of the three different codeword sizes of 648, 1248, and 1944 bits. The most complicated issue with this code is the row weight and column weight flexibility needed to fully support this standard (see the third row of Table 1.1) [Brack-2007].

Table 1.1 LDPC Code parameters for some Next Generation Standards

	Codeword size, n			Code Rate, k/n			Row weight, w_r			Column weight, w_c			Number of ones in H
	#	min	max	#	min	max	#	min	max	#	min	max	
DVB-S2	1	64800	64800	11	1/4	9/10	11	4	30	7	2	13	285120
WiMax 802.16e	19	576	2304	4	1/2	5/6	7	6	20	4	2	6	8448
WiFi 802.11n	3	648	1944	4	1/2	5/6	9	7	22	8	2	12	7128

(# denotes the number of different n, k/n, w_r or w_c values in respective columns.)

1.2 Aim and Outline of the Thesis

The aim of this thesis is to explore pseudo-random and quasi-cyclic LDPC code construction methods among the proposed generation schemes. The topologies in the graph that affect the performance of pseudo-random LDPC codes at short block lengths are explained in detail and the performance comparison of regular and irregular pseudo-random LDPC codes is made. Besides, quasi-cyclic LDPC code construction methods and their girth properties are investigated. Finally, the thesis work is focused on the performance comparison of pseudo-random and quasi-cyclic LDPC codes for the regular and irregular cases using the comparably sized parity check matrices that we construct for that purpose.

Chapter 2 presents a brief overview of the terminology and concepts used in this thesis. The review of LDPC code properties, their graph representations, and two main iterative decoding algorithms are also given in this chapter. Chapter 3 describes the pseudo-random LDPC codes, which are constructed specifically to minimize the occurrence of topologies that cause problems during decoding. These topologies lower the performance by generating closed loops in the graph, such as *short length cycles* and an unfavorable *local girth distribution*. The effects of these topologies on the performance of the pseudo-random codes are discussed at the end of this chapter.

Besides all the advantages, pseudo-random LDPC codes have disadvantages like the encoding complexity and the memory problem since a large amount of information is required to locate the non-zero elements in huge parity check matrices. On the other hand, quasi-cyclic LDPC codes reduce the encoding complexity and memory problem since they have an algebraic construction method which uses the circulant matrices. Quasi-cyclic LDPC codes, their construction methods, cycle properties and performances of the regular and irregular LDPC codes that we have generated in this thesis work are given in Chapter 4. The conclusions in Chapter 5 include the performance comparison of all these codes and discussion of related future work.

CHAPTER 2

LOW DENSITY PARITY CHECK CODES

In this chapter, review of the concepts and terminology related to LDPC codes will be given. After the description of LDPC codes and related parameters in Section 2.1, the Tanner graph and cycle properties of block codes are explained briefly in Section 2.2. The concept of irregular codes and the irregular codes proposed by MacKay are discussed in Section 2.3. Finally, Section 2.4 is a review of the iterative message passing decoding for LDPC codes based on hard decision and soft decision algorithms.

2.1 LDPC Codes

Low Density Parity Check (LDPC) codes discussed in this thesis are special examples of parity check codes which are a class of linear error correcting block codes. An (n, k) linear block code maps 2^k messages to 2^n codewords c, all of which satisfy $cH^T=0$, where H is the parity check matrix of size $(n-k)\times n$. As the name suggests, Low Density Parity Check codes are a sub-class of parity check codes, whose parity check matrix has relatively few non-zero entries and the non-zero entry positions are generally chosen at random. According to the column/row weight distributions, LDPC codes can be divided into two main groups as regular and irregular LDPC codes. A regular (n, w_c, w_r) LDPC matrix is an $(n-k)\times n$ binary parity check matrix having exactly w_c ones in each column and exactly w_r ones in each row, where $w_c<w_r$ and both are very small compared to n.

An irregular LDPC matrix is still sparse, but not all rows and columns contain the same number of ones. In other words, irregular LDPC codes have parity check matrices whose weight per column and/or row is not uniform, but instead governed by an appropriately chosen distribution of weights. Irregular LDPC codes are parameterized by column and row degree distribution polynomials $\lambda(x) = \sum \lambda_i x^{i-1}$ and $\rho(x) = \sum \rho_i x^{i-1}$, where the coefficients λ_i and ρ_i respectively specify the fraction of edges of degree i (column or row weight i), such that $\sum \lambda_i = 1$ and $\sum \rho_i = 1$. Richardson and Urbanke showed by carefully choosing the distributions that performance improvement can be achieved over regular LDPC codes. They presented irregular LDPC codes that perform extremely close (0.0045 dB) to the best possible bound determined by the Shannon capacity formula [Richardson-2001a], for instance their ½ rate code of length 10^7 approaches Shannon bound by 0.0045 dB.

In contrast to the irregular codes, every parity check equation of a regular LDPC code involves exactly w_r bits, and each of these bits is involved in exactly w_c parity check equations. The restriction $w_c < w_r$ is needed to ensure that more than just the all-zero codeword satisfies all of the constraints, or equivalently, to ensure a nonzero code rate. Indeed, the total number of ones in the $m \times n$ parity check matrix H is $mw_r = nw_c$, since there are $m = (n-k)$ rows, each containing w_r ones, and there are n columns, each containing w_c ones.

The code rate R of an (n, k) block code is $R = k / n = 1-((n-k) / n)$, hence $R = 1 - (w_c / w_r)$. The parameters n, w_c, and w_r cannot be chosen independently, but must be related for regular LDPC codes in such a way that $n (w_c / w_r)$ is an integer. For example, a $(w_c, w_r) = (3, 4)$ LDPC matrix exists when $n = 1000$ and $n = 1004$, but not when $n = 1002$. Observe that the fraction of ones in a regular (w_c, w_r) LDPC matrix is w_r/n. The *low density* terminology derives from the fact that this fraction approaches zero as n goes to infinity. In contrast, the average fraction of ones in a purely random binary matrix, with independent components equally likely to be zero or one, is $1/2$.

One measure of the ability of a code to detect errors is its *minimum Hamming distance*. The minimum Hamming distance of a code may be viewed as a convenient measure of how *good* it is, but in fact it is not possible to distinguish between good and very good codes by their minimum distance, without reference to their Tanner graph properties.

2.2 Graph Structure

Any parity check code (including an LDPC code) may be specified by a Tanner graph [Tanner-1981], which is essentially a visual representation of the parity check matrix H. An $m \times n$ parity check matrix H defines a code in which the n bits of each codeword satisfy a set of m parity check constraints. The Tanner graph contains n *variable nodes* (*bit nodes*), one for each codeword bit; and m *check nodes* (*constraint nodes*), one for each of the parity checks. The variable nodes are depicted using circles, while the check nodes are depicted using squares. The check nodes are connected to the variable nodes they check. Specifically, a branch connects the check node i to variable node j if and only if the i^{th} parity check involves the j^{th} bit, or briefly, if and only if the $(i, j)^{\text{th}}$ element H_{ij} of the parity check matrix nonzero. The graph is said to be bipartite because there are two distinct types of nodes, variable nodes and check nodes, and there can be no direct connection between any two nodes of the same type.

Example 1

A 3×6 parity check matrix H and its associated Tanner graph with the 3×6 LDPC matrix are shown in Figure 2.1. The variable nodes are represented by the $n = 6$ circles at the top, while the check nodes are represented by the $m = 3$ squares at the bottom.

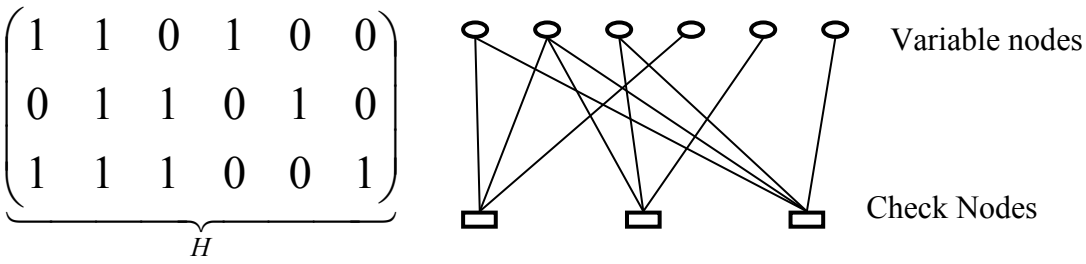


Figure 2.1 Tanner graph representation of a 3x6 LDPC matrix

Degree of a node is the number of branches connected to it. Degrees of 1st to 6^{th} variable nodes in Figure 2.1 respectively are 2, 3, 2, 1, 1 and 1, which are also the respective column weights; and the degrees of 1^{st} to 3^{rd} check nodes or corresponding row weights are 3, 3 and 4, respectively.

In Figure 2.1, degree distribution polynomial for the variable nodes is $\lambda(x) = 3/10 + (4/10)x + (3/10)x^2$ since 3 of the 10 edges have degree 1, 4 of the 10 edges have degree 2 and 3 of the 10 edges have degree 3. Degree distribution polynomial for the check nodes is $\rho(x) = (6/10)x^2 + (4/10)x^3$ by similar reasoning.

A *cycle* is defined as the path through the graph that begins and ends at the same variable node. The length of the cycle is the number of edges traversed. The minimum length of a cycle is four in a bipartite graph. A 4-cycle can also be described as *more than one overlapping* 1*'s between two columns*. An example for a cycle of length four which can be observed both in matrix representation and graph representation is given in Figure 2.2.

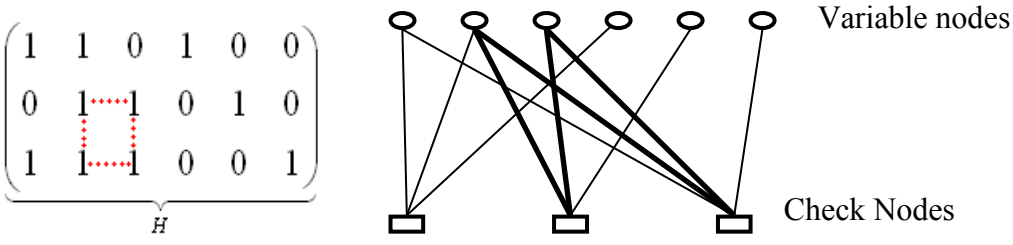


Figure 2.2 Representation of a cycle in matrix and graph representations

Girth is the length of the shortest cycle in the parity check matrix. Similarly, *local girth* of a variable node (column) can be defined as the length of the smallest cycle that is generated by that variable node. Short length cycles degrade the performance of the currently used iterative decoding algorithms, which converge to maximum likelihood decoding only code girth gets large.

2.3 Irregular LDPC Codes

Irregular LDPC codes may have better performances than regular LDPC codes and this can be explained by understanding the main idea behind irregular graphs. Let us first consider a regular low density parity check matrix. From the point of view of a variable node, it is the best to have high degree, since the more information it gets from its check nodes the more accurately it can judge what its correct value should be. In contrast, from the point of view of a check node, it is the best to have low degree, since the lower the degree of a check node, more valuable the information it can transmit back to its neighboring variable nodes. These two rival requirements must be appropriately balanced. Previous work has shown for regular graphs, that the low degree graphs yield the best performance [MacKay & Neal-1996], [MacKay-1999]. On the other hand, irregular graphs have significantly more flexibility in balancing these competing requirements. Message nodes (i.e., variable nodes) with high degree tend to correct their value quickly. These nodes then provide good information to the check nodes, which subsequently provide better information to lower degree message nodes. Irregular graph constructions thus have the potential to lead to a wave effect, where high degree message nodes tend to get corrected first, and then message nodes with slightly smaller degree, and so on. For a Gaussian channel, Luby has shown that, the best performance is achieved if the check node degree is constant (or all check nodes have degrees as close to the same degree as possible) [Luby-2001].

Richardson and Urbanke [Richardson-2001b] and Luby [Luby-2001] defined ensembles of irregular LDPC codes parameterized by the degree distribution

polynomials $\lambda(x)$ and $\rho(x)$, and showed how to optimize these polynomials for a variety of channels. Optimality is in the sense that, assuming message-passing decoding, a typical code in the ensemble is capable of reliable communication in worse channel conditions than the codes outside the ensemble. The worst-case channel condition is called the decoding threshold and the optimization of $\lambda(x)$ and $\rho(x)$ is found by a combination of a density evolution algorithm and an optimization algorithm. Density evolution refers to the evolution of the probability density functions of the various quantities passed around the Tanner graph of the code. The decoding threshold for a given $\lambda(x)$ - $\rho(x)$ pair is determined by evaluation of the probability distribution functions of computed log likelihood ratios (LLR) of the code bits. The separate optimization algorithm optimizes over the $\lambda(x)$ - $\rho(x)$ pairs.

The irregular codes used in this thesis are "93" irregular codes defined by MacKay in [MacKay-1998], which have rows of uniform weight 7, i.e., $\rho(x) = x^6$, and column weight distribution $\lambda(x)=(11/14)x^2+(3/14)x^8$ (3/14 of the edges have column weight 9 and 11/14 of the edges have column weight 3). The name "93" comes from the column weights of 9 and 3; so we prefer to use the notation ("93", 7) for these codes which also shows the uniform row weight 7. The variable nodes that have 9 connections to 9 check nodes are called *elite variable nodes*. In this thesis, 2 different constructions "93a" and "93y" are used, which have the same column/row weight distribution, but different elite variable node connections. The details and structures are as follows:

- **Irregular ("93a", 7) LDPC Code:**

This construction allocates exactly one or two elite variable nodes to each of the check nodes. The structure of the parity check matrix of ("93a", 7) irregular LDPC code is given in Figure 2.3, where integer values inside each circle give the column and row weight of each sub-matrix and empty places are zero matrices. Observing the row weight distribution in the last columns, which correspond to the elite

variable nodes, one can follow that half of the check nodes are connected to two elite variable nodes and the remaining half are connected to a single elite variable node. An example of a ("93a", 7) irregular LDPC matrix generated by our Local Girth Distribution Check algorithm is shown in Figure 2.4 by indicating nonzero elements of the parity check matrix with dots.

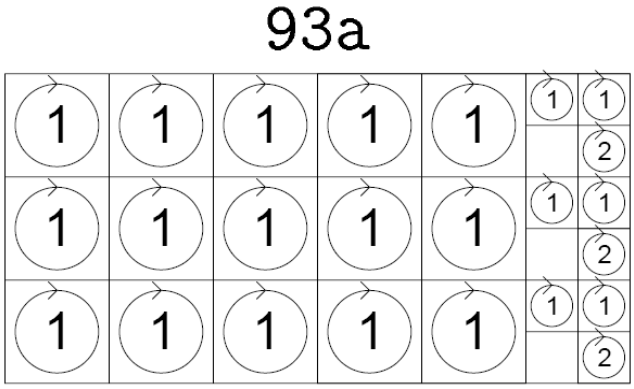


Figure 2.3 Parity check matrix construction of a ("93a", 7) irregular LDPC code where integer values inside each circle give the column and row weight of each sub-matrix and empty places are zero matrices

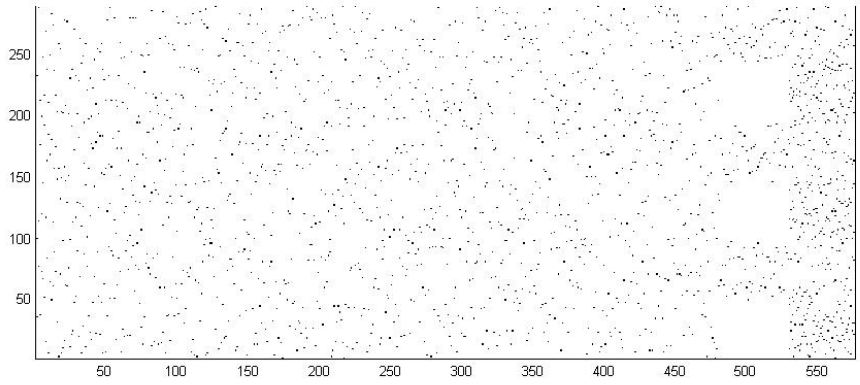


Figure 2.4 Example of a 288×576 parity check matrix for an irregular ("93a", 7) LDPC code

- **Irregular ("93y", 7) LDPC Code:**

In this irregular construction, one third of the check nodes are connected to four elite variable nodes, one third are connected to one and the remaining one third are connected to none of the elite variable nodes. The structure of the ("93y", 7)

irregular LDPC code is given in Figure 2.5 using the same visual notation, where integers in circles indicate the row and column weight of the corresponding sub-matrix. An example of a ("93y", 7) LDPC matrix is shown in Figure 2.6, in which dots show the one's of the parity check matrix.

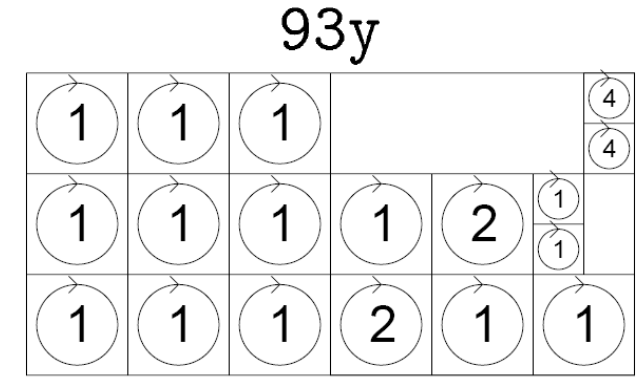


Figure 2.5 Parity check matrix construction of a ("93y", 7) irregular LDPC code where integer values inside each circle give the column and row weight of each sub-matrix and empty places are zero matrices

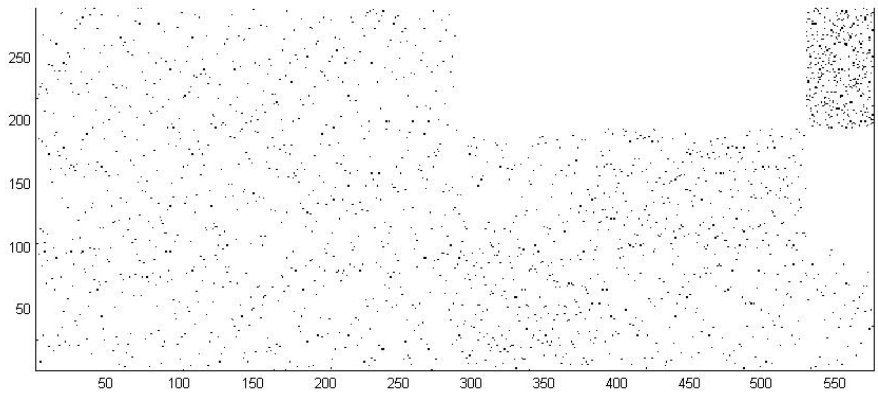


Figure 2.6 Example of a 288×576 parity check matrix for an irregular ("93y", 7) LDPC code

2.4 Decoding by Iterative Message Passing

One of the most attractive features of LDPC codes is that they allow for efficient iterative decoding. There are several algorithms known for iterative decoding of

LDPC codes; Gallager's *bit flipping algorithm*, *the belief propagation decoder*, *the min-sum decoder* etc. Most of these decoding techniques can be described as message passing algorithms, because they operate by exchanging messages.

The decoding operation can be conveniently described on the Tanner graph of a code. In a message passing algorithm, messages are exchanged between nodes along the edges of the Tanner graph and nodes process the incoming messages received via their adjacent edges to determine the outgoing messages. A message along an edge represents the estimate of the bit represented by the variable node associated with the edge. The message can be in the form of a hard decision, i.e., 0 or 1; or a probability vector $[p_0\ p_1]$, where p_i is the probability of the bit taking the value *i,* or in the form of a log likelihood ratio (LLR) $log(p_0/p_1)$ etc. An iteration of message passing consists of a cycle of information passing and processing. The outgoing message from the node is calculated based on the incoming messages along the other edges. The exact fashion in which the outgoing message is calculated depends on the message passing algorithm being used.

The computations in a message passing decoder are localized. In other words, computations at a check node are performed independent of the overall structure of the code. This implies that highly parallel implementations of a message passing decoder are feasible. Further, the computations are distributed, such that, computations are performed by all nodes in the graph. The number of messages exchanged in an iteration of message passing is dependent on the number of edges in the Tanner graph; a sparser Tanner graph means less computations. Thus, it is computationally feasible to use message passing algorithms for decoding long block length LDPC codes.

2.4.1 Bit Flipping Algorithm

The bit flipping algorithm is based on hard decisions. A variable node sends a message to each of the check nodes that it is connected, declaring if it is 1 or 0, and

each check node sends a message to each of the variable nodes to which it is connected, declaring whether the parity check is satisfied or not. The algorithm is as follows [Gallager-1962]:

Step 1 - Initialization: The maximum number of iterations is set to I_{max} and the iteration number is initiated as 1. Each variable node from the received word sends messages to the related check nodes indicating its value.

Step 2 - Parity Update: Using the messages coming from variable nodes, check nodes control if the parity check equations are satisfied or not. If all the equations are satisfied then the algorithm terminates with success. Else, check nodes send messages to variable nodes.

Step 3 - Variable Node Update: The iteration number is increased by one. If I_{max} is reached, the algorithm terminates with failure. Otherwise, the variable nodes which get the largest number of the messages from check nodes as "not satisfied" flip their values. The remaining variable nodes keep their values. Each variable node sends its value to the related check nodes and the algorithm turns back to *Step* 2.

Example 2: The decoding example of the bit-flip algorithm is described in Figure 2.7 where the parity check matrix, sent codeword and received word are given below.

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \; sent\,vector: 00000000 \; received\,vector: 00100000$$

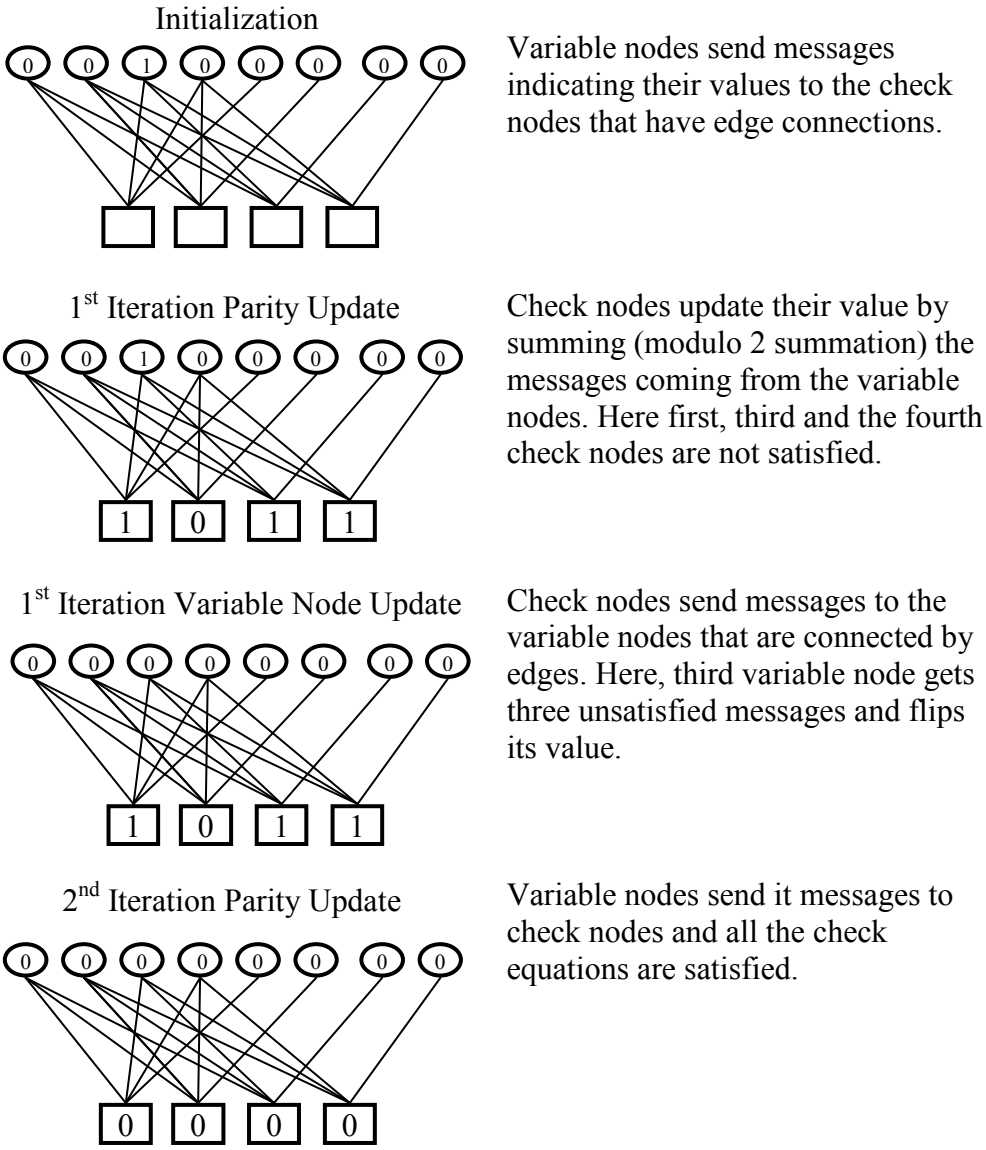


Figure 2.7 Message flow during the error correction of bit-flipping algorithm

2.4.2 Belief Propagation Algorithm

The most widely used message passing algorithm is the belief propagation which is also called the sum-product algorithm. BP decoding corresponds to the probabilistic solution to iterative decoding based on "cycle free Tanner graph"

assumption. Although the algorithm is not optimum due to the inevitable presence of loops in the graphs of all practical parity check sets, BP provides efficient decoding if these loops remain quite long. The processing at the variable and check nodes for a BP algorithm with log likelihood ratios (LLRs) as messages is now described. The main idea is the same with the bit-flip algorithm except the use of probabilistic decision instead of hard decision.

The aim of the belief propagation algorithm is to compute *a posteriori probabilities* (APPs) for each codeword bit, $P_i(1)=P[c_i=1|N]$ and $P_i(0)=P[c=0|N]$ which are the probabilities that the i^{th} bit of the codeword $c = [c_1\ c_2...c_i...c_n]$ is a 1 or 0 conditional on the event N that all parity check constraints are satisfied. The *intrinsic* or *a priori probability*, P_i^{int}, is the original likelihood ratio independent of the knowledge of the code constraints and the *extrinsic probability*, P_i^{ext} represents what has been learnt from each iteration. The extrinsic information obtained from the check nodes in one iteration is used as the priori information for the consequent iteration. The extrinsic bit information obtained from a parity check constraint would be independent of the original a priori probability if there were no cycles.

BP algorithm iteratively computes an approximation of the APP values for each code bit. To compute the extrinsic probability of the i^{th} bit of the received word that comes from the j^{th} parity check equation, we use the following properties. If bit i is assumed to be a 0, the j^{th} parity check equation is satisfied only when an even number of the other received word bits are 1.

Hence, the probability ($P_{i,j}(0)$) that the j^{th} parity check equation is satisfied when bit i equals to 0, is given by [Gallager-1963]

$$P_{i,j}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2p_{i'}), \qquad (2.1)$$

where B_j represents the set of column locations of the bits in the j^{th} parity check equation of the code and p_i is the probability that the i^{th} bit of the received word is equal to 1. Similarly, if bit i is assumed to be a 1 then the j^{th} parity check equation is satisfied when an odd number of the other received word bits are 1. So, when bit i equals to 1, the probability that the j^{th} parity check equation is satisfied is

$$P_{i,j}(1) = \frac{1}{2} - \frac{1}{2}\prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'}). \tag{2.2}$$

Details of (2.1) and (2.2) can be found in the main references [Gallager-1963]. Then, the extrinsic likelihood ratio is found dividing (2.1) by (2.2)

$$P_{i,j}^{ext} = \frac{P_{i,j}(0)}{P_{i,j}(1)} = \frac{\frac{1}{2} + \frac{1}{2}\prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'})}{\frac{1}{2} - \frac{1}{2}\prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'})} = \frac{1 + \prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'})}{1 - \prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'})}. \tag{2.3}$$

One can then use the following identity (2.4) to calculate the extrinsic message from check node j to variable node i.

$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \Rightarrow$$

$$\tanh\left(\frac{1}{2}\ln\left(\frac{1-p}{p}\right)\right) = \frac{e^{\frac{1}{2}\ln\left(\frac{1-p}{p}\right)} - e^{-\frac{1}{2}\ln\left(\frac{1-p}{p}\right)}}{e^{\frac{1}{2}\ln\left(\frac{1-p}{p}\right)} + e^{-\frac{1}{2}\ln\left(\frac{1-p}{p}\right)}} = \frac{\sqrt{\frac{1-p}{p}} - \sqrt{\frac{p}{1-p}}}{\sqrt{\frac{1-p}{p}} + \sqrt{\frac{p}{1-p}}} = 1 - 2p \tag{2.4}$$

Taking the natural logarithm of (2.3), substituting (2.4) for all probabilities (1-2p_i) and then using the definition of the log likelihood ratio,

$$LLR(P_i^{\text{int}}) = \ln\left(\frac{P\left\{i^{th}\, bit\, is\, 0\right\}}{P\left\{i^{th}\, bit\, is\, 1\right\}}\right) = \ln\left(\frac{1-p_i}{p_i}\right) \tag{2.5}$$

One obtains,

$$LLR(P_{i,j}{}^{ext}) = \ln\left(\frac{P_{i,j}(0)}{P_{i,j}(1)}\right) = \ln\left(\frac{1+\prod_{i'\in B_j, i'\neq i}\tanh\left(LLR(P_{i'}{}^{\text{int}})/2\right)}{1-\prod_{i'\in B_j, i'\neq i}\tanh\left(LLR(P_{i'}{}^{\text{int}})/2\right)}\right) \tag{2.6}$$

The estimated LLR of the i^{th} bit at each iteration is then found by combining all information coming from the related check nodes,

$$LLR(P_i) = LLR(P_i^{\text{int}}) + \sum_{j\in A_i} LLR(P_{i,j}{}^{ext}) \tag{2.7}$$

where A_i is the set of row locations of the parity check equations which check on the i^{th} bit of the code.

Belief propagation algorithm [Fossorier-2000b] can be now summarized as follows:

Step 1 - Initialization: The maximum number of iterations is set to I_{max}. The iteration number is initiated as 1. The initial message R_i sent from variable node i to the check node j is the log likelihood ratio defined by (2.5),

$$R_i = LLR(P_i^{\text{int}}) = \ln\left(\frac{1-p_i}{p_i}\right) \tag{2.8}$$

where p_i is the probability that the i^{th} received bit is 1. In (2.10), $L_{i,j}$ denotes the variable node message sent from variable node i to check node j, and R_{i} denotes the

variable node message sent from variable node i to all check nodes that are connected to the variable i. Calling the algorithmic message sent from variable node i to check node j, $L_{i,j}$, the initial value of $L_{i,j}$ is set to $R_i=LLR(P_i^{int})$.

Step 2 - Check-to-variable: The extrinsic message $E_{i,j}$ from the check node j to variable node i is the LLR computed in (2.6), by substituting $L_{i',j}=LLR(P_{i'}^{int})$ for all $i' \in B_j$, where B_j includes the indices of variable nodes connected to the check node j. Hence,

$$E_{i,j} = LLR(P_{i,j}^{ext}) = \ln\left(\frac{1+\prod_{i'\in B_j, i'\neq i}\tanh(L_{i',j}/2)}{1-\prod_{i'\in B_j, i'\neq i}\tanh(L_{i',j}/2)}\right) \tag{2.9}$$

Step 3 - Codeword Test: The combined log likelihood ratio L_i for each variable node i is the sum of the extrinsic messages $E_{i,j}$'s found by (2.9) and the initial message R_i defined by (2.9),

$$L_i = \sum_{j\in A_i} E_{i,j} + R_i \tag{2.10}$$

where A_i denotes the set of all check nodes connected to the i^{th} variable node. Then, for each bit, a hard decision is made:

$$z_i = \begin{cases} 1, L_i \leq 0 \\ 0, L_i > 0 \end{cases} \tag{2.11}$$

If $z = [z_1, ..., z_n]$ is a valid codeword, i.e., if $zH^T = 0$, the algorithm terminates with success.

Step 4 - Variable-to-check: The iteration number is increased by one. If I_{max} is reached, the algorithm terminates with failure. Otherwise the variable nodes send messages to the check nodes. The message $L_{i',j}$ sent by each variable node i to the check nodes j to which it is connected is similar to equation (2.10) in *Step 3*, except that bit i sends to check node j an LLR calculated without using the information from check node j:

$$L_{i,j} = \sum_{j' \in A_i, j' \neq j} E_{i,j'} + R_i \qquad (2.12)$$

Then the algorithm returns to *Step 2*.

In the following, we describe the parameters to be used in the algorithm given above, for BPSK modulation over an AWGN channel with noise variance σ^2. The i^{th} codeword bit is transmitted as ±1, so the channel input is x_i±1. Corresponding channel output is $y_i = x_i + n_i$, where n_i is the noise term. Assuming 1's are transmitted as -1 and 0's are sent as 1 with equal probabilities, the probability p_i in equation (2.8) is calculated as

$$P\{i^{th} \text{ bit is } 0\} = p_i = P(x_i = -1 \mid y_i) = \frac{1}{1 + e^{\frac{2y_i}{\sigma^2}}}, \qquad (2.13)$$

and similarly,

$$P\{i^{th} \text{ bit is } 1\} = 1 - p_i = P(x_i = 1 \mid y_i) = \frac{1}{1 + e^{\frac{-2y_i}{\sigma^2}}}. \qquad (2.14)$$

Substituting (2.13) and (2.14) into (2.8), R_i is found as,

$$R_i = \ln\left(\frac{\frac{1}{1+e^{\frac{-2y_i}{\sigma^2}}}}{\frac{1}{1+e^{\frac{2y_i}{\sigma^2}}}}\right) = \frac{2y_i}{\sigma^2} \tag{2.15}$$

Since the code length of the codes generated in this thesis work is kept almost constant (usually 576, rarely 564 and 900), the maximum number of iterations of the belief propagation decoder is set to 100 as explained in Appendix A.

CHAPTER 3

PSEUDO-RANDOM LDPC CODES

In this chapter, the pseudo-random LDPC codes, which are constructed specifically to minimize the occurrence of topologies that cause problems during decoding are described. Such topologies lower the performance by generating closed loops in the graph such as *short length cycles* and undesirable *local girth distribution*. In Section 3.1, the definitions of *girth*, *local girth distribution* (LGD), *extrinsic message degree* (EMD) and *stopping set* are given under the title of "Performance Improving Criteria for Pseudo-Random LDPC Codes". Details of the three LDPC code construction algorithms implemented in this study, namely, Local Girth Distribution Check (LGDC), ACE Check (ACEC) and Stopping Set Check (SSC) Joint Approach (JA) algorithms are given in Section 3.2. All these algorithms generate LDPC codes with desired nonuniform or uniform column weight distributions but uniform row weight distribution.

In Section 3.3, we compare the performances of the pseudo-random LDPC codes generated by our code construction algorithms, which use different criteria. Section 3.3.1 is devoted to the experimental investigation of the local girth distribution (LGD) as a criterion in the code construction, whereas Section 3.3.2 deals with the criteria of approximate cycle EMD (i.e., ACE) and stopping sets (SS). Hence regular and irregular codes all having parity check matrices of size 288×576 are

generated by LGDC algorithm in Section 3.3.1 and ACEC or joint ACEC/SSC algorithms in Section 3.3.2 for performance comparison.

3.1 Performance Improving Criteria for Pseudo-Random LDPC Codes

The random ensemble consists of codes that are defined only by the block length *n*. With the requirement that the codes be sparse, a low density parity check matrix *H* may be populated by arbitrarily low-weight column vectors. This means that one should expect quite poor performance at short block lengths, since the Tanner graph will most probably contain some undesired topologies. In this section, definitions of the four criteria that affect the performance of the pseudo-random LDPC codes are given; namely, *Girth, Local Girth Distribution* (LGD), *Extrinsic Message Degree* (EMD) and *Stopping Set* (SS).

3.1.1 Girth and Local Girth Distribution

The Tanner graph of a short LDPC code most probably contains quite a few cycles which are short with respect to the average number of iterations required for decoding. As a result, it is observed that short LDPC codes significantly deviate from the predicted performance. It is also known that for short LDPC codes, the performance often varies significantly over the ensemble, especially at high *signal-to-noise ratios* (SNR). An efficient method to search for good LDPC codes in a given ensemble is to search for codes without short cycles, since message-passing algorithms work well if the graph does not contain too many short cycles. Motivated by this, Mao described [Mao-2001] the *girth* as the smallest cycle generated by the parity check matrix of the code, and the *local girth distribution* as the distribution of the length of cycles generated by each column of the parity check matrix. In his work, he randomly generated many LDPC matrices and chose the one that has the local girth distribution with the greatest mean.

The local girth value of each column can be found by summing the parity check matrix columns since cycles can be expressed using column sums. Two columns can generate a 4-cycle if and only if their sum has two or more 2's. In the example given below for m=8, the parity check columns C_1 and C_2 generate a 4-cycle.

$$C_1^T = 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1$$

$$C_2^T = 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1$$

$$(C_1 + C_2)^T = 1 \quad 2 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 2$$

Similarly three columns can generate a cycle of minimum length six if the sum of these columns has three 2's. So, for the parity check columns C_1 and C_2 which do not generate any 4-cycle, if a column C_3 is added to the set, provided that C_3 has no 4-cycles with C_1 or C_2, a cycle of length six is generated whenever there are three 2's in the sum vector, as shown below.

$$C_1^T = 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1$$

$$C_2^T = 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0$$

$$C_3^T = 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1$$

$$(C_1 + C_2 + C_3)^T = 2 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 2$$

Using induction, generation of a cycle can be summarized as:

- A cycle of length $2t$ can be generated by at least t columns.
- If a column set (variable node set) consisting of t columns generate a cycle of length $2t$ (and not less), the sum vector has t many 2's. If the sum vector has values greater than 2, then corresponding columns generate a smaller cycle. This situation can be overcome by eliminating 4-cycles before 6-cycles, etc., i.e., initially discarding the smallest cycles.

3.1.2 Extrinsic Message Degree (EMD) and Approximate Cycle EMD (ACE)

For irregular LDPC codes, not only the length and the distribution of cycles but also their connectivity play important roles in the code performance [Tian-2003]. The graph connectivity of a cycle is defined as the number of connections of the cycle with the rest of the graph. Figure 3.1 shows examples of an 8-cycle with poor graph connectivity and a 4-cycle with good graph connectivity. Short cycles with good graph connectivity as shown in Figure 3.1 (b) are less harmful than long cycles with poor graph connectivity. Poorly connected subgraphs are more vulnerable to channel noise, since they do not have sufficient message flow from the rest of the graph to correct the errors. A high degree variable node has high probability to generate short cycles but it is not much harmful since it has many connections to the rest of the graph.

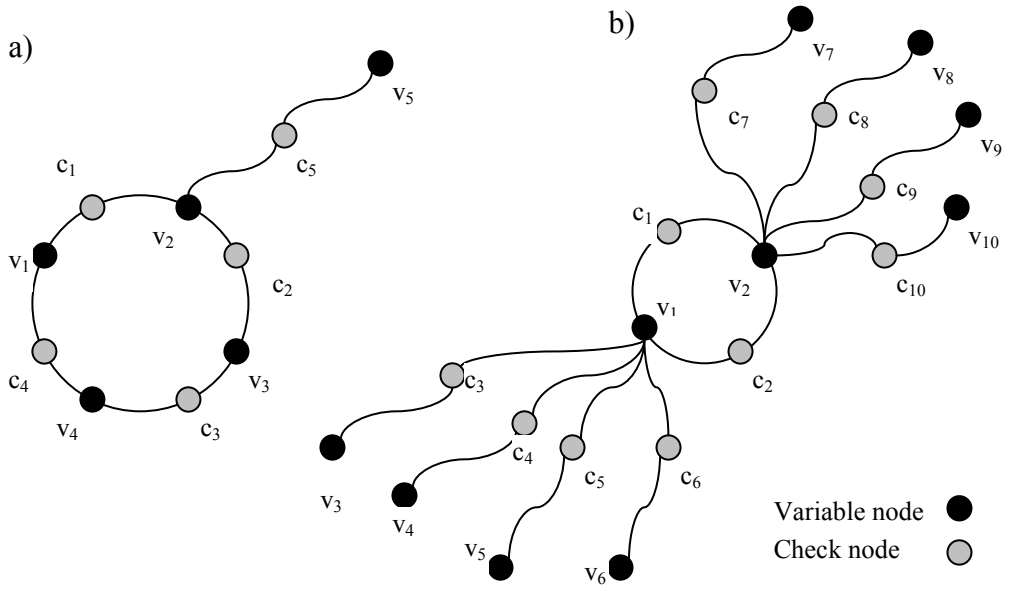


Figure 3.1 8-cycle with poor graph connectivity and 4-cycle with good graph connectivity

Extrinsic information is defined as the information that is collected exclusively from other parts of the system. The system is more capable of repairing errors by

attempting to keep all calculations through extrinsic information. This idea triggers the definition of another performance criterion for irregular codes, namely the *extrinsic message degree* (EMD), which was introduced by Tian [Tian-2003].

EMD *of a variable node set* that generates a cycle is the number of extrinsic check nodes that are connected to this variable node set. The EMD *of a cycle* that does not contain any sub-cycles can be defined as $\sum (d_i - 2)$ where d_i is the degree of the i^{th} variable node in the cycle, since a variable node has to be connected to two check nodes in a cycle to construct that cycle. If there are sub-cycles, the EMD of the cycle is reduced since one or more variable nodes will have more than two connections to the check nodes of this cycle. The *approximate cycle EMD* (ACE) *of a variable node* of degree d is defined as $(d-2)$ and the ACE *of a check node* is 0. So, the ACE *value of a cycle*, whether the cycle includes sub-cycles or not, is approximated as $\sum (d_i - 2)$, where the summation is over all nodes of a cycle.

3.1.3 Stopping Set

Similarly to the EMD, the criterion of *stopping set* is more concerned about the connectivity of the cycles than their length and distribution. Stopping set is defined as the set of variable nodes that all the neighboring check nodes are connected to the variable node set at least twice [Richardson-2002]. The number of variable nodes in the stopping set is the size of the set. In a bipartite graph that is free of degree-1 variables, every stopping set contains cycles, but not vice versa. The only stopping set formed by a single cycle is the one that consists of all degree-2 variable nodes. An example of such a stopping set is described below, where C_i's indicate the columns of the parity check matrix; corresponding to the variable nodes v_i. The stopping set of this example is $\{v_1, v_2, v_3\}$ and its size is 3.

$$C_1^T = 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1$$

$$C_2^T = 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0$$

$$C_3^T = 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1$$

$$(C_1 + C_2 + C_3)^T = 2 \quad 2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 2$$

As for the examples given in Figure 3.2, part (a) demonstrates a stopping set $\{v_1, v_2, v_3\}$ and part (b) shows another stopping set $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. On the other hand, the cycle shown in part (c) does not contain a stopping set, because the check node c_4 has only one connection to the variable node set $\{v_1, v_2, v_3\}$.

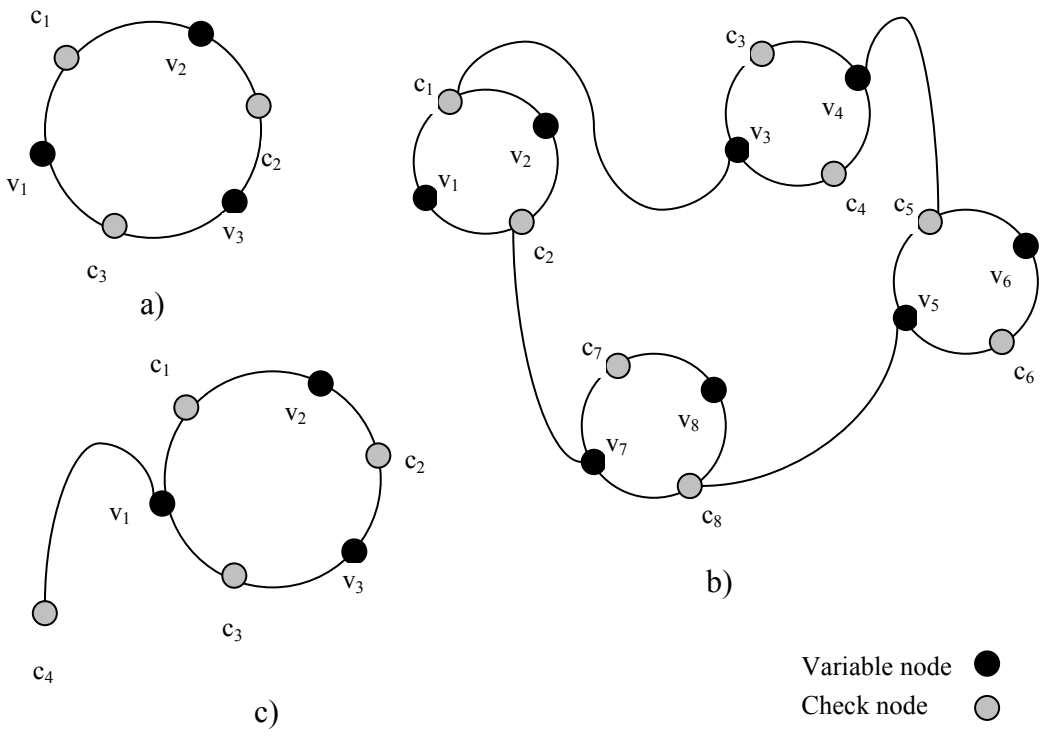


Figure 3.2 Examples of variable node sets generating a stopping sets

3.2 Pseudo-Random LDPC Code Construction Algorithm

In the following, we describe the structure of the algorithms with which we generate our parity check matrices according to the desired criteria. We name the

algorithm, whose flow graph is given in Figure 3.3, either as LGDC (local girth distribution check) or ACEC (approximate cycle EMD (extrinsic message degree) check) or SSC (stopping set check) depending on the criterion used in the second check box. At *the initialization step*, in addition to the constraints related to the used criterion, the column and row size (n, m) of the parity check matrix and the column weight distribution $\lambda(x)$ are specified. An $m\times1$ vector which obeys the $\lambda(x)$ is chosen as the initial parity check matrix H_0. The algorithm constructs the i^{th} parity check matrix of dimension $m\times(i+1)$, by adding a suitable column at each step, and keeping the row weight w_r almost constant.

The process is based on a heuristic approach, where random candidate columns are generated and each candidate is subjected to the constraints of the ensemble, which determine whether it can be permanently added to the previously found sub-matrix H_{i-1} of dimension $m\times i$ that satisfies the desired criteria. For each column to be added, we randomly generate a column vector of length m with a column weight adjusted according to desired $\lambda(x)$. To give the decision of permanently adding this column into the parity check matrix, two conditions have to be met: Firstly, the row weights of each row are checked in order to have nearly uniform row weight distribution; secondly, each variable node is checked with respect to either the local girth, or ACE, or SS criterion. As the size of the parity check matrix grows, it becomes increasingly difficult to find a valid column vector since the check time grows exponentially with the number of iterations. The algorithm terminates when the matrix is generated with all columns satisfying the desired criteria.

In our LGDC algorithm, desired local girth distribution array is specified at the initialization step. For each column, after controlling the row weight constraint, the length of the shortest length cycle generated by that column is evaluated. If it passes the initially set LGD constraint, the column is accepted.

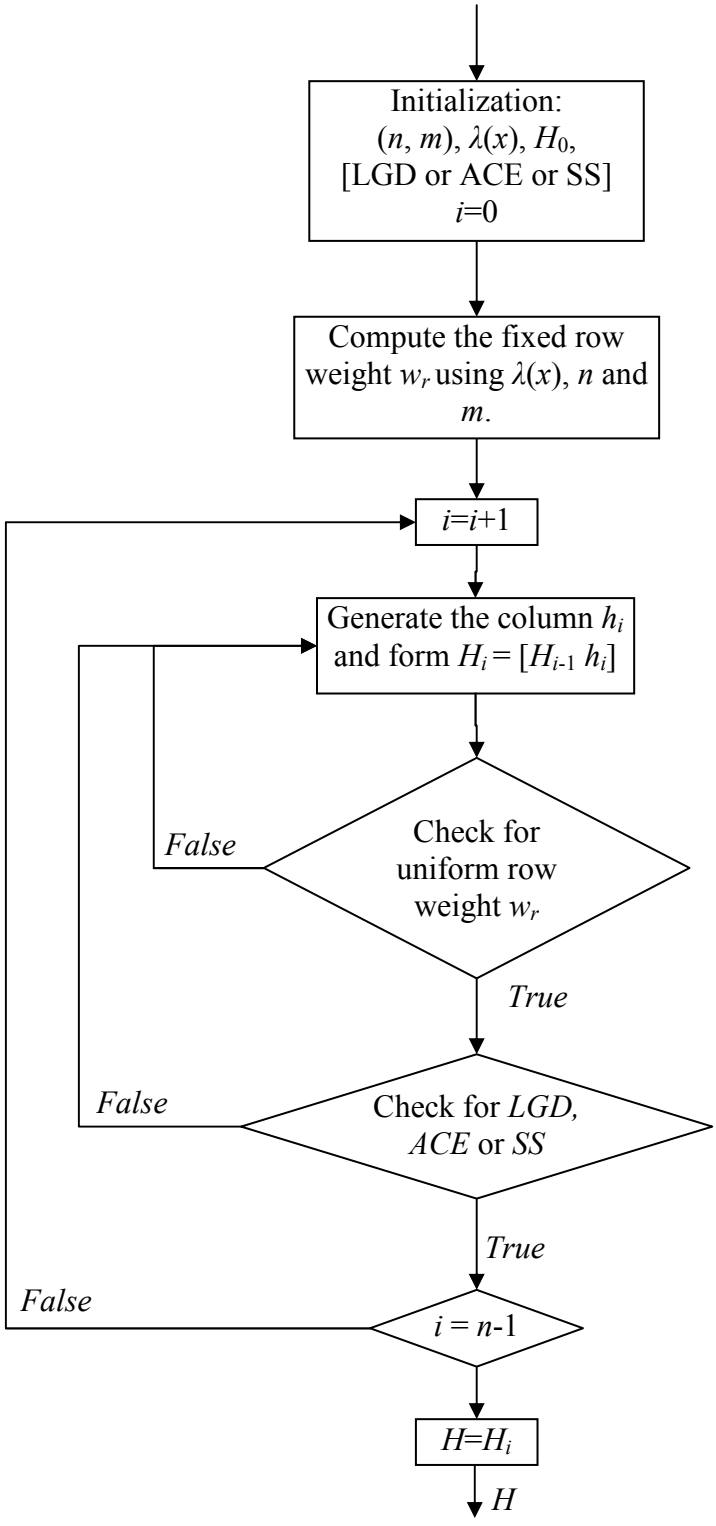


Figure 3.3 Flow graph of the code construction algorithm

The constraint of the ACEC algorithm is defined with two parameters (d_{ACE}, η), such that each cycle either has a length greater than $2d_{ACE}$ or it has an ACE value greater than η. To find the ACE values generated by the newly added column, the algorithm evaluates the ACE values of all the cycles with length shorter than $2d_{ACE}$. If the algorithm finds a cycle with ACE value less than η, it discards the column and tries another column for the vacant position. Otherwise it proceeds to the next step.

Similarly, the SSC algorithm checks the stopping set size. If the size is less than the *stopping set limit,* the algorithm discards the column and generates another column. The criterion of SS is practically more meaningful at medium-to-high degree variable nodes, since small size of stopping sets are more likely to occur at high degree nodes [Ramamoorthy-2004]. On the other hand, the ACEC algorithm works more efficiently with low degree variables. A joint approach where the ACEC algorithm is used at low-degree nodes and SSC algorithm is used at medium-to-high degree variable nodes is claimed [Ramamoorthy-2004] to provide better performance results, which is also tested in our work.

3.3 Effect of Different Criteria on the Performance

In this section, we present our experimental results, which compare the "BER versus SNR" performances of the (576, 288) codes that we construct using different criteria. The first criterion is related to the local girth distribution whereas the second criterion is concerned about the extrinsic message degree. Finally, the third criterion is a combination of extrinsic message degrees and stopping set sizes of the graph. Regular codes with (w_c, w_r) = (3, 6) and irregular codes with structures ("93a", 7) and ("93y", 7) described in Section 2.3 are constructed.

3.3.1 Effect of Local Girth Distribution on the Performance

In order to see the effect of cycles and local girth distribution, regular parity check matrices of size 288×576 with (w_c, w_r) = (3, 6) are generated by our LGDC algorithm. Besides regular matrices, irregular matrices with similar local girth distributions are constructed for comparison purposes. LGD parameters of the generated codes are summarized in Table 3.1.

Table 3.1 Local girth distribution parameters of the generated codes

Code Type	Girth	Mean
Regular	4	6
	6	6
	6	8
Irregular "93a" and "93y"	6	6
	6	8

- **Regular LDPC Codes:**

The 288×576 parity check matrix of a regular LDPC code with uniform column weight 3 that is constructed by our LGDC algorithm is shown in Figure 3.4, where dots indicate the non-zero locations of the matrix.

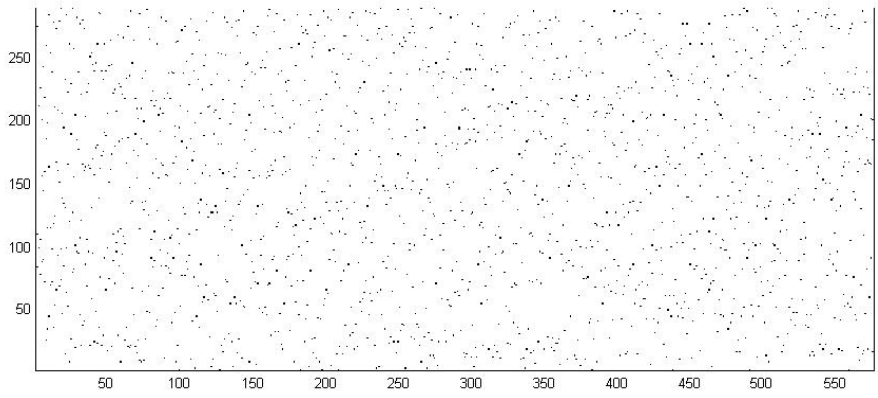


Figure 3.4 The parity check matrix of a (576, 288) regular (3, 6) LDPC code

Figure 3.5, Figure 3.6 and Figure 3.7 show the local girth histograms of the regular LDPC codes generated by our LGDC algorithm. Note that, these local girth distributions are chosen only for illustrative purposes and LDPC codes with better distributions can also be generated by the same algorithm, in longer time.

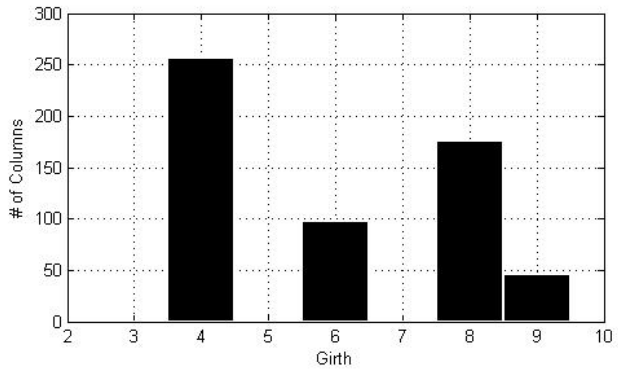


Figure 3.5 Local girth histogram of the parity check matrix of a (576, 288) regular (3, 6) LDPC code, girth=4, mean=5.96

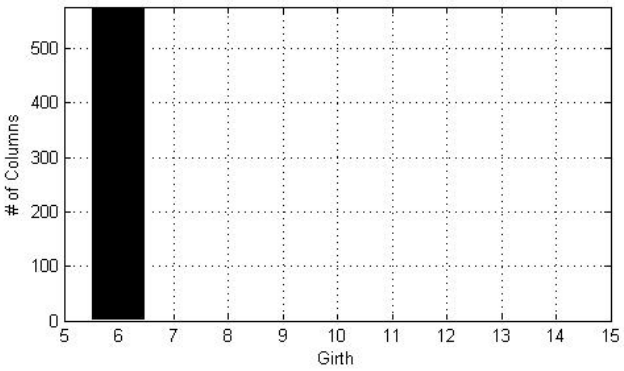


Figure 3.6 Local girth histogram of the parity check matrix of a (576, 288) regular (3, 6) LDPC code, girth=6, mean=6

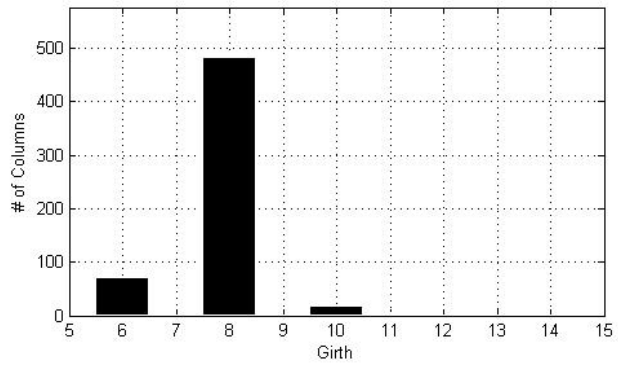


Figure 3.7 Local girth histogram of the parity check matrix of a (576, 288) regular (3, 6) LDPC code, girth=6, mean=7.83

We have measured the "BER versus SNR" performance of these three (576, 288) regular codes using belief propagation decoding, where maximum number of iterations is set to 100. Comparing the performance shown in Figure 3.8 one observes that the code with *girth*=4 *and mean*=6 has the worst, and the code with *girth*=6 and *mean*=8 has the best performance as expected. Since, the performance increase obtained in the former case is much more than that in the latter, to avoid cycles of length four seems to be the main concern about the generation of the LDPC codes.

More specifically, at BER=10^{-3}, increase of girth from 4 to 6 results in an SNR gain of approximately 0.23 dB, whereas keeping the girth at 6, if the mean value of the LGD is raised from 6 to 8, only an SNR gain of ~0.03 dB is obtained.

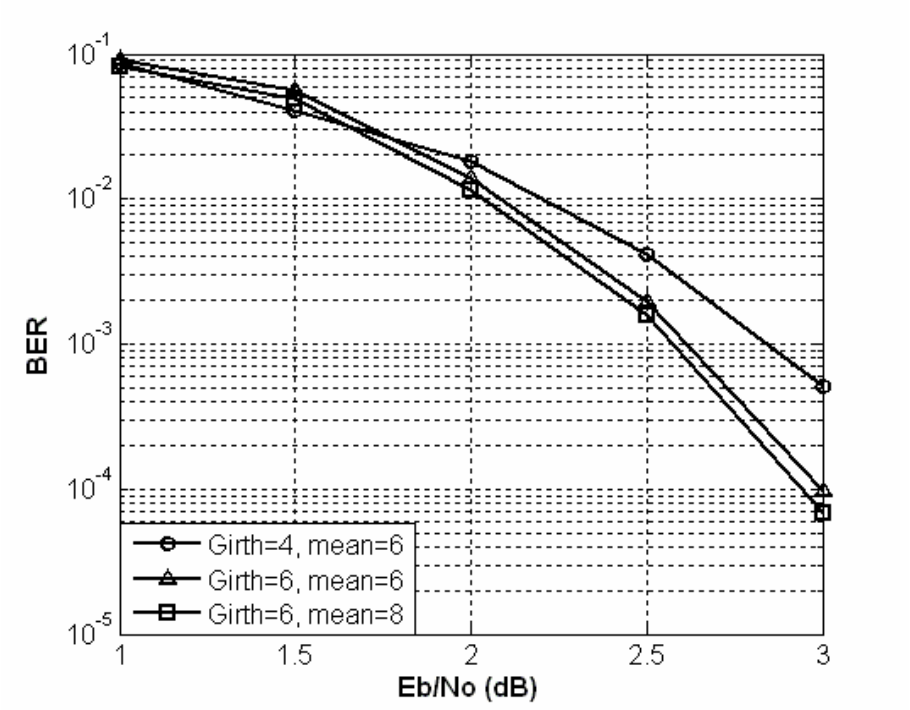


Figure 3.8 Performance of (576, 288) regular (3, 6) pseudo-random codes generated by LGDC algorithm

- **Irregular LDPC Codes:**

In order to generate the irregular LDPC codes, ("93a", 7) and ("93y", 7) structures explained in Section 2.3, we can use an algorithm similar to the one described by the flow graph shown in Figure 3.3, with some additional constraints on pseudo-randomly generated columns. The local girth distributions which are close to those of the regular codes mentioned above are chosen for fair comparison.

Figure 3.9 and Figure 3.10 show the local girth histograms of two examples of the generated irregular LDPC matrices with LGD means of 6 and 8.

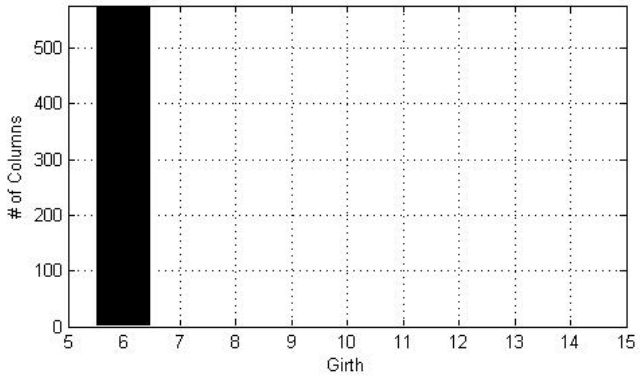


Figure 3.9 Local girth histogram of the parity check matrix of the (576, 288) irregular (both ("93a", 7) and ("93y", 7)) LDPC codes, girth=6, mean=6

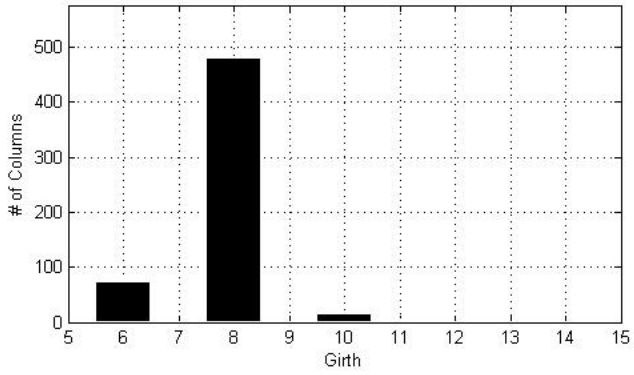


Figure 3.10 Local girth histogram of the parity check matrix of the (576, 288) irregular (both ("93a", 7) and ("93y", 7)) LDPC codes, girth=6, mean=8

We have compared the "BER versus SNR" performances of the ("93a", 7) and ("93y", 7) irregular codes, which have the local girth distributions mentioned above. The decoding algorithm is the belief propagation algorithm with 100 iterations.

As observed in Figure 3.11 and Figure 3.12, the performances of these (576, 288) codes get better with increasing LGD mean. For both ("93a", 7) and ("93y", 7) codes, the SNR gain is approximately 0.1 dB at 10^{-4} BER for an increase from 6 to 8 in the LGD mean. So, simulations give an idea about why the local girth distribution is an effective tool for designing short block length LDPC codes.

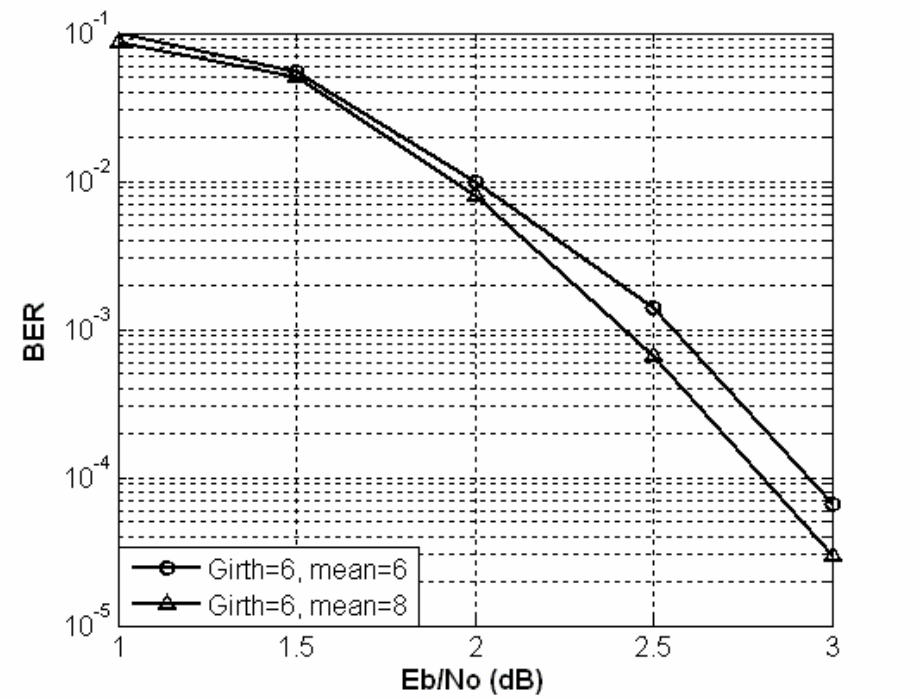


Figure 3.11 Performance of (576, 288) irregular ("93a", 7) LDPC codes

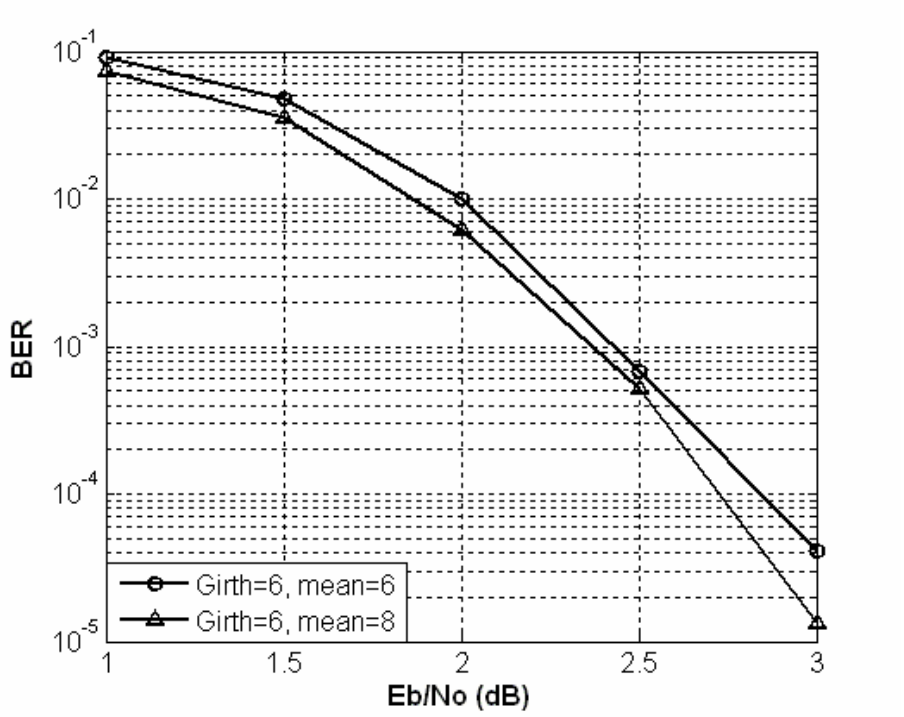
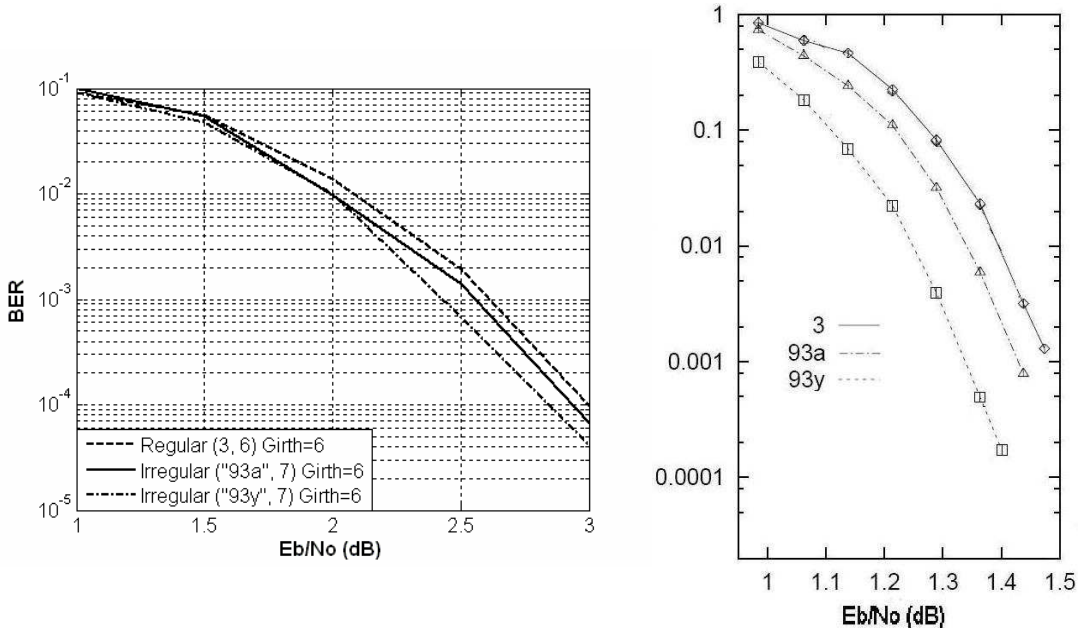


Figure 3.12 Performance of (576, 288) irregular ("93y", 7) LDPC codes

As the last remark in this sub-section, we compare in Figure 3.13 (a) the performances of the (576, 288) regular and irregular LDPC codes constructed by the LGDC algorithm with the same local girth distributions. We observe that regular LDPC codes perform 0.1-0.2 dB inferior to irregular LDPC codes. Also, when we compare the irregular constructions, ("93y", 7) irregular construction is ~0.1 dB superior to ("93a", 7) irregular construction since it has a better elite variable node connection structure as described in Section 2.3. The results are similar to those obtained in [MacKay-1998]. However, MacKay generated regular and irregular LDPC codes with n=9972 and k=m=4986 whose performances shown in Figure 3.13 (b) are much better than our results, mainly because of the longer code lengths.



(a) (b)

Figure 3.13 Performance of regular and irregular LDPC codes

a) of size (576, 288) generated in this work, with girth=6, mean=6

b) of size (9972, 4986) generated in [MacKay-1998]

3.3.2 Effect of Approximate Cycle EMD and Stopping Sets

Irregular ("93a", 7) LDPC codes of size (576, 288) are constructed by our ACEC algorithm and iteratively decoded using the belief propagation decoder, described in Section 2.4.2. In order to fairly compare the effect of optimizing the local girth distribution to that of optimizing the ACE criterion; ACE values of the parity check matrices are chosen as 3 and 4, since for the $(w_c, w_r) = (3, 6)$ regular codes, ACE values of 3 and 4 correspond to cycles of length 6 and 8. Hence, the performances of the codes generated by the LGDC algorithm in Section 3.1.1, having girth values of 6 and 8 are shown together with the performances of the codes designed by the ACEC algorithm with ACE limits $\eta = 3$ and 4 in Figure 3.14.

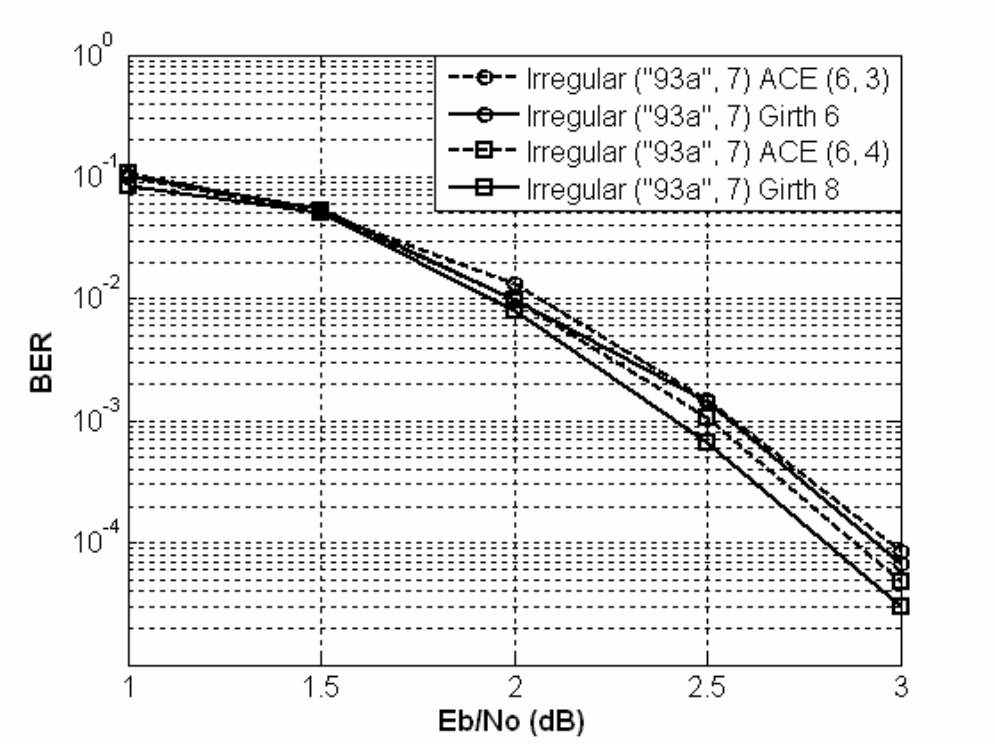


Figure 3.14 Performance of (576, 288) pseudo random irregular ("93a", 7) codes with different ACE and girth values

It can be seen that similar irregular matrices constructed by the ACEC algorithm do not provide any performance increase over those designed by the LGDC algorithm. The reason can be described as follows:

The ACEC algorithm with parameters (d_{ACE}, η) generates graph cycles either longer than $2d_{ACE}$ or having ACE values greater than η. ACE value of a variable node with degree d is (d−2). When two variable nodes with degree higher than (η/2+2) are generated, the algorithm accepts these nodes even if they are connected to the same check nodes, since the ACE value generated by these variable nodes is higher than $2(\eta/2 + 2 - 2)=\eta$.

For the ("93a", 7) codes, degree of a variable node is either 9 or 3. So, even when a candidate variable node of degree 9 has exactly the same check node connections with a previously accepted variable node of degree 9, the corresponding ACE value

is 2×(9−2)=14. The ACEC algorithm with η=3 or 4 accepts the newly added high degree variable node since 14>3 and 4, so it has little control on the high degree variable node additions.

However, for the variable nodes with degree 3, the ACEC algorithm works well, since two overlapping variable nodes ends up with the ACE value of 2×(3−2)=2 and the algorithm rejects the candidate column since 2<3 (2<4). So, the ACEC algorithm works well for low degree variable nodes.

In the literature [Ramamoorthy-2004], a joint approach is proposed, which uses the ACEC for low degree variable nodes and the stopping set check (SSC) for high degree variable nodes. We also implement a joint algorithm (JA) to constant ("93a", 7) codes, where variable nodes of degree 3 are controlled by ACEC and those of degree 9 are controlled by the stopping set check criterion. Irregular parity check matrices of size 288×576 are constructed and iteratively decoded using the belief propagation decoder, described in Section 2.4.2. Performances of the codes produced by the local girth distribution and the ACE criteria are compared with those generated by the joint approach algorithm. For the (576, 288) irregular codes shown in Figure 3.15, the ACE limit is chosen as (d_{ACE}, η) = (7, 3) and stopping set size limit is chosen as 30 (SS=30).

It is observed from Figure 3.15 that the code generated by the joint approach (JA) algorithm is slightly better when it is compared with similar LDPC codes generated by the ACEC or the LGDC algorithms. The improvement is small most probably because of the small size of the parity check matrices.

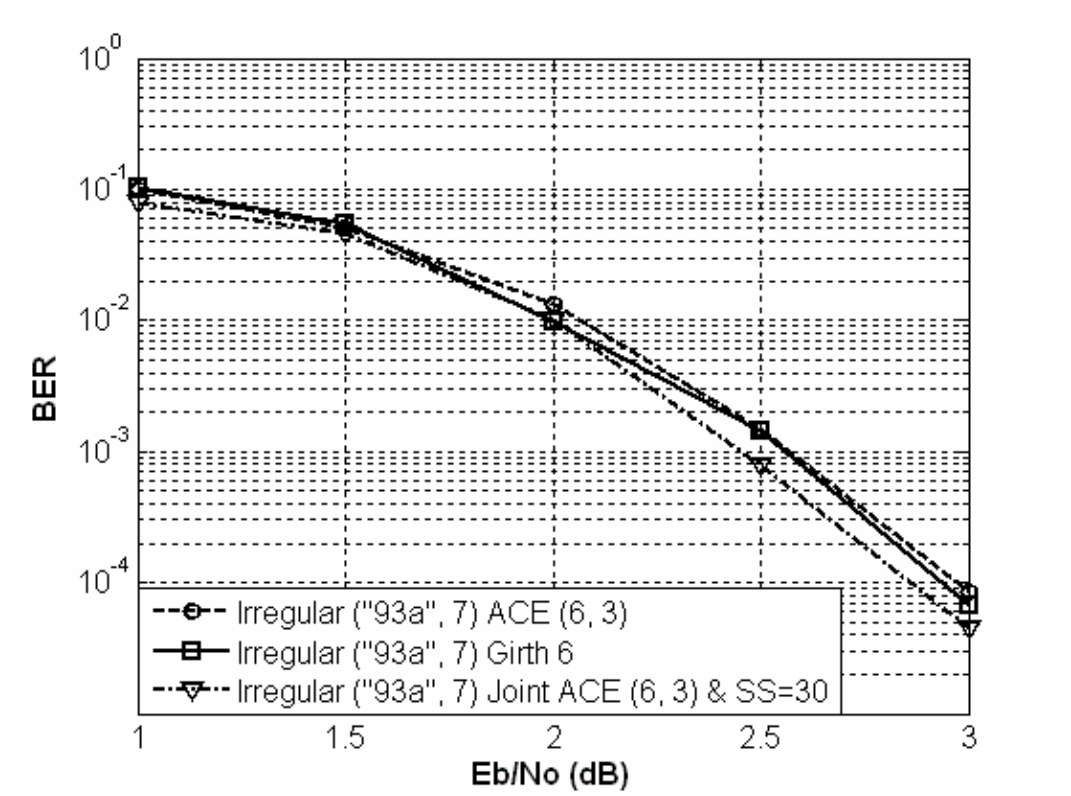


Figure 3.15 Performance of (576, 288) irregular ("93a", 7) codes generated by ACEC, LGDC and Joint Approach algorithms

3.4 Results

Finally, we combine the summarized performance curves of regular and irregular codes designed with respect to different criteria.

As depicted in Figure 3.16, the joint approach algorithm which uses ACE and SS criteria seems superior to both ACEC and the LGDC algorithms. This shows that for short block length irregular LDPC codes, the graph connectivity of cycles is as important as the length and distribution of cycles. On the other hand, the most effective way of increasing performance is simply avoiding the cycles of length four in the graph, before the application of any other sophisticated criterion.

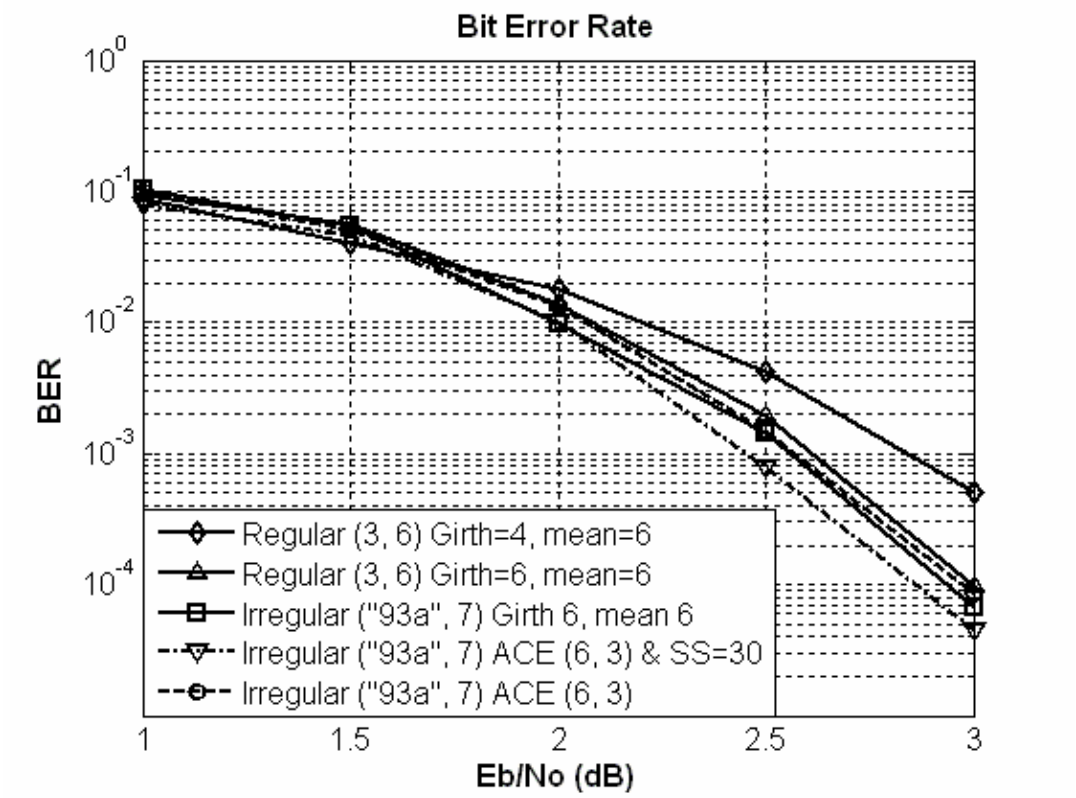


Figure 3.16 Performance of (576, 288) pseudo-random codes generated by the ACEC, LGDC and Joint Approach algorithms

Figure 3.17 shows the performance comparison of the codes generated in this thesis and similar codes generated in [Ramamoorthy-2004] of slightly higher length. The codes generated by Ramamoorthy have the weight distributions $\lambda(x)=0.2186x+0.1470x^2+0.1692x^4+0.0136x^5+0.0517x^6+0.3999x^{16}$ and $\rho(x)=x^8$, where the irregular codes that we generate have weight distributions $\lambda(x)=11/14x^2+3/14x^8$ and $\rho(x)=x^6$. If the BER performances of rate 1/2 codes generated by our LGDC, ACEC and joint approach algorithms are compared with those of Ramamoorthy, although the performance order of ACEC, LGDC and joint approach algorithms is similar, Ramamoorthy's results are much better. The main reason of this performance difference is the existence of highly elite variable nodes of degree 17 (see the last term of $\lambda(x)$) in Ramamoorthy's codes. Almost 40% of the variable node edges belong to these elite variable nodes, which are checked by 17 different parity check equations. Hence the belief propagation algorithm can

decide on the correct codeword more easily by using the reliable messages coming from the elite variable nodes in order to correct the values of the small-degree variable nodes.

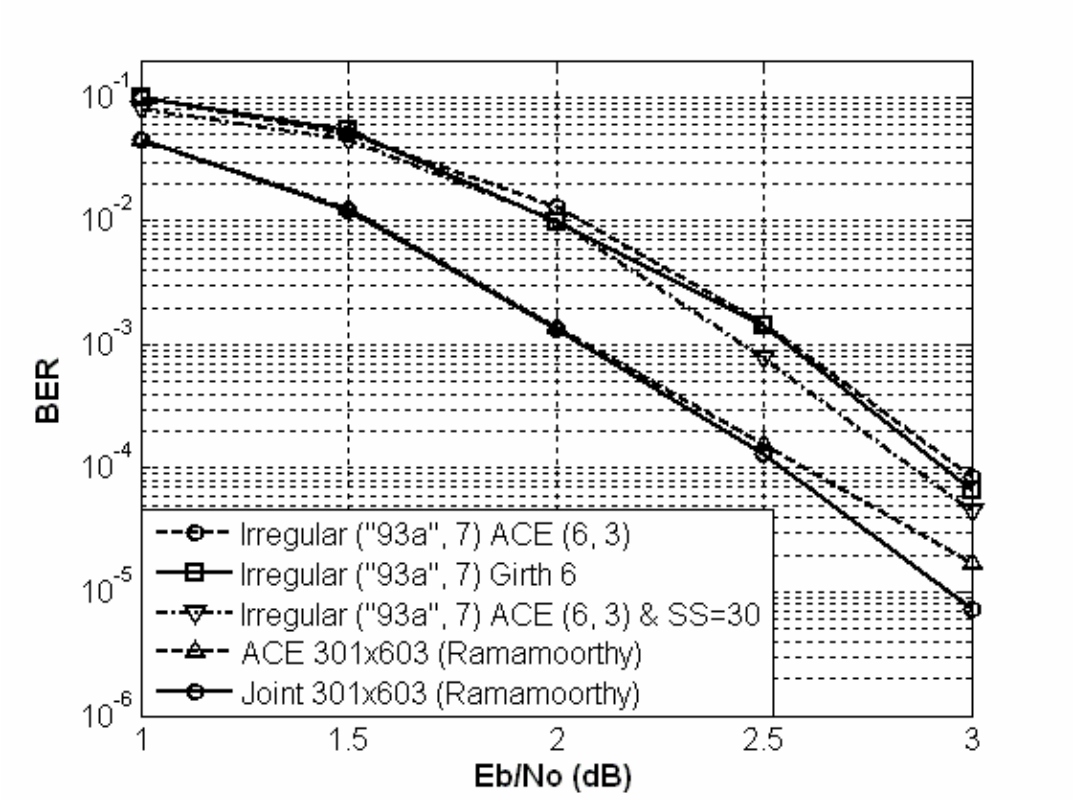


Figure 3.17 Performances of (576, 288) codes generated by our LGDC, ACEC and joint approach algorithms and similar codes generated in [Ramamoorthy-2004].

CHAPTER 4

QUASI-CYCLIC LDPC CODES

Despite the excellent error-correcting properties of some known pseudo-random LDPC codes, complexity resulting from storage issues tends to dominate the system architecture and makes such codes hard to use in actual communication scenarios. High complexity of pseudo-random LDPC codes is a direct consequence of the fact that very large amount of information is necessary to specify positions of the non-zero elements of the huge parity check matrices. Quasi-cyclic LDPC codes are good candidates to solve the memory problem, since their parity check matrices consist of circulant permutation matrices. They also have encoding advantages over pseudo-random LDPC codes since they can be encoded using simple shift-registers, with a complexity linearly proportional to the code length.

In this chapter, construction methods of *quasi-cyclic regular and irregular LDPC* codes are explained and their performance is measured by simulations. Section 4.1 begins with the description of the construction method for the *algebraically structured quasi-cyclic LDPC* codes. This is followed by the irregular structures of type ("93", 7) that we adapt from pseudo-random forms of MacKay to quasi-cyclic form. Then, "BER versus SNR" performances of regular and irregular quasi-cyclic codes are compared. Section 4.2 describes the relationship between the shift values and the cycles in quasi-cyclic LDPC codes, followed by an algorithm to generate girth controlled quasi-cyclic LDPC codes. Then, regular quasi-cyclic LDPC codes are constructed with girth values 4, 6, 8 and 10 and their performances are presented. In Section 4.3 we finally compare both regular and irregular

algebraically constructed quasi-cyclic codes with the pseudo-random codes of the previous chapter.

4.1 Algebraically Structured Quasi-Cyclic LDPC Codes

In this section, first, we will give the description of the algebraically structured regular quasi-cyclic code construction and then we will describe our solution to the construction of irregular algebraically structured quasi-cyclic codes. The parity check matrix of a quasi-cyclic LDPC code consists of permutation matrices, which are usually derived from identity matrices. In Figure 4.1, examples of such sub-matrices are shown, where I^{α} denotes an identity matrix I, whose columns are (α-1) times circularly shifted to the right (or rows are (α-1) times circularly shifted up). Notice that if the size of the sub-matrix is $S \times S$, then $I^{S+1} = I$. More generally, $I^{a} = I^{a(\mathrm{mod}\, S)}$.

$$I = I^5 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad I^2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad I^3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad I^4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 4.1 Sub-matrices of size 4×4 used in quasi-cyclic parity check matrices

4.1.1 Algebraically Structured Regular Quasi-Cyclic LDPC Codes

The parity check matrix of an algebraically structured regular quasi-cyclic LDPC code with size $Sw_c \times Sw_r$ is shown in Figure 4.2, which is first proposed by Tanner in 2004. There are lots of similar quasi-cyclic LDPC codes proposed in the literature, as in Figure 4.3 [Myung-2005] or in Figure 4.4 [Honary-2005].

$$H = \begin{pmatrix} I & I^{a} & \cdots & I^{a^{(\omega_r-1)}} \\ I^{b} & I^{ab} & \cdots & I^{a^{(\omega_r-1)}b} \\ \vdots & \vdots & \vdots & \vdots \\ I^{b^{(\omega_c-1)}} & I^{ab^{(\omega_c-1)}} & \cdots & I^{a^{(\omega_r-1)}b^{(\omega_c-1)}} \end{pmatrix}$$

Figure 4.2 Structure of a regular quasi-cyclic parity check matrix of size $Sw_c \times Sw_r$ proposed by Tanner in 2004

$$H = \begin{pmatrix} I & I & I & I & I \\ I & I^{a} & I^{a^2} & \cdots & I^{a^{(\omega_r-1)}b} \\ I & I^{a^2} & I^{a^3} & \cdots & \vdots \\ I & \vdots & \vdots & \vdots & \vdots \\ I & I^{a^{(\omega_c-1)}} & I^{a^{(\omega_c-1)+1}} & \cdots & I^{a^{(\omega_r-1)}a^{(\omega_c-1)}} \end{pmatrix}$$

Figure 4.3 Structure of a regular quasi-cyclic parity check matrix of size $Sw_c \times Sw_r$ proposed by Myung in 2005

$$H = \begin{pmatrix} I & I & I & I & I \\ I & I & I^{2} & \cdots & I^{(\omega_r-1)} \\ I & I^{2} & I^{4} & \cdots & I^{2(\omega_r-1)} \\ I & \vdots & \vdots & \vdots & \vdots \\ I & I^{(\omega_c-1)} & I^{2(\omega_c-1)} & \cdots & I^{(\omega_r-1)(\omega_c-1)} \end{pmatrix}$$

Figure 4.4 Structure of a regular quasi-cyclic parity check matrix of size $Sw_c \times Sw_r$ proposed by Honary in 2005

The $Sw_c \times Sw_r$ parity check matrix of a regular, quasi-cyclic, (n, w_r, w_c) LDPC code is constructed from sub-matrices of size $S \times S$, where $S=n/w_r$. S must be greater than the number of the sub-matrices ($w_r w_c$) in order not to have short cycles, since repetition of sub-matrices within H is the main cause of 4-cycles. Figure 4.5 (a) is

an example of such a case, where similar rows (or columns), i.e., the 1^{st} and 5^{th}, the 2^{nd} and 6^{th}, etc., all create 4-cycles. Another example can be given using the parity check matrix shown in Figure 4.2, choosing a=b=2 and (w_c, w_r)=(3, 6) as in Figure 4.6. If the sub-matrix size S=48, since I^{64}=$I^{\text{mod48(64)}}$=I^{16} and I^{128}=$I^{\text{mod48(128)}}$=I^{32}, the resulting 144×288 parity check matrix will have repeated sub-matrices which create 4-cycles. On the other hand, with S = 47, the resulting 141×282 parity check matrix in Figure 4.6 will have no repetition of sub-matrices and consequently, no cycles of length four.

Therefore, before determining the size of the parity check matrix, the number of distinct sub-matrices that can be generated for the chosen a, b and S values should be checked. For example, if one chooses S = 56 for the structure in Figure 4.6, only 6 distinct sub-matrices can be generated (i.e., I, I^2, I^4, I^8, I^{16} and I^{32}) instead of 18 distinct sub-matrices.

One final remark is that, even if there are no repetitions of sub-matrices, there still can be cycles of length four as a result of the constant shift differences between consecutive sub-matrices. Example of such a case is given in Figure 4.5 (b), where column pairs like 1-6, 3-8 and 4-5 create cycles of length four as a result of the constant shift between I - I^2 and I^3 - I^4 sub-matrices.

$$\begin{pmatrix} I & I \\ I & I \end{pmatrix} = \left(\begin{array}{cccc:cccc} 1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&0&1&0 \\ \hdashline 0&0&0&1&0&0&0&1 \\ 1&0&0&0&1&0&0&0 \\ \hdashline 0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&0&1&0 \\ 0&0&0&1&0&0&0&1 \end{array}\right) \quad \text{(a)}$$

$$\begin{pmatrix} I & I^2 \\ I^3 & I^4 \end{pmatrix} = \left(\begin{array}{cccc:cccc} 1&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0&1 \\ \hdashline 0&0&0&1&1&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ \hdashline 1&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&1&0 \end{array}\right) \quad \text{(b)}$$

Figure 4.5 Parity check matrices which cause of cycles of length four

Taking all these considerations into account, a sub-matrix size of S = 94 with (w_c, w_r) = (3, 6) generates a quasi-cyclic parity check matrix of size 282×564 shown in **Figure 4.7**, which has the structure shown in Figure 4.6. Using the girth check part of the LGDC algorithm introduced in Section 3.2, we compute the local girth distribution and sketch it in Figure 4.8.

$$H = \begin{pmatrix} I & I^2 & I^4 & I^8 & I^{16} & I^{32} \\ I^2 & I^4 & I^8 & I^{16} & I^{32} & I^{64} \\ I^4 & I^8 & I^{16} & I^{32} & I^{64} & I^{128} \end{pmatrix}$$

Figure 4.6 Parity check matrix of a regular quasi-cyclic LDPC code with the structure given in Figure 4.2 and (w_c, w_r) = (3, 6)

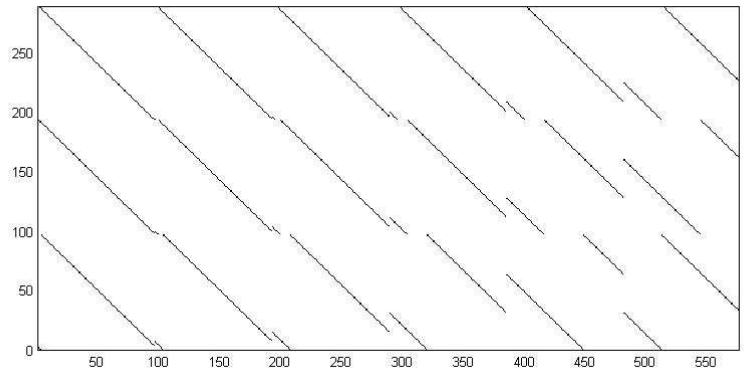


Figure 4.7 The 282×564 parity check matrix of a regular (3, 6) quasi-cyclic LDPC code having the structure given in Figure 4.6

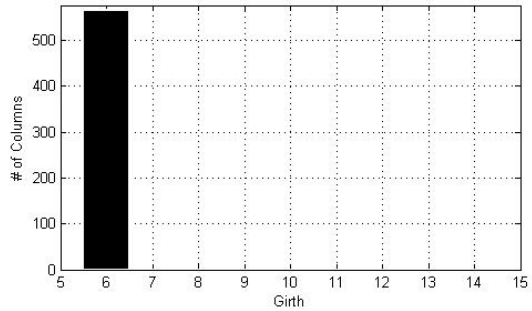


Figure 4.8 Local girth histogram of the parity check matrix of a (564, 282) regular (3, 6) quasi-cyclic LDPC code, whose parity check matrix is shown in **Figure 4.7.**

4.1.2 Algebraically Structured Irregular ("93", 7) Quasi-Cyclic LDPC Codes

In order to construct the algebraically structured irregular codes of this work, we have applied the regular quasi-cyclic construction proposed by Tanner in 2004, to the irregular codes of type “93” described in [MacKay-1998]. We have generated irregular quasi-cyclic LDPC codes by inserting sum of permutation matrices into the parity check matrix H instead of sub-matrices of weights greater than one shown in Figure 2.3 and Figure 2.5. For instance, I^2 and I^4 shown in Figure 4.1 can be used to generate a sub-matrix with column and row weight of 2 as in Figure 4.9.

$$I^2 + I^4 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Figure 4.9 A sub-matrix of H used for irregular quasi-cyclic LDPC code construction

The structure of an irregular (“93a”, 7) quasi-cyclic parity check matrix proposed by MacKay in 1998 is as shown in Figure 2.3 and we propose the quasi-cyclic structure shown in Figure 4.10. Notice that, although most of the sub-matrices of H are of size $S \times S$, sub-matrices shown in the last column are only half size, i.e., $S/2$ by $S/2$. Choosing S=94 , an irregular (“93a”, 7) quasi-cyclic parity check matrix of size 282×564 is demonstrated in Figure 4.11 where dots indicate the nonzero entries. The local girth histogram of the parity check matrix is computed by using the girth check part of the LGDC algorithm described in Section 3.2 and sketched in Figure 4.12.

$$H = \begin{pmatrix} I & I^2 & I^4 & I^8 & I^{16} & I^{32} & I^{64} \\ & & & & & 0 & I^{128} + I^{256} \\ I^2 & I^4 & I^8 & I^{16} & I^{32} & I^{64} & I \\ & & & & & 0 & I^{512} + I^{1024} \\ I^4 & I^8 & I^{16} & I^{32} & I^{64} & I^{128} & I^2 \\ & & & & & 0 & I^{2048} + I^{4096} \end{pmatrix}$$

Figure 4.10 Structure of the parity check matrix for irregular ("93a", 7) quasi-cyclic LDPC code

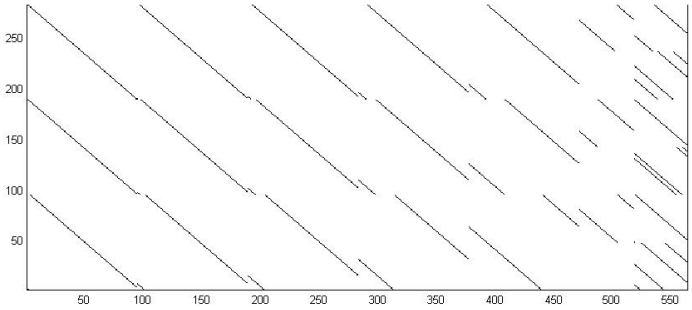


Figure 4.11 The 282×564 parity check matrix of an irregular ("93a", 7) quasi-cyclic LDPC code having the structure given in Figure 4.10

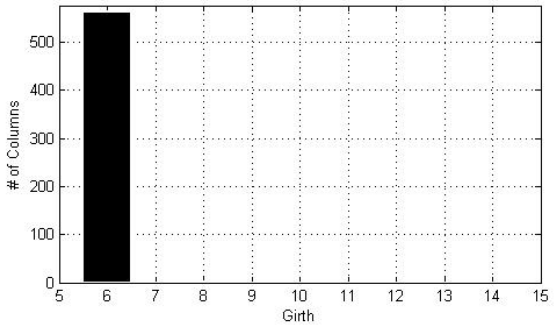


Figure 4.12 Local girth histogram of the parity check matrix of a (564, 282) irregular ("93a", 7) quasi-cyclic LDPC code, whose parity check matrix is shown in Figure 4.11

We similarly propose the irregular ("93y", 7) quasi-cyclic parity check matrix structure shown in Figure 4.13 for the MacKay's irregular ("93y", 7) quasi-cyclic parity check matrix given in Figure 2.5. Now, H has sub-matrices with row and column weights of 1, 2 and 4. Similar to the weight-2 sub-matrices, weight-4 matrices are generated by the sum of 4 permutation matrices, i.e., $I^2 + I^4 + I^7 + I^{17}$.

In **Figure 4.14**, an irregular ("93y", 7) quasi-cyclic parity check matrix of size 282×564 (for S=94) is given, where dots indicate the nonzero entries. We have computed the local girth histogram of the parity check matrix by using the girth check part of the LGDC algorithm described in Section 3.2 and sketched in Figure 4.15.

$$H = \begin{pmatrix} I & I^2 & I^4 & 0 & 0 & \begin{matrix} 0 & 4^i \\ 0 & 4^{ii} \end{matrix} \\ I^2 & I^4 & I^8 & I^{16} & I^{32} + I^{128} & \begin{matrix} I^{256} & 0 \\ I & 0 \end{matrix} \\ I^4 & I^8 & I^{16} & I^{32} + I^{256} & I^{64} & I^{128} \end{pmatrix}$$

$$where\ 4^i = I^8 + I^{16} + I^{32} + I^{64}\ and\ 4^{ii} = I + I^2 + I^4 + I^{128}$$

Figure 4.13 Structure of the parity check matrix of ("93y", 7) irregular quasi-cyclic code

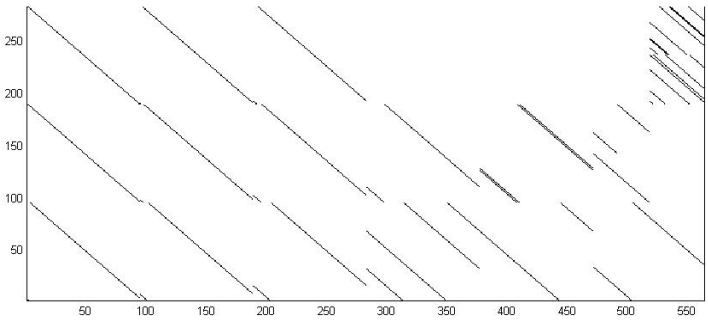


Figure 4.14 The 282×564 parity check matrix for an irregular ("93y", 7) quasi-cyclic LDPC code having the structure given in Figure 4.13

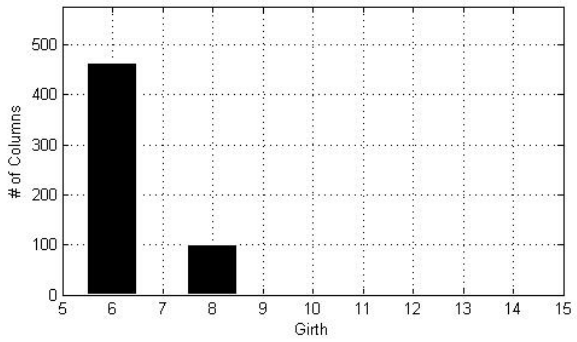


Figure 4.15 Local girth histogram of the parity check matrix of a (564, 282) irregular ("93y", 7) quasi-cyclic LDPC code, whose parity check matrix is shown in **Figure 4.14.**

4.1.3 Performance Comparison of Regular and Irregular Algebraically Structured Quasi Cyclic LDPC Codes

In order to measure the performance of the irregular quasi-cyclic codes that we have proposed, we construct (576, 288) regular and irregular quasi-cyclic LDPC codes, which have the structures given in Figure 4.6 , Figure 4.10 and Figure 4.13 and girth values of 6. They are iteratively decoded using the belief propagation decoder, settling the maximum number of iterations to 100. It can be seen from Figure 4.16 that codes are ordered similarly to the codes shown in Figure 3.13. So, either pseudo-random or quasi-cyclic, irregular ("93y", 7) code has the best and the regular (3, 6) code has the worst performance. Therefore, we can say that the method that we propose for irregular quasi-cyclic LDPC code generation is advantageous.

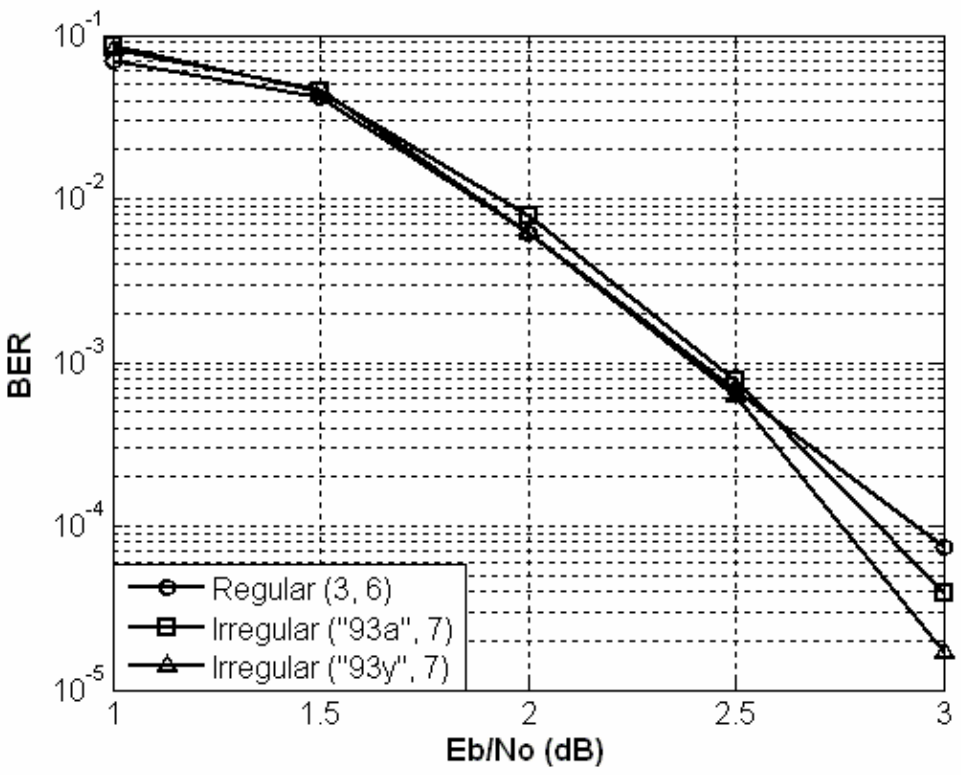


Figure 4.16 Performances of the (564, 282) regular (3, 6) and irregular quasi-cyclic LDPC codes

4.2 Girth Controlled Quasi-Cyclic LDPC Codes

In this section, we review the relations between cycles and shifts values of the sub-matrices in quasi-cyclic LDPC codes as explained by Moura in 2005. We then describe our girth controlled quasi-cyclic code generation algorithm in Section 4.2.1. Examples of regular quasi-cyclic parity check matrices that we construct have size 450×900 and girth values of 4, 6, 8 and 10.

Moura explained that the Tanner Graph of a quasi-cyclic LDPC code contains at least one cycle if and only if there exists a closed path of length $2t$ in the matrix such that its vertices (a_i-times-shifted sub-matrices) satisfy the shift condition given in equation (4.1) where $\oplus$ is the modulo S summation [Moura-2005]. The example cycles of length four and six are shown in Figure 4.17.

$$(\oplus_{i=1}^{i=2t}(-1)^i a_i) = 0 \qquad (4.1)$$

$$\textit{Cycle of Length } 4: = \begin{pmatrix} I & \rightarrow & I^5 \\ \uparrow & & \downarrow \\ I^{20} & \leftarrow & I^{24} \end{pmatrix}; \quad (\oplus_{i=1}^{i=4}(-1)^i a_i) = (-0+4-23+19=0)$$

$$\textit{Cycle of Length } 6: = \begin{pmatrix} I & \rightarrow & I^3 & & - \\ \uparrow & & \downarrow & & \\ - & & I^{24} & \rightarrow & I^{11} \\ \uparrow & & & & \downarrow \\ I^{31} & \leftarrow & - & \leftarrow & I^{20} \end{pmatrix}; \quad (\oplus_{i=1}^{i=6}(-1)^i a_i) = (-0+2-23+10-19+30=0)$$

Figure 4.17 Cycles of length four and six

As the girth value increases, the number of possible ways to generate cycles increases exponentially. The examples for cycle of length six and eight are given in Figure 4.18.

$$Cycle\ of\ Length\ 6: = \begin{pmatrix} I^{a_1} & \rightarrow & I^{a_2} & & - \\ \uparrow & & \downarrow & & \\ I^{a_6} & \leftarrow & - & \leftarrow & I^{a_5} \\ & & \downarrow & & \uparrow \\ - & & I^{a3} & \rightarrow & I^{a_4} \end{pmatrix}$$

$$(\oplus_{i=1}^{i=6}(-1)^i a_i) = -(a_1-1)+(a_2-1)-(a_3-1)+(a_4-1)-(a_5-1)+(a_6-1)=0)$$

$$Cycle\ of\ Length\ 8: = \begin{pmatrix} I^{a_1} & \rightarrow & I^{a_2} & & - & & - \\ \uparrow & & \downarrow & & & & \\ - & & I^{a_3} & \rightarrow & I^{a_4} & & - \\ \uparrow & & & & \downarrow & & \\ - & & - & & I^{a_5} & \rightarrow & I^{a_6} \\ \uparrow & & & & & & \downarrow \\ I^{a_8} & \leftarrow & - & \leftarrow & - & \leftarrow & I^{a_7} \end{pmatrix}$$

$$(\oplus_{i=1}^{i=8}(-1)^i a_i) = -(a_1-1)+(a_2-1)-(a_3-1)+(a_4-1)-(a_5-1)+(a_6-1)-(a_7-1)+(a_8-1)=0)$$

$$Cycle\ of\ Length\ 8: = \begin{pmatrix} I^{a_1} & \rightarrow & I^{a_2} \\ \uparrow & & \downarrow \\ I^{a_4} & \leftarrow & I^{a_3} \\ \uparrow\downarrow & & \\ I^{a_5} & \rightarrow & I^{a_6} \\ \uparrow & & \downarrow \\ I^{a_8} & \leftarrow & I^{a_7} \end{pmatrix}$$

$$(\oplus_{i=1}^{i=8}(-1)^i a_i) = -(a_1-1)+(a_2-1)-(a_3-1)+(a_4-1)-(a_5-1)+(a_6-1)-(a_7-1)+(a_8-1)=0)$$

$$Cycle\ of\ Length\ 8: = \begin{pmatrix} I^{a_1} & \rightarrow & I^{a_2} \\ \uparrow & & \downarrow \\ I^{a_4} & \leftrightarrow & I^{a3} \\ \downarrow & & \uparrow \\ I^{a_5} & \rightarrow & I^{a_6} \end{pmatrix}$$

$$(\oplus_{i=1}^{i=8}(-1)^i a_i) = -(a_1-1)+(a_2-1)-(a_3-1)+(a_4-1)-(a_5-1)+(a_6-1)-(a_7-1)+(a_8-1)=0)$$

Figure 4.18 Cycles of length six and eight

The quasi-cyclic LDPC codes of the form given in Figure 4.2 have girth value of 6 in general (for S equals to a prime number or some of the non-prime numbers); but for some particular values of S, they can also have cycles of length four. For example, if we take S=93 for a (w_c, w_r) = (3, 6) regular matrix given in Figure 4.6, sub-matrices I, I^{32}, I^{4}, I^{128} generate cycles of length four since the shift condition is satisfied as shown in Figure 4.19. As a consequence of the matrix size restrictions, quasi-cyclic LDPC codes with larger girth values (i.e., 8, 10, and 12) can be generated by controlling the shift values of the sub-matrices according to the correspondence between cycles and shifts for the quasi-cyclic LDPC codes.

$$H_{3S\times 6S} = \begin{pmatrix} I & \rightarrow & - & \rightarrow & - & \rightarrow & - & \rightarrow & - & \rightarrow & I^{32} \\ \uparrow & & & & & & & & & & \downarrow \\ - & & - & & - & & - & & - & & - \\ \uparrow & & & & & & & & & & \downarrow \\ I^{4} & \leftarrow & - & \leftarrow & - & \leftarrow & - & \leftarrow & - & \leftarrow & I^{128} \end{pmatrix}$$

$$(\oplus_{i=1}^{i=4}(-1)^{i} a_i) = (-0 + 31 - 127 + 3 = 93 \,(\text{mod } 93) \equiv 0)$$

Figure 4.19 Parity check matrix of a regular quasi-cyclic LDPC code with the structure given in Figure 4.2 and (w_c, w_r) = (3, 6)

4.2.1 Construction of Girth Controlled QC-LDPC Codes

The shift condition for the circulant matrices that generate cycles is given in equation (4.1). Now, we describe our algorithm to construct a girth controlled quasi-cyclic LDPC code by using (4.1). The construction is similar to the structured quasi-cyclic LDPC codes except the places of circulant matrices. These codes are not algebraically structured anymore, since the circulant matrices and their locations are not constant for short block length LDPC codes.
The algorithm is as follows:

Step 1: The sub-matrix size S, column and row weights, w_r and w_c of the parity check matrix and the desired girth value T are the initial parameters. The algorithm generates an empty matrix H_t which has w_r columns and w_c rows. Each location in

the matrix shows the shift value (a-1) of the circularly shifted identity sub-matrix I^a at that location.

$$H_t = \begin{pmatrix} a_1 & a_2 & \cdots & a_{w_r-1} & a_{w_r} \\ a_{w_r+1} & \cdots & \cdots & \cdots & a_{2w_r} \\ a_{2w_r+1} & \cdots & \cdots & \cdots & a_{3w_r} \end{pmatrix}_{w_r \times w_c}$$

$$\updownarrow$$

$$H = \begin{pmatrix} \left(I^{a_1}\right)_{S\times S} & \left(I^{a_2}\right)_{S\times S} & \cdots & \left(I^{a_{w_r-1}}\right)_{S\times S} & \left(I^{a_{w_r}}\right)_{S\times S} \\ \left(I^{a_{w_r+1}}\right)_{S\times S} & \left(\cdots\right)_{S\times S} & \left(\cdots\right)_{S\times S} & \left(\cdots\right)_{S\times S} & \left(I^{2a_{w_r}}\right)_{S\times S} \\ \left(I^{a_{2w_r+1}}\right)_{S\times S} & \left(\cdots\right)_{S\times S} & \left(\cdots\right)_{S\times S} & \left(\cdots\right)_{S\times S} & \left(I^{3a_{w_r}}\right)_{S\times S} \end{pmatrix}_{Sw_c \times Sw_r}$$

Step 2: First column and the first row are filled randomly with shift values a_i, since they can not generate cycles without a second column or row in H_t.

$$H_t = \begin{pmatrix} a_1 & \leftrightarrow & a_2 & a_3 & a_4 & a_5 & a_6 \\ \updownarrow & & \updownarrow & & & & \\ a_7 & \leftrightarrow & ? & - & - & - & - \\ & & & & & & \\ a_8 & & - & - & - & - & - \end{pmatrix}$$

Step 3: The rest of the matrix is filled by random values generated between 0 and (S−1) for the vacant locations each time checking for the shift condition given in equation (4.1) for all the integers $t < T/2$. If a random number at a new location satisfies (4.1) with $t < T/2$, it is discarded and another number is generated. When all vacant locations are filled, the algorithm terminates.

Example: Assume we try to construct a quasi-cyclic LDPC code with girth value of $T = 8$ for $S = 150$, $(w_c, w_r) = (3, 6)$.

Step 1: An empty matrix generated is shown below.

$$H_t = \begin{pmatrix} - & - & - & - & - & - \\ - & - & - & - & - & - \\ - & - & - & - & - & - \end{pmatrix}$$

Step 2: First column and the first row are randomly filled. (For illustrative purposes, we use a specific sequence which clearly is not random.)

$$H_t = \begin{pmatrix} 1 & \leftrightarrow & 2 & 3 & 4 & 5 & 6 \\ \updownarrow & & \updownarrow & & & & \\ 7 & \leftrightarrow & ? & - & - & - & - \\ & & & & & & \\ 18 & & - & - & - & - & - \end{pmatrix}$$

Step 3: For the shift value of the sub-matrix located at (2, 2), values between 0 and 149 except 8 are acceptable since it generates a 4-cycle where $\oplus_{i=1}^{i=4} (-1)^i a_i = 1-2+8-7=0 \pmod{150}$ and the algorithm fills the rest of the vacant positions by preventing the generation of 4-cycles and 6-cycles

$$H_8 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 9 & 11 & 13 & 15 & 17 \\ 18 & 25 & 31 & 36 & 42 & 0 \end{pmatrix}$$

4.2.2 Performance Comparison of Girth Conditioned Quasi Cyclic LDPC Codes

Three regular (3, 6) quasi-cyclic parity check matrices with S=150, and girth values 4, 6 and 8 are generated by the algorithm described in Section 4.2.1. The fourth quasi-cyclic code with girth value 10 is taken from [Moura-2005]. Corresponding parity check matrices are shown in Figure 4.20.

$$H_4 = \begin{pmatrix} 0 & 1 & 3 & 7 & 15 & 31 \\ 1 & 3 & 7 & 15 & 31 & 63 \\ 1 & 2 & 6 & 10 & 18 & 50 \end{pmatrix} \quad H_6 = \begin{pmatrix} 0 & 1 & 3 & 7 & 15 & 31 \\ 1 & 3 & 7 & 15 & 31 & 63 \\ 3 & 7 & 15 & 31 & 63 & 127 \end{pmatrix}$$

$$H_8 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 9 & 11 & 13 & 15 & 17 \\ 18 & 25 & 31 & 36 & 42 & 0 \end{pmatrix} \quad H_{10} = \begin{pmatrix} 80 & 125 & 105 & 104 & 143 & 25 \\ 109 & 85 & 81 & 93 & 80 & 4 \\ 46 & 55 & 66 & 119 & 141 & 135 \end{pmatrix}$$

Figure 4.20 Girth controlled parity check matrices of size 450×900 for regular (3, 6) quasi-cyclic LDPC codes

These regular quasi-cyclic codes are iteratively decoded using the belief propagation decoder, described in Section 2.4.2. The corresponding "BER versus SNR" performances are given in Figure 4.21. One can observe that results are similar to those given in Figure 3.8 for regular pseudo-random codes. The performance of the regular quasi-cyclic LDPC codes also increases as the girth value increases and the largest performance improvement is obtained when cycles of length four are avoided.

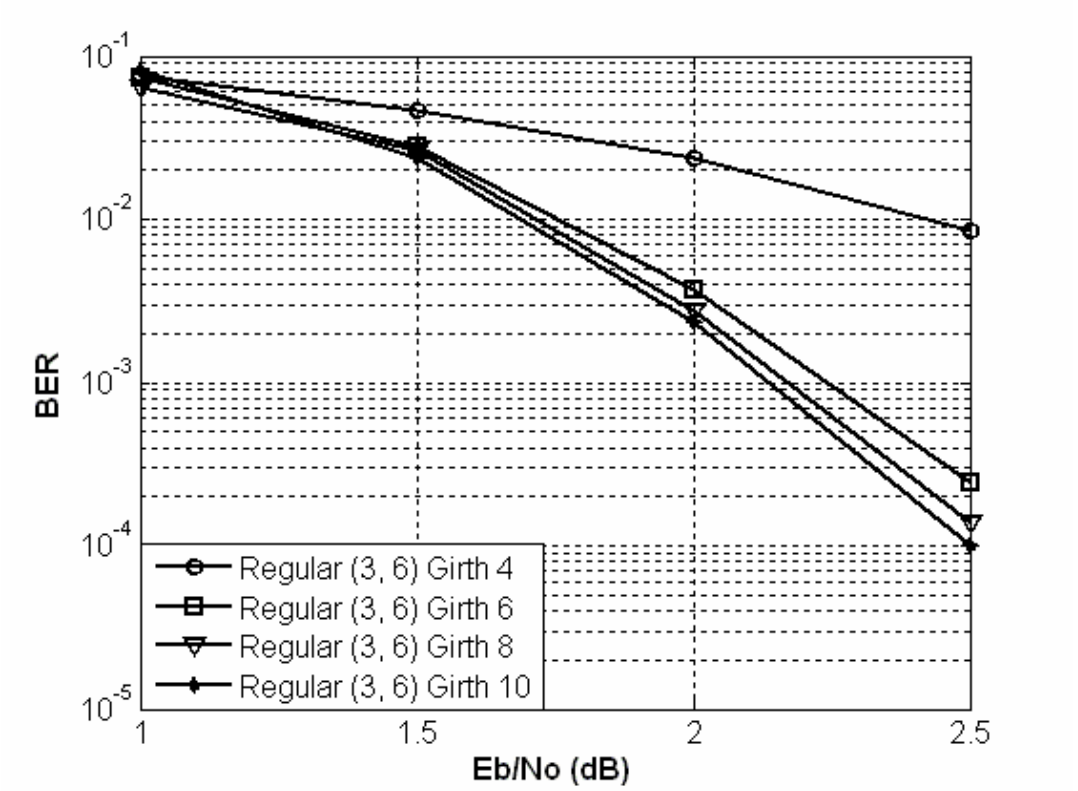


Figure 4.21 Performances of (900, 450) quasi-cyclic regular (3, 6) LDPC codes

4.3 Pseudo-Random versus Quasi-Cyclic LDPC Codes

We finally give a general comparison between quasi-cyclic and pseudo-random LDPC codes with girth 6, for both regular (3, 6) and irregular ("93", 7) cases in Figure 4.22. It is observed that performance of quasi-cyclic codes is approximately 0.1 dB superior to those of pseudo-random codes. However, this performance advantage over pseudo-random codes at short block lengths can not be expected to continue when the block length gets longer. On the contrary, quasi-cyclic codes of high block lengths are inferior [Tanner-2001], because, the main reason for the successful performance of LDPC codes, i.e., the "randomness", becomes more prominent for pseudo-random codes.

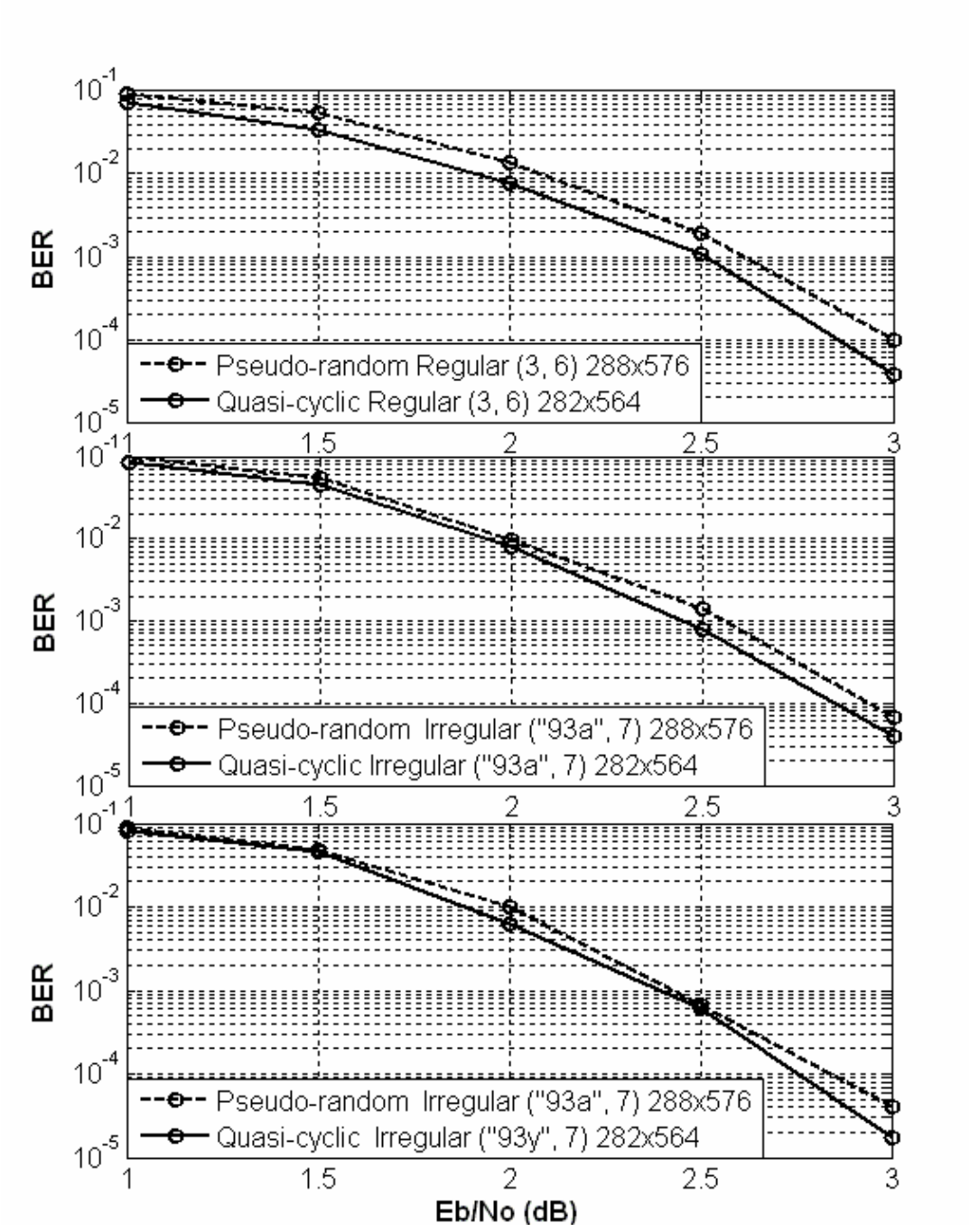


Figure 4.22 Performances of regular and irregular (576,288) pseudo-random and (564, 282) quasi-cyclic LDPC codes

In summary, although the algebraically structured quasi-cyclic LDPC codes considered in this chapter have low encoding complexity, when we sacrifice from their algebraic structure to obtain high girths, they become closer to pseudo-

random codes and a little gain in performance is obtained at the cost of increasing encoding complexity. The algebraic structure and the performance increase obtained by leaving this structure are the competing points about quasi-cyclic codes and either one must be chosen considering the requirements of the application.

CHAPTER 5

CONCLUSION

Randomly constructed LDPC codes are known to perform very good at extremely long block lengths. However, for many engineering applications, short block length is a must to decrease the complexity. In this thesis, we construct short block length pseudo-random and quasi-cyclic LDPC codes and discuss the construction criteria that affect the performance of these codes.

For both of the pseudo-random and quasi-cyclic cases, regular as well as irregular codes are generated and all the codes are decoded by the iterative belief propagation algorithm. We observe that the errors made by the belief propagation decoder are always the detected errors, which occur when the decoder reaches the maximum number of iterations and reports the fact that "it is not possible to find a valid codeword". The maximum number of iterations of the belief propagation decoder is set to 100 for all the codes used in this work, which have comparable block lengths.

We have constructed (576, 288) regular and irregular pseudo-random codes with girth values of 6 by our LGDC (*Local Girth Distribution Check*) algorithm. We have observed that regular LDPC codes are 0.1-0.2 dB inferior to irregular LDPC codes and construction is ~0.1 dB superior to ("93a", 7) irregular construction. Better performance of ("93y", 7) irregular structure is a result of its better elite variable node connections. Our results are similar to those obtained by MacKay, who generated (9972, 4986) regular and irregular LDPC codes [MacKay-1998].

We have also constructed (576, 288) irregular codes via our ACEC (*Approximate Cycle EMD Check*) algorithm. The criteria of ACE and stopping sets are then used together in the *Joint Approach* (JA) algorithm since the ACE alone may not be meaningful for high degree variable nodes of irregular codes. The performance increase obtained when the girth is changed from 4 to 6 is much more than that can be obtained by any other construction criterion. Hence, to avoid cycles of length four should be the main concern about the code generation algorithms.

As for the *algebraically structured quasi-cyclic LDPC* codes, we have proposed some irregular structures combining MacKay's pseudo-random irregular ("93", 7) forms with Tanner's algebraically structured quasi-cyclic construction, which avoid cycles of length four. The results are similar to those obtained for the pseudo-random case, i.e., the regular quasi-cyclic code has the worst and the irregular ("93y", 7) quasi-cyclic code has the best performance. So, we can say that the proposed method for quasi-cyclic irregular LDPC codes is advantageous at short block lengths.

Using the relations between cycles and shift values of the sub-matrices [Moura - 2005], (900, 450) regular (3, 6) quasi-cyclic codes with girth values of 4, 6, 8 and 10 are constructed. When we compare the generated codes, we can say that results are very similar to those obtained for pseudo-random codes.

Although, all the codes generated in this thesis work have short block lengths (576 to 900), iterations of the belief propagation algorithm take more than 30 hours while evaluating the BER performance of a single code, with MATLAB on 3 GHz Intel Pentium IV processor. Since we could not increase the block length of the codes for complexity reasons, all the improvements we obtain are small values like 0.1 or 0.2 dB. The optimization of the computer programs for generation and the performance evaluation of longer code length pseudo-random and quasi-cyclic codes can be a future work.

REFERENCES

- **[Brack-2007]** T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, N.E. L'Insalata, F. Rossi, M. Rovini, L. Fanucci, "Low Complexity LDPC Code Decoders for Next Generation Standards" ,Design, Automation and Test in Europe Conference and Exhibition, DATE '07.

- **[Choi-2005]** Eun-A Choi, Dae-Ik Chang, Deock-Gil Oh, Ji-Won Jung "Low Computational Complexity Algorithms of LDPC Decoder for DVB-S2 Systems" IEEE 2005.

- **[Fan-2000]** J. L. Fan, "Array codes as low density parity check codes," in Proc. 2nd Int. Symp. Turbo Codes and Related Topics, Brest, France, Sept. 2000, pp. 543–546.

- **[Fossosier-2000]** R. Lucas, M. P. C. Fossorier, Yu Kou, Shu Lin "Iterative Decoding of One-Step Majority Logic Decodable Codes Based on Belief Propagation" IEEE Trans. Comm. Theory, 2000.

- **[Fossosier-2004]** M. P. C. Fossorier, "Quasi-cyclic low density parity check codes from circulant permutation matrices," IEEE Trans. Inform. Theory, 2004.

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

- **[Fossosier-2004]** M. P. C. Fossorier, “Quasi-cyclic low density parity check codes from circulant permutation matrices,” IEEE Trans. Inform. Theory, 2004.

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

- **[Gallager-1963]** R. G. Gallager. Low density Parity check Codes. Cambridge, MA: MIT Press, 1963.

- **[Honary-2005]** B. Honary, A. Moinian and B. Ammar, “Construction of well structured quasi-cyclic low density parity check codes.” IEEE Proc. Commun. 2005.

- **[Kschischang-2001]** F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” IEEE Trans. Info. Theory 47: 498–519, 2001.

- **[Luby-2001]** M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. “Improved low density parity check codes using irregular graphs.” IEEE Trans. Info. Theory 47: 585-598. 2001

- **[MacKay-1998]** D. J. C. MacKay, S.T. Wilson and M.C. Davey, “Comparison of Constructions of Irregular Gallager Codes”, IEEE Trans. on Comm., 1998.

- **[MacKay-1999]** D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices", IEEE Trans. on Info. Theory, 1999.

- **[MacKay-2003]** D. J. C. MacKay **"**Information Theory, Inference, and Learning Algorithms" Cambridge University Pres, 2003.

- **[MacKay & Neal-1996]** D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes", Electronics Letters 1996.

- **[Mao-2000]** Y. Mao, A. Banihashemi, and M. Landolsi, "Comparison between low density parity check codes and turbo product codes for delay and complexity sensitive applications," in Proc. 20th Biennial Symp. Comm., Kingston, Ontario, 2000.

- **[Mao-2001]** Y. Mao and A. H. Banihashemi, "A heuristic search for good low density parity check codes at short block lengths," in Proc. IEEE Int. Conf. Communications, vol. 1, Helsinki, Finland, June 2001, pp. 41–44.

- **[Moura-2005]** J. Lu, and J. M. F. Moura, "Partition-and-Shift LDPC Codes." IEEE Trans. on Magnetics, 2005.

- **[Myung-2005]** Seho Myung, Kyeongcheol Yang, and Jaeyoel Kim "Quasi-Cyclic LDPC Codes for Fast Encoding"

- **[Ramamoorthy-2004]** A. Ramamoorthy and R. Wesel "Construction of Short Block Length Irregular Low density Parity check Code" (IEEE Comm. Conf. 2004)

- **[Richardson-2001a]** S. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. "On the Design of Low Density Parity Check Codes within 0.0045 dB of the Shannon Limit." IEEE Comm. Letters, 2:58-60, 2001.

- **[Richardson-2001b]** T. Richardson, A. Shokrollahi, and R. Urbanke. "Design of Capacity Approaching Irregular Low Density Parity Check Codes" IEEE Trans. Info. Theory 47: 619-637. 2001

- **[Richardson-2001c]** T. J. Richardson and R. L. Urbanke, "The capacity of low density parity check codes under message-passing decoding," IEEE Trans. On Info. Theory 47:599–618, 2001.

- **[Richardson-2002]** C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite-length analysis of low density parity check codes on the binary erasure channel," IEEE Trans. on Info. Theory 48: 1570–1579, 2002.

- **[Richardson-2003]** T. Richardson, "Error floors of LDPC codes," in Proc. 41st Annu. Allerton Conf. Communication, Control, and Computing, Monticello, IL, Sep. 2003, pp. 1426–1435.

- **[Rosenthal-2000]** J. Rosenthal and P. O. Vontobel, "Constructions of LDPC codes using Ramanujam graphs and ideas from Margulis," in Proc. 38th Allerton Conf. Communications, Control, and Computing, Monticello, IL, Oct. 2000, pp. 248–257.

- **[Shannon-1949]** C. E. Shannon and W. Weaver, "The Mathematical Theory of Communication" Univ. of Illinois Press. (1949)

- **[Tanner-1981]** B. M. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inform. Theory, 1981.

- **[Tanner-1999]** R. M. Tanner, "On quasi-cyclic repeat accumulate codes" in Proc. 37th Allerton Conf. Communication, Control and Computing, Monticello, IL, Oct. 1999, pp. 249–259.

- **[Tanner-2000]** R. M. Tanner, "A [155; 64; 20] sparse graph (LDPC) code," presented at the Recent Results Session at IEEE International Symposium on Information Theory, Sorrento, Italy, June 2000.

- **[Tanner-2001]** R. M. Tanner, A. Sridharan, and T. E. Fuja "A Class of Group-Structured LDPC Codes" Proc. International Symposium on Comm. Theory and Applications, Ambleside, U.K. 2001.

- **[Tanner-2004]** R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC Block and Convolutional Codes Based on Circulant Matrices" IEEE Trans. On Info. Theory, 2004.

- **[Tanner-2007]** J. Chen, R. M. Tanner, J. Zhang, M. P. C. Fossorier, "Construction of Irregular LDPC Codes by Quasi-Cyclic Extension." IEEE Trans. On Info. Theory, April 2007.

- **[Tian-2003]** Tao Tian, Chris Jones, John D. Villasenor, Richard D. Wesel "Construction of Irregular LDPC Codes with Low Error Floors" IEEE 2003.

- **[Tian-2004]** Tao Tian, Christopher R. Jones, John D. Villasenor, Richard D. Wesel "Selective Avoidance of Cycles in Irregular LDPC Code Construction." IEEE 2004.

- **[Zhang-2006]** Chao Zhang, Xiao-Lin Zhang, Cheng Lu, Zhan Zhang, "The Technical analysis on the China National Standard for Digital Terrestrial TV Broadcasting" 2006

APPENDIX A

REQUIRED NUMBER OF ITERATIONS FOR DECODING

The decoder halts whenever another codeword is found or when it reaches the maximum number of iterations. In many cases 20 iterations suffice, but sometimes even one hundred is not enough for the algorithm to converge to a codeword. Even by setting the maximum number of iterations to 1000 some blocks that are declared failures can be decoded by allowing more iterations. The number of successful decodings missed by limiting the number of iterations is an important issue. Decoding failures usually (nearly all in this thesis) occur because the decoding algorithm converges to a stable configuration in which several checks are failed. In such cases, extra iterations never lead to successful decoding. It is, however, possible for the algorithm to fail to converge to a stable state.

Figure 6.1 shows the distribution of iterations for a regular (576, 288) (3, 6) regular LDPC code with girth mean 6 at 2 dB SNR and Figure 6.2 shows the percentage of the iterations. Out of 1000 blocks, 794 blocks are decoded within 100 iterations, and there are 206 block decoding failures. 33 of the 206 decoding failures can be successfully decoded by increasing the iteration number to 500. This situation is nearly the same for the other codes in this thesis and 100 iterations is a good choice. Figure 6.3 shows the distribution of number of iterations for 1 to 3 dB SNR.

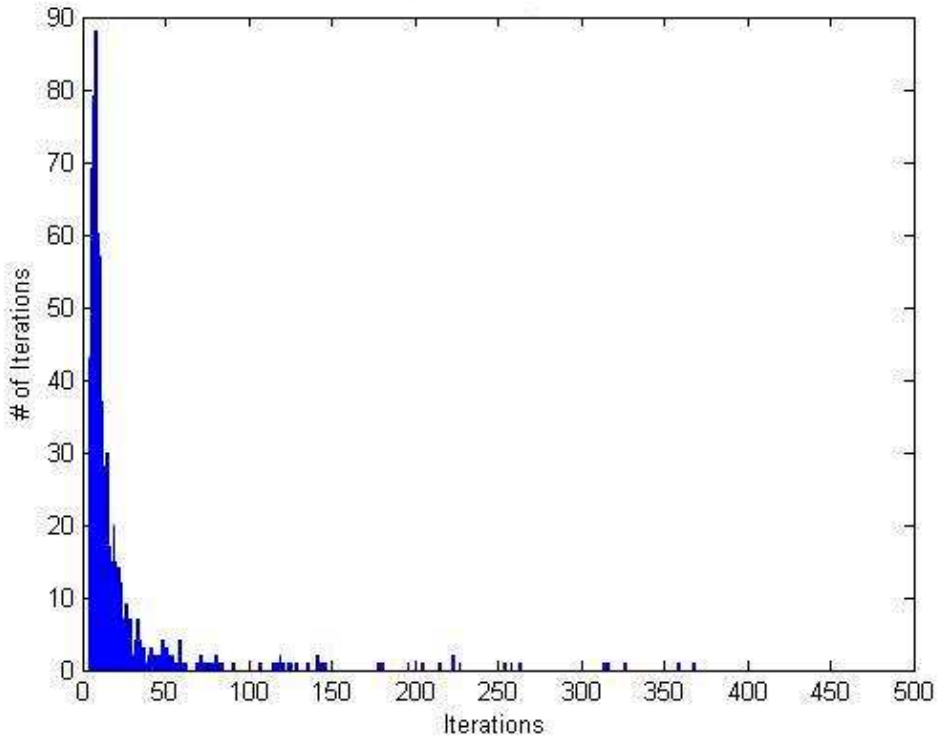


Figure A1 Iteration histogram for (576, 288) (3, 6) regular LDPC code with girth 6 for 2 dB SNR

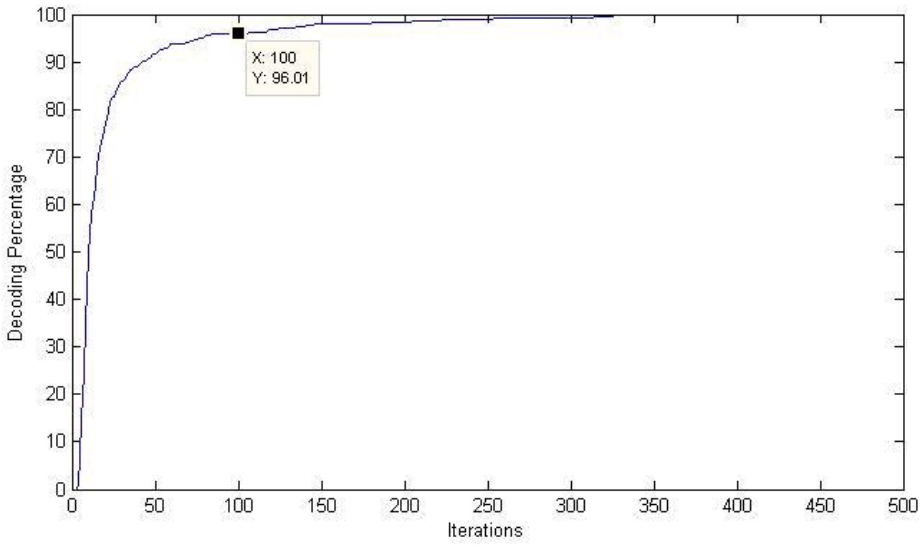


Figure A2 Iteration number convergence curve for 2 dB SNR

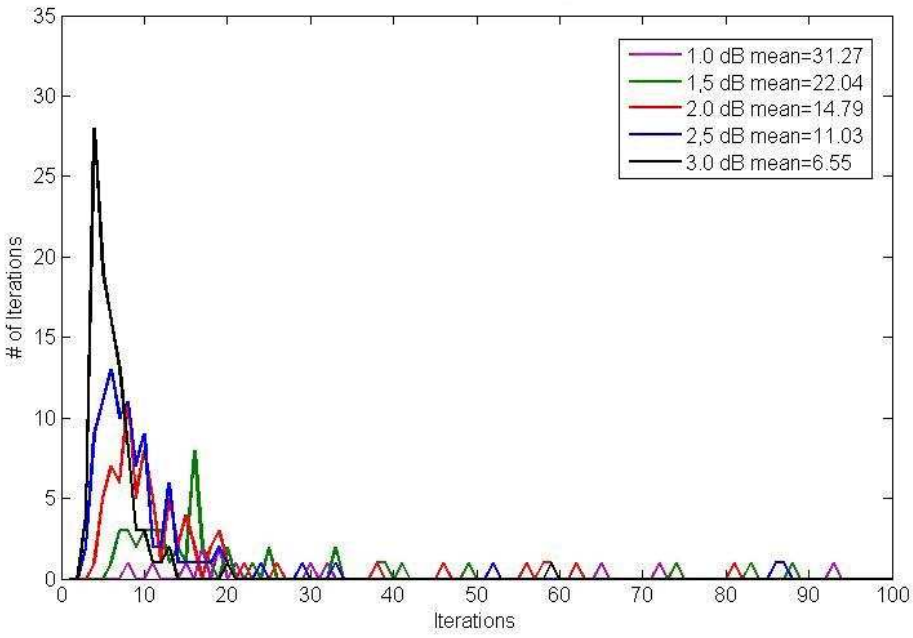


Figure A3 Iteration histograms for (576, 288) (3, 6) regular LDPC code with girth=6