DESIGN AND PERFORMANCE OF CAPACITY APPROACHING IRREGULAR LOW-DENSITY PARITY-CHECK CODES

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

BY

ERİNÇ DENİZ BARDAK

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

SEPTEMBER 2009

Approval of the thesis:

DESIGN AND PERFORMANCE OF CAPACITY APPROACHING IRREGULAR LOW-DENSITY PARITY-CHECK CODES

submitted by **ERİNÇ DENİZ BARDAK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Date: September 10, 2009

ABSTRACT

DESIGN AND PERFORMANCE OF CAPACITY APPROACHING IRREGULAR LOW-DENSITY PARITY-CHECK CODES

Bardak, Erinç Deniz
M. Sc., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Melek Diker Yücel

September 2009, 112 pages

In this thesis, design details of binary irregular Low-Density Parity-Check (LDPC) codes are investigated. We especially focus on the trade-off between the average variable node degree, w_a, and the number of length-6 cycles of an irregular code. We observe that the performance of the irregular code improves with increasing w_a up to a critical value, but deteriorates for larger w_a because of the exponential increase in the number of length-6 cycles. We have designed an irregular code of length 16,000 bits with average variable node degree w_a=3.8, that we call '2/3/13' since it has some variable nodes of degree 2 and 13 in addition to the majority of degree-3 nodes. The observed performance is found to be very close to that of the capacity approaching commercial codes. Time spent for decoding 50,000 codewords of length 1800 at E_b/N_o=1.6 dB for an irregular 2/3/13 code is measured to be 19% less than that of the regular (3, 6) code, mainly because of the smaller number of decoding failures.

Keywords: Irregular LDPC Codes, Length-6 Cycles.

ÖZ

KAPASİTEYE YAKLAŞAN DÜZENSİZ DÜŞÜK YOĞUNLUKLU EŞLİK SAĞLAMASI KODLARININ TASARIM VE PERFORMANSI

Bardak, Erinç Deniz
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Melek Diker Yücel
Eylül 2009, 112 sayfa

Bu tezde, ikili düzensiz Düşük-Yoğunluklu Eşlik-Sağlaması (DYES) kodlarının tasarım ayrıntıları incelenmektedir. Özellikle odaklandığımız konu, düzensiz ortalama değişken düğümü derecesi, w_a, ile 6 uzunluğundaki döngüler arasındaki ödünleşimdir. Bir kodun başarımının, w_a değeri kritik bir değere kadar arttırıldıkça düzeldiği fakat daha büyük w_a değerleri için 6 uzunluğundaki döngülerin sayısındaki üssel artış nedeniyle kötüye gittiği gözlenmektedir. Tasarladığımız 16,000 ikil uzunluğunda, değişken düğümlerinin çoğunluğunun derecesi 3, kalanı da 2 ve 13 olduğu için 2/3/13 diye adlandırdığımız, ortalama değişken düğüm derecesi w_a=3.8 olan düzensiz kodun başarımı, kapasiteye yaklaşan ticari kodlarınkine çok yakındır. E_b/N_o=1.6 dB değerinde 50,000 kod sözcüğünü çözümlediğimiz 1800 uzunluğundaki kodlardan, düzensiz 2/3/13 kod için gereken zamanın, görece az sayıdaki çözümleme hatası nedeniyle, düzenli (3,6) koda göre %19 daha kısa olduğu görülmüştür.

Anahtar Sözcükler: Düzensiz DYES Kodları, 6 Uzunluğundaki Döngüler.

To My Family and My Fiancée

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assoc. Prof. Dr. Melek Diker Yücel for her motivating ideas and valuable guidance.

I would also like to thank to Hande, my fiancée, for giving me all the inspiration I need and for being "the one" for me.

I also wish to thank my mother Sema and my sister Işıl for giving me the encouragement and all kinds of support during my whole work.

I would also like to thank to Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) for their contribution to my Master of Science education with the scholarship.

Finally, I would like to thank my company ASELSAN for the support during this thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

APP	A Posteriori Probability
AWGN	Additive White Gaussian Noise
BCH	Bose and Ray-Chaudhuri
BER	Bit Error Ratio
BPSK	Binary Phase-Shift Keying
DVB-S2	Second Generation Satellite Digital Video
GF	Galois Field
ITU-T	International Telecommunications Union-Telecommunications Standardization Sector
JPL	Jet Propulsion Laboratory
LDPC	Low Density Parity Check
LL	Log Likelihood
RS	Reed-Solomon
RU	Richardson-Urbanke
SNR	Signal to Noise Ratio
WIMAX	Worldwide Interoperability for Microwave Access
10GBase-T	10-Gigabit Ethernet over Twisted-Pair Cabling

CHAPTER 1

INTRODUCTION

In digital communication systems, the main goal is to achieve errorless communication between two points. When data is transmitted over an imperfect and noisy communication channel, there is some probability that the received message will not be identical to the transmitted message. To overcome the effect of noise and reduce the error probability at the receiver, one can improve the physical characteristics of the communication channel by using more reliable components or higher transmission power. One can also use *error control coding* to detect and correct the errors introduced by the channel.

In 1948, Shannon published his seminal paper **[Shannon-1948]** on the limits of reliable transmission of data over unreliable channels, which established the roots of information theory. Given a communication channel, Shannon proved that there exists a parameter, called the capacity of the channel, such that reliable transmission is possible for rates arbitrarily close to the capacity and not possible above it. The researchers, who try to achieve communication rates close to the channel capacity, discovered the first examples of error control codes, which in turn started the development of coding theory.

Coding theory is concerned with the design of powerful error control codes, and practical encoding and decoding systems. A powerful code is expected to detect and correct as many errors at the receiver side as possible. First known error control codes, which were capable of correcting single bit errors in each data block, were introduced to the literature by Richard W. Hamming in 1950 **[Hamming-1950]**. Afterwards, different codes such as the convolutional codes **[Elias-1955]**, and other block codes like the BCH codes **[Bose-Chaudhuri-1960]**, Reed-Solomon (RS) codes **[Reed-Solomon-1960]**, Kerdock codes **[Kerdock-1972]**, Goethals Codes **[Goethals-1974]**, and Goppa Codes **[Goppa-1982]** were found. In 1993, turbo codes, which were the first practical codes to closely approach the Shannon limit, were invented **[Berrou-Glavieux-Thitimajshima-1993]**.

All block codes mentioned above other than Kerdock codes and Goethals codes are linear. Low-Density Parity-Check (LDPC) codes, which we are interested in this work, are also linear block codes which were invented by Gallager in 1962 **[Gallager-1962]**.

1.1 Definition and History of LDPC Codes

LPDC codes are linear block codes which are defined by low density parity-check matrices. Let H be a binary $(n-k)\times n$ matrix with $(n-k)$ linearly independent rows. A linear block code C is defined as the set of vectors $c=(c_1,\dots,c_n)$ such that $H\,c^T=0$. The matrix H is called a parity-check matrix for the code. The code C defined by the parity-check matrix H is said to be an LDPC code if H is *sparse* **[Gallager-1962]**, i.e., has small number of nonzero elements. The sparsity of the parity-check matrix is the key property that allows algorithmic efficiency of LDPC codes.

Tanner graphs (or bipartite graphs), which are proposed to the literature by Michael Tanner in **[Tanner-1981]**, are used to visualize the parity-check matrices of LDPC codes. In a Tanner graph, each column of the parity-check matrix is called a *variable node* and each row is called a *check node*. The variable nodes are connected to the check nodes with edges, which are drawn according to the positions of the nonzero elements in the parity check matrix.

There are mainly two kinds of LDPC codes; the regular and irregular ones. A *regular* LDPC code has a parity-check matrix *H*, in which every column has the same weight

w_v and every row has the same weight w_c. On the other hand, the columns and rows of the parity-check matrix of an *irregular* LDPC code do not have uniform weight distribution. In the literature, irregular LDPC codes have been shown to outperform the regular ones.

The performance of an LDPC code is affected also by another important parameter, called the *cycle*. In a Tanner graph, a *cycle* is defined as the series of connected edges that starts from and ends at the same variable node. The *length of a cycle* is defined as the number of edges that it contains. Small cycles, such as length-4 and length-6, deteriorate the performance of a code.

The most widely used algorithms for decoding LDPC codes are *message passing algorithms*, which are also known as *iterative algorithms*. These algorithms are called *iterative* since messages are passed from variable nodes to check nodes, and from check nodes to variable nodes at each round of the algorithm. The sparsity property of LDPC codes lowers the complexity of the operations done in each iteration and makes the iterative algorithms suitable for decoding of LDPC codes.

LDPC codes were invented by Robert Gallager in 1962 **[Gallager-1962]**. Due to the requirement of high complexity computations, LDPC codes had been ignored for a long time. Also at that time Reed-Solomon and convolutional codes were considered to be perfectly suitable for error control coding, which was another reason for LDPC codes to be neglected.

36 years after from its invention, the studies done by MacKay and Neil attracted the attention of the communication society on LDPC codes again. In 1996, MacKay and Neil **[MacKay-Neal-1996]** showed that optimally decoded LDPC codes can reach information rates within 1 dB to the Shannon limit. MacKay and Neil's codes were regular LDPC codes. In 1998, Luby et al. **[Luby-Mitzenmacher-Shokrollahi-Spielman-1998]** proposed irregular LDPC code structures whose performances are better than the regular ones. In 2001, an analytical way of designing irregular LDPC

codes, called *density evolution*, was developed by Richardson et al. **[Richardson-Shokrollahi-Urbanke-2001]** to construct irregular LDPC codes which outperform the regular ones and even turbo codes. Again in 2001, the best known LDPC code was proposed by Chung et al. **[Chung-Forney-Shokrollahi-Urbanke-2001]**. The code they proposed was an irregular LDPC code of rate ½, codeword length 10^7, and of performance only 0.0045 dB away from the Shannon limit.

All codes mentioned above are binary codes. There also exist non-binary LDPC codes introduced to the literature **[Davey-MacKay-1998]**, **[Davey-1999]**. Figure 1.1**,** which is given in **[MacKay-2005]**, is a very good visualization to compare the binary and non-binary regular and irregular LDPC codes with each other and also with other outstanding error correcting codes such as turbo codes.

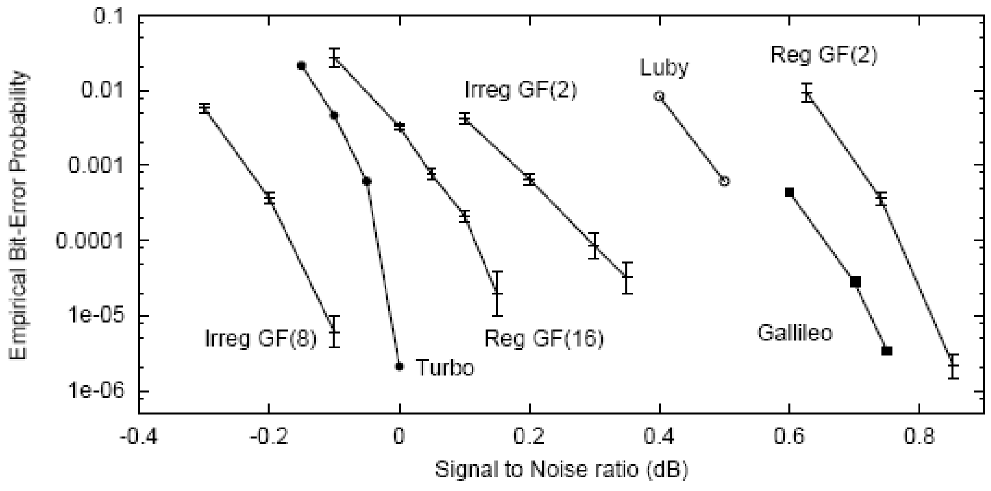


Figure 1.1 Comparison of regular binary Gallager codes with irregular codes, codes over GF(q), and other outstanding codes of rate 1/4.

In Figure 1.1, Irreg GF(8) is an irregular LDPC code over GF(8),with codeword length of 48,000 bits given in **[Davey-1999]**; Turbo is a JPL turbo code with codeword length of 65,536 bits given in **[MacKay-2005]**; Reg GF(16) is a regular LDPC code over GF(16) with codeword length of 24,448 bits given in **[Davey-**

MacKay-1998]; Irreg GF(2) is an irregular binary LDPC code with codeword length of 16,000 bits given in **[Davey-1999]**; Luby is an irregular binary LDPC code with codeword length of 64,000 bits given in **[Luby-Mitzenmacher-Shokrollahi-Spielman-1998]**; Galileo is a JPL turbo code used in the space-craft Galileo, given in **[Swanson-1988]**; Reg GF(2) is a regular binary LDPC code with codeword length of 40,000 bits given in **[MacKay-1999]**.

Today, LDPC codes became one of the most important error correcting codes used in several areas of communication. LDPC codes are the main codes used in very important standards such as:

- IEEE WIMAX (Worldwide Inter-operability for Microwave Access) 802.16e Standard
- Digital Video Broadcasting – Satellite - Second Generation (DVB-S2)
- 10GBase-T Ethernet Standard
- ITU-T G.hn/G.9960 Standard for networking over power lines, phone lines and coaxial cable
- China National Standard for Digital Terrestrial TV Broadcasting standard

1.2 Aim and Organization of the Thesis

The aim of this thesis is the investigation of the performance of irregular LDPC codes with different variable node degree distribution polynomials. We design regular and irregular codes and analyze the circumstances under which the performance of the codes improve or deteriorate. Specifically, we study the effects of the *average variable node degree* and *the number of length-6 cycles* parameters on the performance of the codes, and try to reveal the *trade-off* between these two parameters.

In Chapter 2, we review the literature on Low-Density Parity-Check (LDPC) codes after discussing the preliminaries of linear block codes and LDPC codes. We discuss the regular and irregular code construction methods that are used in our work. Finally, we explain the decoding algorithms that we employ to decode the LDPC codes that we generate and use in the simulations.

In Chapter 3, we investigate the performances of the regular codes with variable node degree w_v=3 and the irregular codes generated by MacKay and Neil's methods. After verifying the correctness of the software that we develop for simulations, we find the distribution of block errors in terms of wrongly decoded codewords and decoding failures of the belief propagation decoding algorithm. We examine the conditions that lead to wrong codewords and comment on the choice of the maximum number of iterations to be used. We then study the performances of many irregular LDPC codes that have different variable node degree distribution polynomials. Properly designing these polynomials, we generate irregular codes with desired properties. Using the generated codes, we investigate the effects of the average variable node degree and the number of length-6 cycles on the performance. Then, we design an irregular code of length 16,000 bits and compare its performance with a capacity approaching commercial code which is used in DVB-S2 standard. We also measure the decoding times needed for some irregular and regular codes to understand the effect of average variable node degree on the decoding times.

In Chapter 4, we summarize our work and give suggestions for further studies.

CHAPTER 2

LOW DENSITY PARITY CHECK CODES

In this chapter, we review the literature on Low-Density Parity-Check (LDPC) codes after discussing the preliminaries on linear block codes. Section 2.1 covers the overview of linear block codes and LDPC codes. In Section 2.2, we give the Regular Code Construction, Irregular 2A Code Construction and Irregular Pseudo-Random Code Construction methods of MacKay and Neil that we have used in his work. In Section 2.3, we discuss the encoding methods of LDPC codes. Finally, in Section 2.4, we give the details of the log-likelihood and the likelihood decoding algorithms utilized in our simulations.

2.1 Overview of Linear Block Codes and LDPC Codes

LDPC codes are linear block codes that have parity-check matrices in which the number of the nonzero elements is much less than the number of zero's. Before giving detailed information about LDPC codes, it will be better to briefly discuss the main properties of linear block codes.

An (n, k) *block code* is a rule of converting a sequence of source symbols of length k into a sequence of n, where $n > k$. The linear (n, k) code over GF(q) is a subspace C of the vector space GF$(q)^n$. The elements of C are n-dimensional vectors, called the codewords. Let the source message $\overline{m} = (m_0, m_1, ..., m_{k-1})$ be an arbitrary vector in GF$(q)^k$. By the linear transformation

$$\overline{c} = \overline{m}\,G = \sum_{i=0}^{k-1} m_i\, g_i = m_0\, g_0 + m_1\, g_1 + ... + m_{k-1}\, g_{k-1}, \tag{2.1}$$

one can generate all q^k codewords $\bar{c} = (c_0, c_1, ..., c_{n-1})$ in *C*, provided that the $k \times n$ matrix *G* is of rank *k*. Then, *G* is called the *generator matrix* of the code; because it has *k* linearly independent row vectors $\overline{g_i}$ spanning the subspace *C*.

$$G = \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \cdots \\ \bar{g}_{k-1} \end{bmatrix} \tag{2.2}$$

An (*n, k*) *linear block code* has the $(n-k) \times n$ *parity-check* matrix *H*, whose rows are orthogonal to the rows of *G*, hence

$$G \times H^T = 0_{k \times (n-k)}. \tag{2.3}$$

Therefore, the codeword $\bar{c} = (c_0, c_1, ..., c_{n-1})$ generated by (2.1) satisfies

$$\bar{c} \times H^T = \bar{0}. \tag{2.4}$$

The generator matrix of a linear block code in systematic form can be expressed as

$$G = \begin{bmatrix} \bar{g}_0 \\ \bar{g}_1 \\ \cdots \\ \bar{g}_{k-1} \end{bmatrix} = [I_k \; P], \tag{2.5}$$

where I_k is the $k \times k$ identity matrix and *P* is a $k \times (n-k)$ matrix. The corresponding parity-check matrix in systematic form can be found using (2.3) as

$$H = [P^T \; I_{n-k}]. \tag{2.6}$$

An important parameter of a block code is its rate *k*/*n*, that is the number of information symbols divided by the number of codeword symbols. Consider a parity-check matrix, H of an LDPC code whose size is $m \times n$. If there are m parity check symbols, k=n−m; so the code rate can also be expressed as

$$Rate = \frac{k}{n} = \frac{n-m}{n} = 1 - \frac{m}{n}. \tag{2.7}$$

Rate of a code plays an important role in its error correction performance. When the rate is increased, the fraction, m/n, of the parity-check symbols is decreased in a codeword. In this case, more information is sent with less number of parity-check symbols. This may sound good since the speed of information transmission increases. However, the error correction capability of the code is obviously hurt since less number of parity-check equations exist to be used to correct the erroneous symbols. This explains the trade-off between information density and the error correction capability of the code. Therefore, the rate of a code is crucial and should be chosen according to the characteristics of the communication channel.

After this brief review of linear block codes, we can now give some information about low-density parity-check codes. A *low-density parity-check* (LDPC) code is a linear block code that has a parity-check matrix, H, every row and column of which is `sparse' **[Gallager-1962]**. As emphasized by the word ‘sparse’, an LDPC code contains very small number of nonzero elements in the parity-check matrix H as compared to its size.

There are mainly two kinds of LDPC codes; the regular and irregular ones. A regular LDPC code has a parity-check matrix H, in which every column of H has the same weight w_v and every row has the same weight w_c. On the other hand, the columns and rows of the parity-check matrix of an irregular LDPC code do not have uniform weight distribution.

Tanner graphs (or bipartite graphs), which are proposed to the literature by Michael Tanner in **[Tanner-1981]**, are used to visualize the parity-check matrices of LDPC codes. In a Tanner graph, each column of the parity-check matrix is called a variable node and each row is called a check node. The variable nodes are connected to the check nodes with edges, which are drawn according to the positions of the nonzero elements in the parity check matrix. The parity check matrix of a linear block code and its corresponding Tanner graph is illustrated in Figure 2.1:

$$\mathrm{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

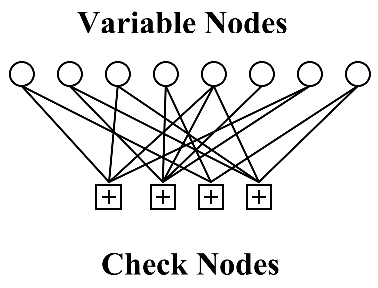


Figure 2.1 The parity-check matrix of a (8, 4) code and the corresponding Tanner graph

Each edge in a Tanner graph corresponds to a nonzero entry in the parity-check matrix *H*. Therefore, in terms of edges, the weight of a column (row) is the number of edges emanating from the corresponding variable (check) node. The number of nonzero elements in a column (row) is said to be the *degree* of that variable (check) node.

One way to express the weight distributions of variable and check nodes is to use *degree distribution polynomials*. The variable node degree distribution polynomial of an LDPC code is of the form

$$\lambda(x) = \sum_{i=1}^{d_v} \lambda_i x^{i-1} \tag{2.8}$$

and the check node degree distribution polynomial is of the form

$$\rho(x) = \sum_{j=1}^{d_c} \rho_j x^{i-1}. \tag{2.9}$$

In these equations λ_i is the fraction of edges emanating from the variable nodes of degree *i*, and ρ_j is the fraction of edges emanating from the check nodes of degree *j*. In other words, λ_i is the number of variable nodes of degree *i* divided by the total number of the variable nodes, and ρ_j is the number of check nodes of degree *j* divided by the total number of the check nodes. d_v and d_c are the maximum variable and check node degrees, respectively. Since λ_is and ρ_js denote the fraction of variable and check node degrees, they must sum up to one. Hence,

$$\begin{aligned} \sum_{j=1}^{d_v} \lambda_j &= 1 \\ \sum_{j=1}^{d_c} \rho_j &= 1. \end{aligned} \tag{2.10}$$

The main properties of an LDPC code can be understood by looking at its degree distribution polynomials. For an irregular LDPC code, the degrees of variable and check nodes may differ from each other. In this case there exists more than one coefficient λ_i (or ρ_j) in the variable (or check node) degree distribution polynomials of the code. The variable and check node degree distribution polynomials of the irregular LDPC code in Figure 2.1 are

$$\lambda(x) = \frac{1}{8}x^0 + \frac{6}{8}x + \frac{1}{3}x^2, \; \rho(x) = \frac{1}{4}x^2 + \frac{2}{4}x^3 + \frac{1}{4}x^4. \tag{2.11}$$

As can be seen, the given LDPC code has an irregular degree structure for both of its variable and check nodes: $1/8$ of the variable nodes have degree 1, $6/8$ of them have degree 2, and $1/8$ of them have degree 3. The check node degree distribution of this code is also irregular: $1/4$ of the check nodes have degree 3, $2/4$ of them have degree 4, and $1/4$ them have degree 5.

A regular LDPC code has all variable node degrees equal to some constant w_v, and all check node degrees equal to some constant w_c. Therefore, there exist only one coefficient λ_{w_v} in the variable node degree distribution polynomial, and one coefficient ρ_{w_c} in the check node degree distribution equation of the code. Considering (2.2) it is not difficult to see that both λ_{w_v} and ρ_{w_c} are equal to 1. As an example, the degree distribution polynomials of a regular LDPC code with $w_v = 3$ and $w_c = 6$ are

$$\lambda(x) = x^2, \ \rho(x) = x^5. \tag{2.12}$$

One can now define the rate of an LDPC code in terms of the coefficients λ_i and ρ_j. Consider an LDPC code with variable node degree distribution polynomial $\lambda(x) = \sum_{i=1}^{d_v} \lambda_i x^{i-1}$, and check node degree distribution polynomial $\rho(x) = \sum_{j=1}^{d_c} \rho_j x^{i-1}$. Let E be the total number of edges in the Tanner graph of this code. Then the number of variable nodes which have degree i can be expressed as $\frac{E\lambda_i}{i}$. Hence, the total number of variable nodes is $\sum_{i=1}^{d_v} \frac{E\lambda_i}{i}$. Similarly the total number of check nodes is $\sum_{j=1}^{dc} \frac{E\rho_j}{j}$. Therefore we can rewrite the rate of this code in terms of the coefficients λ_i and ρ_j as follows:

$$Rate = 1-\frac{m}{n} = 1-\frac{\sum_{i=1}^{d_v}\frac{E\lambda_i}{i}}{\sum_{j=1}^{dc}\frac{E\rho_j}{j}} = 1-\frac{\sum_{i=1}^{d_v}\frac{\lambda_i}{i}}{\sum_{j=1}^{dc}\frac{\rho_j}{j}} \tag{2.13}$$

We have mentioned earlier in this chapter that the rate of an LDPC code is crucial in the error correcting capability of the code. The performance of an LDPC code is affected also by another important parameter, called the *cycle*. In a Tanner graph, a *cycle* is defined as the series of connected edges that starts from and ends at the same variable node. The *length of a cycle* is defined as the number of edges that it contains. As an example, consider the parity-check matrix *H*, and its Tanner graph given in Figure 2.2. *H* has one cycle of length 4 created by its bold entries and the bold edges in the Tanner graph are the ones that form the cycle.

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

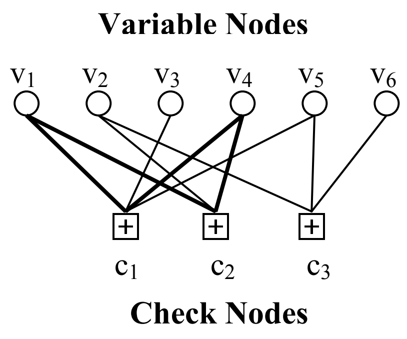


Figure 2.2 A sample parity-check matrix for a (6, 3) code and its Tanner graph. The cycle of length-4 is shown as bold entries in the matrix and bold edges in the graph

In a Tanner graph there may be several cycles of different length. Obviously, the minimum length that a cycle can have is four. Especially the length-4 cycles, deteriorate the decoding performance of LDPC codes; therefore, to avoid cycles of length-4 is one of the most important things to take into account in the construction of an LDPC code. Cycles with lengths greater than four also decrease the performance of LDPC codes; however, their effect is not as significant as the length-4 ones.

The smallest cycle length in the Tanner graph is called the *girth* of the code. LDPC codes with larger *girth* values have been shown to result in better error correcting performance.

2.2 MacKay & Neil Construction Methods for LDPC Codes

There are a great number of different LDPC code construction methods in the literature such as quasi-cyclic construction **[Tanner-2004]**, **[Moura-2005]**, pseudo-random construction **[Moinian-2006]**, **[Bonello-2008]**, combinatorial approach **[Krishnan-2007]**, **[Johnson-2008]**, and finite geometry techniques **[Kou-2000]**, **[Aly-2008]**. In this section we will explain MacKay & Neil's techniques that we have used for constructing regular and irregular LDPC codes.

2.2.1 Regular LDPC Code Construction

In this work, we have used the MacKay and Neal's Method for regular LDPC code construction **[MacKay-Neal-1996]**. In this algorithm, construction of an $m \times n$ parity-check matrix starts with forming the leftmost column of the matrix. At first step, desired number of w_v 1's are placed randomly in the first column. After that, the remaining columns are formed one-by-one from left to right. In the construction of these remaining columns, two things are taken into account. Any column which will be added should not have more than one overlap between any of the present columns in order to avoid length-4 cycles. The second thing to be considered when adding a new column is that, the positions of the 1's of the column should be selected from the rows with weights smaller than the desired row weight w_c. By this construction method, the parity check matrix of an $m \times n$ regular LDPC code that is free from length-4 cycles can be obtained.

We have implemented a software which is capable of generating regular parity-check matrices of any length and any rate using McKay and Neil's method. In this work, all the regular matrices we have used are constructed using this software. The software we implemented for regular matrix generation is explained in Appendix A.

2.2.2 Irregular LDPC Code Construction by 2A Method

The 2A method of constructing irregular LDPC codes was introduced to the literature by MacKay and Neal **[MacKay-Neal-1996]**. The method has the following rules for constructing the parity-check matrix:

- In the parity-check matrix of the code whose size is $m \times n$, up to $m/2$ of the columns are designated 'weight-2 columns', and these are constructed such that there is zero overlap between any pair of columns.
- The remaining columns are made at random with weight-3, with the weight per row as uniform as possible, and overlap between any two columns of the entire parity-check matrix no greater than 1.

As can be understood from the above rules for construction, the parity-check matrix generated by this method has some of its variable nodes with degree 2, and some of them with degree 3. Therefore, the variable node degree distribution of such a matrix can be written in generic form which is given in equation (2.14).

$$\begin{gathered} \lambda(\mathrm{x}) = \lambda_2 \mathrm{x} + \lambda_3 \mathrm{x}^2 \\ \lambda_2 + \lambda_3 = 1 \end{gathered} \tag{2.14}$$

The irregular codes generated by this method look very similar to regular codes. The only difference of these irregular codes is that they have some of their columns with weight-2 instead of weight-3. In the related paper of MacKay **[MacKay-1999]**, it is said that the weight-2 columns are introduced to the parity-check matrix because they “guessed” that these columns may lead the code to a better performance than the regular ones.

The part of the matrix with weight-2 columns has the important property that there exist zero overlap between any pair of the columns. This property is quite essential to avoid cycles of any length which may be caused by the weight-2 columns. For a moment let us ignore this property and see what may happen. Let us consider the below figure which is a part of the bipartite graph of an LDPC code which is constructed randomly:

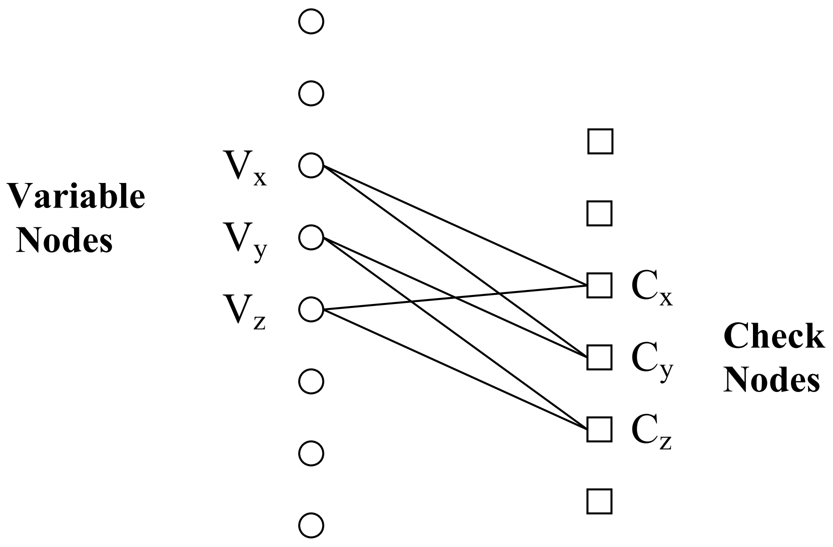


Figure 2.3 A part of the bi-partite graph of an irregular LDPC code which is constructed randomly

In Figure 2.3, it can be easily seen that the edges emanating from the variable nodes v_x, v_y and v_z result in a cycle of length-6 which distorts the performance of the code. However, if the actual proposed property of the 2A method was preserved, there would be no chance to have any cycles generated by the degree-2 variable nodes.

The performance of the irregular codes generated by the 2A method will be investigated in detail in Section 3.3.

2.2.3 Irregular LDPC Code Construction by Pseudo-Random Method

This method works similar to the regular matrix construction method. However, in this case all the columns do not have the same weight. The degrees of the variable nodes are defined by the degree distribution polynomial of the code and there is more than one degree value that a variable node can have.

In this method, as in the regular case, the columns of the irregular matrix are constructed from left to right. The weights of the columns and number of columns with each column weight are defined by the variable node degree distribution polynomial. Starting from the leftmost column, first the columns with the smallest weight are constructed, then the columns with next greater degree are constructed and this process continues until all columns with each weight are constructed. All of the columns are constructed such that the overlap between any two columns of the matrix is not greater than one in order to avoid length-4 cycles.

In this work we have generated many different irregular parity-check matrices with different lengths and different variable node degree distributions using these methods. The performances of the irregular LDPC codes are analyzed in detail and compared with the regular ones in the following chapters.

2.3 Encoding of LDPC Codes

Consider an LDPC code defined by the parity-check matrix *H*. As we have discussed in Section 2.1, a codeword *c* of this code is generated by $c = \overline{m}G$, where $\overline{m}$ is the source message and *G* is the generator matrix. In this codeword generation process, which is called encoding, the main point is to have low encoding complexity. The parity-check matrix *H* is a sparse matrix. However, the generator matrix *G*, is not a sparse matrix; hence, the encoding time by $c = \overline{m}G$ is proportional to n^2, where n is the codeword length.

To reduce the encoding complexity, Richardson and Urbanke proposed a method where the parity-check matrix H is directly used to encode codewords **[Richardson-Urbanke-2001]**. In this method, which is called the RU algorithm, the parity-check matrix is transformed into an approximate lower-triangular form, by performing basic row and column operations only. The approximate lower-triangular form for an $m \times n$ parity-check matrix is shown in Figure 2.4.

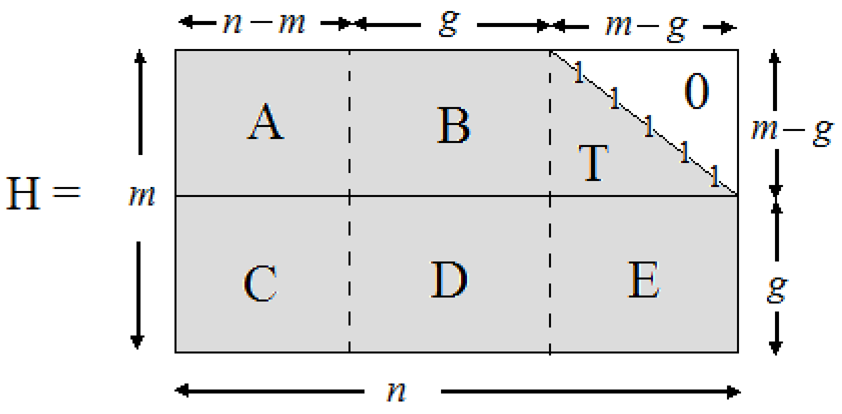


Figure 2.4 Example of a parity-check matrix in approximate lower triangular form

Suppose that m is a vector of message block. According to the RU algorithm, the codeword after decoding is $c = (m, p_1, p_2)$ where p_1 and p_2 are the parity parts. It is shown that $p_1 = -\phi^{-1}(-ET^{-1}A + C)s^T$ and $p_2{}^T = -T^{-1}(As^T + Bp_1{}^T)$, where $\phi = -ET^{-1}B + D$. The encoding complexity of the algorithm is then shown to be proportional to $n + g^2$. Therefore, when n >> g, the encoding complexity is proportional to n.

The RU method works on the given parity-check matrices. In the literature, there are other methods that use the idea of spending effort on the construction the LDPC codes in order to have low encoding complexities. In **[Mackay-Wilson-Davey-1999]** and **[Xia-He-Xu-Cai-2008],** design methods of LDPC codes are given such that the encoding complexities of the codes are proportional to *n*.

2.4 Decoding of LDPC Codes by Belief Propagation Algorithm

The most widely used algorithms for decoding LDPC codes are *message passing algorithms* which are also known as *iterative algorithms*. These algorithms are called *iterative* since messages are passed from variable nodes to check nodes, and from check nodes to variable nodes at each round of the algorithm. The messages from message nodes to check nodes are computed based on the observed value of the message node and some of the messages passed from the neighbouring check nodes to that message node **[Shokrollahi-2003]**. In this section we will investigate the most commonly used decoding algorithm called the *belief propagation algorithm*.

Belief propagation algorithm is a message passing algorithm, in which the messages passed between variable and check nodes at each round of the algorithm are random variables. In this algorithm, the calculations of these random variables are done separately assuming that they are *statistically independent*. This assumption would be true for a code, which contains no cycles of any length. Almost every LDPC code contains cycles. However, this algorithm works quite well for decoding LDPC codes whose cycles are long enough **[Shokrollahi-2003]**.

In our study, we have implemented the *log-likelihood belief propagation decoding* and the *likelihood belief propagation decoding*. These two techniques are quite similar to each other as explained below.

Likelihood Decoding

In likelihood decoding, the messages passed between variable and check nodes are the likelihood values of these bits. The algorithm is composed of 4 steps.

First Step: Initialization

At the first step of decoding, the only messages to be sent are the messages from variable nodes to check nodes which are calculated using the observed values of these bits. This observed information for each bit is used to calculate the likelihood ratio of each bit.

$$Q_i = P_{i,initial} = \frac{1-p_i}{p_i} \tag{2.15}$$

In (2.15), p_i is the probability that the bit c_i of the received codeword $\bar{c} = [c_1\, c_2 \ldots c_n]$ is 1. Q_i is the initial value for the message that is sent from the i^{th} variable node to its related check nodes. In an algorithmic manner, we will call the message sent from i^{th} variable node to the j^{th} variable node $L_{i,j} = p_i$, and the initial value of $L_{i,j} = Q_i$.

Second Step: Check Node Response

In this step, check nodes calculate their response messages $R_{j,i}(0)$ and $R_{j,i}(1)$ to be sent to the variable nodes.

$$\begin{aligned} R_{j,i} &= P(c_i = 0 | y_j) \\ R_{j,i} &= P(c_i = 1 | y_j) \end{aligned} \tag{2.16}$$

Here, $P(c_i = 0 | y_j)$ and $P(c_i = 1 | y_j)$ are the probabilities that the bit c_i is 0 or 1 given the conditional event y_j that the j^{th} parity-check equation is satisfied.

To be able to compute $P(c_i = 0 | y_j)$ and $P(c_i = 1 | y_j)$, we will use an expression which is proved by Gallager in his work **[Gallager-1963]**. Gallager showed that the probability $P(c_i = 0 | y_j)$ that the j^{th} parity-check equation is satisfied if the bit c_i is equal to 0 can be expressed as

$$P(c_i = 0 | y_j) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2p_{i'}), \tag{2.17}$$

where B_j is the set of bits included in the j^{th} parity-check equation and p_i is the probability that the bit c_i is equal to 1.

Then, the probability $P(c_i = 1 | y_j)$ that the j^{th} parity-check equation is satisfied if the bit c_i is equal to 1 is

$$P(c_i = 1 | y_j) = 1 - P(c_i = 0 | y_j) = \frac{1}{2} - \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2p_{i'}). \tag{2.18}$$

Therefore, $R_{j,i}(0)$ and $R_{j,i}(1)$ can be calculated as the following.

$$\begin{aligned} R_{j,i}(0) &= \frac{1}{2} + \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2p_{i'}) \\ R_{j,i}(1) &= \frac{1}{2} - \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2p_{i'}) \end{aligned} \tag{2.19}$$

where B_j is the set of bits included in the j^{th} parity-check equation.

The $R_{j,i}$ values are then sent from each check node to the variable nodes that are connected to it, and the algorithm continues with the third step.

Third Step: Codeword Test

This step is the decision step of the algorithm. The check node responses together with the initially observed information are used to decide whether the bit c_i is a 1 or 0. For each bit i the following calculations are done.

$$D_i(0) = k_{i,j}(1 - p_i)\prod_{j' \in A_i} R_{j',i}(0) \tag{2.20}$$

In the equations, the set A_i is the set of check nodes that are connected to the variable node i, and the constants $k_{i,j}$ are chosen such that $D_i(0) + D_i(1) = 0$. These $D_i(0)$ and $D_i(1)$ values are then used in the inequalities given below to decide whether the bit c_i is a 1 or 0. Let the decided sequence be $\overline{m} = [m_0 \;\; m_1 \ldots m_{n-1}]$.

$$m_i = \begin{cases} 1, & D_i(0) \le D_i(1) \\ 0, & else \end{cases} \tag{2.21}$$

If the vector $\overline{m}$ is a valid codeword, that is to say $\overline{m}H^T = 0$, the algorithm successfully terminates here, and outputs $\overline{m}$ as the decoded codeword. If $\overline{m}$ is not a valid codeword, the algorithm continues with the fourth step.

Fourth Step: Variable Node Response

In this step, the first thing to be done is to increase the iteration number by one and check if the maximum number of iterations, say $I_{\max}$, has been reached. If $I_{\max}$ has been reached, the algorithm terminates with failure and outputs the last decided value of $\overline{m}$.

If $I_{\max}$ has not been reached yet, variable node responses are calculated using equation (2.22), and these responses are sent form each variable node to the related check nodes.

$$\begin{aligned} L_{i,j}(0) &= l_{i,j}(1-p_i) \prod_{j' \in A_i, j' \neq j} R_{j',i}(0) \\ L_{i,j}(1) &= l_{i,j} p_i \prod_{j' \in A_i, j' \neq j} R_{j',i}(1) \end{aligned} \tag{2.22}$$

such that the constants $l_{i,j}$ are chosen to satisfy $L_{i,j}(0) + L_{i,j}(1) = 1$.

After the responses are sent to the variable nodes, the algorithm continues with the second step.

Log-likelihood Decoding

This type of decoding is similar to the likelihood decoding. However, in this case, the messages sent between variable nodes and check nodes are the log-likelihood ratios, not the likelihood values, of these bits. The steps of log-likelihood decoding are same as the likelihood decoding. In this section, we will mention the similarities and differences of log-likelihood decoding compared to likelihood decoding and modify the equations of likelihood decoding in order to suit the log-likelihood decoding.

First Step: Initialization

At this step, the messages to be sent from variable nodes to check nodes are calculated using the following equation.

$$Q_i = LLR(P_{i,initial}) = \ln\left(\frac{1-p_i}{p_i}\right) \tag{2.23}$$

Similar to the likelihood decoding, we will call the message sent from i^{th} variable node to the j^{th} variable node $L_{i,j} = \ln\left(\frac{1-p_i}{pi}\right)$, and the initial value of $L_{i,j} = Q_i$.

Second Step: Check Node Response

In this step, the check node responses are calculated using the following equation.

$$R_{j,i} = \ln\left(\frac{P\left(c_i = 0 \middle| y_j\right)}{P\left(c_i = 1 \middle| y_j\right)}\right) \tag{2.24}$$

Similar to likelihood decoding, $P\left(c_i = 0 \middle| y_j\right)$ and $P\left(c_i = 1 \middle| y_j\right)$ are computed using (2.17) and (2.18). Therefore, $R_{j,i}$ can be expressed as

$$R_{j,i} = \ln\left(\frac{P(c_i = 0 | y_j)}{P_{ij}(c_i = 1 | y_j)}\right) = \ln\left(\frac{\frac{1}{2} + \frac{1}{2}\prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'})}{\frac{1}{2} - \frac{1}{2}\prod_{i' \in B_j, i' \neq i}(1 - 2p_{i'})}\right). \tag{2.25}$$

At this point the following identity of tangent hyperbolic and natural logarithm functions will be used.

$$\tanh\left(\frac{1}{2}\ln\left(\frac{1-x}{x}\right)\right) = \frac{e^{\frac{1}{2}\ln\left(\frac{1-x}{x}\right)} - e^{-\frac{1}{2}\ln\left(\frac{1-x}{x}\right)}}{e^{\frac{1}{2}\ln\left(\frac{1-x}{x}\right)} + e^{\frac{1}{2}\ln\left(\frac{1-x}{x}\right)}} = \frac{\sqrt{\frac{1-x}{x}} - \sqrt{\frac{x}{1-x}}}{\sqrt{\frac{1-x}{x}} + \sqrt{\frac{x}{1-x}}} = 1 - 2x \tag{2.26}$$

Also note that

$$L_{i,j} = \ln\left(\frac{1-p_i}{p_i}\right). \tag{2.27}$$

Then, $R_{j,i}$ can be calculated as

$$R_{j,i} = \ln\left(\frac{1 + \prod_{i' \in B_j, i' \neq i} \tanh\left(\frac{\tanh\left(L_{i',j}/2\right)}{2}\right)}{1 - \prod_{i' \in B_j, i' \neq i} \tanh\left(\frac{\tanh\left(L_{i',j}/2\right)}{2}\right)}\right). \tag{2.28}$$

Third Step: Codeword Test

Similar to likelihood decoding, this step is the decision step. In this step, firstly the following calculation is done.

$$L_i = Q_i + \sum_{j \in A_i} R_{j,i} \tag{2.29}$$

After this calculation, the decision for each bit is done using the following equation.

$$m_i = \begin{cases} 1, & L_i \leq 0 \\ 0, & L_i > 0 \end{cases} \tag{2.30}$$

If the decided vector $\overline{m}$ is a valid codeword, the algorithm stops here. Else, the algorithm continues with the fourth step.

Fourth Step: Variable Node Response

In this step, similar to likelihood decoding, iteration number is increased and checked whether it has reached the value I_{max}. If so, the algorithm terminates with failure and the last decided codeword is outputted. If not, the variable node responses are calculated using the following equation.

$$L_{i,j} = Q_i + \sum_{j' \in A_i, j' \neq j} R_{j',i} \tag{2.31}$$

The responses of the variable nodes calculated using (2.31) are then sent to the check nodes and the algorithm continues with the second step.

The steps of both likelihood decoding and log-likelihood decoding algorithms are illustrated in Figure 2.5.

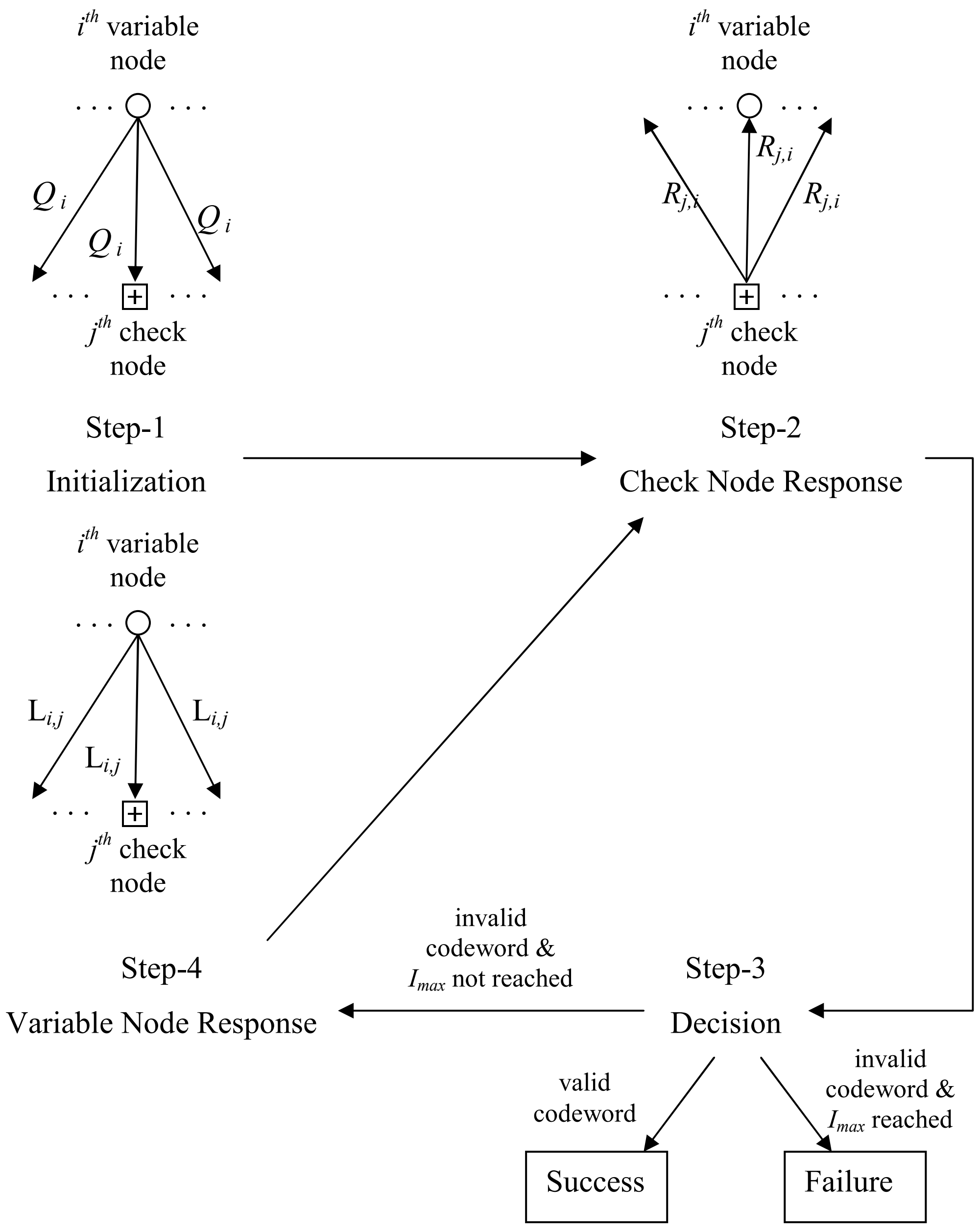


Figure 2.5 Illustration of likelihood and log-likelihood decoding algorithms

CHAPTER 3

SIMULATION RESULTS

In this chapter, we mainly investigate the performance of regular and irregular LDPC codes. Irregular LDPC codes with different variable node degree distribution polynomials are generated and their bit error ratio (BER) versus input bit energy divided by noise spectral density of the channel (E_b/N_0) performance is compared with that of the regular codes.

Section 3.1 is intended as a preliminary related to the understanding of the fundamental concepts about LDPC codes and to the control of our implementation. In Section 3.2, we examine two different types of errors made by the decoder; failures and wrong decisions. We show that decoding failures are much more likely, i.e., the decoder algorithm reaches to the end of iterations without deciding on any codeword but still correcting some erroneous bits. The frequency of wrong decisions, i.e., decisions on a wrong codeword different from the sent one is extremely small and approaches to 0 for sufficiently long codes.

In the remaining sections, irregular LDPC codes are constructed using either of the MacKay's construction methods discussed in Section 2.2. The codes generated by the 2A method are examined in Section 3.3 for various values of the average variable node degree. Two different types of irregularities, having variable node degrees of either 2/3/4 or 1/3/5 generated by pseudo-random construction, are investigated in Sections 3.4 and 3.5, respectively. The effects of these kinds of irregularities on the performance of irregular LDPC codes are discussed.

In Sections 3.6 and 3.7, we consider codes with variable node degrees 2/3/*i*, where the highest degree *i* is chosen from the set {9, 11, 13, 19}. We have specifically studied the *trade-off between the average variable node degree and the number of length-6 cycles* in detail.

In Section 3.8, we design a 2/3/13 irregular code of length 16,000 bits and compare its performance with a commercially used LDPC code of length 64,800 bits. Then, in Section 3.9, we analyze the decoding times of some regular and irregular codes. We investigate the effect of the average variable node degree on the decoding time of the codes.

Section 3.10 deals with the check node degree distributions of the generated LDPC codes, which occur randomly as a result of the construction algorithm. Section 3.11 is a summary of the work done.

The properties of the codes, the communication channel, and the decoder that are used in the simulations are as described below.

- **Code Properties:** In each simulation, we use a randomly generated regular or irregular LDPC code of specific length ($576 \leq n \leq 1800$), and of rate ½ according to one of the construction methods described in Section 2.2.
- **Communication Channel Properties:** We use additive white Gaussian noise (AWGN) channel with Binary Phase Shift Keying (BPSK) modulation in our simulations. Each bit of a codeword to be sent is first modulated with BPSK, then a random noise sample of given power is added.
- **Decoder Properties:** We use the log-likelihood decoding algorithm with the maximum number of iterations, $I_{\max}$, chosen as 50. So, if the decoder cannot decide on a valid codeword, it yields the vector that it arrives at the 50th iteration as the decoder output.

In the simulations, for each E_b/N_0 value, we count the block errors, almost all of which are shown in Section 3.2 to occur as a result of decoding failures. The simulation is stopped when 20 block errors are counted. Then we calculate the Bit

Error Ratio (BER) value for this level of E_b/N_0, and present the results in BER versus E_b/N_0 value curves.

We have used MATLAB as software development tool. In our work, we have done many simulations all of which take long time. In order to shorten the time needed to complete the simulations, we have worked on optimizing our decoder software and also run our simulations in more than 4 computers in parallel. In Section 3.1, the performances of two different decoders, namely log-likelihood and likelihood decoders are compared. The time consumption of the log-likelihood decoder that we worked on optimizing its software is less than that of the likelihood decoder. Therefore, we may say that our optimization wok on the decoder software has been useful.

3.1 Experimental Preliminaries and Software Control

In this section, we investigate the preliminaries related to the understanding of the fundamental concepts about LDPC codes and control the correctness of our implementation. Firstly, we compare the performance of two different decoders, *log-likelihood and likelihood decoder*. Then, we review the effect of the codeword length of an LDPC code on its performance.

3.1.1 Decoder Comparison

In Section 2.4, we have described two types of decoding algorithms called likelihood and log-likelihood decoding. Although the performance of the two algorithms is expected to be the same, in **[Uzunoğlu-2007]** the simulation results of these two decoders were found to be not exactly the same. This is why we start by comparing the performance of these decoders.

We have done decoding simulations of different length codes using both of the decoders and obtained the graphs shown in Figure 3.1 for n=576 and in Figure 3.2 for n=896. It is seen that both of the decoders yield exactly the same results, which is not surprising. The only difference that we have observed is the time consumption of the two decoding softwares.

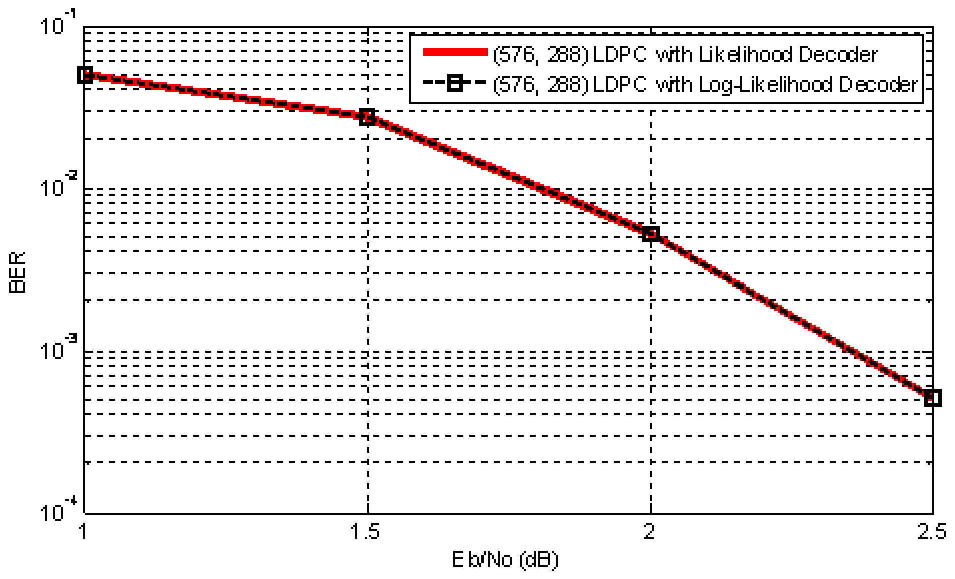


Figure 3.1 Performance comparison of log-likelihood and likelihood decoders for a regular (576, 288) low-density parity-check code

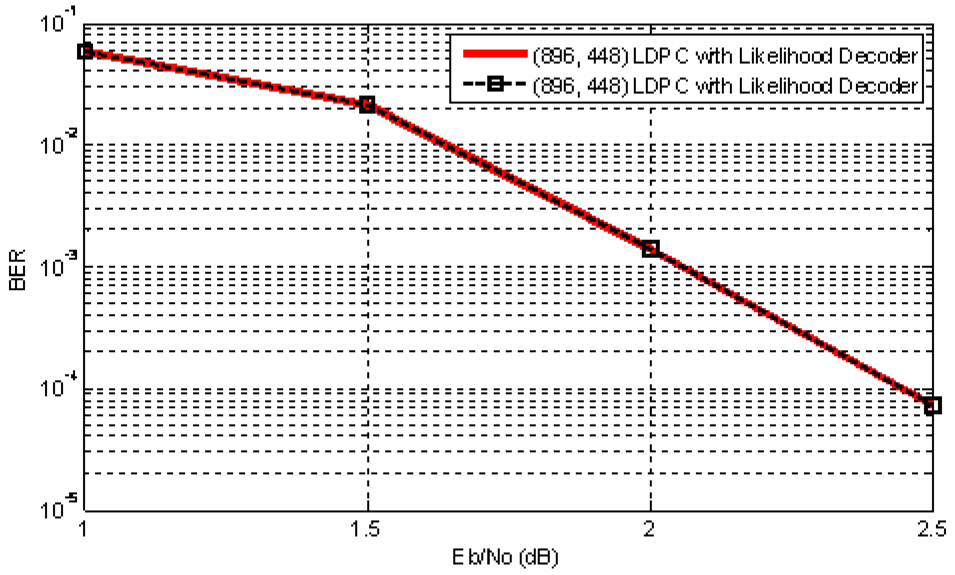


Figure 3.2 Performance comparison of log-likelihood and likelihood decoders for a regular (896, 448) low-density parity-check code

The time spent for the simulations of (576, 288) and (896,448) codes with log-likelihood and likelihood decoders are given in Table 3.1.

Table 3.1 Time consumptions of log-likelihood and likelihood decoders

LDPC Code	Time Spent with log-likelihood decoder	Time Spent with likelihood decoder
(576, 288)	402.297 seconds	460.703 seconds
(896, 448)	2933.02 seconds	3452.86 seconds

As can be seen from Table 3.1, our software for the likelihood decoder spends approximately 15% more time than the log-likelihood decoder. This is an optimization issue to be studied on the implementation of the decoder software. We did not work on this optimization since both decoders give out the same results. Instead, we decided to use the log-likelihood decoder in the rest of our simulations.

In **[Uzunoğlu-2007]** the simulation results of these two decoders were not exactly the same, the log-likelihood decoder consistently performing slightly worse than the likelihood decoder. One possible reason for this erroneous result may be the cumulative effect of the rounding errors in the software written for the log-likelihood decoder. A related graph taken from **[Uzunoğlu-2007]** is given in Figure 3.3, where the abbreviation of “LL Decoders” stands for the log-likelihood decoder, and “APP Decoder” stands for the likelihood decoder.

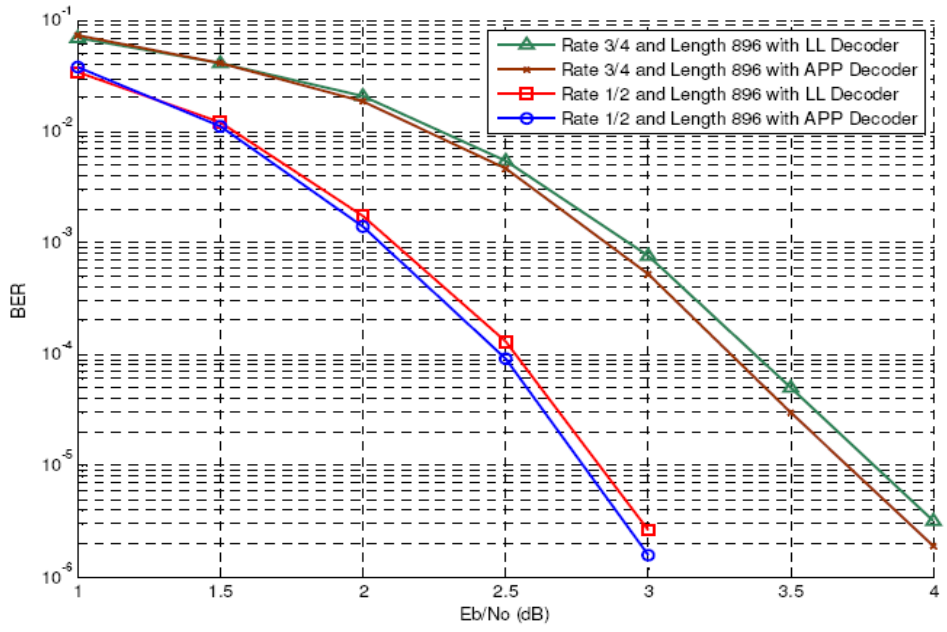


Figure 3.3 Performance comparison of log-likelihood and likelihood decoders for regular (896, 448) and (896, 224) codes taken from **[Uzunoğlu-2007]**

3.1.2 Effect of Codeword Length on the BER Performance

In the literature of LDPC codes, it is a well-known fact that the performances of LDPC codes with long codeword lengths are better than those of the shorter length codes. In this section, we observe the amount of performance improvement brought by an increase in the length from 576 to 2700 bits, for rate ½ codes.

In the simulations, we randomly generate regular (3, 6) LDPC codes, i.e., the variable node degree w_v= 3 and the check node degree w_c=6. A sample graph containing the BER performances of rate ½ and regular (3, 6) LDPC codes with different codeword lengths is given in Figure 3.4.

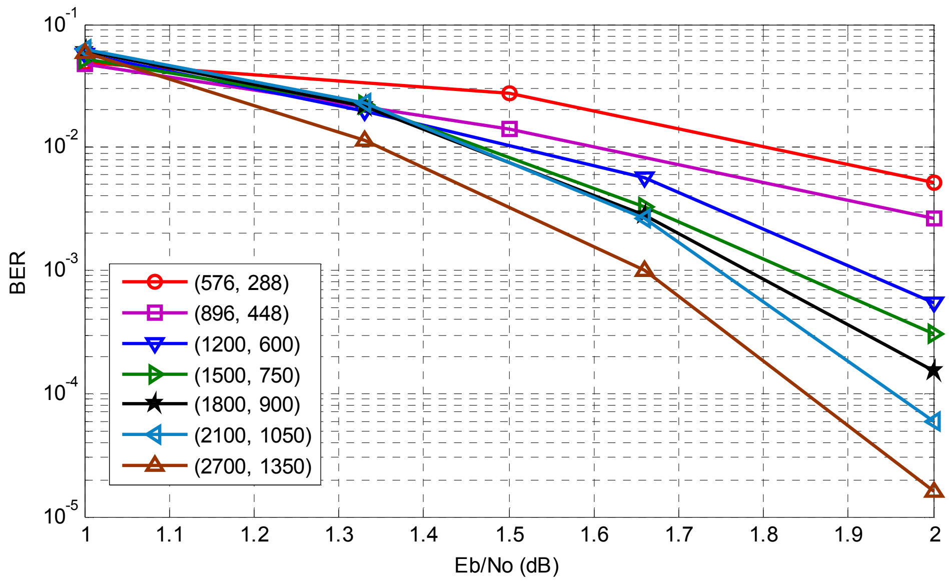


Figure 3.4 Performance comparison of (576, 288), (896, 448), (1200, 600), (2100, 1050), (1800, 900) and (2700, 1350) regular (3, 6) codes

When the graph is examined, it is observed that if the worst performing (576, 288) code is compared with others at the BER of 10^{-2}, (2700, 1350) code is 0.45 dB better, and (1200, 600) code is 0.3 dB better. The performance gain of (2700, 1350) code over (1200, 600) code is around 0.25 dB at the BER of 10^{-3}. It seems that doubling the codeword length results in a performance gain like 0.2-0.3 dB for BER's around 10^{-2} or 10^{-3}.

3.2 Wrong Codewords at the Decoder Output

In this section, we analyze the codewords at the output of the decoder in terms of their relation between the input codewords. We first give the definitions of different situations that we may face at the decoder output and then we investigate the conditions which lead to these different situations.

When a received sequence enters our decoder, we may have three different cases at the output of the decoder. These cases are:

1) Success:

The situation that we call "Success" occurs when the output of the decoder is exactly the same as the initially sent codeword. This means that our decoder has successfully decoded the noisy input and obtained the transmitted codeword without any error.

2) Failure:

The situation that we call "Failure" occurs when the decoder is not able to find a valid codeword for the related parity check matrix. In this case the maximum number of iterations for the decoding algorithm is reached and the output has erroneous bits as compared to the initially sent codeword.

3) Wrong Codeword:

The situation that we call "Wrong Codeword" occurs when the output of the decoder is a valid codeword for the related parity check matrix, but it is different from the transmitted codeword. In this case the decoding process ends up with an error vector equal to the difference between the transmitted and wrongly decoded codewords.

We have made simulations on various codes with different code lengths to investigate when we get wrong codewords at the output of the decoder. We have used codes with codeword lengths of 50, 100, 200, 400, 600, 800, and 1000 bits. All of the codes are regular with rate ½, variable node degree 3 and check node degree 6; i.e., regular (3, 6) codes. Random noise samples for each level of SNR in the set {0.5, 1, 1.5, 2, 2.5, 3} dB are calculated and added to the sent codeword. The maximum number of iterations $I_{\max}$ is set to 50, so that the log-likelihood decoder either finds a codeword (either correct or wrong) in less than 50 iterations or it fails at the 50th iteration and yields an erroneous word. We have sent a thousand codewords for each SNR level. The simulation is stopped either when 20 codeword errors for each SNR level is reached or all the thousand codewords are sent.

The error distribution results of three simulations, each one using a different seed for generating the random noise samples, are given in Table 3.2 where “S” (success) refers to the number of codewords decoded successfully, “F” (failure) refers to the number of decoding failures (which output encoded words with bit errors), and “W” (wrong codeword) refers to the number of valid codewords, which are different from the sent codeword.

Table 3.2 Distribution of 20 block errors (between decoding failures and wrong codewords) for rate 1/2 codes of different lengths, when the maximum number of iterations is set to 50.

Codeword Length	**50**			**100**			**200**			**400**			**600**			**800**			**1000**		
	1st Simulation																				
SNR	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W
0.5 dB	9	18	2	8	19	1	8	20	0	1	20	0	1	20	0	0	20	0	0	20	0
1 dB	18	20	0	21	19	1	14	20	0	11	20	0	6	20	0	12	20	0	10	20	0
1.5 dB	42	19	1	39	19	1	52	20	0	36	20	0	39	20	0	45	20	0	66	20	0
2 dB	123	19	1	88	19	1	79	20	0	255	20	0	393	20	0	981	19	0	984	16	0
2.5 dB	156	18	2	278	17	3	301	19	1	991	9	0	995	5	0	1000	0	0	1000	0	0
3 dB	195	20	0	627	18	2	991	9	0	999	1	0	999	1	0	1000	0	0	1000	0	0
	2nd Simulation																				
SNR	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W
0.5 dB	18	20	0	19	20	0	8	20	0	3	20	0	3	20	0	0	20	0	0	20	0
1 dB	15	17	3	12	19	1	7	20	0	10	20	0	15	20	0	14	20	0	16	20	0
1.5 dB	21	20	0	33	18	2	46	20	0	31	20	0	34	20	0	69	20	0	83	20	0
2 dB	106	18	2	79	20	0	162	20	0	150	20	0	420	20	0	649	20	0	984	16	0
2.5 dB	195	18	2	323	18	2	282	20	0	986	14	0	996	4	0	998	2	0	1000	0	0
3 dB	132	19	1	654	19	1	984	16	0	1000	0	0	1000	0	0	998	1	0	1000	0	0
	3rd Simulation																				
SNR	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W	S	F	W
0.5 dB	16	20	0	11	19	1	5	20	0	2	20	0	1	20	0	1	20	0	0	20	0
1 dB	17	20	0	17	20	0	14	20	0	12	20	0	5	20	0	22	20	0	3	20	0
1.5 dB	46	20	0	63	18	2	29	20	0	59	20	0	54	20	0	50	20	0	74	20	0
2 dB	124	19	1	130	20	0	94	20	0	191	20	0	241	20	0	544	20	0	983	17	0
2.5 dB	226	16	4	263	20	0	301	20	0	984	16	0	997	3	0	999	1	0	999	1	0
3 dB	170	20	0	590	20	0	987	13	0	998	2	0	1000	0	0	1000	0	0	1000	0	0

When Table 3.2 is investigated, it is seen that for codeword lengths of 50 and 100 bits, the decoder may output wrong codewords. However, for codeword lengths greater than 200 bits, we never face any wrong codewords; i.e., all of the bit errors occur as a result of decoding failures. Our simulations show that when the codeword length is large enough and the maximum number of iterations is chosen suitably, there is no wrong codeword at the decoder output. In LDPC coding, codeword lengths are chosen very long for excellent performance, they are practically on the order of thousands of bits; so all the bit errors are the result of decoding failures arrived at the pre-specified maximum number of iterations.

In Table 3.2, the wrong codewords appear at the decoder output only for block lengths shorter than 200. We suspect that this may be the result of the maximum allowed number of iterations, $I_{max} = 50$, which is relatively high as compared to the codeword length. In order to investigate the effect of this parameter on the number of wrong codewords, we have made a similar simulation setting the maximum number of iterations to 20, for codeword lengths of 50, 100 and 200. (see Table 3.3)

Table 3.3 Distribution of 20 block errors (between decoding failures and wrong codewords), when maximum number of iterations is set to 20.

Codeword Length	**50**			**100**			**200**		
	1st Simulation								
SNR	S	F	W	S	F	W	S	F	W
0.5 dB	13	20	0	5	20	0	2	20	0
1 dB	20	19	1	14	20	0	10	20	0
1.5 dB	45	17	3	27	18	2	40	20	0
2 dB	110	19	1	127	20	0	98	20	0
2.5 dB	135	18	2	127	20	0	241	20	0
3 dB	173	16	4	559	20	0	993	7	0
	2nd Simulation								
SNR	S	F	W	S	F	W	S	F	W
0.5 dB	15	18	2	4	20	0	1	20	0
1 dB	39	20	0	13	20	0	9	20	0
1.5 dB	37	20	0	30	19	1	42	20	0
2 dB	53	19	1	120	19	0	72	20	0
2.5 dB	190	19	1	165	20	0	381	20	0
3 dB	407	18	2	488	18	2	724	20	0

Table 3.3 (cont'd)

Codeword Length	50			100			200		
	3rd Simulation								
SNR	S	F	W	S	F	W	S	F	W
0.5 dB	18	19	1	3	20	0	6	20	0
1 dB	18	20	0	16	19	1	15	20	0
1.5 dB	37	17	3	42	20	0	24	20	0
2 dB	91	20	0	86	19	1	63	20	0
2.5 dB	175	17	3	155	20	0	315	20	0
3 dB	423	19	1	423	18	2	986	14	0

Table 3.3 shows that the number of wrong codewords obtained at the output of the decoder considerably reduces while $I_{\max}$ is decreased from 50 to 20. This is also not a surprising result since decoding failures are more likely to occur when the algorithm is allowed to perform less number of iterations. On the other hand, setting $I_{\max}$ equal to 20 seems to be large enough for successful decoding of the transmitted codewords for the codes of length shorter than 200.

After these observations, we wanted to compare the erroneously decoded vectors, i.e., failures and wrong codewords, with the initial received vectors. In other words, when a received vector is decoded to be a failure or wrong codeword at the decoder output, we compared the bit errors of the received words before decoding with those of the words after decoding.

We have used a rate ½ regular (3, 6) code with codeword length 50. Again, random noise samples for each level of SNR in the set {0.5, 1, 1.5, 2, 2.5, 3} dB is added to the sent codewords. This time, instead of 20 block errors, we stopped our simulations when 200 block errors are counted for each SNR level. Table 3.4 shows the total number of erroneous bits of the actual received words and that of the words which are decided as failure or wrong decision at the decoder output.

Table 3.4 The number of erroneous bits of blocks, which are decided as failure or wrong decision at the decoder, before and after they are decoded.

	Failures=193		**Wrong Codewords=7**	
	Before Decoding	After Decoding	Before Decoding	After Decoding
0.5 dB	1604	1334	68	72
	Failures=182		**Wrong Codewords=18**	
	Before Decoding	After Decoding	Before Decoding	After Decoding
1 dB	1511	1239	133	127
	Failures=183		**Wrong Codewords=17**	
	Before Decoding	After Decoding	Before Decoding	After Decoding
1.5 dB	1452	1170	123	125
	Failures=172		**Wrong Codewords=28**	
	Before Decoding	After Decoding	Before Decoding	After Decoding
2 dB	1299	1052	185	193
	Failures=168		**Wrong Codewords=32**	
	Before Decoding	After Decoding	Before Decoding	After Decoding
2.5 dB	1207	1037	207	221
	Failures=166		**Wrong Codewords=34**	
	Before Decoding	After Decoding	Before Decoding	After Decoding
3 dB	1146	1084	196	242

When Table 3.4 is inspected, it can be seen that the number of bit errors introduced by the communication channel is always reduced whenever there is a decoding failure. However, for the case of wrong codewords, the number of erroneous bits is either very close to the initial value or more than that.

In the last simulations, we collected all the wrong codewords and seen that the number of 1's of the codeword with minimum weight is 5. Considering that all-zero word is a codeword for all block codes, one can say that the minimum distance $d_{\min}$ of this code is smaller than or equal to 5. The smallness of $d_{\min}=5$ value with respect to 20 iterations is the main reason to have more wrong codewords at the decoder output, as compared to higher length codes that have higher $d_{\min}$ values.

To sum up all the observations of this section, while decoding in 50 iterations, we have detected wrongly decoded codewords at the decoder output only for codeword length shorter than 200 bits. If the maximum number of iterations is reduced to 20, wrongly decoded codewords disappear for length-200 codes and can only be seen for codes of length smaller than 100 bits. So, as a handy rule of thumb, we can say that *wrong codewords do not occur if the maximum number of iterations parameter of the decoding algorithm is less than one tenth of the block length*. Useful LDPC codes are chosen very long for excellent performance, which guarantees not having any wrongly decoded codeword at the decoder output. So, practically all decoding errors come from decoding failures.

Moreover, whenever a decoding failure occurs, i.e., a legitimate codeword cannot be arrived at the decoder output, the number of bit errors seems to be reduced slightly at the end of the maximum number of iterations of the message passing decoding algorithm.

3.3 Irregular 2A Codes

In Section 2.2.2, we have described the 2A method of MacKay and Neil. In this section, we investigate the BER performances of the LDPC codes with variable node degrees varying between 2.75 and 3 are found by simulations. The LDPC codes we use in this section are constructed using our software for constructing irregular 2A codes, which is explained in Appendix A.

Using our software, we have generated different length irregular LDPC codes. For each codeword length, five different irregular matrices with different average variable node degree values are constructed. The performances of these irregular codes and a regular code with the same length are compared.

The variable node degree distributions of the irregular 2A matrices that we have constructed for each codeword length are given in Table 3.5.

Table 3.5 The variable node degree distributions of the irregular 2A matrices that we have constructed for codeword lengths of 576 and 896

Matrix Number	**Average Variable Node Degree** w_a	**Variable Node Degree Distribution**
1	2.75	$\lambda_1(x) = 0.25x + 0.75x^2$
2	2.80	$\lambda_2(x) = 0.2x + 0.8x^2$
3	2.85	$\lambda_3(x) = 0.15x + 0.85x^2$
4	2.90	$\lambda_4(x) = 0.1x + 0.9x^2$
5	2.95	$\lambda_5(x) = 0.05x + 0.95x^2$

The reason of constructing five different irregular matrices for each codeword length is to see the effect of *the number of weight-2 columns* in the parity-check matrix. By this way, one can observe the performance of different irregular 2A matrices with different average variable degrees.

In the simulations, we have first used parity check matrices of size 288×576; i.e., the codeword length is 576 and the code rate is ½ . The simulation results for the five (n, k)=(576, 288) irregular codes with different average variable node degrees, Av{w_v}= w_a, close to 3; along with the performance of a (576, 288) regular (3, 6) (i.e., (w_v, w_c)=(3, 6)) code are given in Figure 3.5.

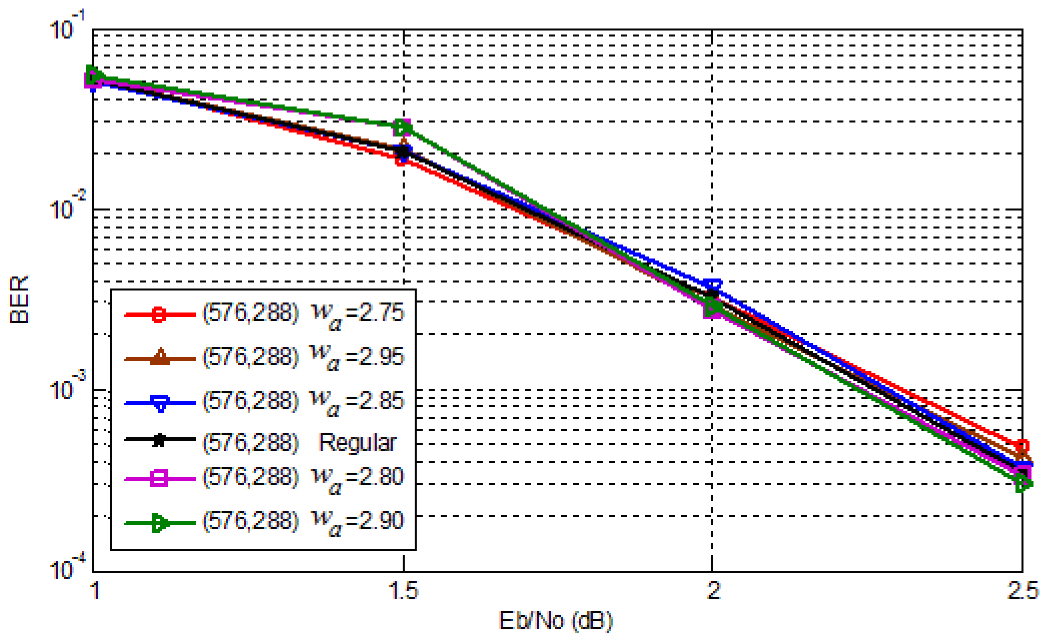


Figure 3.5 Performance comparison of (576, 288) regular and irregular 2A codes, where w_a denotes the average variable node degree of the code.

We should note here that the check node degrees of the 2A matrices are kept as uniform as possible as it is mentioned in **[MacKay-Neil-1996]**. For a code with average variable node degree w_a, we have tried to make the degrees of the check nodes close to the average check node degree value which is $\frac{n}{m} w_a = 2 w_a$. In Appendix B, check node degree distribution of sample irregular 2A codes are visualized.

When Figure 3.5 is investigated, it is seen that all irregular 2A codes having w_a values close to 3, and the regular (3, 6) code have similar performances. Nevertheless, the irregular code with average node degree 2.90 seems to have the best performance. To compare this irregular code with the regular one only these two performance curves are given in Figure 3.6.

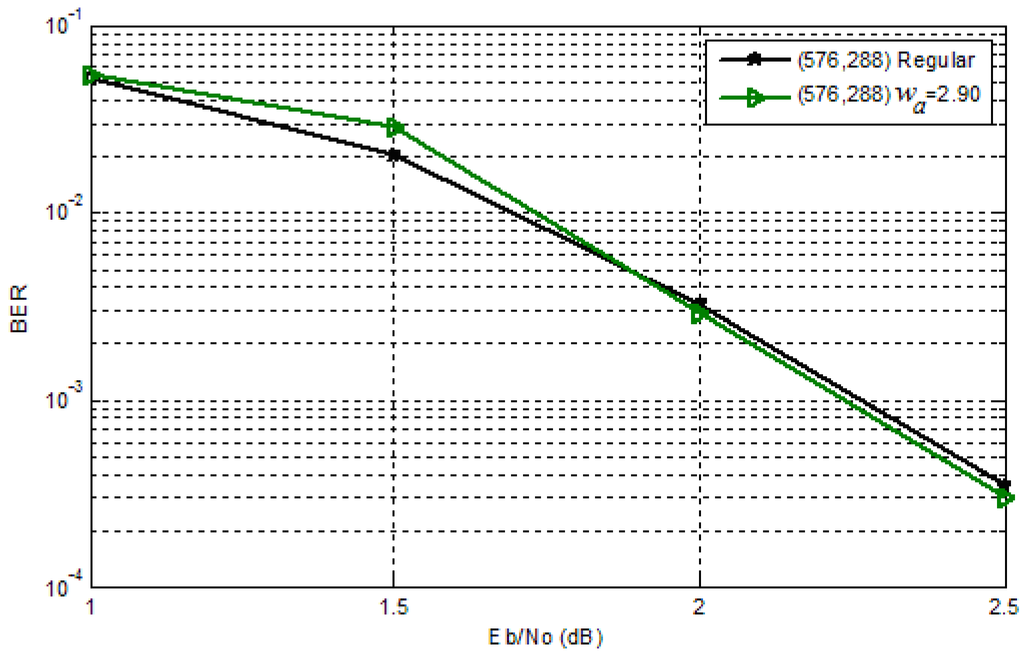


Figure 3.6 Performance comparison of (576, 288) regular and irregular 2A codes, where w_a denotes the average variable node degree of the code.

Figure 3.6 shows that for rate ½ codes of length 588, the irregular 2A code with average variable node degree 2.90 has slightly better performance than that of the regular one with variable node degree 3. This result led us to simulate longer length 2A codes in order to see whether the irregularity of this type always improves the performance of the code.

Secondly we have simulated the performance of (896, 448) regular and irregular codes with average variable node degrees around 3 and obtained Figure 3.7.

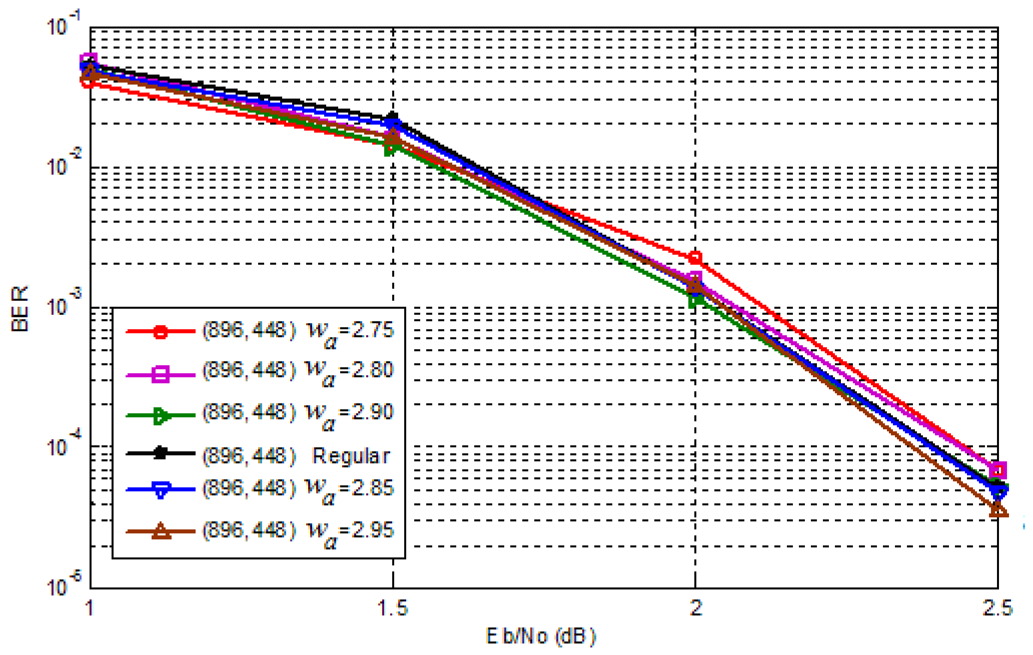


Figure 3.7 Performance comparison of (896, 448) regular and irregular 2A codes, where w_a denotes the average variable node degree of the code.

For codeword length of 896 bits, again we could not observe a noticeable improvement in the performance of the code. After these observations, we have decided to further increase the codeword length and see the effect of it on the performance of the code. We have simulated codes with codeword lengths 1200, 1500, 1800, 2100, 2700 and 3600 bits. The simulation results for all of these different codeword length codes are given in Figure 3.8 to Figure 3.13. In the graphs, we included only the best performance irregular code together with the regular code in order to see the difference clearly.

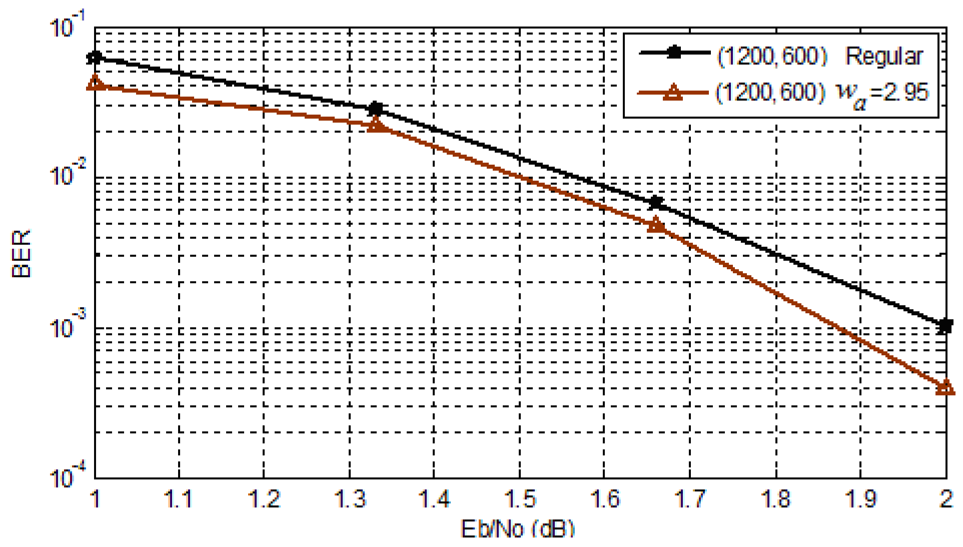


Figure 3.8 Performance comparison of (1200, 600) regular and irregular 2A codes, with the average variable node degrees $w_a = 3$ and $w_a = 2.95$, respectively.

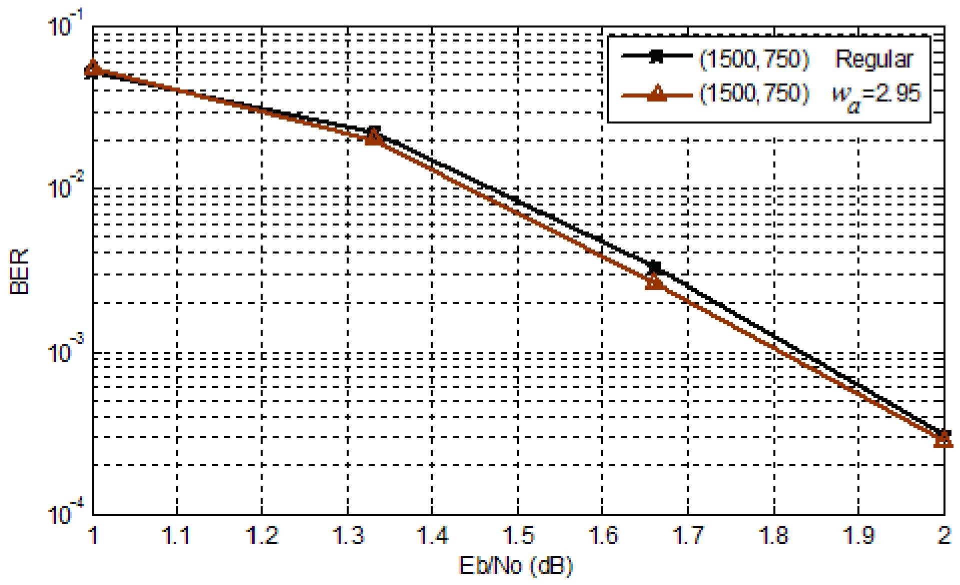


Figure 3.9 Performance comparison of (1500, 750) regular and irregular 2A codes, with the average variable node degrees $w_a = 3$ and $w_a = 2.95$, respectively.

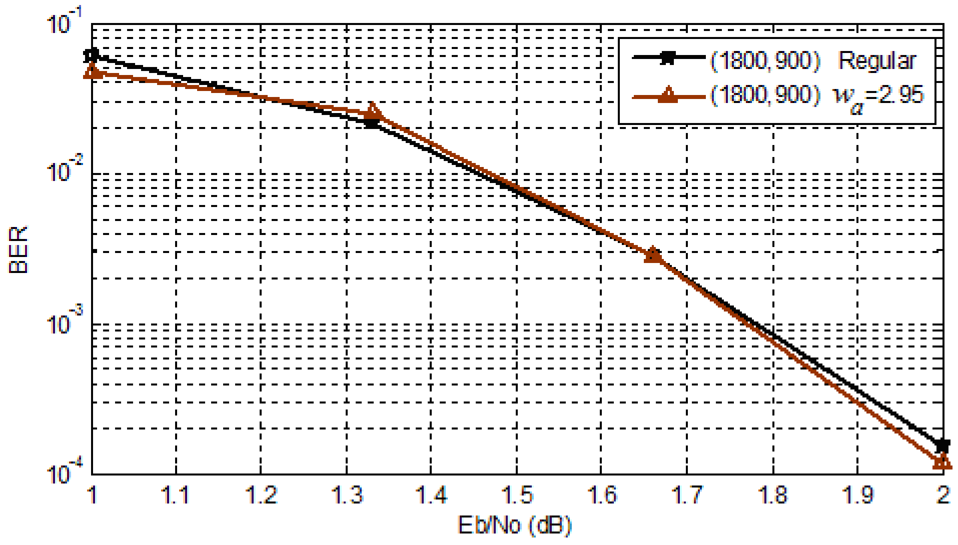


Figure 3.10 Performance comparison of (1800, 900) regular and irregular 2A codes, with the average variable node degrees $w_a = 3$ and $w_a = 2.95$, respectively.

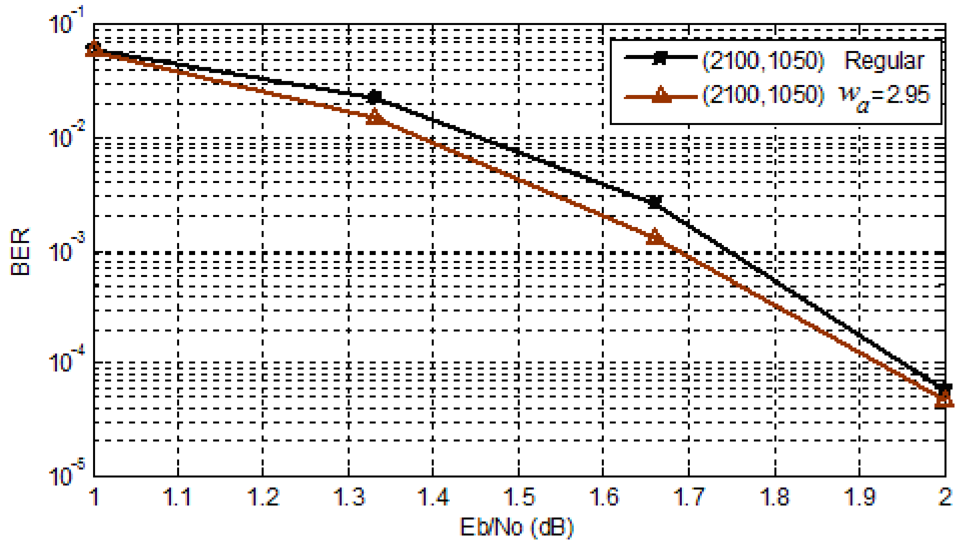


Figure 3.11 Performance comparison of (2100, 1050) regular and irregular 2A codes, with the average variable node degrees $w_a = 3$ and $w_a = 2.95$, respectively.

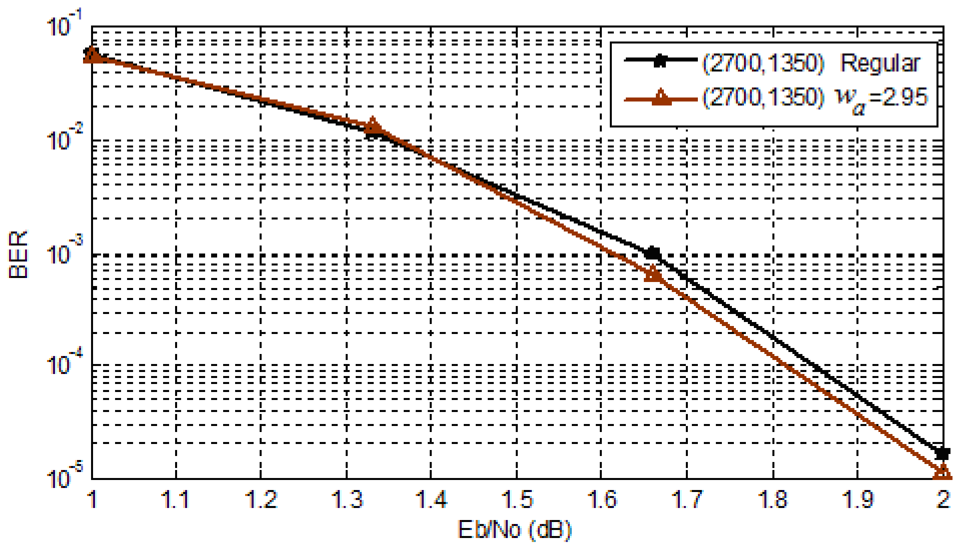


Figure 3.12 Performance comparison of (2700, 1350) regular and irregular 2A codes, with the average variable node degrees w_a = 3 and w_a = 2.95, respectively.

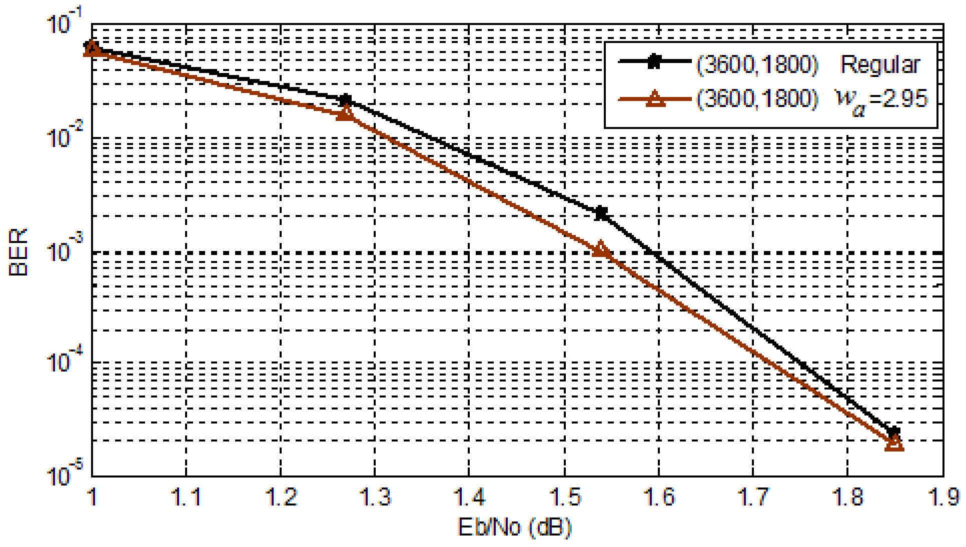


Figure 3.13 Performance comparison of (3600, 1350) regular and irregular 2A codes, with the average variable node degrees w_a = 3 and w_a = 2.95, respectively.

When we examine Figures 3.8 to 3.13, we see that the codes with average variable node degree of 2.95 have always slightly better performance than regular ones. At first sight, the reason for this is not very clear. Also, in **[MacKay-Neal-1996]**, the authors do not give a lucid reason for using weight-2 columns in their design.

The codes with average variable node degree of 2.95 have almost regular structures, where 95 % of the columns are weight-3 and 5 % are weight-2. Therefore, it does not make much sense to claim that the irregularity of the code is the reason for the better performance. After some thinking, we conjecture that one possible reason for the better performance of the codes with average variable node degree of 2.95 may be their local girth distribution. In Section 2.2.2, we have discussed that the non-overlapping property of the degree-2 columns decreases the number of length-6 cycles. Therefore, the irregular codes constructed with the 2A method have less number of length-6 cycles as compared to regular (3, 6) code. In Table 3.6, we present the total number of length-6 cycles for the (1200, 600) regular (3, 6) codes and (1200, 600) irregular codes that we construct by the 2A method.

Table 3.6 Total number of length-6 cycles for (1200, 600) regular and irregular codes constructed by the 2A method

Matrix Number	**Average Variable Node degree**	**Variable Node Degree Distribution**	**Total Number of Length-6 Cycles**
1	2.75	$\lambda_1(x) = 0.25x + 0.75x^2$	128
2	2.80	$\lambda_2(x) = 0.2x + 0.8x^2$	135
3	2.85	$\lambda_3(x) = 0.15x + 0.85x^2$	149
4	2.90	$\lambda_4(x) = 0.1x + 0.9x^2$	165
5	2.95	$\lambda_5(x) = 0.05x + 0.95x^2$	184
6	3	$\lambda_6(x) = x^2$	203

In Table 3.6, it is seen that the regular code has the largest number of length-6 cycles, which decreases with decreasing average variable node degree of the code.

If we consider only from the cycle point of view, we expect that the code with less number of length-6 cycles has the best performance. However, there exists a trade-off between the number of length-6 cycles and the average node degree of a code. As the average variable node degree increases, each variable of the codeword is checked by larger number of equations, which in turn improves the performance. On the other hand, an increase in the average variable node degree also increases the 1's of the parity check matrix, hence the number of cycles, which degrades performance by thwarting the correct decision process with repeated use of some variable bits in the same check equations. For instance, a cycle of length 6 involves 3 variables, say V_1, V_2, V_3, used in 3 different check equations, C_1, C_2 and C_3 in different pairs; say (V_2, V_3) in the check equation C_1, (V_1, V_3) in C_2, and (V_1, V_2) in C_3 (see Figure 2.3). So, if all the variables V_1, V_2, V_3, were decoded incorrectly, the three check equations C_1, C_2 and C_3 would all be satisfied and those three bit errors would be undetectable.

In order to see the effect of the number of length-6 cycles more closely, we have generated many LDPC codes with identical parameters and investigated their performances. From the parity-check matrices that we have generated, we have selected five different (1200, 600) regular codes (with variable node degree 3), and five different (1200, 600) irregular codes with average variable node degree of 2.95. The total numbers of length-6 cycles for all of these regular and irregular matrices are given in Table 3.7.

Table 3.7 Total number of length-6 cycles for (1200, 600) regular and irregular codes constructed by the 2A method

Matrix Number	**Average Variable Node Degree**	**Variable Node Degree Distribution**	**Total Number of Length-6 cycles**
1	3	$\lambda_1(x) = x^2$	203
2	3	$\lambda_2(x) = x^2$	181
3	3	$\lambda_3(x) = x^2$	171
4	3	$\lambda_4(x) = x^2$	157
5	3	$\lambda_5(x) = x^2$	151
6	2.95	$\lambda_6(x) = 0.05x + 0.95x^2$	174
7	2.95	$\lambda_7(x) = 0.05x + 0.95x^2$	172
8	2.95	$\lambda_8(x) = 0.05x + 0.95x^2$	167
9	2.95	$\lambda_9(x) = 0.05x + 0.95x^2$	149
10	2.95	$\lambda_{10}(x) = 0.05x + 0.95x^2$	139

From the parity-check matrices given in Table 3.7, we have firstly selected and simulated the performance of the regular matrix with the lowest number of length-6 cycles, which is matrix-5, and the performance of the irregular matrix with the highest number of length-6 cycles, which is matrix-6. In Figure 3.14 that shows the results of the simulations, we see that the regular code, which has less number of length-6 cycles than the irregular one, has better performance.

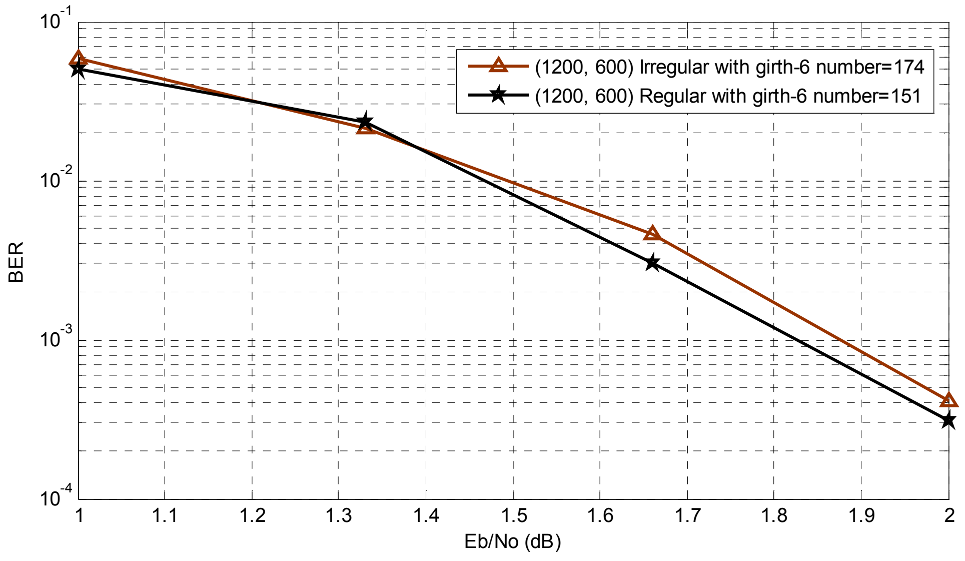


Figure 3.14 Performances of matrix-5 and matrix-6 of Table 3.7

As the second example, we have compared the regular matrix with the highest number of length-6 cycles (matrix-1), and the irregular matrix with the lowest number of length-6 cycles (matrix-10). The performances of these codes are given in Figure 3.15. In this case, the irregular matrix, which has less number of length-6 cycles than the regular one, has better performance than the regular one.

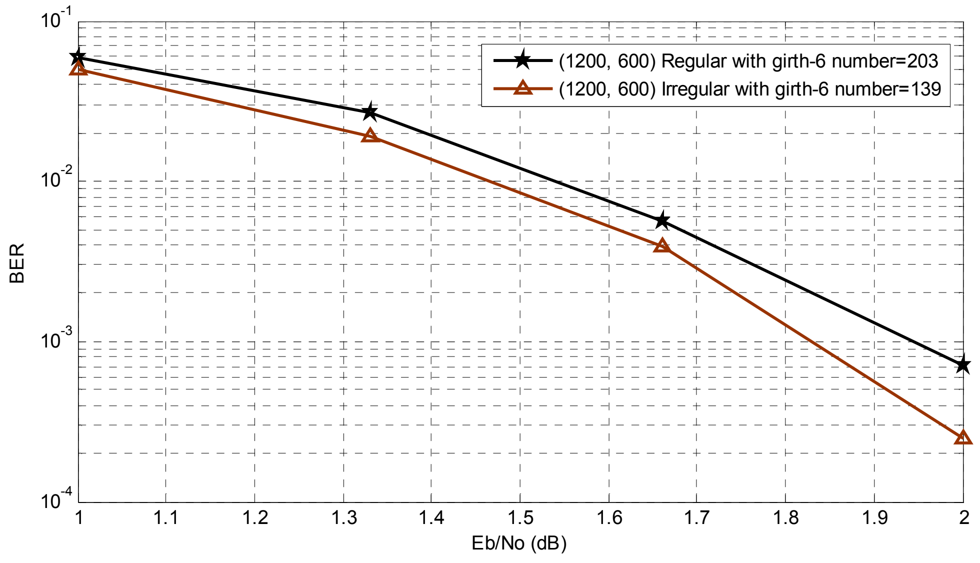


Figure 3.15 Performances of matrix-1 and matrix-10 of Table 3.7

Finally, we have simulated the performances of the regular and irregular LDPC codes having the same number of length-6 cycles, namely, matrix-3 & matrix-7 of Table 3.7. In Figure 3.16 that presents the results, one can observe that the performance of the regular matrix is slightly better than the irregular one.

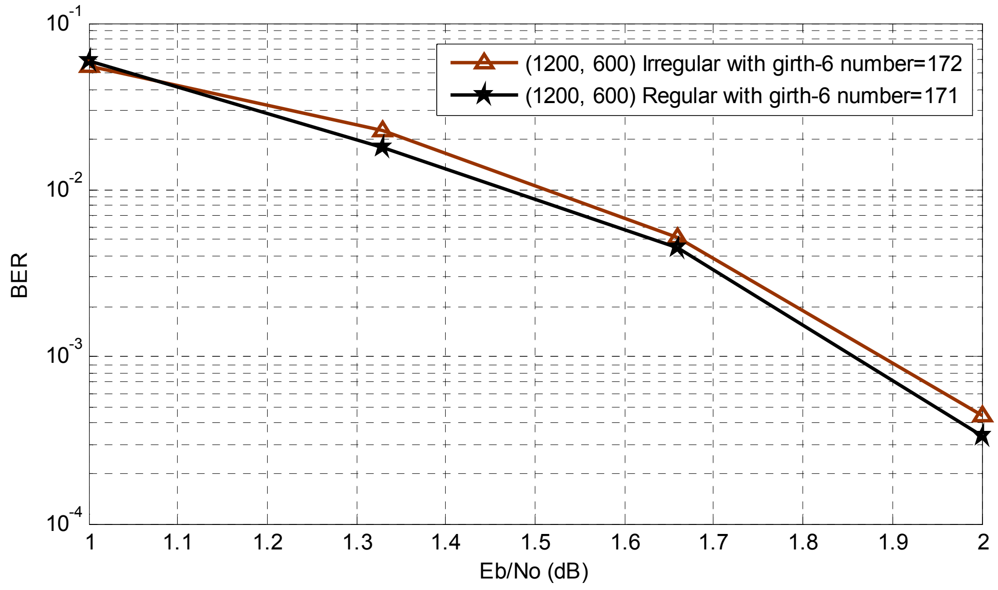


Figure 3.16 Performances of matrix-1 and matrix-10 of Table 3.7

Considering all the cases that we have investigated, and comparing the (3, 6) regular codes with the irregular 2A codes of average variable node degree 2.95, we can say that the codes with less number of length-6 cycles have always better performances. The number of 1's in the parity check matrix of an irregular 2A code is smaller than the number of 1's in the parity-check matrix of a regular code. Therefore, it is a great probability that a randomly chosen irregular 2A code has less number of length-6 cycles than a regular code. Because of this fact, the irregular 2A codes of average variable node degree 2.95, which is almost equal to 3, have better performances than the regular codes for most of the time.

3.4 Effect of 2/3/4 Irregularity

In this section, we investigate the effect of adding a slight pseudo-random irregularity to regular (3, 6) codes. Instead of a parity check matrix with all the variable nodes of degree-3, we randomly generate some columns with degree-2 and some with degree-4. The number of columns of degree-2 is kept equal to the number of columns of degree-4 in order to have average variable node degree equal to 3. To construct the matrices, we use our software for constructing pseudo-random irregular LDPC code generation, which is explained in Appendix A.

The variable node degree distribution polynomials of the constructed irregular matrices are given in Table 3.8. As these 4 different polynomials indicate, the percentage of the weight-2 and weight-4 columns are chosen as 5%, 10%, 20% and 33% respectively. All the remaining columns in the parity check matrices are of weight 3. We name all these codes as 2/3/4 codes and corresponding irregularity as 2/3/4 irregularity.

Table 3.8 The variable node degree distributions of the 2/3/4 irregular matrices

Percentage of Degree-2 and Degree-4 Variable Nodes	**Variable Node Degree Distribution**
5 %	$\lambda_1(x) = 0.05x + 0.9x^2 + 0.05x^3$
10 %	$\lambda_2(x) = 0.1x + 0.8x^2 + 0.1x^3$
20 %	$\lambda_3(x) = 0.2x + 0.6x^2 + 0.2x^3$
33 %	$\lambda_4(x) = 0.33x + 0.34x^2 + 0.33x^3$

In AWGN channel, we have simulated the BER performances of the codes defined by the pseudo-randomly generated parity check matrices. We have used rate ½ codes at two different lengths, to generate (576, 288) and (896, 448) codes. In Figure 3.17 and Figure 3.18, the BER versus SNR curves of the irregular codes and that of the regular code are given for codeword lengths of 576 and 896 respectively.

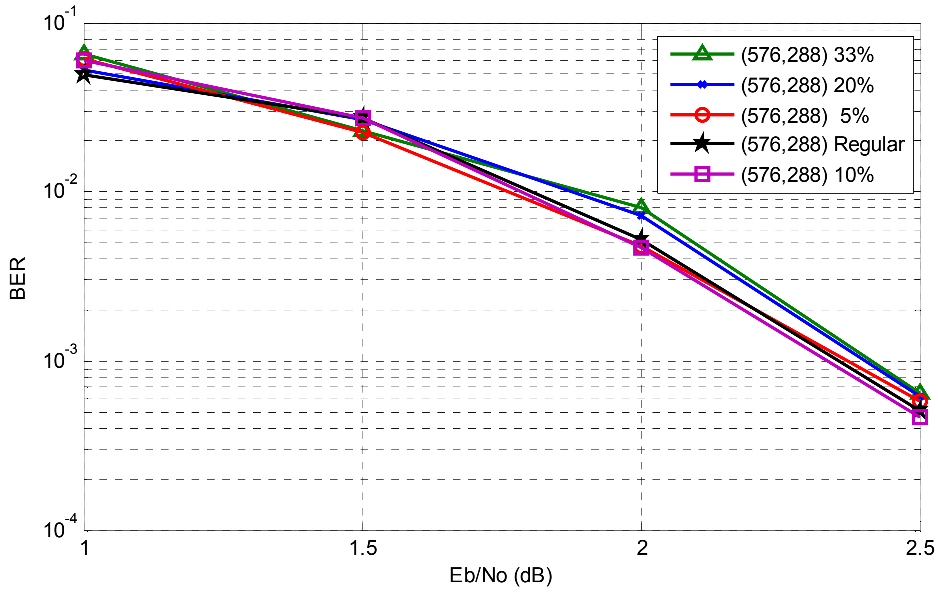


Figure 3.17 Performance comparison of (576, 288) codes defined in Table 3.8

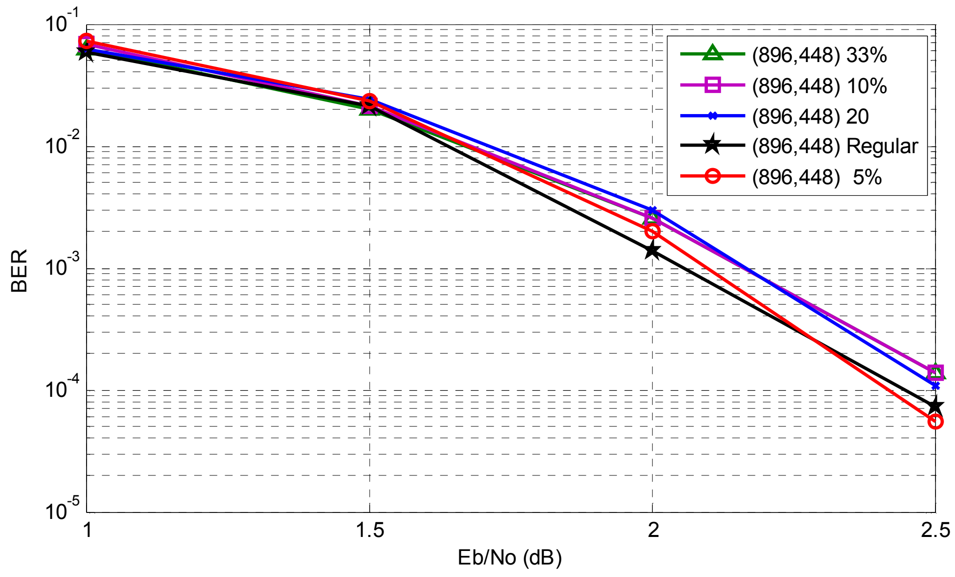


Figure 3.18 Performance comparison of (896, 488) codes defined in Table 3.8

Figures 3.17 and 3.18 do not offer a clear idea about the effect of adding this slight irregularity to the regular codes. In order to see whether there exists a repeatable difference, we have done two more simulations for each of the codes. In these new simulations, the are different random noise generation seeds from the first simulations. The results are given in Figures 3.19, 3.20, 3.21 and 3.22.

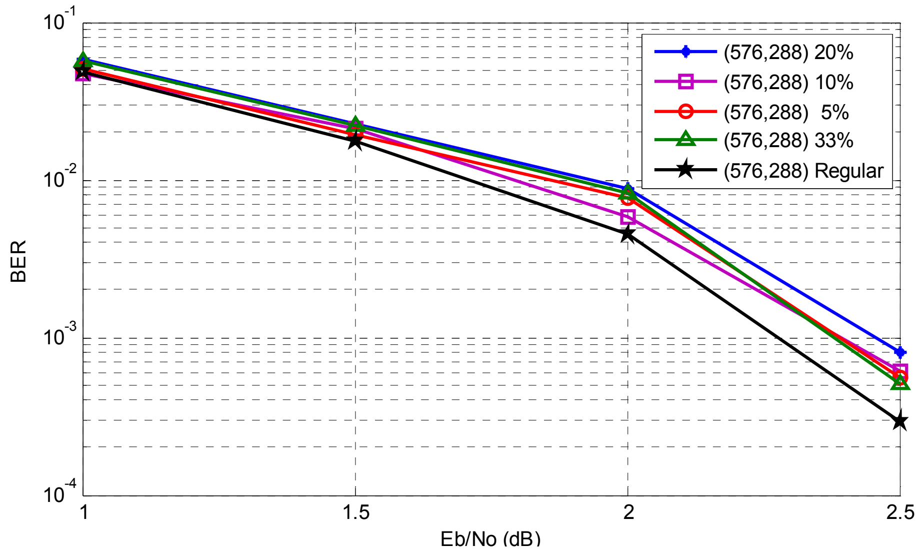


Figure 3.19 Performance comparison of (576, 288) codes defined in Table 3.8 with a second seed for random noise generation

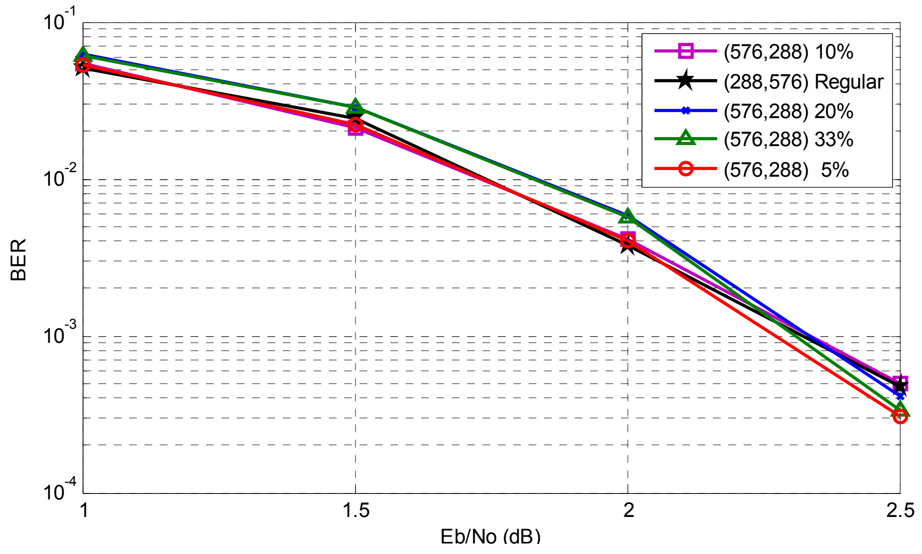


Figure 3.20 Performance comparison of (576, 288) codes defined in Table 3.8 with a third seed for random noise generation

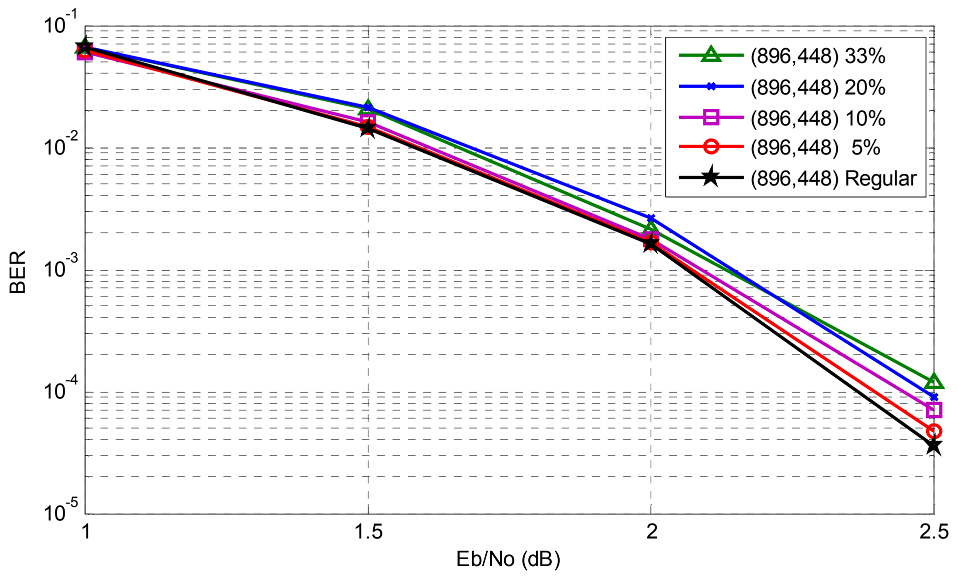


Figure 3.21 Performance comparison of (896, 488) codes defined in Table 3.8 with a second seed for random noise generation

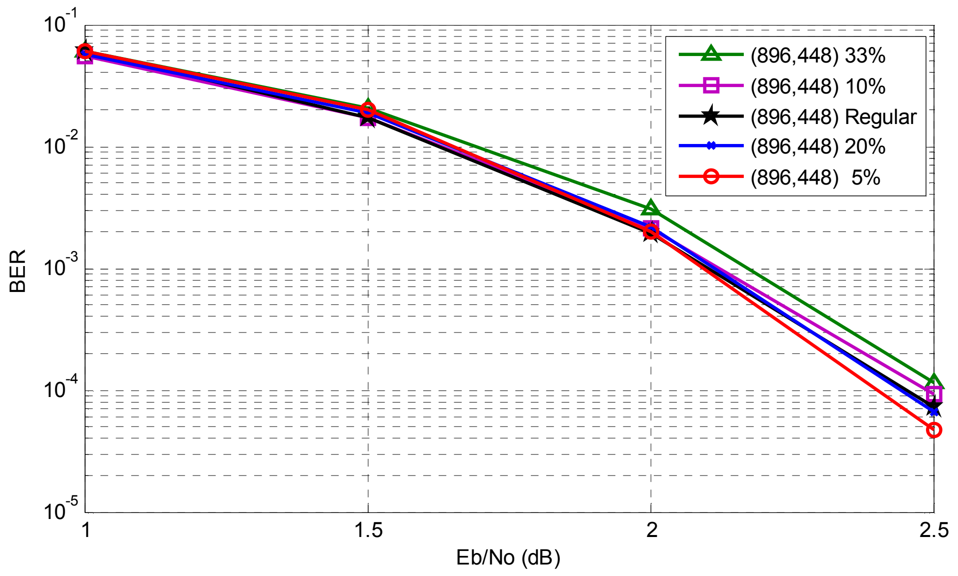


Figure 3.22 Performance comparison of (896, 488) codes defined in Table 3.8 with a third seed for random noise generation

When all simulations results are investigated, we can say that for a given codeword length, all codes have nearly the same BER performance. Because, the code that has the best performance changes when noise generation seeds are changed. This change cannot be the result of the different number of length-6 cycles mentioned in the previous section, since in three different simulations that use different noise seeds, the codes of Table 3.8 and the regular (3, 6) code are kept as the same as before, i.e., they are not regenerated.

Therefore we conclude that, the irregular codes of type 2/3/4 have almost the same BER performance as a regular (3, 6) code of the same length. In the following sections we will simulate the effect of different kinds of irregularity on the BER performance of the codes.

3.5 Effect of 1/3/5 Irregularity

In this section, we investigate another irregular structure for MacKay and Neil's pseudo-randomly generated LDPC codes. In the parity-check matrix of the irregular code, some variable nodes will have degree-1, some of them will have degree-3 and some of them will have degree-5. In order to have the average degree equal to 3, we will make the number of degree-1 variable nodes equal to that of the degree-5 variable nodes. To construct the matrices, we use our software for constructing pseudo-random irregular LDPC code generation, which is explained in Appendix A.

To see the effect of the mentioned degree distribution on the performance of LDPC codes and compare with the regular (3, 6) code, we have made simulations with four different irregular codes. The variable node degree distribution polynomials of these codes are given in Table 3.9, where the percentages of variable nodes with degrees 1&5 are chosen as 5, 10, 20 and 33 percent respectively. We name all these codes as 1/3/5 codes and corresponding irregularity as 1/3/5 irregularity.

Table 3.9 The variable node degree distributions of the 1/3/5 irregular matrices

Parity-Check Matrix	**Percentage of Degree-1 and Degree-5 Variable Nodes**	**Variable Node Degree Distribution**
H576_5	5 %	$\lambda_1(x) = 0.05x^0 + 0.9x^2 + 0.05x^4$
H576_10	10 %	$\lambda_2(x) = 0.1x^0 + 0.8x^2 + 0.1x^4$
H576_20	20 %	$\lambda_3(x) = 0.2x^0 + 0.6x^2 + 0.2x^4$
H576_33	33 %	$\lambda_4(x) = 0.33x^0 + 0.34x^2 + 0.33x^4$

In our simulations we have only used 576 as the codeword length. Because, the simulations for the codes with given degree distributions indicated an interesting result. We have monitored that, it is so likely to have wrong decisions at the output of the decoder.

After realizing that these codes having 1/3/5 irregularity are not successful examples of the LDPC code design, we have run some simulations so as to present the results given in Table 3.10, where "S" (success) refers to the number of codewords decoded successfully by the decoder, "F" (failure) refers to the number of cases that the decoder fails and yields blocks with errors and "W" (wrong codeword) refers to the number of codewords that are wrongly decoded. The matrices H576_5, H576_10, H576_20, H576_33 refer to the parity-check matrices of the (576, 288) irregular codes that have the degree distributions given in Table 3.9.

Table 3.10 Simulation results for the irregular codes given in Table 3.9

Parity-Check Matrix	H576_5			H576_10			H576_20			H576_33		
	First Simulation											
SNR	S	F	W	S	F	W	S	F	W	S	F	W
1 dB	11	19	1	18	20	0	13	14	6	1	8	12
1.5 dB	79	20	0	100	20	0	36	11	9	4	0	20
2 dB	482	17	3	429	8	12	73	2	18	9	0	20
2.5 dB	2107	8	12	1338	3	17	191	0	20	12	0	20
	Second Simulation											
SNR	S	F	W	S	F	W	S	F	W	S	F	W
1 dB	22	20	0	36	18	2	19	12	8	2	9	11
1.5 dB	64	19	1	115	19	1	38	3	17	7	1	19
2 dB	244	13	7	397	12	8	76	3	17	12	1	19
2.5 dB	2484	7	13	1172	3	17	136	1	19	15	0	20
	Third Simulation											
SNR	S	F	W	S	F	W	S	F	W	S	F	W
1 dB	17	18	2	23	20	0	19	17	3	2	10	10
1.5 dB	66	18	2	106	17	3	53	7	13	10	5	15
2 dB	260	16	4	412	10	10	123	2	18	11	1	19
2.5 dB	1967	6	14	889	0	20	185	0	20	12	0	20

Investigation of Table 3.10 deepens the discussion given in Section 3.2 about the *distribution of wrong decisions versus decoding failures* in 20 block errors made by the decoder, by adding a new dimension: if the LDPC code (say, its variable node degree distribution polynomial) is not chosen suitably, belief propagation algorithm of the LDPC decoder is more likely to give wrong decisions rather than decoding failures, especially at low noise levels. We see that the number of wrong decisions at the output of the decoder is higher for the parity-check matrices with more number of variable nodes of degree-1 and degree-5. Actually, this issue is related with the variable nodes of degree-1 in these matrices. In a Tanner graph, a degree-1 variable node is connected to a single check node and receives information only from this single node. This fact increases the probability for a degree-1 variable node, to be misguided by the wrong information coming from the check node that it is connected to. Therefore, one should avoid variable nodes of degree-1 in LDPC code design.

In order to see the effect of the wrong codewords on the performances of the codes, we have simulated the performances of the codes defined in Table 3.9. Figure 3.23 shows the results of the simulations.

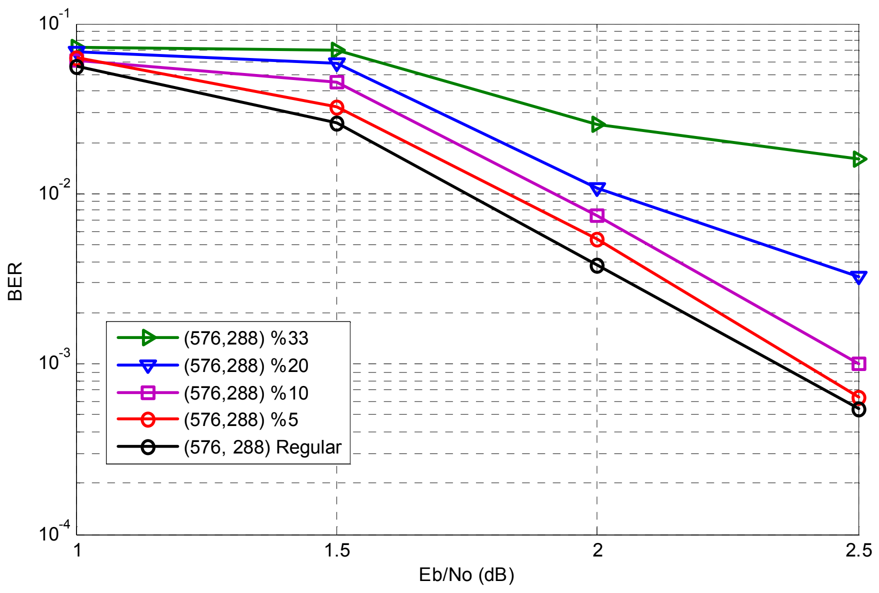


Figure 3.23 Performances of the 1/3/5 codes defined in Table 3.9

In Figure 3.23, we see that performances of the codes with more number of degree-1 nodes are worse than the codes with less number of degree-1 nodes. We can say that the wrong decisions which are led by the degree-1 nodes distort the performance of the codes. The performance loss occurring for the codes with more number of degree-1 nodes is an obvious reason to avoid degree-1 nodes in LDPC code design.

3.6 High Degree Variable Nodes Connected to 9 or 19 Check Equations

In the previous sections, we have seen that the performances of irregular codes, which have average variable node degrees around 3, are not noticeably better than the regular codes. Because of this fact, we have decided to increase the average variable node degree w_a of the constructed irregular codes by adding more high-weight columns to the parity-check matrix.

The experiments we have done in previous sections also showed that having weight-2 columns are helpful to improve the performance of the codes since they lower the number of length-6 cycles. Considering this fact, we have decided to have some weight-2 columns in our irregular parity-check matrices together with weight-3 and higher-weight columns. Particularly, we have compared the performances of 2/3/9 irregular and 2/3/19 irregular codes.

We have constructed irregular LDPC codes with rate $1/2$ and codeword length 1800. In the parity-check matrices of the irregular codes, a fixed 20% of the columns are chosen of weight-2. The remaining columns are composed of weight-3 and higher-weight columns in such a way that the average variable node degree of the parity-check matrix is equal to some specific value. This specific value, which is greater than 3, is adjusted by changing the numbers of the weight-3 and higher-weight columns. In some of the parity-check matrices, we have used weight-9 columns and in some of them we have used weight-19 columns. In Figure 3.24, the generic structure for the irregular matrices is given.

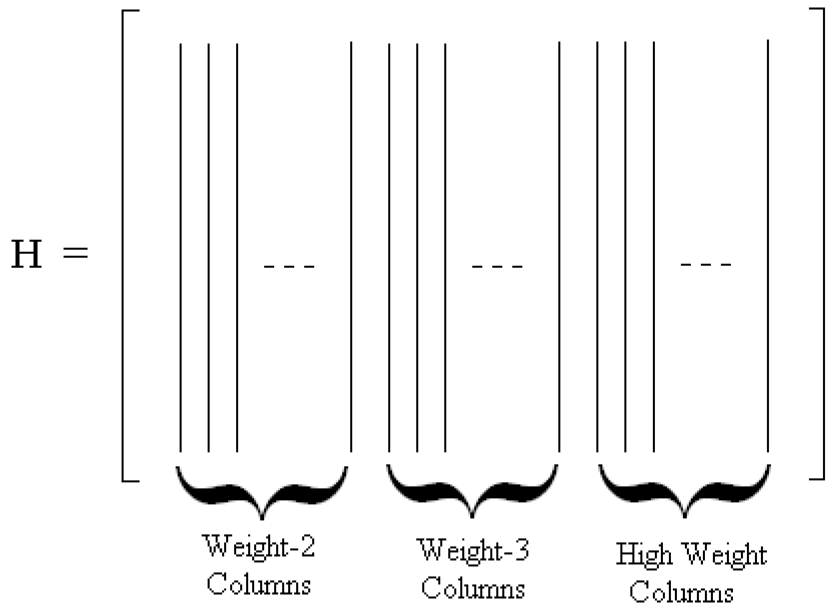


Figure 3.24 The generic structure of the pseudo-random irregular matrices of this work

3.6.1 Effect of the Average Variable Node Degree

We have constructed the irregular codes given in Table 3.11, which shows the variable node degree distributions, average variable node degrees, and the number of higher weight columns. We name each code with 2/3/9 irregularity as 9-3.2, 9-3.4 or 9-3.6 and each code with 2/3/19 irregularity as 19-3.2, 19-3.4 or 19-3.6; where the first number indicates the highest variable node degree and the second number shows the average variable node degree. We have simulated the performance of these codes in AWGN and obtained Figure 3.25, which also includes the performance of a regular (3, 6) code for comparison.

In Figure 3.25, we see that the irregular code with the best performance is the Code 19-3.6 and the one with the worst performance is 19-3.2. However, all the irregular codes have better performances than the regular one.

Table 3.11 The variable node degree distributions and the number of high weight columns of 2/3/9 and 2/3/19 irregular codes of rate ½ and length 1800, with average variable node degrees of 3.2, 3.4 and 3.6

Code Abbreviation	Variable Node Degree Distribution Polynomial $\lambda(x)$.	Average Variable Node Degree w_a	Number of High Weight Columns
9-3.2	$0.2x^1 + 0.733x^2 + 0.067x^8$	3.2	120
9-3.4	$0.2x^1 + 0.7x^2 + 0.1x^8$	3.4	180
9-3.6	$0.2x^1 + 0.67x^2 + 0.13x^8$	3.6	240
19-3.2	$0.2x^1 + 0.775x^2 + 0.025x^{18}$	3.2	45
19-3.4	$0.2x^1 + 0.762x^2 + 0.038x^{18}$	3.4	68
19-3.6	$0.2x^1 + 0.75x^2 + 0.05x^{18}$	3.6	90

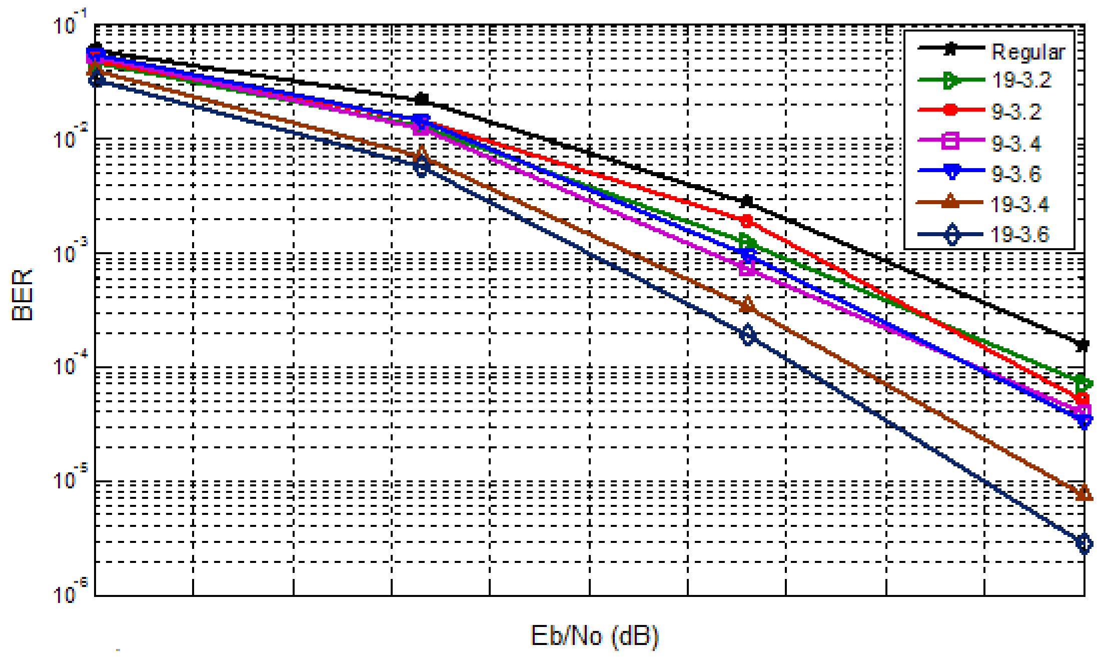


Figure 3.25 Performances of the codes described in Table 3.11

Looking at Figure 3.25, we can say that the performance of the irregular codes gets better when their average variable node degrees are increased. This actually makes sense because the codes with greater average variable node degrees have greater number of high weight columns. These variable nodes defined by these high weight columns are decoded correctly with high probability and the correct information coming from these nodes helps the decoding of the other variable nodes which have lower degrees.

At this point, it will be quite helpful to investigate the results of these simulations in three different cases, where the average node degrees are 3.2, 3.4 and 3.6.

Case 1: Average Variable Node Degree is 3.2

In this case, the code 9-3.2 with weight-9 columns has a better performance than Code 19-3.2 with weight-19 columns. Code 19-3.2 has 45 weight-19 columns, whereas Code 9-3.2 has 120 weight-9 columns.

Case 2: Average Variable Node Degree is 3.4

In this case, as opposed to Case 1, Code 19-3.4 with weight-19 columns has a better performance than Code 9-3.4 with weight -9 columns. Code 19-3.4 has 68 weight-19 columns, whereas Code 9-3.4 has 180 weight-9 columns.

Case 3: Average Variable Node Degree is 3.6

In this case, similar to Case 2, the code with weight-19 columns has a better performance than the code with weight -9 columns. Code 19-3.6 has 90 weight-19 columns, whereas Code 9-3.6 has 240 weight-9 columns.

In all three cases, since the average node degree is kept constant, the number of weight-9 columns is greater than the number of weight-19 columns. In Case 1, this difference makes the performance of the matrix with weight-9 columns better. However, when the average node degree is increased as in Cases 2 and 3, the codes with weight-19 columns have better performances than the codes with weight-9 columns, because the variables connected to 19 check equations improve the decoding performance more effectively than the variables checked by 9 equations. However, we may say that this is possible when the number of such variables (or weight-19 columns in the parity-check matrix) exceeds some threshold value.

After observing this data, we have constructed different matrices with the same parameters given in Table 3.11, to see whether the results of Figure 3.25 are repeatable. The resulting graph containing four different matrices for each parameter set are given in Figure 3.26.

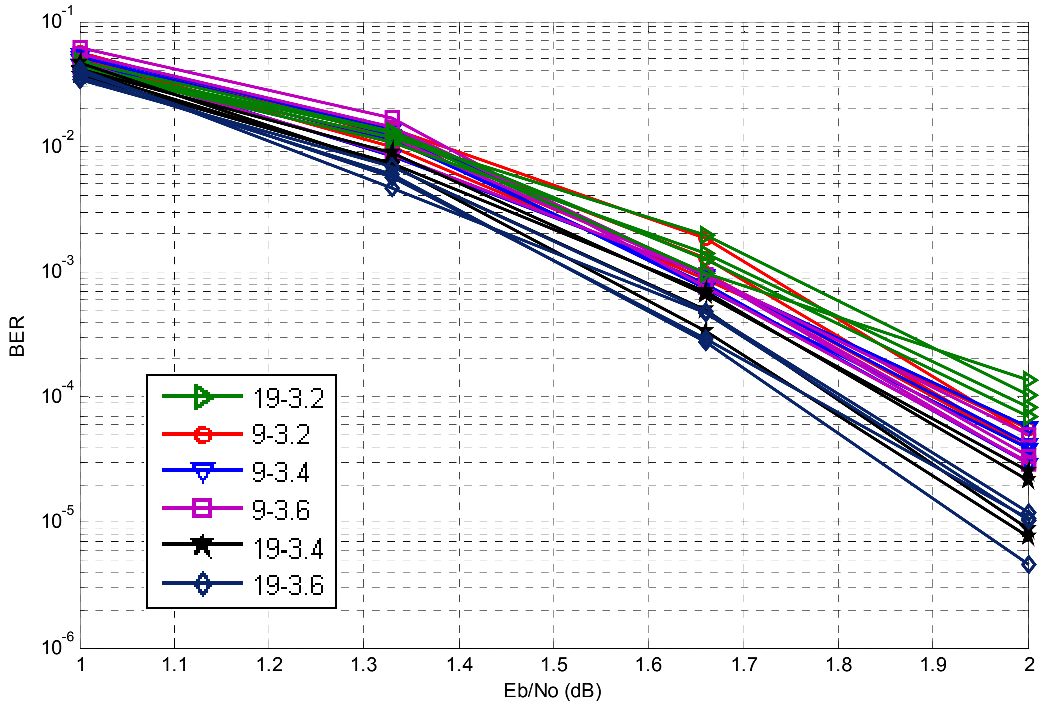


Figure 3.26 Performances of many codes with the parameters described in Table 3.11

The results in Figure 3.26 are very similar to the results in Figure 3.24, which confirms repeatability. The performances of different codes with the same parameters vary by small amounts. For example, at the BER value of 10^{-3}, the performances of the 19-3.2 codes vary by approximately 0.1 dB. This variation in performance may be due to the number of length-6 cycles contained in these codes. To see whether this deduction is true, we have counted the number of length-6 cycles for all the codes in Figure 3.26 and presented the results in the following section.

3.6.2 Effect of the Number of Length-6 Cycles

In order to have better perception about the effect of the number of length-6 cycles, say N_6, on the code performance, we have counted the number of length-6 cycles for the 24 different codes, whose performances are given in Figure 3.26. Those pseudo-randomly generated codes possess either of the 6 groups of parameters given in Table 3.11 (namely the groups 9-3.2, 9-3.4, 9-3.6 with 2/3/9 irregularity and groups 19-3.2, 19-3.4, 19-3.6 with 2/3/19 irregularity). Since we have constructed 4 codes for each group of Table 3.11, there are 24 different codes and 24 different values for the number of length-6 cycles.

Table 3.12 shows the total number of length-6 cycles for all the 24 codes in Figure 3.26. E_b/N_o values required for a specific BER value (of 2×10^{-4}) are also included in ascending order for each group of code parameters, so that the performance within the group is ranked in descending order in Table 3.12.

Table 3.12 The number of length-6 cycles of some 2/3/9 and 2/3/19 codes of rate ½ and length 1800

Code Abbreviation	**E_b/N_o for a BER of 2×10^{-4}**	**Number of Length-6 Cycles N_6**
9-3.2	1.85	952
	1.86	987
	1.87	975
	1.89	1049
19-3.2	**1.90**	**2216**
	1.92	**2363**
	1.95	**2558**
	1.97	**2570**
9-3.4	1.82	1911
	1.83	1922
	1.84	1954
	1.87	1978
19-3.4	**1.73**	**4976**
	1.76	**5151**
	1.80	**5274**
	1.81	**5259**
9-3.6	1.83	3254
	1.84	3293
	1.84	3341
	1.87	3389
19-3.6	**1.70**	**9159**
	1.72	**9565**
	1.75	**9894**
	1.76	**9930**

In Table 3.12, it is observed that within the set of codes with the same parameters, increasing number of length-6 cycles leads to worse performance for almost all cases. For instance, among the codes that has weight-19 columns and average variable node degree 3.6, the worst performance code has 9159 length-6 cycles, whereas the best performance code has 9930 length-6 cycles. This difference leads to an improvement of 0.06 dB in the E_b/N_o value for the BER level of 2×10^{-4}. There also exist some exceptions; for example, in the set of 19-3.2 codes, the code with 987 many length-6 cycles has a better performance than the code with 975 many length-6 cycles. However, the number of length-6 cycles for these two codes are very close, and such a slight difference is not repeatable when the random noise samples are initiated by a different seed.

From Table 3.12, we can also see that, for a given average variable node degree w_a, the 2/3/19 codes (that have degree-19 variable nodes) have much more length-6 cycles than the 2/3/9 codes (that have degree-9 variable nodes). For variable node degrees of 3.4 and 3.6, even though the 2/3/19 codes have more length-6 cycles, they have better performances than the 2/3/9 codes. This is a normal result since the number of length-6 cycles is obviously not a primary comparison element for codes with different parameters.

3.7 Joint Effect of Average Variable Node Degree and Length-6 Cycles

After observing the performances of codes with average variable node degrees 3.2, 3.4 and 3.6, we have decided to investigate the joint effect of the average variable node degree (w_a) and the number of length-6 cycles (N_6), on some codes that have greater average variable node degree values. We have generated (1800, 900) irregular codes with column weight distributions of 2/3/9, 2/3/11 and 2/3/13. We have kept the ratio of weight-2 columns as 20% but adjusted the average variable node degree between 3.2 and 4.4 by changing the ratio of high-weight columns to weight-3 columns.

3.7.1 Codes with High Degree Variable Nodes of Degree 9

For this experiment, we have first used irregular codes that have degree-9 variable nodes as high degree nodes. Table 3.13 shows the variable node degree distribution polynomials and the number of high weight columns of these irregular matrices.

Table 3.13 Variable node degree distributions and the number of high weight columns of the 2/3/9 and 2/3/19 irregular codes of rate ½ and length 1800, with average variable node degrees of 3.8, 4, 4.2 and 4.4

Code Abbreviation	Variable Node Degree Distribution Polynomial $\lambda(x)$	Average Variable Node Degree w_a	Number of Weight-9 Columns
9-3.8	$0.2x^1 + 0.633x^2 + 0.167x^8$	3.8	300
9-4	$0.2x^1 + 0.6x^2 + 0.2x^8$	4	360
9-4.2	$0.2x^1 + 0.567x^2 + 0.233x^8$	4.2	420
9-4.4	$0.2x^1 + 0.533x^2 + 0.267x^8$	4.4	480

In Figure 3.27, the performances of these codes together with the previously generated codes with average variable node degree values 3.2, 3.4 and 3.6 are given.

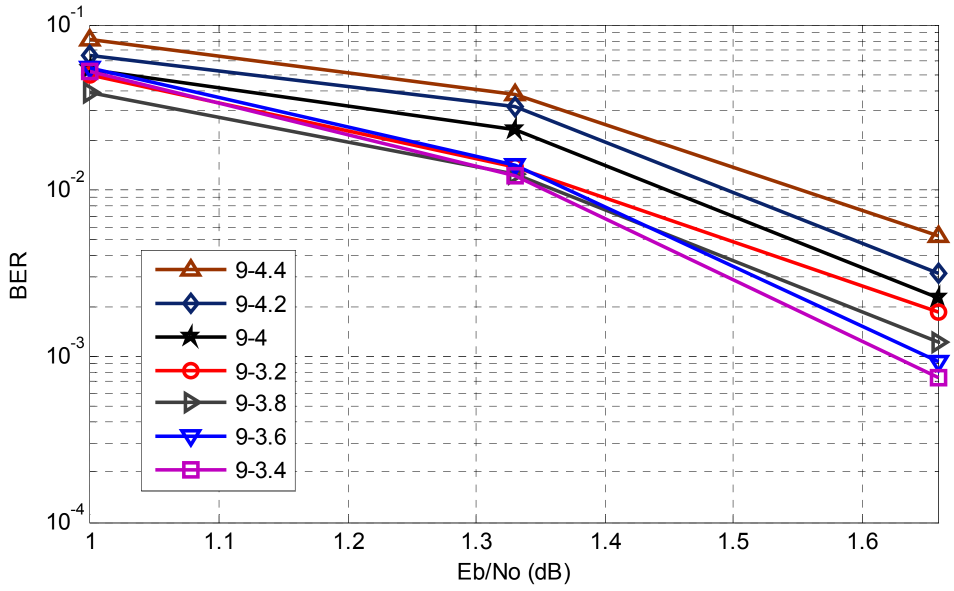
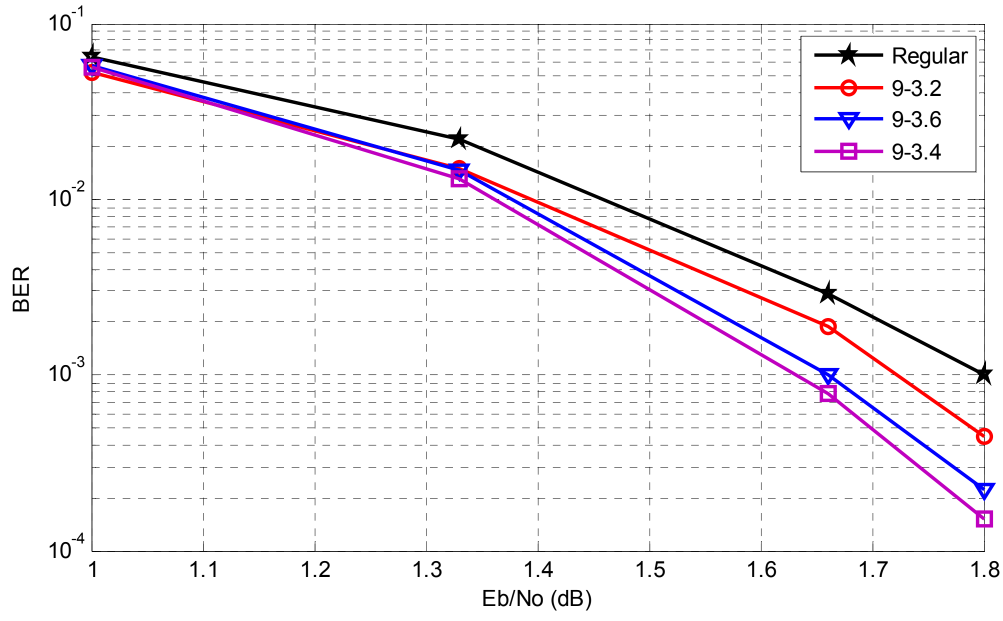
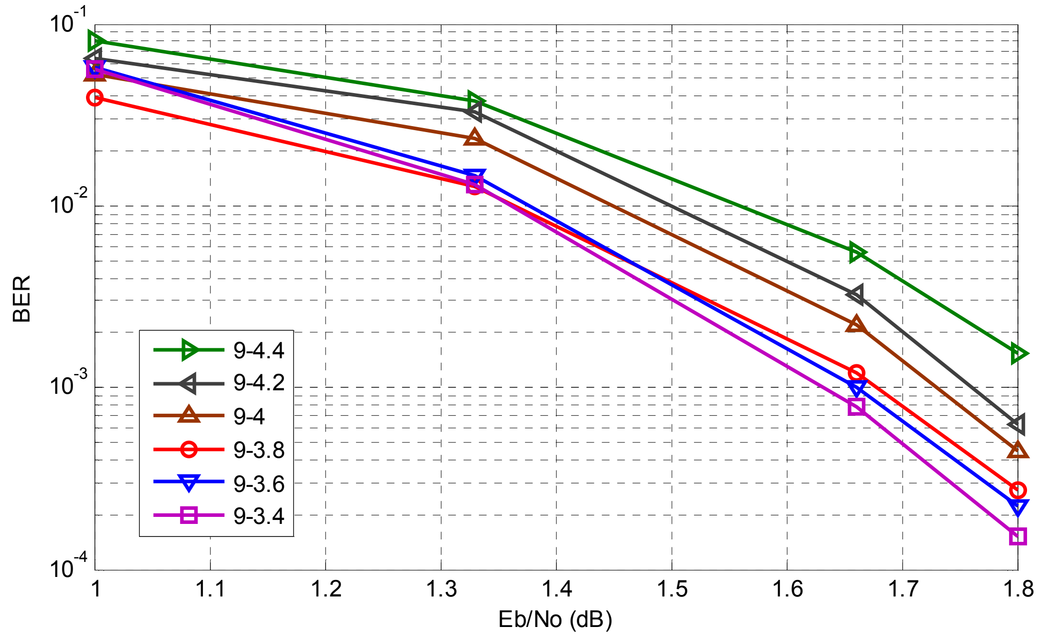


Figure 3.27 Performances of 2/3/9 irregular codes that have average variable degrees from 3.2 to 4.4

From Figure 3.27, we see that the performances of the irregular codes are getting better up to the average variable node degree of 3.6. After this point, the performances of the codes start to get worse. Here, it will be better to see the performances of these codes in two separate graphs. The first graph, given in Figure 3.28 (a), includes the codes whose performances are getting better with increasing average variable node degree, and the second graph, given in Figure 3.28 (b), includes the codes whose performances are getting worse with increasing average variable node degree.



(a)



(b)

Figure 3.28 Performances of the irregular codes with weight-9 columns. The codes whose performances are getting better with increasing average variable node degree are shown in (a), and the codes whose performances are getting worse are shown in (b).

Figure 3.28 (a) shows the performances of the irregular codes with average variable node degrees of 3.2 and 3.4 together with the performance of a regular code of the same codeword length. One can see that increasing the average variable node degree from 3.2 to 3.4 improves the decoding performance of these codes. We should also note that the irregular codes have better performances than the regular one. However, in Figure 3.28 (b), where the average variable node degrees range from 3.4 to 4.4, we see that the performance gets worse with increasing average variable node degree.

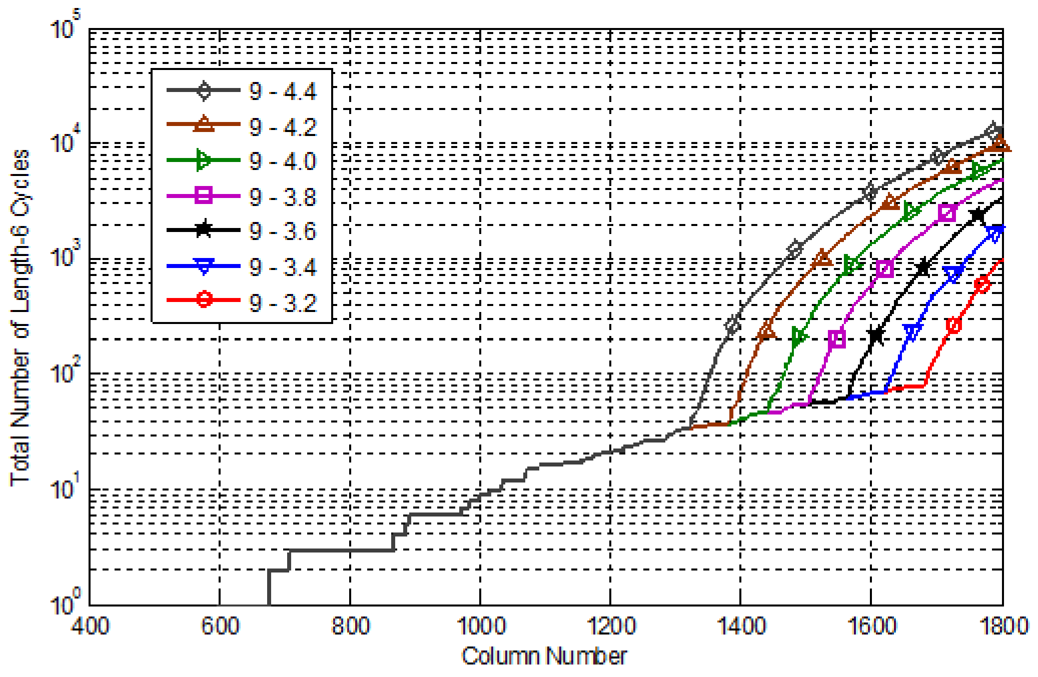
When the total number of high weight columns in an irregular matrix is increased, one expects to have a better performance. However, we have seen in this example that as the number of high-weight columns is increasing, the performance becomes worse beyond some threshold value of the average node degree. The most important reason behind this fact may be the number of length-6 cycles contained in the code. In order to have a clear idea, we have counted the number of length-6 cycles of these codes. Table 3.14 shows the number of length-6 cycles together with other necessary information.

Table 3.14 Number of length-6 cycles of the 2/3/9 codes of rate ½ and length 1800, with average variable node degrees from 3.2 to 4.4

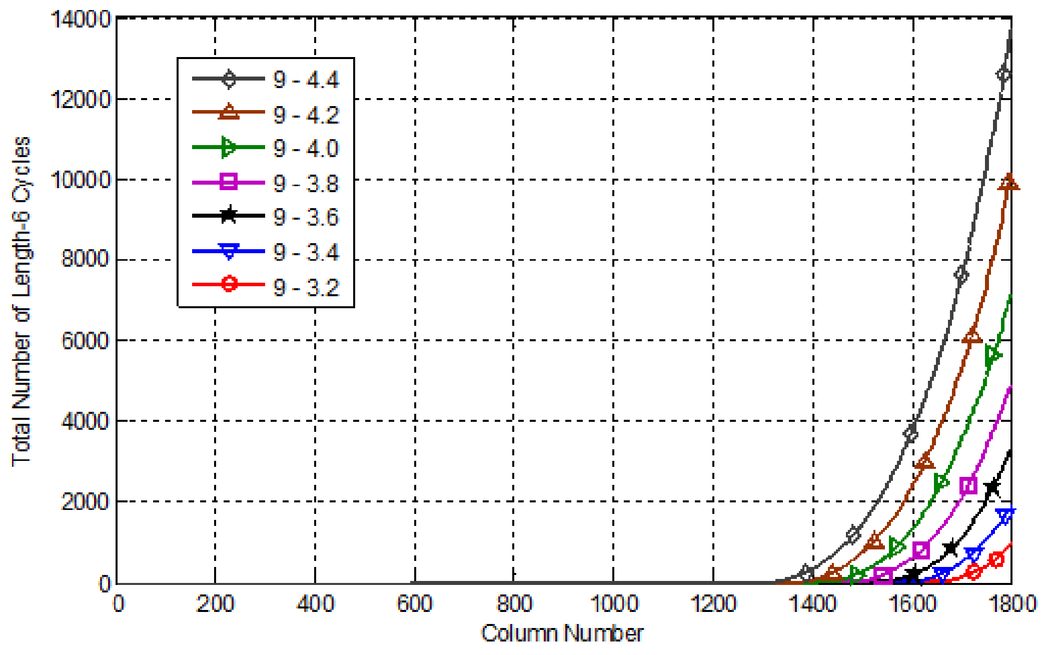
Code Abbreviation	**Average Variable Node Degree**	**Number of Weight-9 Columns**	**E_b/N_o for a BER of** 5×10^{-3}	**Total Number of Length-6 Cycles, N_6**
9-3.2	3.2	120	1.50	975
9-3.4	3.4	180	1.43	1922
9-3.6	3.6	240	1.45	3293
9-3.8	3.8	300	1.46	4784
9-4	4	360	1.55	7164
9-4.2	4.2	420	1.59	10117
9-4.4	4.4	480	1.66	13755

Unsurprisingly, the number of length-6 cycles increases with increasing number of weight-9 columns. However, an increase in the number of length-6 cycles up to 1922, which is observed for 180 many weight-9 columns, seems acceptable; since it does not deteriorate the performance. When the number of weight-9 columns is further increased to 240 and 300 the total number of length-6 cycles becomes 3293 and 4784, and the performance of the codes becomes slightly worse than the previous cases. After this point adding new weight-9 columns leads to huge rises in the number of length-6 cycles (Increasing the number of weight-9 columns from 300 to 360 makes the number of length-6 cycles 71264, from 360 to 420 makes it 10117, and from 420 to 480 makes it 13755).

In the parity-check matrix construction method that we use, columns of the matrix are formed one-by-one from left to right. Each new column introduces new length-6 cycles to the matrix that are added to the total number of length-6 cycles. Motivated by this, we have counted the length-6 cycles introduced by each column for the matrices given in Table 3.14 and obtained the graph given in Figure 3.29, where the vertical axes in parts (a) and (b) show the number of length-6 cycles in logarithmic and linear scales respectively. The horizontal axis shows the column number. For a specific column *x*, the value shown on the vertical axis is the cumulative number of length-6 cycles after the generation of column *x*, starting from the first column. When *x* is the last column, the value shown on the vertical axis corresponds to the overall number of length-6 cycles for the generated code.



(a)



(b)

Figure 3.29 The total number of length-6 cycles after the generation of each column of the parity-check matrix for the codes defined in Table 3.14. The vertical axis is given in: (a) logarithmic scale, (b) linear scale.

The number of length-6 cycles for all the matrices is zero up to the 600th column. This is an expected result since the first 20 % of the columns are weight-2 columns which have no overlap between each other. We have shown previously in this work that zero-overlap weight-2 columns do not cause any length-6 cycles. After the last weight-2 column, following columns are of weight-3. These weight-3 columns slightly increase the number of length-6 cycles. However, length-6 cycles start to increase in considerable amounts at the starting point of the construction of weight-9 columns.

In Figure 3.29 (a), these starting points for the weight-9 columns are seen very clearly. For example, for the matrix with average variable node degree of 4.4, the number of length-6 cycles starts to increase rapidly near the 1321st column which is the first weight-9 column.

The important point here is that, the increase in the number of length-6 cycles accelerates when more and more weight-9 columns are added. In other words, a weight-9 column causes more and more length-6 cycles as the number of previously added weight-9 columns increases. This situation is seen clearly in Figure 3.30, where the horizontal axis again shows the column number, but the vertical axis shows the number of length-6 cycles caused by each individual column instead of the cumulative value. It is observed from Figure 3.30 that the number of length-6 cycles introduced by a column increases with the column number. More specifically, the number of length-6 cycles caused by each new weight-9 column increases almost exponentially. Actually, this is the reason of rapid increase in the total number of length-6 cycles as we observed in Figure 3.29. If we zoom the last part of the graph given in Figure 3.29 (b), we clearly see the exponential increase in the number of length-6 cycles in Figure 3.31, as the number of weight-9 columns is increased.

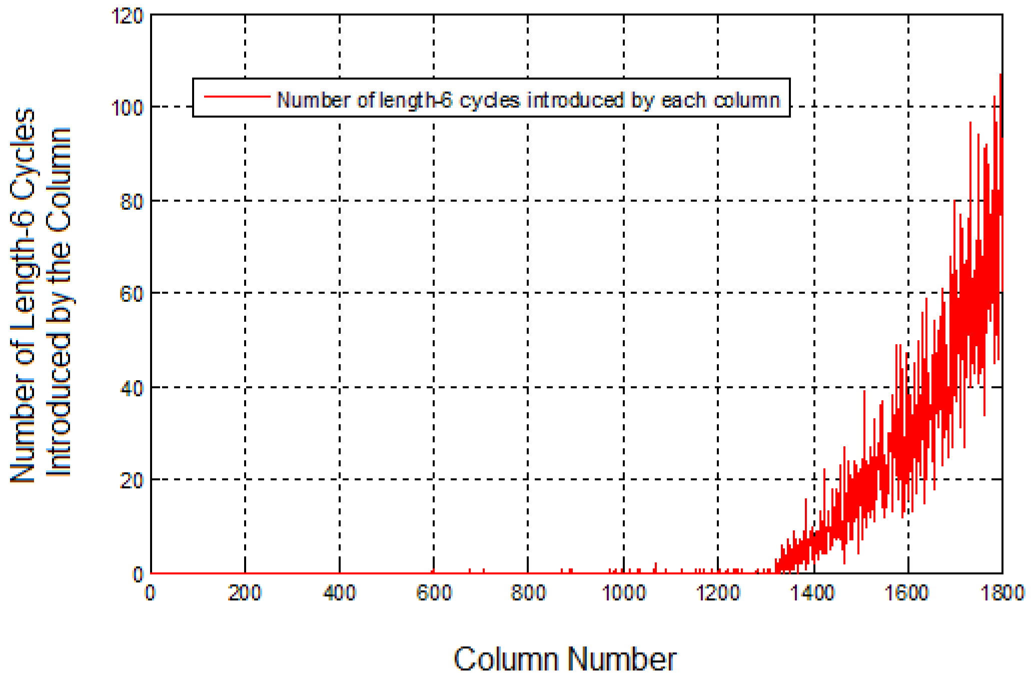


Figure 3.30 The number of length-6 cycles introduced by each column of the parity-check matrix of the code 9-4.4.

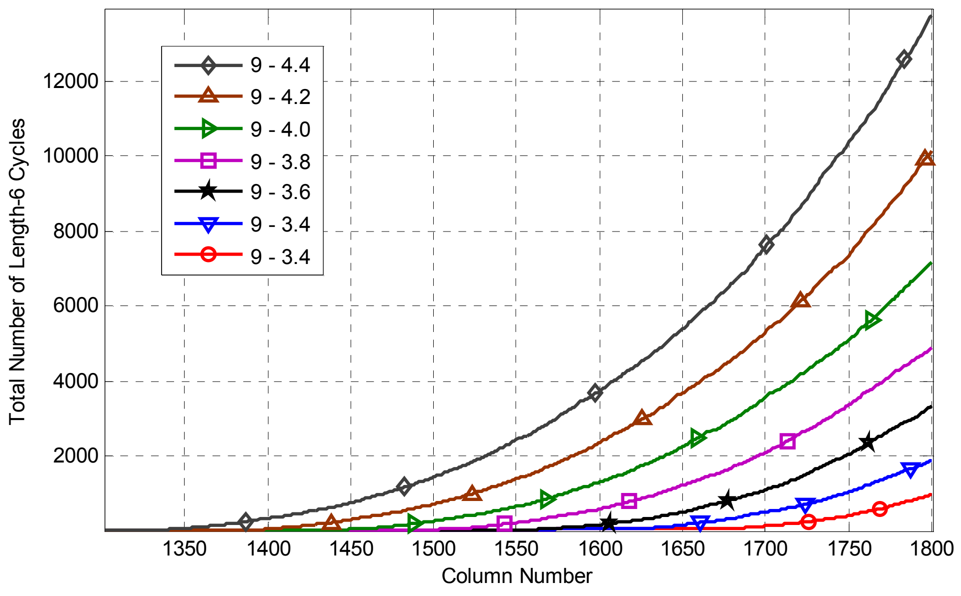


Figure 3.31 Last part of the graph given in Figure 3.29 (b)

When Figure 3.31 is more closely investigated, one sees that as the average variable node degree is increased from 3.2 to 3.4, the numbers of length-6 cycles for the matrices are fairly close to each other. However, as the increase in the average variable node degree is continued from 3.6 to 4.4, the difference between the numbers of the length-6 cycles of the generated codes gets larger.

Beyond the average node degree value of 3.4, the performances of the irregular codes become worse as the number of weight-9 columns is increased. As we said earlier, the weight-9 columns that are expected to improve the performance start to lead to worse performance because of the huge increase in the number of length-6 cycles. Below the average variable node degree of 3.4, the decoding performance improvement supplied by the weight-9 columns suppresses the effect of length-6 cycles. However, above this average degree, the number of length-6 cycles increases so much that the improvement supplied by the weight-9 columns can not cancel the effect of the length-6 cycles.

3.7.2 Extension to High Variable Node Degrees of 11 and 13

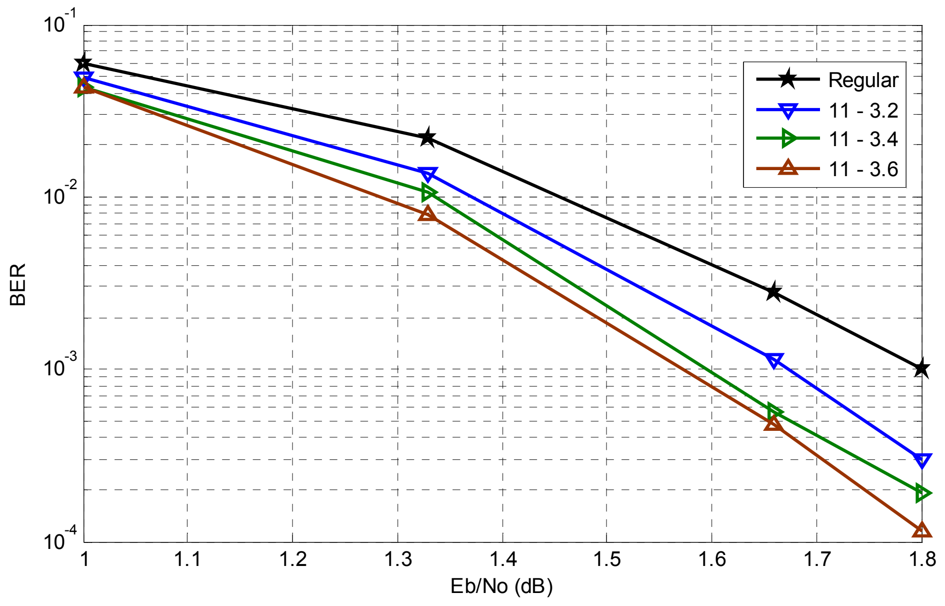
In order to see whether this situation is a general case, we have repeated the same experiments for other irregular LDPC codes which have weight-11 and weight-13 columns. Table 3.15 shows the properties of the new irregular codes that we have generated.

Table 3.15 Variable node degree distributions and the number of high weight columns of of the 2/3/11 and 2/3/13 irregular codes of rate ½ and length 1800

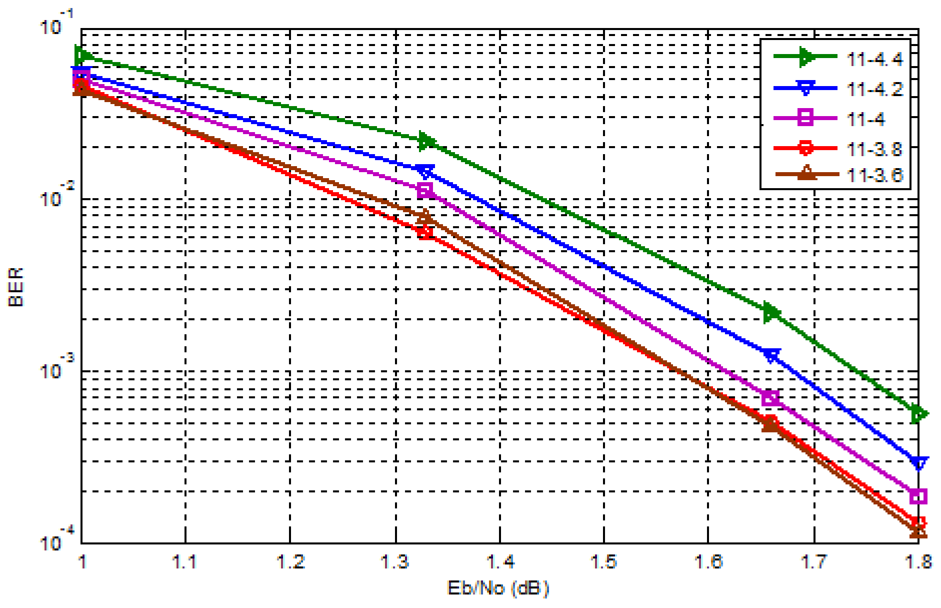
Code Abbreviation	**Variable Node Degree Distribution Polynomial** $\lambda(x)$	**Average Variable Node Degree**	**Number of High Weight Columns**
11-3.6	$0.2x^1 + 0.7x^2 + 0.1x^{10}$	3.6	180
11-3.8	$0.2x^1 + 0.675x^2 + 0.125x^{10}$	3.8	225
11-4	$0.2x^1 + 0.65x^2 + 0.15x^{10}$	4.0	270
11-4.2	$0.2x^1 + 0.625x^2 + 0.175x^{10}$	4.2	315
11-4.4	$0.2x^1 + 0.6x^2 + 0.2x^{10}$	4.4	360
13-3.6	$0.2x^1 + 0.72x^2 + 0.08x^{12}$	3.6	144
13-3.8	$0.2x^1 + 0.7x^2 + 0.1x^{12}$	3.8	180
13-4.0	$0.2x^1 + 0.68x^2 + 0.12x^{12}$	4.0	216
13-4.2	$0.2x^1 + 0.66x^2 + 0.14x^{12}$	4.2	252
13-4.4	$0.2x^1 + 0.64x^2 + 0.16x^{12}$	4.4	288

After constructing the matrices given in Table 3.15, we have simulated their performances. Figure 3.32 and Figure 3.33 show the results of the simulations for the codes with irregularities 2/3/11 and 2/3/13, respectively.

In Figure 3.32, where the performances of 2/3/11 irregular codes with different average variable node degrees w_a are shown, we see that in the range from 3.2 to 3.6, increasing w_a improves the performance. However, if w_a further increases from 3.6 to 4.4, the performances of the codes get worse because of the rapidly increasing number of length-6 cycles.

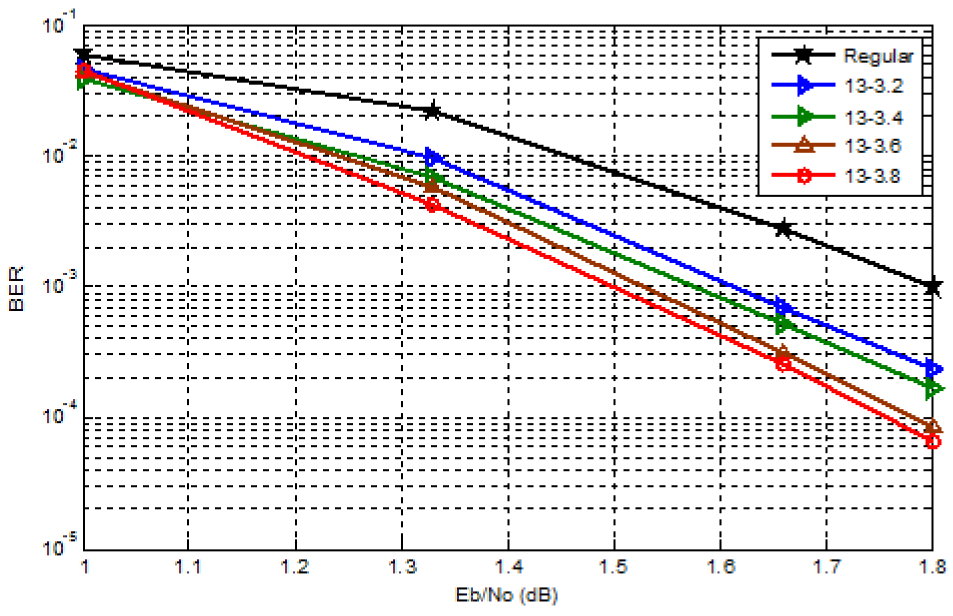


(a)

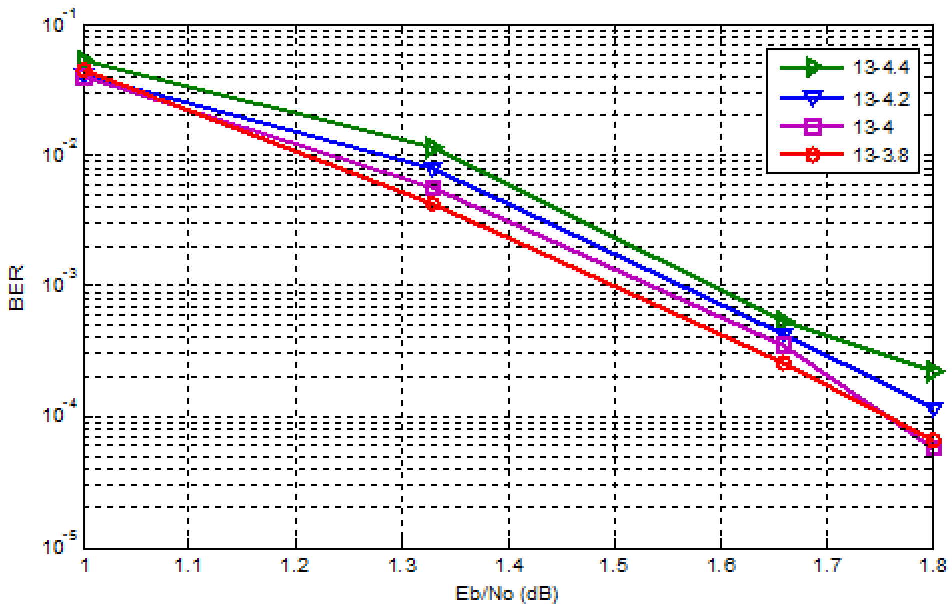


(b)

Figure 3.32 Performances of the 2/3/11 irregular codes. The codes whose performances are getting better with increasing average variable node degree are shown in (a), and the codes whose performances are getting worse are shown in (b).



(a)

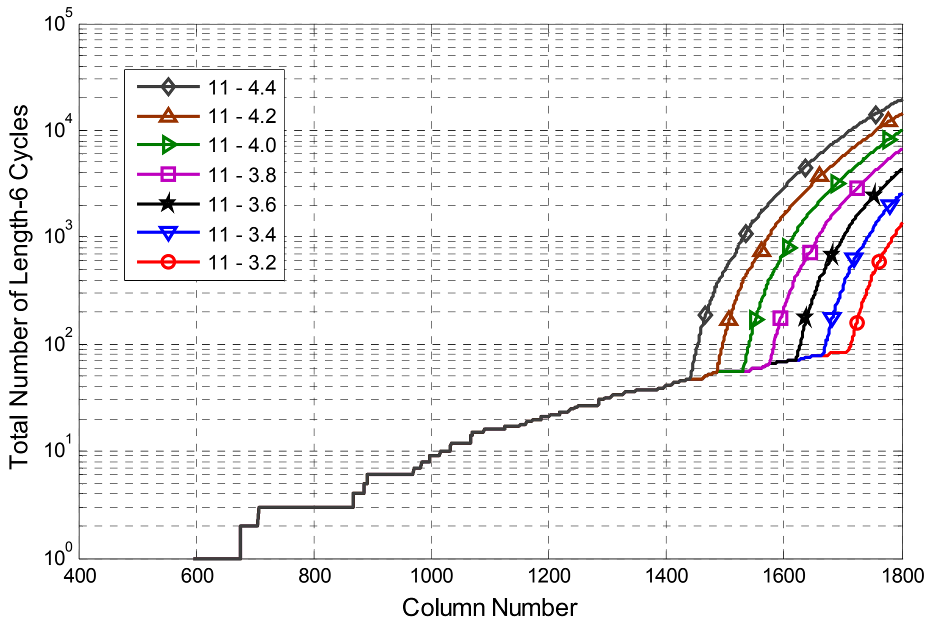


(b)

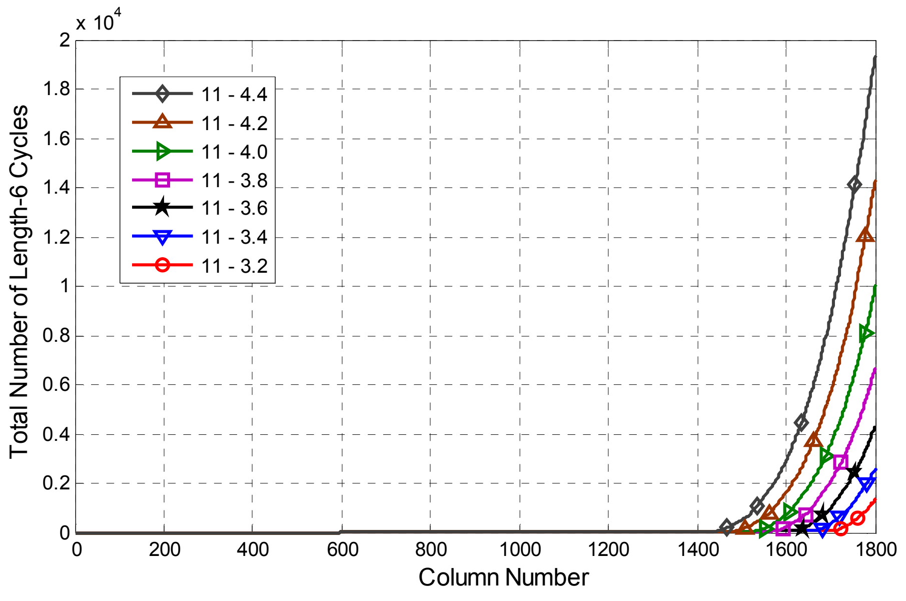
Figure 3.33 Performances of the 2/3/13 irregular codes. The codes whose performances are getting better with increasing average variable node degree are shown in (a), and the codes whose performances are getting worse are shown in (b).

In Figure 3.33, where the performances of the 2/3/13 irregular codes are shown, we see that in the range from 3.2 to 3.8, increasing the average variable node degree w_a improves the performance. However, for w_a growing between 3.8 and 4.4, the performances of the codes get worse because of the rapidly increasing number of length-6 cycles.

These results are quite similar to the case of 2/3/9 irregular codes. In order to compare the rate of increase of length-6 cycles with the previous case, we have counted the number of length-6 cycles at each step of column construction for these new codes. The analyses results for 2/3/11 irregular codes are given in Figure 3.34 and Figure 3.35.



(a)



(b)

Figure 3.34 The change in the total number of length-6 cycles for the 2/3/11 irregular codes in Table 3.15 versus each new generated column. Vertical axis is given in: (a) logarithmic scale (b) linear scale.

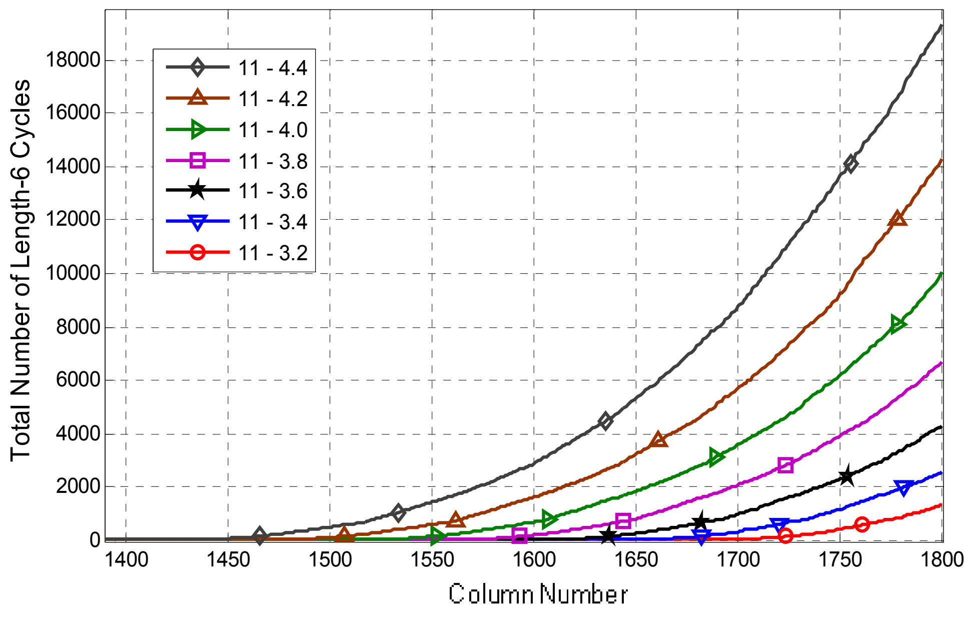
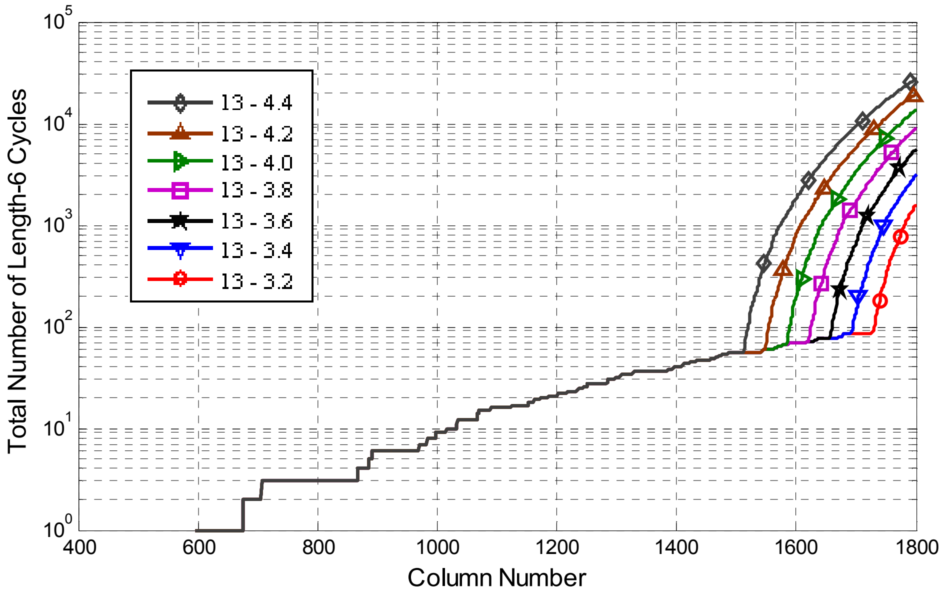


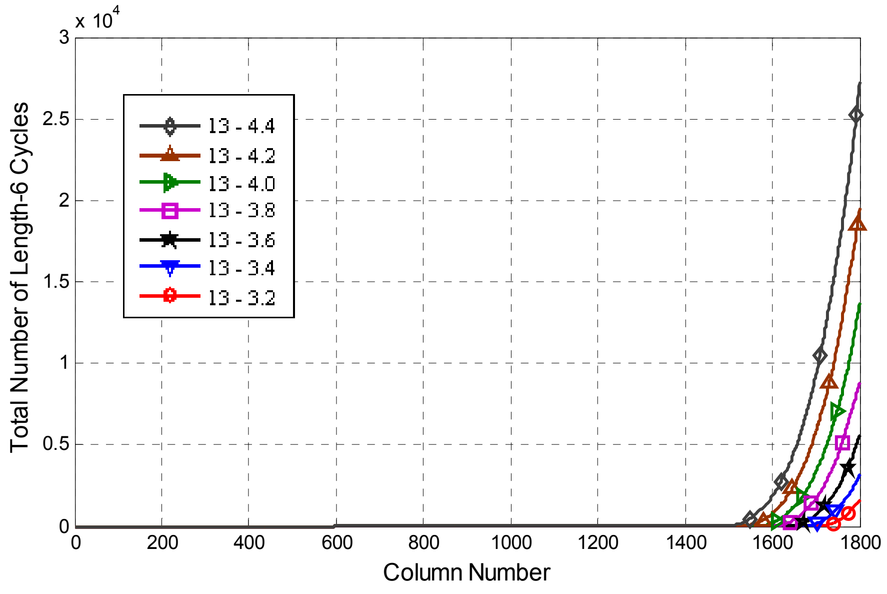
Figure 3.35 Last part of the graph given in Figure 3.34 (b).

In Figure 3.35, we see that the numbers of length-6 cycles of the 2/3/11 irregular codes with average variable node degree $w_a \leq 3.6$, are fairly close to each other. However, for the 2/3/11 irregular codes with w_a larger than 3.6, the total number of length-6 grows very hastily. This rapid increase in the number of length-6 cycles again suppresses the performance improvement supplied by the weight-11 columns and distorts the performance of the codes as shown in Figure 3.32 (b).

Finally, the number of length-6 cycles for the 2/3/13 irregular codes are given in Figure 3.36 and Figure 3.37 for different values of w_a between 3.2 and 4.4.



(a)



(b)

Figure 3.36 The change in the total number of length-6 cycles for the 2/3/13 irregular codes in Table 3.15 versus each new generated column. Vertical axis is given in: (a) logarithmic scale (b) linear scale.

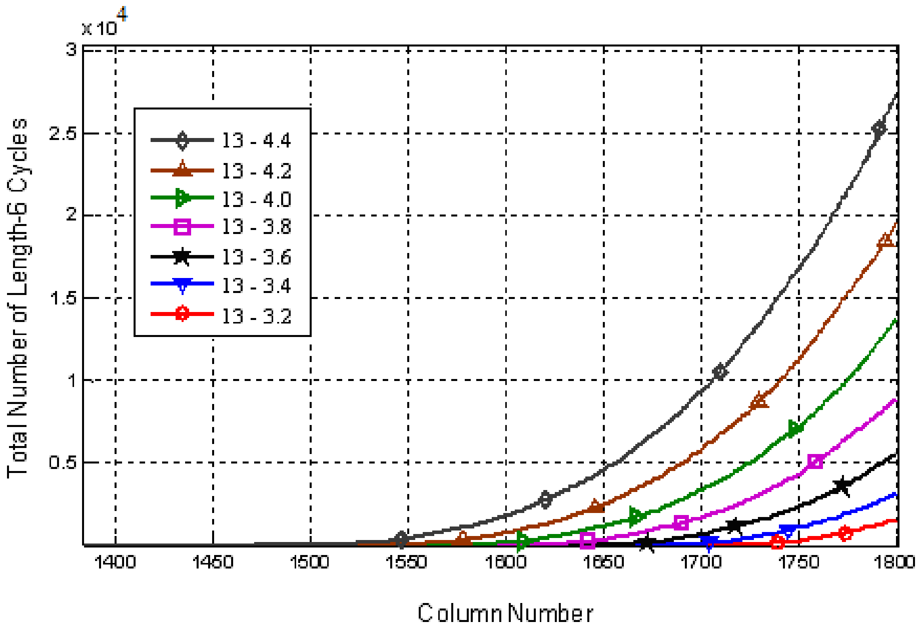


Figure 3.37 Last part of the graph given in Figure 3.36 (b).

Similar to the previous cases, we observe that the rapid increase in the number of length-6 cycles seems to start at some value of w_a, which is around 3.8 for 2/3/13 irregular codes. This explains the performance curves in Figure 3.33, where we identify that the performance improves as w_a grows between 3.2 and 3.8, but further increase of w_a from 3.8 to 4.4 deteriorates the performance. This situation is again similar to the codes with 2/3/9 and 2/3/11 irregularities. However, the critical point for the average variable node degree w_a seems to be around 3.4 for the 2/3/9 codes, 3.6 for the 2/3/11 codes and 3.8 for the 2/3/13 codes. This is understandable, since compared to the weight-9 columns, less number of weight-11 columns are needed to achieve a given value of the average variable node degree, w_a. Similarly, less number of weight-13 columns are needed to achieve a given w_a compared to the weight-9 and weight-11 columns. When small number of high weight columns is added to a matrix, the number of length-6 cycles that it introduces to the code is also small. Therefore, it is a normal result for the codes with weight-13 columns to start losing performance at a w_a value greater than that of the codes with weight-11, and for the codes with weight-11 columns to start losing performance at a w_a value greater than that of the codes with weight-9. Figure 3.38, we sketch the total number of length-6 cycles versus the average variable node degree, w_a, for the mentioned three codes, having the irregularities of 2/3/9, 2/3/11 and 2/3/13.

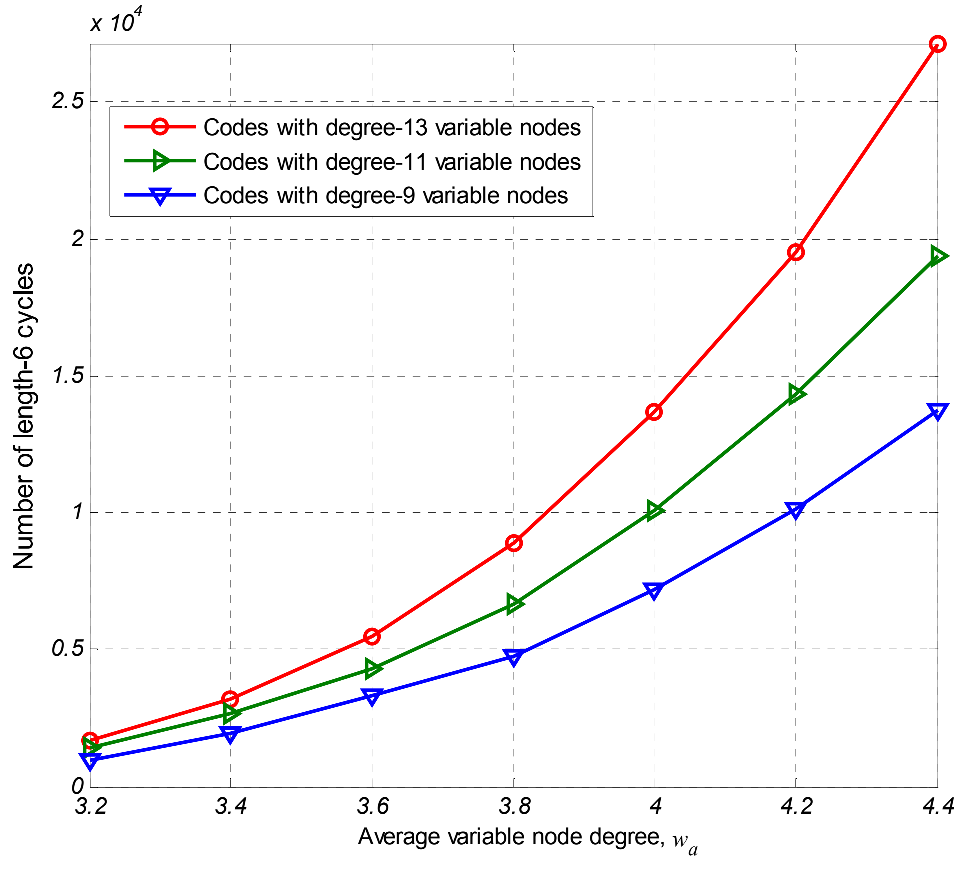


Figure 3.38 Number of length-6 cycles versus average variable node degree, w_a, of 2/3/9, 23/11 and 2/3/13 codes.

To sum up the results we have obtained in this section, we can say that adding high weight columns improves the performance of the LDPC codes up to some point but distorts the performance thereafter, because of the huge increases in the number of length-6 cycles that start to occur. That rapid increase of length-6 cycles suppresses and begins to cancel the decoding improvement brought by the high weight columns of the parity-check matrix.

3.7.3 Some Codes with Fixed Number of Length-6 Cycles

In this section, we investigate the performance of some codes with different but close codeword lengths n and average variable node degrees w_a; but the same number of length-6 cycles. In order to generate such codes, we have used Figure 3.29 (b), where the distributions of the number of length-6 cycles of different 2/3/9 codes are given. For the parent codes 9-4.4, 9-4.2, 9-4, 9-3.8, 9-3.6 and 9-3.4 with w_a's ranging from 3.4 to 4.4 (see Table 3.14), we have noted the number of columns value, say $C_{9\text{-}4.4}$, $C_{9\text{-}4.2}$, $C_{9\text{-}4}$, $C_{9\text{-}3.8}$, $C_{9\text{-}3.6}$, $C_{9\text{-}3.4}$, where the number of length-6 cycles is around 2000. Using this information, we have obtained some new codes whose parity-check matrices are formed by the first $C_{[code\ abbreviation]}$ columns of the parity-check matrices of the original codes. (For example, the first $C_{9\text{-}4.4}$=1531 columns of the parity-check matrix of the parent 9-4.4 code form the parity-check matrix of a new code of length 1531.) The new codes, which contain nearly the same number of length-6 cycles, have codeword lengths ranging from 1531 to 1800 and w_a's between 3.4 and 3.59. Since the number of check nodes remain the same as that of the parent code, the rate k/n of each code is also different but close to 0.5. The parameters of the new codes are given in Table 3.16.

Table 3.16 Parameters of the new codes obtained from the 2/3/9 irregular codes of rate ½ and length 1800, having the number of length-6 cycles around 2000

Abbreviation Used for the New Code and Its Parent Code	**Average Variable Node Degree w_a & Rate k/n for the New Code**	**Size of the New Parity-Check Matrix**	**Variable Node Degree Distribution Polynomial $\lambda(x)$ for the New Code**	**Total Number of Length-6 Cycles N_6**
9-3.4 & 9-3.4	3.40 & 0.5	900×1800 $(C_{9-3.4} = 1800)$	$0.2x + 0.7x^2 + 0.1x^8$	1922
9-3.44 & 9-3.6	3.44 & 0.48	900×1747 $(C_{9-3.6} = 1747)$	$0.205x + 0.686x^2 + 0.109x^8$	2006
9-3.48 & 9-3.8	3.48 & 0.47	900×1695 $(C_{9-3.8} = 1695)$	$0.211x + 0.673x^2 + 0.116x^8$	1998
9-3.51 & 9-4.0	3.51 & 0.45	900×1638 $(C_{9-4.0} = 1638)$	$0.219x + 0.658x^2 + 0.123x^8$	1993
9-3.54 & 9-4.2	3.54 & 0.43	900×1583 $(C_{9-4.2} = 1583)$	$0.225x + 0.645x^2 + 0.130x^8$	1997
9-3.59 & 9-4.4	3.59 & 0.41	900×1531 $(C_{9-4.4} = 1531)$	$0.234x + 0.625x^2 + 0.141x^8$	2007

As can be seen from Table 3.16, the codes with smaller codeword lengths and rates have greater w_a's. We have shown in Section 3.1.2 that increasing codeword length makes the performance of a code better. However, we have also shown in Section 3.3 that increasing the average variable node degree with fixed number of length-6 cycles also increases the performance. In Figure 3.39, we present the performances of the codes given in Table 3.16.

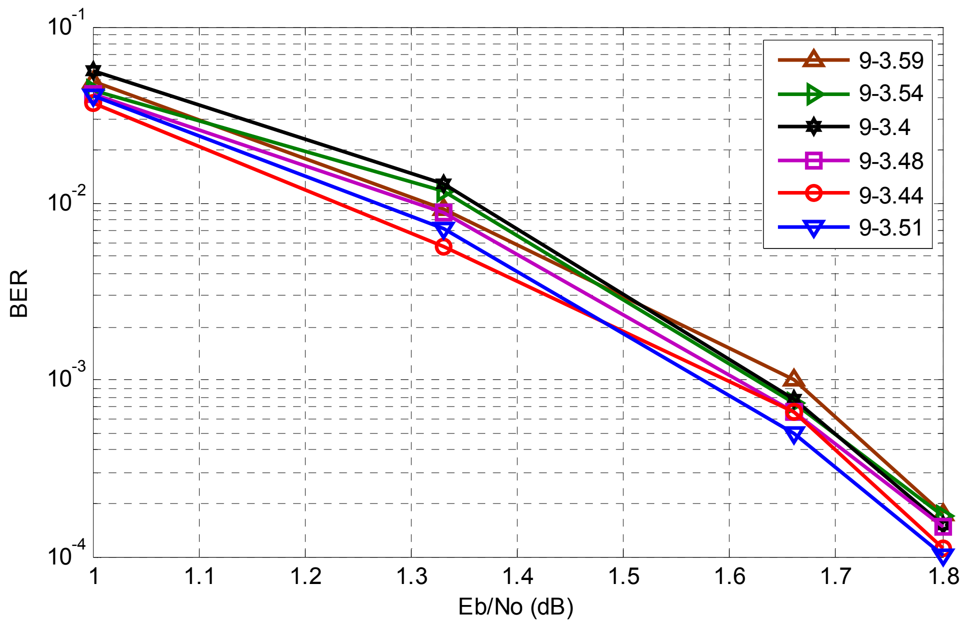


Figure 3.39 Performances of the codes given in Table 3.16.

It is observed that all the codes have nearly the same performance. This may be considered as an expected result. Because, as the codeword length decreases from 1800 to 1531, the performance of the code gets worse; however, the average variable node degree increases from 3.4 to 3.59 (and the rate decreases from 0.5 to 0.41) in parallel, which in turn improves the performance. Combining these effects, it is not surprising to see that the performance of a code with longer codeword length n and lower w_a, can be very similar to the performance of a code with shorter codeword length and greater w_a, whenever their number of length-6 cycles are comparable. In Figure 3.39, one can examine the canceling effects of the codeword length n and the average variable node degree w_a.

3.8 Codes with Very Long Codeword Lengths

In this section, we investigate the performances of codes with very long codeword lengths and compare them with some commercially used LDPC codes. We have selected 16000 bits as the codeword length and constructed a regular (3, 6) code and an irregular 2/3/13 code with average variable node degree w_a=3.8, which was found to be the optimum for the 2/3/13 codes. Figure 3.40 shows the performances of these two codes together with the performance of a regular (64800, 32400) code which is used in DVB-S2 standard.

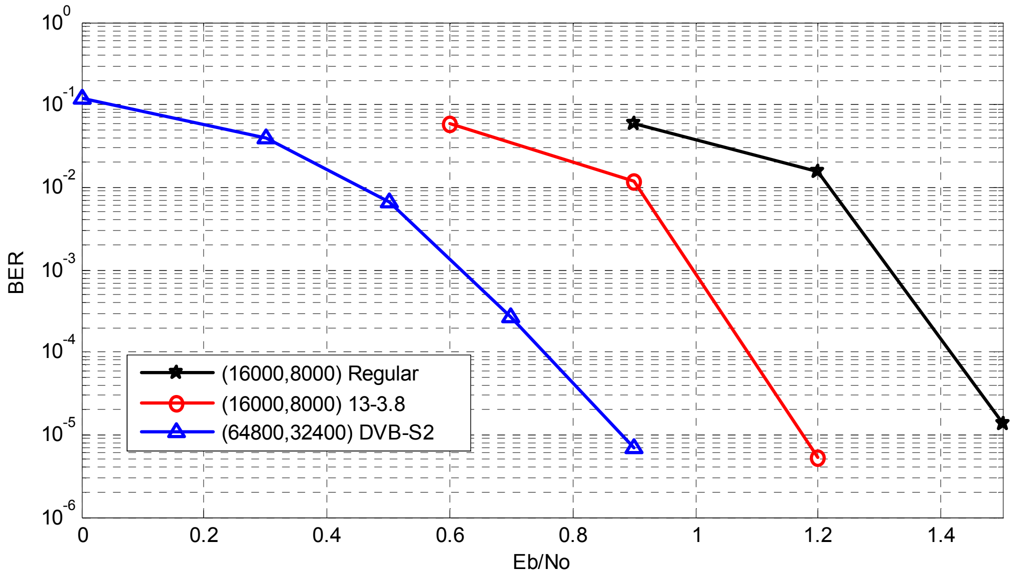


Figure 3.40 Performances of a regular code and an irregular 2/3/13 code of codeword length 16000 together with the performance of a regular code of codeword length 64800 used in DVB-S2 standard

In Figure 3.40, we see that the code with codeword length of 64800 has the best performance. This is normal since there is a great difference in the codeword lengths 64800 and 16000, which is the dominant factor in determining the performance in this case. However, we also see that irregular 2/3/13 code has a performance improvement of nearly 0.35 dB at the BER level of 10^{-3} compared to the regular code of the same codeword length.

Considering these, we can say that, for very long codeword lengths, our irregular code design leads quite an improvement in the performance. This performance improvement helps to approach the performances of commercially used LDPC codes.

3.9 Decoding Times for Regular and Irregular LDPC Codes

At each iteration of decoding, calculations are done according to the distribution of the 1's in the parity-check matrix. In this work, we have shown that the irregular codes with average variable node degrees greater than 3 ($w_a > 3$) have better performances than the regular (3, 6) codes. However, greater w_a means that there exist more 1's in the parity-check matrix, which implies that more calculations are done at each iteration. In this section, we investigate the decoding times needed for the regular and irregular codes that we have designed and used in this work.

In order to compare the decoding times needed for regular and irregular LDPC codes, we have done decoding simulations for four different LDPC codes of codeword length 1800 and rate ½. We have used a regular code, an irregular 2/3/11 code with w_a=3.6, an irregular 2/3/13 code with w_a=3.8 and an irregular 2/3/19 code with w_a=4. For each of the three irregular codes, we have kept the ratio of degree-2 nodes as 20%, and the relative distribution of degree-3 and higher degree nodes are adjusted according to the specific w_a value. In the simulations, we have sent 50000 words at the E_b/N_o value of 1.6 dB. The value of I_{max} is set to 50. The results of the simulations are given in Table 3.17.

Table 3.17 Simulation results for the decoding times of sample regular and irregular codes

Code Abbreviation	Number of Successes	Number of Failures	Total Number of Iterations	Average Time Spent for one Iteration (seconds)	Total Decoding Time (hours)	Total Decoding Time Relative to Regular
Regular	46173	3827	900513	0.0391	9.7765	100%
11-3.6	49273	727	726086	0.0413	8.3225	85%
13-3.8	49588	412	683642	0.0419	7.9630	81%
19-4	49817	183	627862	0.0427	7.4493	76%

In Table 3.17, we see that the average time spent for each iteration increases with increasing average variable node degree. This can be considered as a normal result since more calculations are done at each step for the codes with greater w_a's. However, we also see that, with increasing w_a, the number of successes increases, whereas total decoding time and total number of iterations decreases.

In **[MacKay-2005]**, decoding time for an LDPC is code is said to be proportional to the number of operations done at each iteration. Simply, we may think each iteration as two steps. At first step, the variable nodes send information to the check nodes and at the second step the check nodes send information back to the variable nodes. For a code of length n and rate R, the first step includes $\frac{1}{R} \times w_a$ operations per variable node, so $\frac{1}{R} \times w_a \times n$ operations in total and the second step includes w_a operations per check node so $w_a \times n \times (1 - R)$ operations in total. Therefore, at each decoding iteration, $\frac{1 + R - R^2}{R} \times n \times w_a$ operations are done. In our case, the code length and the rate are constant. Hence, one may expect that the average time spent for one iteration is proportional to w_a. In Table 3.17, we see that average time spent for one iteration is 0.0391 for the regular code for which w_a =3. If we consider the 2/3/13 code with w_a =3.8 (abbreviated as 13-3.8), we expect that the average time spent for one iteration is $\frac{3.8}{3} \times 0.0391 = 0.0495$. However, in Table 3.17, we see that this value is actually 0.0419. This may be as a result of our implementation details. In our implementation of the decoding algorithm, many operations are done at a time using array structures. When the number of operations increases, the size of the array that contains these operations gets larger. Because of that, the time spent for a specific number of operations is not directly proportional to the number of equations but slightly less than that.

As can be seen from the number of successes and failures, the code with the best performance among the codes given in Table 3.17 is the 2/3/19 code with w_a=4. Although the average time spent for each iteration is the greatest for this code, total decoding time is the smallest. Figure 3.41, which shows the iteration histograms for the codes, will be helpful to understand the reason for this.

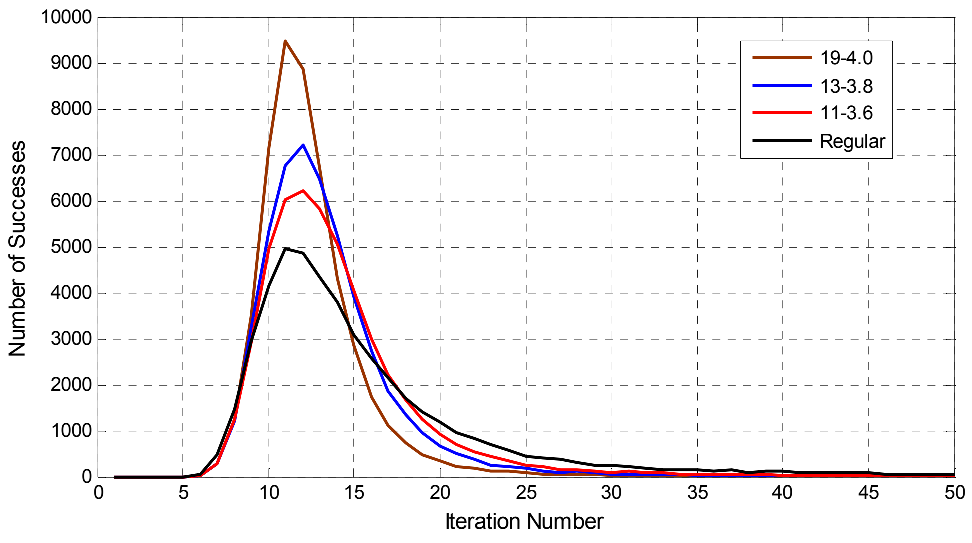


Figure 3.41 Iteration histograms for the regular and irregular codes

In Figure 3.41, we see that that a success is more likely to occur at the 11th or 12th iteration. As I_{max} is set to 50, all decoding failures occur at the 50th iteration. Since the total number of successes is the greatest and the number of failures is the smallest for the 2/3/19 code, it is quite normal that the total number iterations and the total decoding time for this code is the smallest.

Considering all the results, we can say that the average time spent for each iteration is greater for the codes with greater w_a's. However, as these codes with greater w_a's are designed to have better performances, the total number of iterations to decode same number of words and the total decoding times are smaller for these codes.

3.10 Random Distribution of Check Node Degrees

As a final consideration, we will discuss the check node degree distributions of the irregular codes that we generate for this work. Random construction algorithms we have used, do not pay any attention to the check node degrees. Instead, they take the required values of n, k and the given variable node degree polynomial $\lambda(x)$ as input and arrive at random number of variables entering each parity-check equation. So, the check node degrees are established randomly as a result of the 2A or pseudo-random construction algorithm. In this section, we explore the check node degree distributions for some of the generated codes. As representatives of three different groups, we choose **i)** five 2A codes of length 1200, **ii)** four 2/3/4 codes of length 576, and **iii)** four 1/3/5 codes of length 576. In each case, we construct the codes according to the desired variable node degree distribution polynomials, count the frequency of resulting check node degrees and compare the check node degree distributions within the group.

i) Irregular 2A Codes

Table 3.18 shows the number of check nodes at each degree (i.e., the number of rows at each weight) for different codes with irregular 2A matrices of codeword length 1200 and average variable node degrees w_a (i.e., the column weights), from 2.95 to 2.75.

The check node degree distributions given in Table 3.18 are sketched in Figure 3.42. Notice that although the check node degrees are distributed randomly, their average value (see the last column) is equal to $2w_a$, because the parity-check matrix is of size 600×1200.

Table 3.18 Number of rows at each weight for 600×1200 parity-check matrices of 2A irregular codes, where w_a shows the average column weight.

	Row Weight						**Average Row Weight**
w_a	**2**	**3**	**4**	**5**	**6**	**7**	
2.95	-	-	12	114	396	78	5.9
2.90	2	8	45	118	307	120	5.8
2.85	5	11	57	142	256	129	5.7
2.80	5	27	61	155	214	138	5.6
2.75	6	29	78	154	212	121	5.5

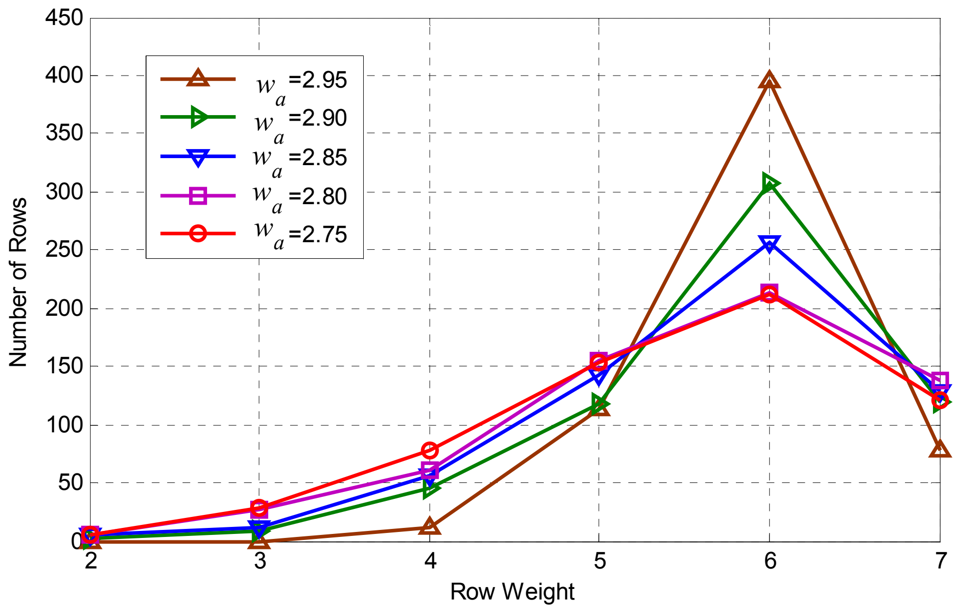


Figure 3.42 Check node degree distributions given in each row of Table 3.18.

In Figure 3.42, it can be seen that the check node degrees of the codes are concentrated around the degree value of 6, which shows that the check node degree distributions of the matrices are nearly uniform. For codes with decreasing w_a, the number of nodes with degree 6 decreases because of the decreasing number of 1's in the parity-check matrices.

ii) Irregular 2/3/4 Codes

Table 3.19 shows the number of check nodes at each degree (i.e., the number of rows at each weight) for different codes with irregular 2/3/4 matrices of codeword length 576 and average variable node degrees percentages of the degree-2 and degree-4 variable nodes of the code. The check node degree distributions given in Table 3.18 are sketched in Figure 3.43.

Table 3.19 Number of rows at each weight for 288×576 parity-check matrices of the 2/3/4 irregular codes

Percentage of Degree-2 and Degree-4 Variable Nodes	**Row Weight**					**Average Row Weight**
	3	**4**	**5**	**6**	**7**	
5 %	2	2	11	250	23	6.0069
10 %	2	5	10	244	27	6.0035
20 %	2	5	9	245	27	6.0069
33 %	1	4	13	244	26	6.0069

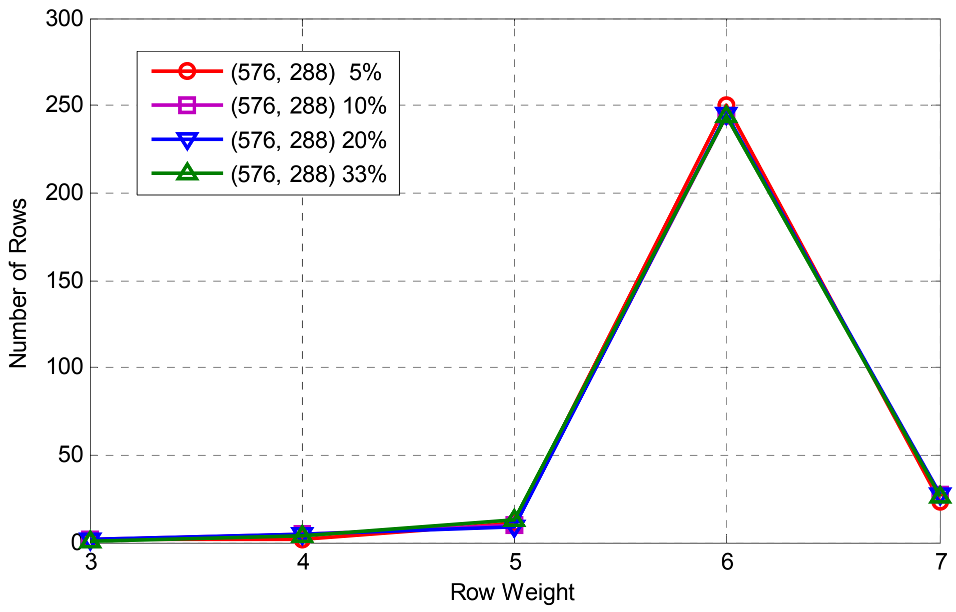


Figure 3.43 Check node degree distributions given in each row of Table 3.19.

In Figure 3.43, it can be seen that the check node degrees of the codes are again concentrated around the degree value of 6, which shows that the check node degree distributions of the matrices are nearly uniform. All of the codes have almost the same check node degree distribution since they have the same number of 1's in their parity-check matrices. Since the average variable node degree w_a is 3 for all the codes of this group, average check node degree is very close to 6.

iii) Irregular 1/3/5 Codes

Table 3.20 shows the number of check nodes at each degree (i.e., the number of rows at each weight) for different codes with irregular 1/3/5 matrices of codeword length 576 and average variable node degrees percentages of the degree-1 and degree-5 variable nodes of the code. The check node degree distributions given in Table 3.20 are sketched in Figure 3.44.

Table 3.20 Number of rows at each weight for 288×576 parity-check matrices of the 1/3/5 irregular codes

Percentage of Degree-1 and Degree-5 Variable Nodes	Row Weight					Average Row Weight
	3	4	5	6	7	
5 %	1	5	10	245	27	6.0139
10 %	1	2	15	246	24	6.0069
20 %	2	2	13	244	27	6.0139
33 %	2	3	12	243	28	6.0139

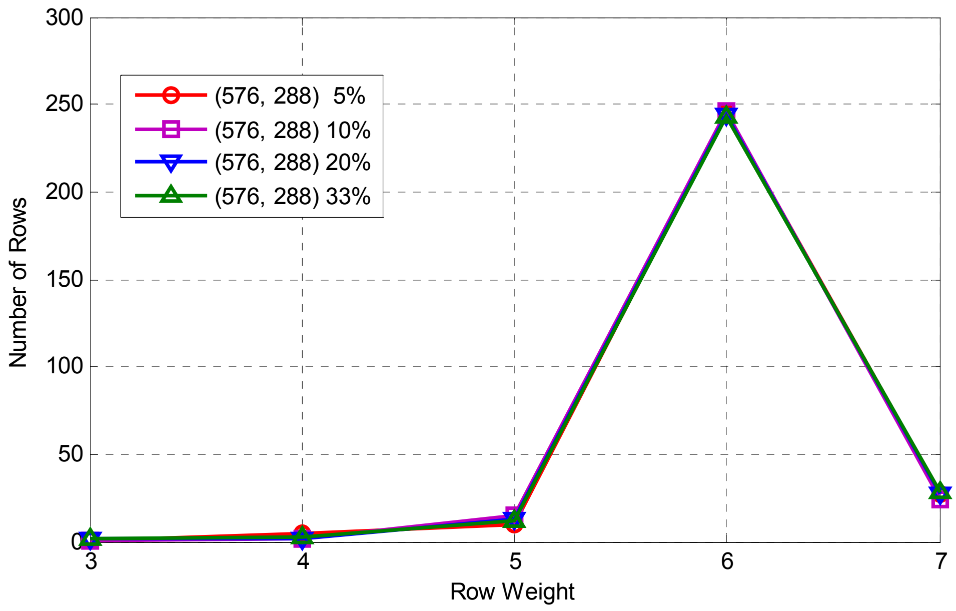


Figure 3.44 Check node degree distributions given in each row of Table 3.20.

In Figure 3.44, it can be seen that the check node degrees of the codes are concentrated around the degree value of 6, which shows that the check node degree distributions of the matrices are nearly uniform. All of the codes have almost the same check node degree distribution since they have same number of 1's in their parity-check matrices. It is also interesting to observe that both of the 2/3/4 and 1/3/5 pseudo-randomly constructed codes have more impulse-like distribution for the check node degrees as compared to the 2A construction.

3.11 Summary

Through simulations performed in this work on the performance of LDPC codes, the conditions that lead to “wrong codewords” at the output of the message-passing decoder are investigated in Section 3.2. When the “maximum number of iterations” parameter $I_{\max}$ of the decoder is set to 50, all errors contributing to the output BER come from the decoding failures for the codes with length n larger than 200, and there is no wrongly decoded codeword at all. On the other hand, when $n< 200$, one may have a few wrong codewords at the decoder output. For short code lengths such as 50 or 100, the probability of having wrong codewords decreases if $I_{\max}$, which is initially chosen as 50, is set to a smaller value, like 20. Since $d_{\min}$ values of short codes are quite small (say as small as 5 for a regular (3, 6) code of length 50), the decoding algorithm may possibly force itself to decide on a wrong codeword in 20 iterations which is quite large as compared to $d_{\min}$; and this probability is even higher when $I_{\max}$ is set to 50. However; in real cases, where the code length n and minimum distance $d_{\min}$ are much larger than the maximum number of iterations parameter of the decoding algorithm, $I_{\max}$, the probability of a wrongly decoded codeword approaches to 0.

In Section 3.3, we have compared the performances of the (1200, 600) irregular 2A codes which have average variable node degree values slightly less than 3 with that of the regular (3, 6) code. We have seen the surprising result that the irregular codes with the variable node degree distribution polynomial $\lambda_5(x) = 0.05x + 0.95x^2$ (hence the average variable node degree w_a=2.95) have slightly better performance than the regular codes with the variable node degree distribution polynomial $\lambda(x) = x^2$, (hence the variable node degree 3). In order to explain this observation, we have counted the number of length-6 cycles (N_6) of several codes. We have seen that the codes with w_a = 2.95 have in general less N_6 than that of the regular (3, 6) code. In order to be sure that this is the reason for better performance, we have generated many different irregular codes with w_a = 2.95, and many different regular codes of variable node degree 3. Comparing the performance of all these codes, we have found that the code with less N_6, has always better performance. Since there are less number of 1's in the parity-check matrix of an irregular code with $w_a = 2.95$, it is a greater probability that the irregular code has less N_6 than that of a regular (3,6) code. This is the reason for irregular 2A codes with $w_a = 2.95$ to have better performance in most of the cases.

In the remaining sections, we have investigated the performance of pseudo-randomly generated (1800, 900) irregular codes with different variable node degree distribution polynomials. In Sections 3.4 and 3.5, we have generated irregular codes that have average variable node degree 3. The 2/3/4 codes, for which the number of variable nodes of degree-2 is chosen to be equal to number of variable nodes of degree-4 in order to have $w_a = 3$, are shown to have a performance that is almost the same as that of the regular codes of the same length. Then, we have used degree-1 and degree-5 variable nodes instead of the degree-2 and degree-4 variable nodes, to generate 1/3/5 irregular codes. We have noticed that there exists an unacceptable number of wrong codewords at the decoder output, resulting from degree-1 nodes. So we concluded that degree-1 nodes (meaning variable nodes connected to single check equations) should never be included in the design of powerful LDPC codes.

We have then generated (1800, 900) irregular codes with average variable node degrees greater than 3 and investigated their performance in Sections 3.6 and 3.7. In order to have average variable node degrees greater than 3, we have added high weight columns to the parity-check matrices to construct 2/3/9, 2/3/11, 2/3/13 and 2/3/19 irregular codes. We have found that the codes with w_a greater than 3 have better performances with increasing w_a up to some level; however, further increase in w_a distorts the performance. To explain the reason behind this performance loss, we have counted the number of length-6 cycles, N_6, of the 2/3/9, 2/3/11 and 2/3/13 codes that are generated. We have observed that up to some value of the average variable node degree w_a, the number of length-6 cycles of the codes remain fairly close to each other. However, for larger w_a, huge increases in the number of length-6 cycles suppress the improving effect of the high degree variable nodes and distorts the code performance. Therefore, the LDPC code designers should consider the trade-off between the average variable node degree and the number of length-6 cycles and explore the optimum value of w_a, which may be different for different irregularities. For the example codes of this work, the best performing 2/3/9 irregular code has the average variable node degree w_a =3.4, the best performing 2/3/11 irregular code has w_a =3.6 and the best performing 2/3/13 irregular code has w_a =3.8. This is understandable, since as compared to the weight-9 columns, less number of weight-13 columns are needed to reach to w_a = 3.8. Because the contribution of less number of high degree variable nodes to N_6 is smaller, the optimum w_a for the 2/3/13 codes is greater than that of the 2/3/9 codes. In Section 3.7.3, some codes with the same $N_{6,}$ but slightly different values of n and w_a are compared and it is seen that their performances are nearly identical.

Then, in order to see whether our irregular code design method leads to capacity approaching performances for very long codeword lengths, we have designed a 2/3/13 irregular code of length 16,000 bits with average variable node degree w_a=3.8. We have compared the performance of this code with a commercially used code of length 64,800 and observed that the performance of our 2/3/13 irregular code is very close to that of the commercial one.

We have also measured the decoding times for a regular, and 2/3/11, 2/3/13 and 2/3/19 irregular codes of length 1800. We have observed that the average time spent for one iteration is almost proportional to w_a. However, the time spent for decoding the same number of codewords decreases with increasing average variable node degree. We have seen that at the time spent for decoding 50,000 codewords at E_b/N_o=1.6 dB for the 2/3/13 irregular code with w_a=3.8 is 19% less than that of the regular code. The main reason for this is found to be the smaller number of decoding failures obtained for the irregular code.

As a final consideration, we have discussed the check node degree distributions of some irregular codes that are used in this work, and shown that the utilized construction methods create check node degree distributions, which remain close to uniform, as long as the desired variable node degree distributions fed to the construction algorithm are close to uniform.

CHAPTER 4

CONCLUSIONS

In this work, performances of randomly generated regular and irregular binary LDPC codes are investigated and the effects which improve or deteriorate the performance are analyzed. The performance of the codes, all of which are constructed as rate ½ codes that are free from length-4 cycles, are studied using "BER versus SNR" curves obtained by the belief propagation decoding algorithm that employs the log-likelihood function.

Using an optimal decoding algorithm, the performance of a regular LDPC would be better with increasing variable node degree, w_v. However, a code with large w_v has a dense Tanner graph in which the belief propagation algorithm makes poor progress. Therefore, one expects that the optimum w_v value is small. In fact, in **[MacKay-2005]**, it is shown that the optimum value of w_v is 3. Considering this fact, we have constructed all the regular matrices that we have used in this work with $w_v = 3$. Since the rate of the codes are ½, their check node degrees are $w_c = 6$. The irregular codes that we generate also have average variable node degrees w_a close to 3.

We have observed that, for short codes with small $d_{\min}$ values, the decoder seldomly decides on a wrong codeword, if the number of decoding iterations is sufficiently larger than $d_{\min}$. So, as a handy rule of thumb, we conjecture that *wrong codewords do not occur if the maximum number of iterations parameter of the decoding algorithm*, $I_{\max}$, *is less than one tenth of the block length*. In real cases, where the code length n and minimum distance $d_{\min}$ are much larger than $I_{\max}$, the probability of a wrongly decoded codeword approaches to 0. Useful LDPC codes are chosen very long for excellent performance, so practically all decoding errors come from decoding failures.

We have compared the performances of many regular (3, 6) and irregular 2A codes defined in **[MacKay-Neil-1996]** with $w_a = 2.95$. In all cases, we have seen that the codes with less number of length-6 cycles have better performance, independent of the regular or irregular structure. As compared to the regular (3, 6) code, there are less number of 1's in the parity-check matrix of an irregular code with $w_a = 2.95$; moreover, degree-2 variable nodes of the irregular 2A codes are designed in such a way that they do not cause any length-6 cycles. Therefore, it is a greater probability that the number of length-6 cycles of the irregular 2A code is less than that of a regular (3, 6) regular code, which is the reason for the better performance of the irregular code in most cases.

Knowing that the best performance of regular LDPC codes are obtained for $w_v = 3$, we have constructed irregular codes that have average variable node degree $w_a = 3$. In case of the 2/3/4 codes, where the variable nodes are of degrees 2, 3 and 4, we have seen that the performances of the irregular codes are almost the same as regular codes. In case of the 1/3/5 codes, we have observed that wrong codewords occur at the decoder output. We have also noticed that increasing number of degree-1 nodes increases the number of wrong codewords at the decoder output. Because, degree-1 variable nodes rely only on the information coming from a single check node; and it is quite probable in the decoding process that they lead to wrong codewords. Since all 2/3/4 codes have nearly the same BER performance, we conclude that there is no use in adding degree-4 columns to an irregular 2A code of average node degree 2.95. One should also avoid variable nodes of degree-1 in LDPC code design.

As we could not obtain a performance improvement for irregular codes with $w_a = 3$, we decided to increase the value of w_a. In the previous parts of this work, we have shown that carefully designed weight-2 columns of a parity-check matrix lower the number of length-6 cycles and therefore improve the performance. So, we have included a fixed number of carefully designed weight-2 columns in the parity-check matrices of the irregular codes and constructed the remaining columns as weight-3 and higher-weight columns in order to have 2/3/*i* codes (where *i*=9, 11, 13, 19) with a desired w_a value. For $w_a > 3$, we have seen that increasing w_a up to some critical level improves the performance, but further increase of w_a deteriorates the performance. To explain the reason for this behavior, we have counted the number of length-6 cycles of the codes and shown that up to that critical value of w_a, the number of length-6 cycles increase slowly with increasing w_a. However, for larger w_a, we have noticed an exponential increase in the number of length-6 cycles, which suppresses the improvement brought by high degree variable nodes. This w_a value, which may be called *the optimum average variable node degree*, changes with the weight of the higher weight columns in the parity-check matrix. For the 2/3/9, 2/3/11 and 2/3/13 codes of this work, the optimum w_a's have been shown to be around 3.4, 3.6 and 3.8, respectively. This increase in critical w_a is normal, since the 2/3/*i* code with larger *i* requires less number of high-degree columns in the parity-check matrix to arrive at the same w_a, which in turn contributes less to the total number of length-6 cycles.

It is not right to pay more attention to the number of length-6 cycles than it deserves. For example, at variable node degrees of 3.4 and 3.6, even though the 2/3/19 codes have more cycles of length-6, they have better performance than the 2/3/9 codes. This is normal, since the number of length-6 cycles is not a primary comparison factor for the codes with different parameters.

In order to see the whether our design methods work well to approach capacity for very long codeword lengths, we have designed an 2/3/13 irregular code of length 16,000 bits. We have observed that the performance of the 2/3/13 irregular code is very close to that of a commercial DVB-S2 code of length 64,800.

We have also measured the decoding times for some regular and irregular codes of codeword length 1800. We have observed that the decoding time per iteration increases with increasing average variable node degree, w_a, as expected. However, we have also seen that the total time spent for a fixed number of codewords is much less in the case of irregular codes, having high average variable node degrees. For example, at E_b/N_o=1.6 dB, 50,000 codewords of 2/3/11 2/3/13 and 2/3/19 irregular codes having w_a values of 3.6, 3.8 and 4 respectively, are decoded in 85%, 81% and 76% of the time required for the regular (3, 6) code. This is mainly because of the higher number of successes that are arrived at the 11th-12th iterations, and smaller number of decoding failures that can only be decided upon at the last (i.e., 50th) iteration of the decoding algorithm.

To sum up, one can improve the performance of an irregular LDPC code by avoiding weight-1 columns, using a small percentage of carefully designed weight-2 columns in the parity-check matrix and increasing the average variable node degree up to some optimum value depending on the structure of the code. The performance of an irregular code with the optimum average variable node degree may approach the capacity for very large codeword lengths. Also, we measure the decoding times for strong irregular codes as much less than that of the regular code with similar parameters.

Future work may incorporate the design of irregular codes with special variable node degree polynomials having smaller number of length-6 cycles than those presented in this work, having much larger average variable node degrees.

REFERENCES

- **[Aly-2008]** S.A. Aly, “A Class of Quantum LDPC Codes Constructed From Finite Geometries”, IEEE Global Telecommunications Conference, 2008.

- **[Bonello-2008]** N. Bonello, S. Chen, and L. Hanzo, “Multilevel Structured Low-Density Parity-Check Codes” IEEE International Conference on Communications, 2008.

- **[Chung-Forney-Shokrollahi-Urbanke-2001]** S.-Y. Chung, G. D. Forney Jr., T. Richardson, and R. Urbanke, “On the design of low-density paritycheck codes within 0.0045 dB of the Shannon limit,” IEEE Communications Letter, vol. 5, no. 2, pp. 58–60, Feb. 2001.

- **[Chung-Richardson-Urbanke-2001]** S. Y. Chung, T. J. Richardson, R. L. Urbanke, “Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation”, IEEE Transactions on Information Theory, Vol. 47, No. 2, February 2001.

- **[Davey-1999]** M. C. Davey, “Error-correction using Low-Density Parity-Check Codes.”, Univ. of Cambridge PhD dissertation, 1999.

- **[Davey-MacKay-1998]** M. C. Davey and D. J. C. MacKay, “Low density parity check codes over GF(q).”, IEEE Communications Letters 2 (6): 165{167, 1998.

- **[Elias-1955]** P. Elias, “Coding for Noisy Channels”, The 3rd London Symposium , pp. 61-76, Sep. 1955.

- **[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.

- **[Goethals-1974]** P. Delsarte and J. M. Goethals, Alternating Bilinear Forms over GF(q), J, Combinatorial Theory, Series A Vol. 19, 26-50, 1975.

- **[Goppa-1982]** V. D. Goppa "Algebraico-Geometric Codes", Math. USSR Izoestiya, vol. 21, pp. 75-91, 1983.

- **[Johnson-2008]** S.J Johnson, S.R Weller, "Combinatorial Interleavers for Systematic Regular Repeat-Accumulate Codes", IEEE Transactions on Communications, 2008.

- **[Kerdock-1972]** A. M. Kerdock "A Class of Low-Rate Nonlienar Binary Codes", Inform. Control Vol.20, 182-187, 1972.

- **[Kou-2000]** Y. Kou, S.Lin, M.P.C. Fossorier, "Low density parity check codes: construction based on finite geometries" IEEE Global Telecommunications Conference, 2000.

- **[Krishnan-2007]** K. M. Krishnan, R. Singh, L. S. Chandran, P. Shankar, "A Combinatorial Family of Near Regular LDPC Codes", IEEE International Symposium on Information Theory, 2007.

- **[Leiner-2005]** B. M. J. Leiner, " LDPC Codes – A Brief Tutorial", 2005.

- **[Luby-Mitzenmacher-Shokrollahi-Spielman-1998]** M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low-density codes and improved designs using irregular graphs.", In Proc. 30th Annu. Sym. Theory of Computing, pages 249–258, 1998

- **[MacKay - 2005]** D. J. C. MacKay, "Information Theory, Inference, and Learning Algorithms", Cambridge University Press, 2003.

- **[MacKay-1999]** D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices", IEEE Transactions on Information Theory, Vol. 45, No. 2, March 1999.

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

- **[Mackay-Wilson-Davey-1999]** D. J. C. Mackay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes", IEEE Transactions on Communications, 47(10):1449-1454 October 1999.

- **[Moinian-2006]** A. Moinian, B. Honary, E. Gabidulin, "Generalized quasi-cyclic LDPC codes for wireless data transmission", IET International Conference on Wireless, Mobile and Multimedia Networks, 2006.

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

- **[Richardson-Shokrollahi-Urbanke-2001]** T. J. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low-density parity check codes", IEEE Trans. Inform. Theory, vol. 47, no. 2, pp. 619–637, Feb. 2001.

- **[Richardson-Urbanke-2001]** T. J. Richardson and R. Urbanke, “Efficient Encoding of Low-Density Parity-Check Codes”, IEEE Trans. Inform. Theory, vol. 47, no. 2, Feb. 2001.

- **[Shannon-1948]** C. E. Shannon, “A mathematical theory of communication," Bell System Technical Journal, vol. 27, pp. 379{423, 1948.

- **[Shokrollahi-2003]** A. Shokrollahi, “LDPC Codes: An Introduction”, Digital Fountain, Inc. April 2, 2003.

- **[Swanson-1988]** L. Swanson, “A new code for Galileo.”, In Proc. 1988 IEEE International Symposium Info. Theory, pp. 94{95, 1988.

- **[Tanner-1981]** R. M. Tanner, “A recursive approach to low complexity codes,” IEEE Trans. Inform. Theory, vol. 27, no. 5, pp. 533–547, Sept. 1981.

- **[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.

- **[Uzunoğlu-2007]** C. Uzunoğlu, “Performance Comparison of Message Passing Algorithms for Binary and Non-Binary Low Density Parity Check (LDPC) Codes”, MSc. Thesis in Electrical and Electronics Engineering Department, METU, 2007.

- **[Xia-He-Xu-Cai-2008]** D. Xia, H. He, Y. Xu, Y. Cai, “A Novel Construction Scheme with Linear Encoding Complexity for LDPC Codes”, 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2008.

APPENDIX A

LDPC CODE CONSTRUCTION SOFTWARES

Regular Code Construction Software

Our software that constructs regular LDPC codes by the MacKay and Neil's regular LDPC code construction method takes the size $m \times n$ and the variable node degree w_v as the input parameter, and gives the parity-check matrix H of an regular (w_v, w_c) LDPC code of rate *m/n* at the output. The algorithm of constructing the parity-check matrix is given below

- Starting from the leftmost column, one-by-one construct weigth-3 columns.
- When constructing a new column, select the positions of the 1's of the column from the rows with weights smaller than the desired row weight w_c.
- When constructing a new column, make sure that number of overlaps between any two columns of the entire parity-check matrix no greater than 1.

Irregular 2A LDPC Code Construction Software

Our software that constructs irregular LDPC codes by the 2A method takes the size n and the average variable node degree w_a as the input parameters, and gives the parity-check matrix H of an irregular LDPC code of rate ½ at the output. The algorithm of constructing the parity-check matrix is given below

- Using the size of the matrix and the desired average node degree, calculate the number of weight-2 columns, which cannot exceed $n/4$ for rate ½ codes.

- Construct the weight-2 columns of the parity-check matrix such that there is zero overlap between any pair of columns
- Make the remaining columns with weight-3, with weight per row as uniform as possible, and number of overlaps between any two columns of the entire parity-check matrix no greater than 1.

Pseudo-Random Irregular Code Construction Software

Our software that constructs irregular LDPC codes by MacKay and Neil's pseudo-random construction method takes the size *n,* the average variable node degree w_a, and the desired variable node degree values as the input parameters, and gives the parity-check matrix *H* of an irregular LDPC code of rate ½ at the output. The algorithm of constructing the parity-check matrix is given below.

- Using the size of the matrix and the desired average node degrees, calculate the number of columns with each weight.
- Starting from the leftmost column, construct the smallest weight columns first, then construct the columns with next greater degree and continue this process until the last column with the greatest weight is constructed.
- When constructing a column, make sure that the number of overlaps between any two columns of the entire parity-check matrix no greater than 1.