PERFORMANCE COMPARISON OF MESSAGE PASSING DECODING ALGORITHMS FOR BINARY AND NON-BINARY LOW DENSITY PARITY CHECK (LDPC) CODES

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

BY

CİHAN UZUNOĞLU

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

DECEMBER 2007

Approval of the thesis:

PERFORMANCE COMPARISON OF MESSAGE PASSING DECODING ALGORITHMS FOR BINARY AND NON-BINARY LOW DENSITY PARITY CHECK (LDPC) CODES

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

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

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

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

Examining Committee Members:

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

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

Prof. Dr. Mete Severcan
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. Dr. Ali Özgür Yılmaz
Electrical and Electronics Engineering Dept., METU _____

Reyhan Ergün
MST-YMM-REH, ASELSAN Inc. _____

Date: December 7, 2007

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

Name, Surname: Cihan Uzunoğlu

Signature :

ABSTRACT

PERFORMANCE COMPARISON OF MESSAGE PASSING DECODING ALGORITHMS FOR BINARY AND NON-BINARY LOW DENSITY PARITY CHECK (LDPC) CODES

Uzunoğlu, Cihan
M. Sc., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Melek Diker Yücel
December 2007, 80 pages

In this thesis, we investigate the basics of Low-Density Parity-Check (LDPC) codes over binary and non-binary alphabets. We especially focus on the message passing decoding algorithms, which have different message definitions such as a posteriori probabilities, log-likelihood ratios and Fourier transforms of probabilities. We present the simulation results that compare the performances of small block length binary and non-binary LDPC codes, which have regular and irregular structures over GF(2),GF(4) and GF(8) alphabets. We observe that choosing non-binary alphabets improve the performance with careful selection of mean column weight by comparing LDPC codes with variable node degrees of 3, 2.8 and 2.6, since it is effective in the order of GF(2), GF(4) and GF(8) performances.

Keywords: Message Passing Decoding Algorithms, Binary and Non-binary LDPC Codes.

ÖZ

MESAJ AKTARIMLI ÇÖZÜMLEME ALGORİTMALARININ İKİ VE ÇOK DEĞİŞKENLİ DÜŞÜK YOĞUNLUKLU EŞLİK KONTROL (DYEK) KODLARINDA BAŞARIM KARŞILAŞTIRMASI

Uzunoğlu, Cihan
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Melek Diker Yücel
Aralık 2007, 80 sayfa

Bu tezde, iki ve çok değişkenli alfabeler için Düşük Yoğunluklu Eşlik Kontrol (DYEK) kodlarının temelleri incelenmektedir. Özellikle odaklandığımız konu, iki ve çok değişkenli DYEK kodları için sonsal olasılıklar ve bu olasılıkların logaritmik olabilirlikleri ve Fourier dönüşümleri gibi farklı tiplerdeki mesajları kullanan mesaj aktarımlı çözümleme algoritmalarıdır. Benzetim sonuçlarıyla kısa blok uzunluğuna sahip, düzenli ve düzensiz yapılardaki iki ve çok değişkenli DYEK kodlarının, GF(2), GF(4) ve GF(8) alfabelerindeki başarım karşılaştırmaları verilmiştir. Değişken düğüm yoğunluğu 3, 2.8 ve 2.6 olan DYEK kodlarının GF(2), GF(4) ve GF(8) başarım sıralamasındaki etkisi gözlenerek, dikkatli seçilmiş ortalama kolon yoğunluğuna sahip çok değişkenli alfabe seçiminin başarımı iyileştirdiği görülmüştür.

Anahtar Kelimeler: Mesaj Aktarımlı Çözümleme Algoritmaları, İki ve Çok Değişkenli Düşük Yoğunluklu Eşlik Kontrol Kodları

To My Wife

ACKNOWLEDGEMENTS

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

I would also like to thank my wife for giving me encouragement during this thesis and all kind of support during my whole work.

Finally, I would like to thank my company 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
BER	Bit Error Ratio
BP	Belief Propagation
BPSK	Binary Phase-Shift Keying
DVB-S2	Second Generation Satellite Digital Video
FEC	Forward Error Correction
FT	Fourier Transform
GF	Galois Field
LDPC	Low Density Parity Check
LL	Log Likelihood
MLD	Maximum Likelihood Decoding
MP	Message Passing
SNR	Signal to Noise Ratio
WiFi	Wireless LAN
WiMax	Worldwide Interoperability for Microwave Access

CHAPTER 1

INTRODUCTION

Main problem of communication systems is the presence of noise over the channel and the main goal of system designers is to mitigate the effect of noise as efficiently as possible. Error-correcting codes are developed for correcting errors when messages are transmitted through noisy channels. A noisy communication channel is usually characterized by a probabilistic model, which specifies a transition probability $p(y|x)$ of receiving a sequence y, when a sequence x is sent over the channel. A model for a typical communication system is shown in Figure 1.1. The error-correcting code of the system encodes the data and constructs x by adding certain amount of redundancy to the message m in order to detect and correct errors of the received sequence y.

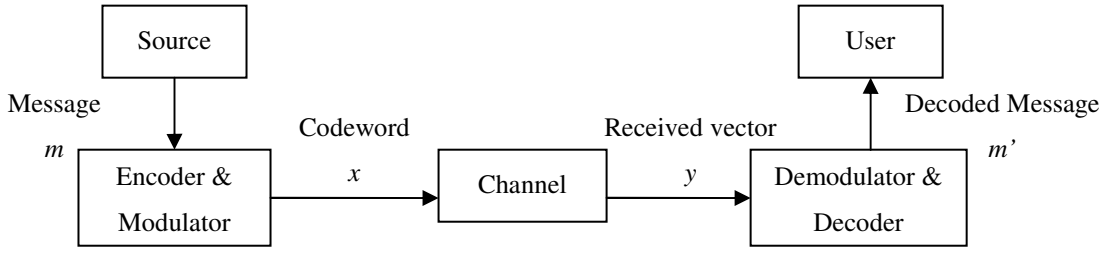


Figure 1.1 Block diagram of a typical communication system

1.1 Shannon's Theorem and Coding History

Shannon proved that there exists an upper bound C (called the capacity of the channel) to the rate of reliable transmission over the channel with a given probabilistic channel model [Shannnon-1948]. This bound is a function of the available bandwidth and the signal-to-noise ratio (SNR) of the channel. For example, in the case of the additive white Gaussian noise (AWGN) channel, the achievable capacity of the channel, in the units of bits/sec, is given by

$$C = B \log_2 \left(1 + \frac{S}{N} \right) \tag{1.1}$$

where B is the channel bandwidth in Hz, S is the average signal power, and N is the average noise power.

Shannon proved his theorem by showing that an arbitrarily low probability of error can result using some codes at a rate below but arbitrarily close to capacity, and that this probability of error approaches to zero exponentially as the length of the codeword tends to infinity. The codes used in his proof were random codes, which are hard to use in practice. If a code is to be used in a practical communication system, one needs computationally efficient algorithms for encoding and decoding.

Since Shannon's original work, there has been enormous interest in explicit construction of codes, together with development of efficient encoding and decoding algorithms, to achieve the promised arbitrarily small probability of error at rates close to the channel capacity.

In 1950, Hamming introduced the very first class of codes that are capable of correcting any single errors [Hamming-1950]. Afterwards, many error correction codes were developed such as Golay, Bose-Chaudhuri-Hocquenghem (BCH) [Bose-Chaudhuri-1960], and Reed-Solomon (RS) [Reed-Solomon-1960] codes with algebraic decoding algorithms and convolutional codes with trellis-based decoding algorithms to achieve gains in overall system performance.

Until the advent of the turbo codes in 1993, practical coding schemes fell quite short of the Shannon limit [Berrou-Glavieux-Thitimajshima-1993]. Turbo codes spectacularly improved on all previous codes, marking a breakthrough in the construction of error correcting algorithms that almost achieve the Shannon capacity. Another class of codes which are now known to have near-capacity-achieving performance is low-density parity-check (LDPC) codes, originally discovered by Gallager in 1962 [Gallager-1962].

LDPC codes share a philosophy which is similar to that of turbo codes: they have a randomly constructed code structure, and they are decoded using iterative algorithms. However, in spite of their excellent properties, LDPC codes had been largely forgotten since they were extremely large in order to achieve this optimum capacity for their time, in terms of what was technologically possible. In 1996, LDPC codes were rediscovered by Mackay and Neal [Mackay-Neal-1996]. This has sparked major research in the coding field because of the near-Shannon-limit performance and simple description of the LDPC codes.

Since that time, researchers have demonstrated that LDPC codes can achieve remarkable performance extremely close to the Shannon capacity. Special forms of these codes, known as irregular LDPC codes, have been shown to outperform even turbo codes. Also, different types of decoding methods have been developed for different technologies where they are used.

The codes approaching capacity most closely will definitely become the choice for next-generation communications standards, ranging from mobile communication (IEEE802.16) to broadcasting [Nozawa-2005]. For instance, current WiMax 802.16e and WiFi 802.11n standards use LDPC codes with varying code rates from ½ to ⅚ and codes length from 576 to 2304 bits [Brack-Alles-Lehnigk-Emden 2007].

LDPC codes have also become the new standard for satellite broadcasted high-definition TV (DVB-S2) in 2004. In the system, the LDPC code is used with BCH

codes as the inner code with code length of 64800 bits different code rates from ¼ to ⁹⁄₁₀ and [Brack-Alles-Lehnigk-Emden 2007].

1.2 Aim and Organization of the Thesis

Since LDPC codes have great performance when they have large block length, to be applicable over small block length applications, one of the main tools is the usage of non-binary alphabet. The aim of this thesis work is to compare the performance of LDPC codes over non-binary alphabet with that for the binary case using message passing decoding algorithms, and to investigate the parameters that affect LDPC decoding performance.

We first explore the basic concepts regarding LDPC codes; how they are constructed, encoded, and especially how they are decoded. We discuss the main idea behind the message passing decoding algorithms and implement some of these algorithms defined in literature for binary and non-binary alphabets with different message types, such as probability, log-likelihood and Fourier transform messages. We evaluate the performance of LDPC codes over GF(2), GF(4) and GF(8) alphabets, in terms of the "bit error ratio versus signal to noise ratio" curves.

The thesis is organized as follows:

Chapter 2 is a review of the basic concepts and principles of LDPC codes, encoding and message passing decoding algorithms both for binary and non-binary cases.

In Chapter 3, the parameters that affect the decoding performance are discussed. Simulation settings and comparison parameters are defined. Experimental results are provided and discussed for the defined parameters.

Finally Chapter 4 summarizes our conclusions and suggestions for further research.

CHAPTER 2

LOW DENSITY PARITY CHECK CODES

This chapter provides a brief review of the extensive literature on linear block codes in addition to the basics and key developments in the area of LDPC codes. Section 2.1 is a brief overview of linear block codes. In Section 2.2, we discuss the basics, encoding and decoding issues on LDPC codes. We then focus on different types and details of message passing decoding algorithms. In Section 2.3 we provide the basics and general description of q-ary LDPC codes. Then, message passing decoding algorithms over GF(q) and some different versions are discussed.

2.1 Basics of Linear Block Codes

The basic principle of error-control codes is to introduce redundancy to the information so that detection and correction of errors can be possible during transmission. Introduction of redundancy is accomplished by adding extra check bits such that produced codewords are sufficiently distinct from each other to enable reconstruction of the initial message even when there occur errors during the transmission. Most of the known good error-correcting codes belong to a class of codes called linear codes, which base their mathematical foundation on linear algebra.

2.1.1 Definition

By definition, a q-ary (n, k) linear block code C is a k-dimensional subspace of GF(q)^n, which is a vector space over the Galois Field (GF) with q elements. This definition implies that a linear block code contains a set of n-dimensional vectors, called *codewords*, chosen from GF(q)^n such that;

- the sum of two codewords is a codeword,
- the scalar product of any codeword by a field element is a codeword.

There are q^k codewords in C and each codeword can be uniquely associated with a vector of k information symbols, defined over the same q-ary alphabet.

To measure the efficiency of the code, the *code rate* is defined as;

$$R_c = \frac{k}{n} \tag{2.1}$$

in the sense that k symbols of information are transmitted in n uses of channel .

2.1.2 Matrix Description

A n-dimensional code vector;

$$c = [c_1 \quad c_2 \quad \dots \quad c_n]$$

can be represented as a linear combination of k basis vectors g_i, such that;

$$c = uG = \sum_{i=1}^{k} u_i g_i = u_1 g_1 + u_2 g_2 + \dots + u_k g_k \tag{2.2}$$

where u is a k-dimensional row vector of information symbols u_i,

$$u = \begin{bmatrix} u_1 & u_2 & \dots & u_k \end{bmatrix}$$

g_i is an n-dimensional row vector,

$$g_i = \begin{bmatrix} g_{i1} & g_{i2} & \dots & g_{in} \end{bmatrix}$$

and the $k \times n$ matrix G is called the generator matrix of the (n, k) code C whose rows are the k basis vectors g_i, $i=1,\ldots,k$, which span C, so

$$G = \begin{bmatrix} g_1 \\ g_2 \\ \dots \\ g_k \end{bmatrix}$$

The generator matrix G can be converted to systematic form by using column permutations and elementary row operations defined over matrix operations. The resulting matrix G_{sys} is composed of two sub-matrices:

$$G_{sys} = \begin{bmatrix} I_k & P \end{bmatrix}$$

where I_k is the $k \times k$ identity matrix and P is the $k \times (n-k)$ parity matrix.

Since C is a k-dimensional subspace in an n-dimensional space, there is an $(n-k)$-dimensional dual space $C^{\perp}$, generated by the rows of an $(n-k) \times n$ matrix H, called the parity-check matrix such that [Mackay-2005];

$$GH^T = \vec{0} \tag{2.3}$$

If G is systematic, then H_{sys} can be described as;

$$H_{sys} = \begin{bmatrix} P^T & I_{n-k} \end{bmatrix}$$

such that,

$$G_{sys} H_{sys}^T = \vec{0}$$

2.1.3 Encoding

Equation (2.2) indicates an encoding rule for linear block codes that can be implemented in a straightforward way. If G is in systematic form, encoding is systematic such that

$$c = uG_{sys} = \begin{bmatrix} u & uP \end{bmatrix}$$

By this encoding method, codeword can be constructed by first repeating the information symbols and then adding the corresponding parity check symbols to the tail of the information symbols.

2.1.4 Decoding

Error Detection

For a codeword c, using (2.2) and (2.3) one shows that,

$$Hc^T = H(uG)^T = HG^T u^T = \vec{0} \tag{2.4}$$

The vector $s^T = Hc^T$ is defined as the *syndrome vector*.

This fact is exploited for error detection as follows. Suppose a codeword c is transmitted over a channel and received as c', where;

$$c' = c + e \qquad c' \in GF(q)^n$$

Then the following syndrome checking can be used for determining whether the received vector c' is a codeword or not.

$$s^T = Hc'^T = H(c+e)^T = Hc^T + He^T = He^T \tag{2.5}$$

If $s=eH^T \neq \vec{0}$, this implies that the received word c' is not a codeword. By this way one immediately detects errors.

However, if the channel introduces an error pattern such that the received vector c' is also a codeword satisfying $Hc'^T=0$ where $c'=c+e$, there is no way of determining whether c' is the transmitted codeword or not. In this case, an undetected error pattern occurs.

The key point on this situation is the *Hamming distance* $d_H(c,c')$ of the codewords with respect to each other, where the *Hamming distance* is defined as the number of places in which they differ.

If the minimum Hamming distance of all codewords in C is called as d_{min}, then detection of errors up to Hamming weight of $d_{min}-1$ is possible [Mackay-2005].

Error Correction

Usually it is a more complex task to correct errors than merely to detect them. If the minimum Hamming distance of C is d_{min}, correction of errors up to Hamming weight $(d_{min}-1)/2$ is possible. This is as follows:

Assume c_1 is sent, c_1' is received with $d_H(c_1, c_1')\leq t$ and $d_{min}\geq 2t+1$. Assume some other c_2 with $d_H(c_2, c_1')\leq t$. Then by triangle inequality;

$$d_H(c_1,c_2) \leq d_H(c_1,c_1') + d_H(c_2,c_1') \leq 2t \tag{2.6}$$

which is contradicting with $d_{min}\geq 2t+1$. So such a codeword c_2 is not valid and c_1 is the nearest codeword to c_1'. So, nearest neighbor decoding corrects the error [Mackay-2005].

There are error-correction decoding methods available in literature. The optimal error correction decoding method, in the sense of minimizing the average probability of codeword error, namely the *maximum likelihood decoding* (MLD)

has a complexity which increases exponentially with the code length n. But, sub-optimal decoding methods, which provide good approximations to the optimal solution while requiring much lower complexity, are available [Robertson-Villebrun-Hoeher-1995]. These sub-optimal decoding methods are mainly based on graphical approaches and iterative message passing (belief propagation) idea. In the next section, we will discuss these subjects in conjunction with LDPC codes.

2.2 Background on LDPC Codes

Low density parity check (LDPC) codes are linear block codes which have sparse parity check matrices and high block length. LDPC codes can also be considered as a class of linear block codes that can be described by graphical representations and decoded through an iterative manner. We first introduce the definition and graphical representation of LDPC codes and then give an overview of the history on LDPC codes. Following this, we mention the construction and decoding methods.

2.2.1 Definition and Representation of LDPC Codes

Low density parity check (LDPC) codes are high block length linear block codes, which have parity check matrices almost entirely filled with zeros and very sparse non-zero elements.

Given a parity-check matrix H of size $m \times n$, where m is the number of parity symbols in a codeword and n is the length of the codeword, the rate of the code given by (2.1) can also be expressed as

$$R_c = \frac{k}{n} = \frac{n-(n-k)}{n} = 1 - \frac{m}{n} \tag{2.7}$$

where $k=n-m$ is the length of the information vector [Mackay-2005].

An $m \times n$ parity-check matrix specifies m linear parity equations, since a codeword c must satisfy (2.4) for each row i of $H_{i,j}$, hence

$$\sum_{j=1}^{n} H_{i,j}.c_j = 0, \qquad i = 1,...,m \tag{2.8}$$

where summation is defined modulo-2 for binary codes. Since H is low density, there are only a small number of nonzero terms in H matrix. If $H_{i,j} \neq 0$, we say that the i'th parity equation checks the j'th code bit, or that the j'th code bit is checked by the i'th parity equation .

LDPC codes are often represented in graphical form by a *Tanner graph*. The *Tanner graph* consists of two sets of vertices: n vertices for the codeword symbols (called *variable nodes*), and m vertices for the parity-check equations (called *check nodes*). Considering binary codes, an edge joins a variable node to a check node if that codeword bit is included in the corresponding parity-check equation. So the number of edges in the *Tanner graph* is equal to the number of nonzero elements in the parity-check matrix.

Below is an example of a binary parity-check matrix:

$$H_{4\times 6} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Considering the location of one's in H matrix with respect to the row and column view we can draw a Tanner graph as illustrated in Figure 2.1.

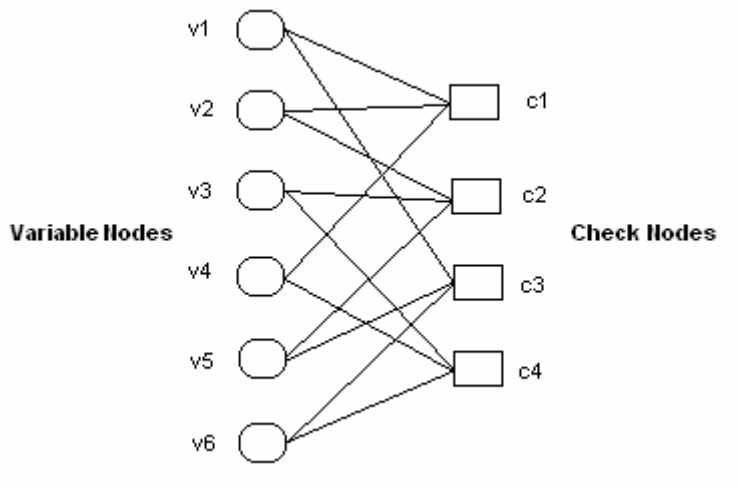


Figure 2.1 Tanner graph representation of *H* matrix defined in (2.9)

On the Tanner graph of an LDPC code, the number of edges incident upon a node is called the *degree* of the node. LDPC codes can be divided into two groups with respect to their degree of node.

If all variable nodes have the same degree d_v and all check nodes have the same degree d_c, the code is known as a *regular*-(d_v, d_c) code, otherwise it is called *irregular*.

As an example, the *H* matrix defined in (2.9) has d_v=2 for all variable nodes and d_c=3 for all check nodes in the Tanner graph such that it is regular.

An irregular LDPC code ensemble is specified by a variable node degree sequence $\lambda(x)$ and a check node degree sequence $\rho(x)$,

$$\lambda(x) = \sum_{i=1}^{d_v} \lambda_i x^i \qquad and \qquad \rho(x) = \sum_{j=1}^{d_c} \rho_j x^j \tag{2.10}$$

where d_v is the maximum variable degree, d_c is the maximum check degree, λ_i represents the fraction of edges that variable nodes have degree i, and ρ_j represents the fraction of edges that check nodes have degree j.

Let e be the number of edges in the graph. So the number of variable nodes of degree i is $\frac{e\lambda_i}{i}$. Therefore the total number of variable nodes n is the summation over all degrees is

$$n = \sum_{i=1}^{d_v} \frac{e\lambda_i}{i} \tag{2.11}$$

Similarly the total number of check nodes is;

$$m = \sum_{i=1}^{d_c} \frac{e\rho_i}{i} \tag{2.12}$$

Since the dimension of the code is $k=n-m$, we find the code rate R_c for irregular codes as:

$$R_c = \frac{k}{n} = 1 - \frac{m}{n} = 1 - \frac{\sum_{i=1}^{d_c} \frac{\rho_i}{i}}{\sum_{i=1}^{d_v} \frac{\lambda_i}{i}} \tag{2.13}$$

Another crucial definition is a *cycle* on Tanner graph representation. A *cycle* in a Tanner graph is a sequence of connected vertices which start and end at the same vertex in the graph, and which contain other vertices no more than once. The *length of a cycle* is the number of edges it contains, and the *girth of a graph* is the size of its smallest cycle [Mackay-2005].

As an example, a cycle of size 6 is shown in bold in Figure 2.2.

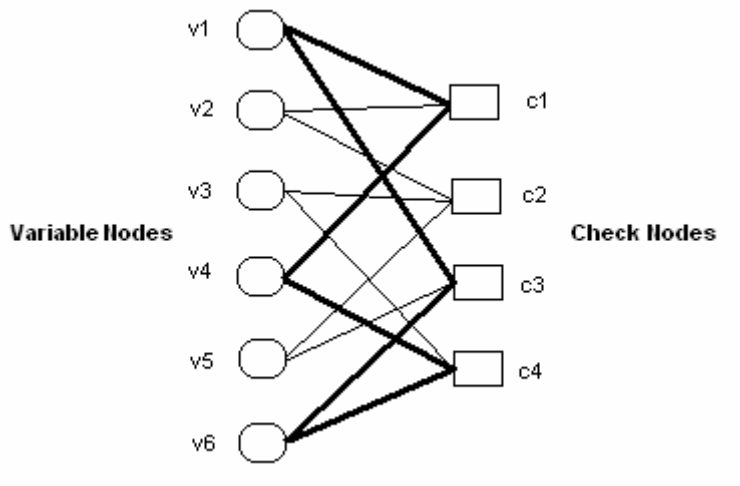


Figure 2.2 Tanner graph representation of *H* matrix defined in (2.9) with cycle length 6

2.2.2 History

The subject of iterative decoding dates back to the work of Gallager on LDPC binary codes published in 1962 [Gallager-1962]. Gallager provided a random construction of codes and showed that iterative computation of *a posteriori* probabilities for decoding these codes could achieve a probability of bit error decreasing exponentially with block length. LDPC codes were generally forgotten by the coding theory community because of technological limitations at that time. Nearly 20 years later in 1981, Tanner [Tanner-1981] introduced a graphical representation of LDPC by Tanner graphs. He also introduced the min-sum and sum-product algorithms, both falling into the category of message passing algorithms and proved that they converge on cycle-free graphs.

LDPC codes were rediscovered in 1996 by Mackay and Neal [Mackay-Neal-1996], who demonstrated by extensive simulations that iterative decoding on bipartite graphs can approach the Shannon limit to within about 1 dB, at a code rate of ½ on the AWGN channels. The LDPC codes considered in [Tanner-1981] and [Mackay-Neal-1996] were regular LDPC codes whose graph representations have fixed

degrees for variable and check nodes. Luby et al. [Luby-Mitzenmacher-Shokrollahi-Spielman-1998] proposed improved LDPC codes using irregular graph structures. Their idea was later extended by Richardson et al. [Richardson-Shokrollahi-Urbanke-2001], who developed an analytical approach, called density evolution, to design irregular LDPC codes that exhibit better performance than turbo codes and regular LDPC codes, as illustrated in Figure 2.3. The performance of the irregular LDPC code shown in Figure 2.3 was the best known result as of 1999. The distance to the Shannon limit was further reduced to 0.0045 dB by a carefully designed rate ½ irregular LDPC code with a block length of 10^7 bits in 2001 [Chung-Forney-Richardson-Urbanke-2001].

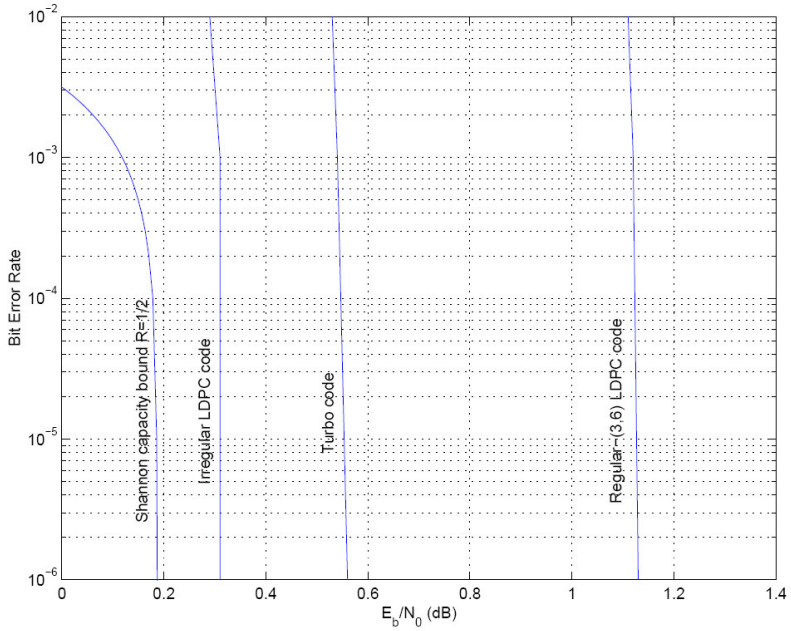


Figure 2.3 Performance comparison between three codes: a turbo code, a regular LDPC code, and an irregular LDPC code, for code length 10^6 and rate ½ over AWGN channels. [Richardson-Shokrollahi-Urbanke-2001]

2.2.3 Construction of LDPC Codes

There are numerous methods for the construction of LDPC parity check matrices. We mention two common of these techniques in this section. The goal of

construction is to create a parity check matrix that has low weight rows and columns, a large girth value, and results in good performance. While either a structured or a pseudo-random approach may be used for construction, it has been found that a higher degree of randomness generally results in codes that perform better than those that are highly structured.

Gallager Method for Regular LDPC Code Construction

First construction method for a regular LDPC code with parity check matrix H, which contains w_c one's per columns and w_r one's per row, was proposed by Gallager [Gallager-1962], as follows:

Initially construct a submatrix with size $(m/w_c)\times n$ where the i'th row contains its 1's in columns iw_r+1 to $(i+1)w_r$, so each column has a single 1. Then divide H matrix into blocks of w_c submatrices, where the first submatrix is the one that was constructed initially. Subsequently let the other w_c-1 submatrices simply be random column permutations of the first. Allowing all column permutations with equal probability, we get an ensemble of (n, w_c, w_r) parity-check matrices. An example of this method can be seen in Figure 2.4.

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

Top Block

Permutation of Top Block

Another Permutation of Top Block

Figure 2.4 The parity check matrix for (12, 3, 4)-regular Gallager code

As a remark, note that this type of a parity check matrix is not in the systematic form. By performing matrix operations it can then be translated to systematic form as $H_{sys} = \left[P^T \quad I_{n-k}\right]$. Also notice that column and row weight pair (w_c, w_r) corresponds to variable node and check node degrees (d_v, d_c) mentioned in Section 2.2.1.

Mackay and Neal Method for Regular and Irregular LDPC Code Construction

Another construction method is proposed by Mackay and Neal for either regular LDPC codes with given variable node degree d_v and check node degree d_c or irregular LDPC codes with given degree distribution polynomials $\lambda(x)$ and $\rho(x)$ [Mackay-Neal-1996]. In this method, columns of H matrix are defined at a time from left to right. The weight of each column is chosen according to the specified variable degree distribution and the locations of the non-zero entries in each column are chosen randomly from those rows which are not yet full.

If at any point there are rows with more positions unfilled than there are columns remaining to be added, the row degree distributions for H cannot be met. The process is then either started again or back tracked by a few columns, until the correct row degrees are obtained. By this method, the resultant H matrix is in non-systematic form since locations of the non-zero entries in each column are chosen randomly. Again by performing matrix operations it can be translated to systematic form.

Cycles are important from the performance point of view, and while constructing with this method, cycles (at least 4-cycles) can be taken into account. This method can be adapted to avoid 4-cycles without disturbing the variable degree distribution, by rejecting each column to be added if it will cause a 4-cycle with any of the already chosen columns.

2.2.4 Encoding

Encoding of k-dimensional information vectors can be performed as explained in Section 2.1.3, to yield an n-dimensional code vector $c = uG$. Since LDPC codes are useful especially for very large values of n, the main point in the sight of encoding is the encoding complexity. When a parity-check matrix H is rearranged into systematic form $H_{sys} = \left[P^T \quad I_{n-k}\right]$ by matrix operations, an explicit generator matrix can also be obtained as $G_{sys} = \left[I_k \quad P\right]$. But when the parity-check matrix H and the generator matrix G are defined in systematic form both of them loose the sparseness property. Even if one sacrifices systematic forms, a sparse H matrix does not mean a sparse G matrix. When G is not sparse, encoding complexity time is proportional to n^2.

However there are some methods that make modifications in the structure of H matrix such that encoding complexity can be reduced. An example for these methods has been shown [Mackay-Wilson-Davey-1999] with encoding complexity proportional to n.

2.2.5 Decoding

Decoding problem can be simply defined as finding the codeword which most resembles what is received at the channel output. In other words, the rule is to find the best estimate that satisfies the Maximum Likelihood (ML) condition; i.e., choose the codeword $\hat{x} \in C$ with maximum *a posteriori* probability $P(\hat{x}$ was transmitted $|\ y$ is received), or with minimum Hamming distance $d_H(\hat{x}, y)$, where y is the received vector.

The ML-decoding strategy (MLD) guarantees optimal results, but requires extensive search through the vector space of the code. On the other hand, there are also sub-optimal methods which do not search in the overall space.

Main idea behind the sub-optimal methods is based on the graphical representation known as Tanner graph and message passing iterations. Through Tanner graph, received information is passed to variable nodes and check nodes in an iterative manner and decoding is also performed in an iterative manner by focusing on the information carried by passing messages.

2.2.5.1 Message-Passing Algorithms

The algorithms using the iterative decoding idea can be named as message passing (MP) or belief propagation (BP) algorithms, since a message or information about received symbols is passed forward and backward through the nodes in the Tanner graph. Different message-passing algorithms are named with respect to the type of messages passed or the type of operation performed at the nodes. The type of messages differs as hard or soft, where hard information is the binary decision made according to the soft information which is usually given in terms of probabilistic values. For example, the messages are binary in bit-flipping decoding; and they are the probabilities which represent a level of belief about the value of the codeword symbols, in belief propagation decoding. Soft message passing decoding is also called the sum-product decoding since the calculations at the variable and check nodes are made by using sum and product operations.

Firstly we will mention the bit-flipping algorithm to see the idea behind iterative decoding and then describe the sum-product algorithm to see the advantages of probabilistic information.

2.2.5.1.1 Bit-Flipping Algorithm

The bit-flipping algorithm is a hard-decision message-passing algorithm for LDPC codes [Gallager-1962]. Before giving the algorithm, let's start by defining the messages which are passed along the edges of the Tanner graph.

- V_j is the set of variable nodes connected to check node j.
- V_j/i is the set of variable nodes connected to check node j excluding variable node i.
- C_i is the set of check nodes connected to variable node i.
- C_i/j is the set of check nodes connected to variable node i excluding check node j.
- Q_{ji} is the check node message passed from check node j to variable node i
- I_i is the variable node message, obtained by using check node messages Q_{ji} connected to variable node i.
- D_j is the decision message, obtained by using all variable node messages I_i connected to check node j.

The algorithm works as follows:

1) Initialization

Firstly, iteration number is initialized to 1. A binary (hard) decision r_i about each received bit y_i of the received vector y is made and passed to the decoder to initialize the variable node messages. The messages along the Tanner graph edges are binary values, $I_i = r_i$ and the variable node message vector is $I = [I_1, \ldots, I_n]$.

2) Check Node Update

After initialization, each variable node sends its message declaring if it is a one or a zero, and each check node sends a message to each connected variable node, declaring what binary value the variable node should have based on the information available to the check node.

The check node determines that its parity-check equation is satisfied if the modulo-2 sum of the all involved variable values is zero. The check node message Q_{ji} from check node j to variable node i is,

$$Q_{ji} = \sum_{i' \in V_j / i} I_{i'} \quad \text{mod } 2 \tag{2.14}$$

Note that addition is defined in modulo-2.

3) Variable Node Update

If the majority of the messages received by a variable node are different from its received value, the variable node flips its current value. That is;

$$\begin{aligned} &\text{if majority of } Q_{ji} \text{ is different from } r_i \\ &\quad I_i = r_i + 1 \quad \text{mod } 2 \\ &\text{end} \end{aligned} \tag{2.15}$$

4) Decision

The bit-flipping decoder calculates the parity-check equations for the set of variable nodes V_j that are connected to the check node j.

$$D_j = \sum_{i' \in V_j} I_{i'} \quad \text{mod } 2 \tag{2.16}$$

Whenever a valid codeword has been found, the algorithm is immediately terminated by checking if all of the parity-check equations are satisfied. Then, the decoded codeword is $\hat{c}_i = I_i$. Otherwise, the iteration number is increased by one and algorithm continues with step 2, check node update. That is,

$$\begin{aligned}
&\text{if } D_j = 0 \text{ (for } j = 1,...,m) \text{ or } i = i_{\max} \\
&\quad \hat{c}_i = I_i \\
&\quad \text{terminate} \\
&\text{else} \\
&\quad i = i + 1 \\
&\quad \text{go to step 2 (check node update)} \\
&\text{end}
\end{aligned} \tag{2.17}$$

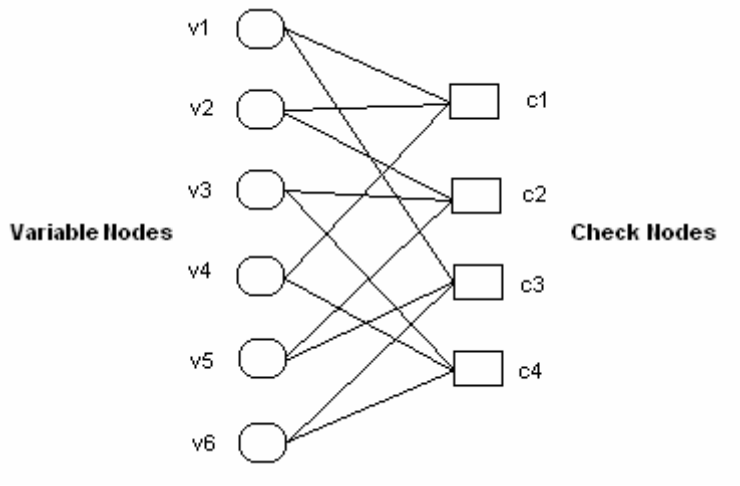
The steps 2, 3 and 4 are repeated until all of the parity-check equations are satisfied, or until some maximum number of decoder iterations has passed and the decoder gives up.

Example 1

Assume the parity check matrix is the one given by (2.9),

$$H_{4\times 6} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

the codeword $c = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1]$ is sent over the BSC channel with probability of error p=0.2, and the received hard word is $r = [1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1]$ with an error in the first bit.



1) Bit flipping algorithm starts with initialization of the variable node message;

$$I = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

2) Then the first check node update process using (2.14)

$$Q_{11} = I_2 + I_4 = 0 + 0 = 0$$
$$Q_{12} = I_1 + I_4 = 1 + 0 = 1$$
$$Q_{14} = I_1 + I_2 = 1 + 0 = 1$$

The same process is applied by other check nodes and:

$Q_{22} = I_3 + I_5 = 1 + 1 = 0$ $\quad Q_{31} = I_5 + I_6 = 1 + 1 = 0$ $\quad Q_{43} = I_4 + I_6 = 0 + 1 = 1$

$Q_{23} = I_2 + I_5 = 0 + 1 = 1$ $\quad Q_{35} = I_1 + I_6 = 1 + 1 = 0$ $\quad Q_{44} = I_3 + I_6 = 1 + 1 = 0$

$Q_{25} = I_2 + \imath_3 = 0 + 1 = 1$ $\quad Q_{36} = I_1 + I_5 = 1 + 1 = 0$ $\quad Q_{46} = I_3 + I_4 = 1 + 0 = 1$

3) Now variable node update will be done. For the first variable node;

$I_1 = 1 \quad Q_{11} = 0 \quad Q_{31} = 0 \quad \Rightarrow I_1 = 0$ since both of the check node messages are 0

Similarly, variable node update continues as follows, where the existence of a single message not matching with the received bit is not sufficient for flipping the bit:

$$I_2 = 0 \quad Q_{12} = 1 \quad Q_{22} = 0 \quad \Rightarrow I_2 = 0$$
$$I_3 = 1 \quad Q_{23} = 1 \quad Q_{43} = 1 \quad \Rightarrow I_3 = 1$$
$$I_4 = 0 \quad Q_{14} = 1 \quad Q_{44} = 0 \quad \Rightarrow I_4 = 0$$
$$I_5 = 1 \quad Q_{25} = 1 \quad Q_{35} = 0 \quad \Rightarrow I_5 = 1$$
$$I_6 = 1 \quad Q_{36} = 0 \quad Q_{46} = 1 \quad \Rightarrow I_6 = 1$$

Hence only the first bit of the variable message is flipped and new variable message vector is:

$$I = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1]$$

4) Decision step checks whether the parity equations are satisfied:

$$D_1 = I_1 + I_2 + I_4 = 0 + 0 + 0 = 0$$
$$D_2 = I_2 + I_3 + I_5 = 0 + 1 + 1 = 0$$
$$D_3 = I_1 + I_5 + I_6 = 0 + 1 + 1 = 0$$
$$D_4 = I_3 + I_4 + I_6 = 1 + 0 + 1 = 0$$

All of the parity checks are satisfied in the first iteration. So the decoded codeword is,

$$\hat{c} = [0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1]$$

which is the one that was sent, so the decoding is successful.

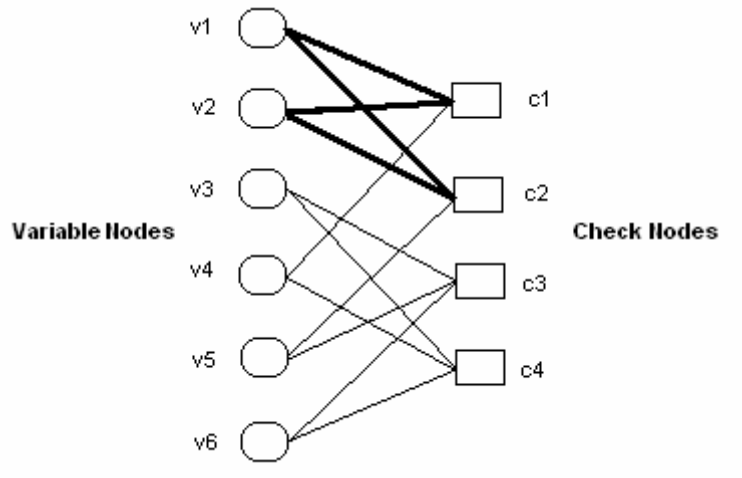
Although bit-flipping algorithm is based on the principle that a codeword bit involved in a large number of check equations will be corrected, the effect of short cycles in the graph may cause uncorrectable errors as in the following example.

Example 2

To see the effect of 4-cycle, assume the parity check matrix as defined below, having the 4-cycle indicated by bold 1's of the matrix;

$$H_{4\times 6} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & 0 & 1 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

From the Tanner graph one can also observe this cycle of length 4.



The codeword $c = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1]$ is sent over the BSC channel with probability of error p=0.2, and the received hard word is $r = [1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1]$ with an error in the first bit as in the previous example.

1) Bit flipping algorithm sets the variable node message;

$$I = [1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1]$$

2) The check node update process starts using (2.14)

$$Q_{11} = I_2 + I_4 = 0 + 0 = 0 \qquad Q_{21} = I_2 + I_5 = 0 + 0 = 0$$
$$Q_{12} = I_1 + I_4 = 1 + 0 = 1 \qquad Q_{22} = I_1 + I_5 = 1 + 0 = 1$$
$$Q_{14} = I_1 + I_2 = 1 + 0 = 1 \qquad Q_{25} = I_1 + I_2 = 1 + 0 = 1$$

$$Q_{33} = I_5 + I_6 = 0 + 1 = 1 \qquad Q_{43} = I_4 + I_6 = 0 + 1 = 1$$
$$Q_{35} = I_3 + I_6 = 1 + 1 = 0 \qquad Q_{44} = I_3 + I_6 = 1 + 1 = 0$$
$$Q_{36} = I_3 + I_5 = 1 + 0 = 1 \qquad Q_{46} = I_3 + I_4 = 1 + 0 = 1$$

3) Now variable node updates are done as follows;

$$
\begin{aligned}
&I_1 = 1 \quad Q_{11} = 0 \quad Q_{21} = 0 \quad \Rightarrow I_1 = 0 \\
&I_2 = 0 \quad Q_{12} = 1 \quad Q_{22} = 1 \quad \Rightarrow I_2 = 1 \\
&I_3 = 1 \quad Q_{33} = 1 \quad Q_{43} = 1 \quad \Rightarrow I_3 = 1 \\
&I_4 = 0 \quad Q_{14} = 1 \quad Q_{44} = 0 \quad \Rightarrow I_4 = 0 \\
&I_5 = 0 \quad Q_{25} = 1 \quad Q_{35} = 0 \quad \Rightarrow I_5 = 0 \\
&I_6 = 1 \quad Q_{36} = 1 \quad Q_{46} = 1 \quad \Rightarrow I_6 = 1
\end{aligned}
$$

So, the first and second bits of the variable message are flipped and new variable message vector is:

$$I = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

4) Decision step checks whether parity equations are satisfied:

$$
\begin{aligned}
D_1 &= I_1 + I_2 + I_4 = 0 + 1 + 0 = 1 \\
D_2 &= I_1 + I_2 + I_5 = 0 + 1 + 0 = 1 \\
D_3 &= I_3 + I_5 + I_6 = 1 + 0 + 1 = 0 \\
D_4 &= I_3 + I_4 + I_6 = 1 + 0 + 1 = 0
\end{aligned}
$$

First and second parity checks are not satisfied so the iteration number is incremented and decoding continues with check node update step.

2) In the check node update, check node messages are updated as:

$$
\begin{aligned}
&Q_{11} = 1 \quad Q_{21} = 1 \; Q_{33} = 1 \; Q_{43} = 1 \\
&Q_{12} = 0 \quad Q_{22} = 0 \; Q_{35} = 0 \; Q_{44} = 0 \\
&Q_{14} = 1 \quad Q_{25} = 1 \; Q_{36} = 1 \; Q_{46} = 1
\end{aligned}
$$

3) Then the variable nodes are updated as,

$$I_1 = 0 \quad Q_{11} = 1 \quad Q_{21} = 1 \quad \Rightarrow I_1 = 1$$
$$I_2 = 1 \quad Q_{12} = 0 \quad Q_{22} = 0 \quad \Rightarrow I_2 = 0$$
$$I_3 = 1 \quad Q_{33} = 1 \quad Q_{43} = 1 \quad \Rightarrow I_3 = 1$$
$$I_4 = 0 \quad Q_{14} = 1 \quad Q_{44} = 0 \quad \Rightarrow I_4 = 0$$
$$I_5 = 0 \quad Q_{25} = 1 \quad Q_{35} = 0 \quad \Rightarrow I_5 = 0$$
$$I_6 = 1 \quad Q_{36} = 1 \quad Q_{46} = 1 \quad \Rightarrow I_6 = 1$$

Then first and second bits are again flipped and new variable message vector is:

$$I = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

4) In decision step whether check parity equations are satisfied:

$$D_1 = I_1 + I_2 + I_4 = 1 + 0 + 0 = 1$$
$$D_2 = I_1 + I_2 + I_5 = 1 + 0 + 0 = 1$$
$$D_3 = I_3 + I_5 + I_6 = 1 + 0 + 1 = 0$$
$$D_4 = I_3 + I_4 + I_6 = 1 + 0 + 1 = 0$$

Again, first and second parity checks are not satisfied, so the iteration number is incremented and the algorithm turns to the initial state.

In further iterations, the first two bits continue to flip their values together such that one of them is always incorrect and the algorithm fails to converge. As a result of the cycle of length four, each of the first two codeword bits are involved in the same two parity-check equations, and so when both of the parity-check equations are unsatisfied, it is not possible to determine which bit is in error. This is the performance degrading effect of length-4 cycles in decoding.

After mentioning the iterative decoding idea with examples of the bit-flipping algorithm that uses hard decisions; we now describe the soft decision decoding method by means of the sum-product algorithm.

2.2.5.1.2 Sum-Product Algorithm

The sum-product algorithm is a soft decision message-passing algorithm, whose aim is to compute the *a posteriori* probability (APP) of each received symbol. The decision is made choosing the maximum probability given the channel observation, after each iteration step [Wiberg-1996]. It is similar to the bit-flipping algorithm described in the previous section, but messages passed back and forth are constructed from probabilities.

The algorithm can be summarized as follows: Nodes compute a metric message for each edge, based on the incoming messages as the node's belief. After each step, messages are examined and a tentative decision is produced. Until either decoding produces a codeword, or a predefined maximum number of iterations is reached, messages are iteratively updated.

The message from a variable node to a check node for binary transmission;

$$I_{v\to c} = \left(P(x_i = 0 \mid y), P(x_i = 1 \mid y)\right) \tag{2.18}$$

is the current local value at variable node v conveying the assumption of v on its parity.

Similarly, the opposite message, which is from the check node to a variable node, is a bias on the correctness of variable node v, as computed by the check node c. The check node checks whether the incoming message satisfies the parity check equation.

Basically, messages are *a posteriori* probabilities which are computed in parallel either in the form of (2.18) or the likelihood ratio;

$$L(x_i) = \frac{P(x_i = 0 \mid y)}{P(x_i = 1 \mid y)} \tag{2.19}$$

or its logarithm;

$$LL(x_i) = \ln\left(\frac{P(x_i = 0 \mid y)}{P(x_i = 1 \mid y)}\right) \tag{2.20}$$

Likelihood message is used for cutting the memory usage to half by compressing variables into a one-dimensional likelihood ratio. However during the computation, numerical division problems arise when $P(x_i = 1 \mid y)$ approaches to 0. This problem may be overcome by using some constraints on the value of $P(x_i = 1 \mid y)$ like using maximum or minimum limited numbers or using messages as log-likelihood ratios since extreme values are naturally scaled by ln(.) function.

After defining the message types, we will introduce the methods dedicated to these message types. We will mention the details of decoding algorithm with different messages types in the following sections.

- **A Posteriori Probability (APP) Decoder:**

Let's start by updating the message definitions with the probabilities which are passed along the edges of the Tanner graph.

- $I_{ij}(v) = P(v_i = v \mid \{y_i\}_{i \in C_i / j})$, $v \in \{0, 1\}$, is the message passed from variable node i to check node j, where y_i is the received noisy symbol and v_i denotes the value of the variable node i.
- $Q_{ji}(v) = P(\text{parity - check equation } c_j \text{ is satisfied} \mid v_i = v)$, $v \in \{0, 1\}$, is the message passed from check node j to variable node i.

The algorithmic messages $I_{ij}(v)$ and $Q_{ji}(v)$ are also random variables, as the received symbol y_i.

$I_{ij}(v)$ is expressed as;

$$
\begin{aligned}
I_{ij}(v) &= P(v_i = v \mid \{y_i\}_{i \in C_i / j}) \\
&= \frac{P(v_i = v) P(\{y_i\}_{i \in C_i / j} \mid v_i = v)}{P(\{y_i\}_{i \in C_i / j})} \\
&= k_{ij} P_v \prod_{j' \in C_i / j} Q_{j'i}(v)
\end{aligned}
\tag{2.21}
$$

where $P_i = P(v_i = v \mid y_i)$, $v \in \{0, 1\}$ and k_{ij} are the normalization factors chosen to satisfy $\sum_v I_{ij}(v) = 1$. Note that Bayes' Theorem is used in the second line and with the independence assumption on y_i, product of all other check nodes' votes for state v is used in the third line of (2.21).

Figure 2.5 illustrates the computation of $I_{ij}(v)$;

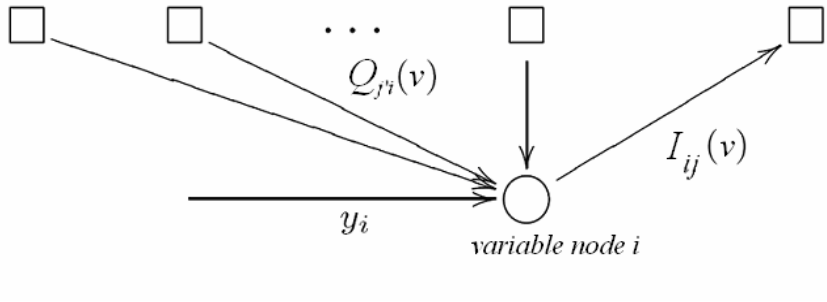


Figure 2.5 Computation of $I_{ij}(v)$

One obtains the check to variable node message $Q_{ji}(v)$ using the following lemma by Gallager.

Lemma [Gallager]

Consider a sequence of m independent binary digits $c = [c_1 \quad c_2 \quad ... \quad c_m]$ in which $P_r(c_i=1) = p_i$.

Then the probability that c contains an even number of 1's can be found as;

$$\frac{1 + \prod_{i=1}^{m} (1 - 2p_i)}{2}$$

and the probability that c contains an odd number of 1's is;

$$\frac{1-\prod_{i=1}^{m}(1-2p_i)}{2}$$

Proof:

Consider the function;

$$\prod_{i=1}^{m}(1-p_i+tp_i)$$

Observe that if this is expanded into a polynomial in t, the coefficient of t^i is the probability of i 1's. The function $\prod_{i=1}^{m}(1-p_i-tp_i)$ is identical, except that all the odd powers of t are negative. Adding these two functions, all the even powers are doubled, and the odd terms cancel out. Finally letting t=1 and dividing by 2, the result is the probability of an even number of 1's. Indeed,

$$\frac{\prod_{i=1}^{m}(1-p_i+tp_i)+\prod_{i=1}^{m}(1-p_i-tp_i)}{2}=\frac{1+\prod_{i=1}^{m}(1-2p_i)}{2}$$

thus proving the lemma.

Considering this lemma with $p_i = I_{ij}(1) = P(v_i = 1 \mid \{y_i\}_{i \in C_i / j})$;

$$\begin{aligned} Q_{ji}(v) &= P(\text{parity - check equation } c_j \text{ is satisfied} \mid v_i = v) \\ Q_{ji}(1) &= \frac{1-\prod_{i' \in V_j / i}\left(1-2I_{i'j}(1)\right)}{2} \\ Q_{ji}(0) &= \frac{1+\prod_{i' \in V_j / i}\left(1-2I_{i'j}(1)\right)}{2} \end{aligned} \tag{2.22}$$

Because, in order that the parity check equation is satisfied, it must contain an odd number of 1's for v=1 and must contain an even number of 1's for v=0.

Figure 2.6 illustrates the computation of $Q_{ij}(v)$;

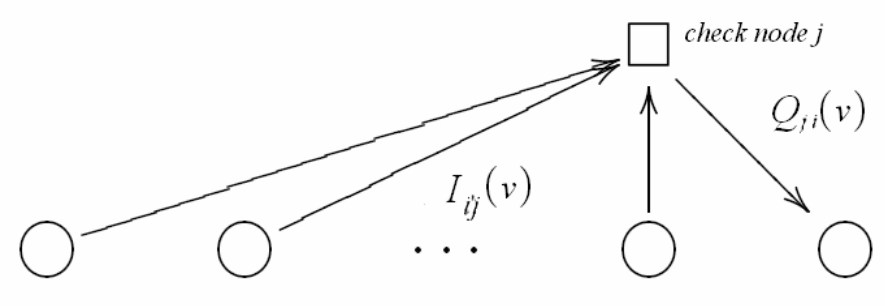


Figure 2.6 Computation of $Q_{ji}(v)$

The algorithm is described as in the following:

1) Initialization

The algorithm starts with the initialization of iteration number with 1 and the $I_{ij}(v)$ values such that y_i is the received channel sample;

$$\begin{aligned} I_{ij}(1) &= P(v_i = 1 \mid y_i) = P_i \\ I_{ij}(0) &= 1 - P(v_i = 1 \mid y_i) = 1 - P_i \end{aligned} \tag{2.23}$$

where $P_i = P(v_i = 1|y_i)$, for $1 \leq i \leq n$.

2) Check Node Update

Update $Q_{ji}(b)$, for all i, j, using;

$$Q_{ji}(1) = \frac{1 - \prod_{i' \in V_j / i} \left(1 - 2I_{i'j}(1)\right)}{2}$$
$$Q_{ji}(0) = 1 - Q_{ji}(1) \tag{2.24}$$

3) Variable Node Update

Update $I_{ij}(b)$, for all i, j, using;

$$I_{ij}(1) = k_{ij} P_i \prod_{j' \in C_i / j} Q_{j'i}(1)$$
$$I_{ij}(0) = k_{ij} (1 - P_i) \prod_{j' \in C_i / j} Q_{j'i}(0) \tag{2.25}$$

such that the constants k_{ij} are chosen to satisfy $I_{ij}(0) + I_{ij}(1) = 1$.

4) Decision

Compute;

$$D_i(1) = k_{ij} P_i \prod_{j' \in C_i} Q_{j'i}(1)$$
$$D_i(0) = k_{ij} (1 - P_i) \prod_{j' \in C_i} Q_{j'i}(0) \tag{2.26}$$

such that the constants k_i are chosen to satisfy $D_i(0) + D_i(1) = 0$. Then each bit is decided as;

$$z_i = \begin{cases} 1 & \text{if } D_i(0) < D_i(1) \\ 0 & \text{else} \end{cases} \tag{2.27}$$

After decision for each bit is made, the parity check equations are checked by computing Hc^T, where $c = (z_1, \quad ... \quad z_n)$. If all parity check equations are satisfied, i.e. $Hc^T = \vec{0}$, or if a certain maximum number of iterations is reached, the algorithm stops; else goes back to the second step, check node update step, and iteration number is incremented.

- **Log-Likelihood (LL) Decoder**

The APP Decoder includes multiplications of probabilities on the variable node update and check node update steps as in equation (2.25) and (2.26),

$$Q_{ji}(1) = \frac{1 - \prod_{i' \in V_j / i} (1 - 2I_{i'j}(1))}{2} \qquad Q_{ji}(0) = \frac{1 + \prod_{i' \in V_j / i} (1 - 2I_{i'j}(1))}{2}$$

$$I_{ij}(1) = k_{ij} P_i \prod_{j' \in C_i / j} Q_{j'i}(1) \qquad I_{ij}(0) = k_{ij} (1 - P_i) \prod_{j' \in C_i / j} Q_{j'i}(0) \tag{2.28}$$

which are costly, but the LL decoder uses additions instead which is more preferable. LL decoder has the same steps as defined in the APP decoder with different message definitions as the log-likelihood ratios;

$$LL(x_i) = \ln\left(\frac{P(x_i = 0 \mid y)}{P(x_i = 1 \mid y)}\right) \qquad LL(I_{ij}) = \ln\left(\frac{I_{ij}(0)}{I_{ij}(1)}\right)$$

$$LL(Q_{ij}) = \ln\left(\frac{Q_{ji}(0)}{Q_{ji}(1)}\right) \qquad LL(D_i) = \ln\left(\frac{D_i(0)}{D_i(1)}\right) \tag{2.29}$$

The derivation of parameters such as check node message $LL(Q_{ji})$, variable node message $LL(I_{ij})$ and decision message $LL(D_i)$ are as follows:

- To obtain the check node message $LL(Q_{ji})$, replace $Q_{ji}(0)$ with $Q_{ji}(0)=1-Q_{ji}(1)$ in (2.28) and obtain

$$1-2Q_{ji}(1)=\prod_{i'\in V_j/i}\left(1-2I_{i'j}(1)\right)$$

Use the following identity;

$$\tanh(\frac{1}{2}\ln(\frac{Q_{ji}(0)}{Q_{ji}(1)}))=Q_{ji}(0)-Q_{ji}(1)$$

Then;

$$\tanh(\frac{1}{2}LL(Q_{ji}))=\prod_{i'\in V_j/i}\tanh\left(\frac{1}{2}LL(I_{i'j})\right)$$

$$LL(Q_{ji})=2.\tanh^{-1}\left(\prod_{i'\in V_j/i}\tanh\left(\frac{1}{2}LL(I_{i'j})\right)\right) \qquad (2.30)$$

- To obtain the variable node message $LL(I_{ij})$ we do the following:

$$\frac{I_{ij}(0)}{I_{ij}(1)}=\frac{k_{ij}(1-P_i)\prod_{j'\in C_i/j}Q_{j'i}(0)}{k_{ij}P_i\prod_{j'\in C_i/j}Q_{j'i}(1)}$$

$$\ln\left(\frac{I_{ij}(0)}{I_{ij}(1)}\right)=\ln\left(\frac{1-P_i}{P_i}\right)+\sum_{j'\in C_i/j}\ln\left(\frac{Q_{j'i}(0)}{Q_{j'i}(1)}\right) \qquad (2.31)$$

$$LL\left(I_{ij}\right)=LL\left(x_i\right)+\sum_{j'\in C_i/j}LL\left(Q_{j'i}\right)$$

- To obtain the decision message $LL(D_i)$ we do the same procedure for the $LL(I_{ij})$:

$$\frac{D_i(0)}{D_i(1)} = \frac{k_{ij}(1-P_i)\prod_{j'\in C_i} Q_{j'i}(0)}{k_{ij}P_i\prod_{j'\in C_i} Q_{j'i}(1)}$$

$$\ln\left(\frac{D_i(0)}{D_i(1)}\right) = \ln\left(\frac{1-P_i}{P_i}\right) + \sum_{j'\in C_i}\ln\left(\frac{Q_{j'i}(0)}{Q_{j'i}(1)}\right) \tag{2.32}$$

$$LL(D_i) = LL(x_i) + \sum_{j'\in C_i} LL(Q_{j'i})$$

One follows the algorithm as defined for APP decoder; initialization, check node update, variable node update and decision with the message definitions given above. In addition to message updates decision step is updated as below:

$$z_i = \begin{cases} 1 & \text{if } LL(D_i) < 0 \\ 0 & \text{else} \end{cases} \tag{2.33}$$

Similarly, if all parity check equations are satisfied, i.e. $Hc^T = \vec{0}$ where $c = (z_1, \ \ldots \ z_n)$, or if a certain maximum number of iterations is reached the algorithm stops, else goes back to the second step, check node update step, and iteration number is incremented.

To reduce the complexity of the algorithm there are some assumption that can be made on the LL decoder. One example for these methods is as follows:

- $LL(I_{ij})$ can be divided into its sign α_{ij} and its magnitude β_{ij} so as to rewrite;

$$\alpha_{ij} = sign\left(LL\left(I_{ij}\right)\right) \qquad \beta_{ij} = \left|LL\left(I_{ij}\right)\right|$$

$$LL(Q_{ji}) = 2.\tanh^{-1}\left(\prod_{i'\in V_j/i} \tanh\left(\frac{1}{2}LL(I_{i'j})\right)\right)$$

$$LL(Q_{ji}) = \left(\prod_{i'\in V_j/i} \alpha_{ij}\right) 2.\tanh^{-1}\left(\prod_{i'\in V_j/i} \tanh\left(\frac{1}{2}\beta_{ij}\right)\right)$$

$$LL(Q_{ji}) = \left(\prod_{i'\in V_j/i} \alpha_{ij}\right) 2.\tanh^{-1} \ln^{-1} \ln\left(\prod_{i'\in V_j/i} \tanh\left(\frac{1}{2}\beta_{ij}\right)\right)$$

$$LL(Q_{ji}) = \left(\prod_{i'\in V_j/i} \alpha_{ij}\right) 2.\tanh^{-1} \ln^{-1}\left(\sum_{i'\in V_j/i} \ln.\tanh\left(\frac{1}{2}\beta_{ij}\right)\right)$$

$$LL(Q_{ji}) = \left(\prod_{i'\in V_j/i} \alpha_{ij}\right) f\left(\sum_{i'\in V_j/i} f\left(\beta_{ij}\right)\right)$$

$$where \quad f(\beta) = -\ln\left(\tanh\left(\frac{1}{2}\beta\right)\right) = \ln\left(\frac{e^{\beta+1}}{e^{\beta-1}}\right) \tag{2.34}$$

The property for this function is that $f^{-1}(\beta) = f(\beta)$ for $\beta > 0$ and the term corresponding to the smallest β_{ij} in (2.34) dominates the function,

$$f\left(\sum_{i'\in V_j/i} f\left(\beta_{ij}\right)\right) \cong \min_{i'\in V_j/i} \beta_{ij} \tag{2.35}$$

As a result, (2.34) is reduced to:

$$LL(Q_{ji}) = \left(\prod_{i'\in V_j/i} \alpha_{ij}\right) \min_{i'\in V_j/i} \beta_{ij} \tag{2.36}$$

With this assumption the product of the signs can be calculated by using modulo-2 addition and finding the minimum simplifies the calculations.

2.3 Non-binary LDPC Codes

The motivation for this section is the paper [Davey-Mackay-1998], where authors consider LDPC codes over higher order finite fields and report a significant improvement in decoding over the performance of binary LDPC codes. So, we focus on non-binary LDPC codes and their decoding issues.

2.3.1 Description of LDPC codes over GF(q)

The description of a low-density parity-check matrix H and generator matrix G over GF(q) is similar to the binary case. The elements of these matrices are defined over finite fields GF(q) with q elements where $q=p^s$, that p is some prime integer and s is an integer. We shall call the elements of GF(q) with $q>2$, symbols, whereas in the binary case we call them bits.

A q-ary LDPC code is defined as a subspace of the vector space GF(q)^n and consists of a set of codewords satisfying the constraints specified by an $m \times n$ sparse parity-check matrix H such that;

$$\sum_{j=1}^{n} H_{ij} c_j = 0, \qquad i = 1,...,m \tag{2.37}$$

where each element of the parity check matrix, $H_{ij} \in$ GF(q), $c_j \in$ GF(q), addition and multiplication are defined over GF(q).

The graphical representation of a q-ary code which is known as *Tanner graph* is also similar to the binary one, but each edge $e(v_i, c_j)$ of the graph is weighted such that $w(v_i, c_j) = H_{ij}$.

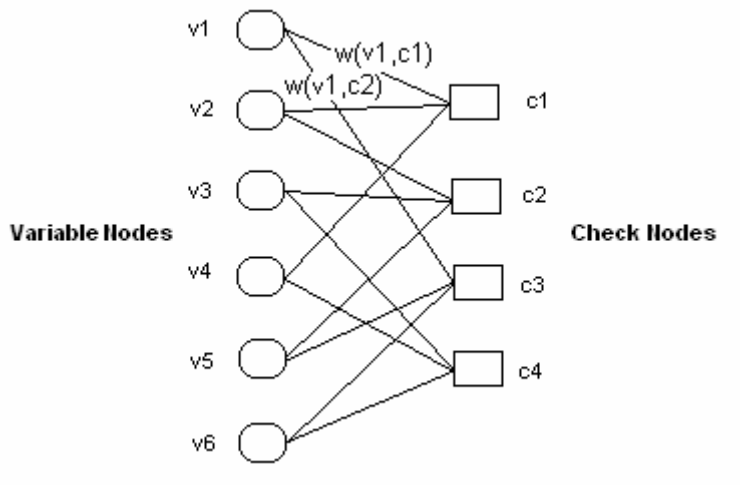


Figure 2.7 Tanner graph for a q-ary LDPC code

As with binary LDPC codes, we say that LDPC code in GF(q) is *regular* if all variable nodes in its Tanner graph have the same degree, and all check-nodes have the same degree. Otherwise, we say it is *irregular*.

Mackay showed that going from binary to non-binary field may reduce the number of cycles in the Tanner graph. If we associate an $i \times i$ matrix for every element in GF(q) where $q=2^i$ then we can substitute the H matrix in GF(q) domain with an equivalent H matrix which is i times longer in each dimension in binary domain.

As an example, let H matrix in GF(4) be given as;

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 3 & 0 & 2 \end{bmatrix}$$

The equivalent binary H matrix is;

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

where $0=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}, 1=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}, 2=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}, 3=\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}$ are the elements which uniquely satisfy GF(4) addition and multiplication [Davey-Mackay-1998].

Then, the Tanner graph for binary and non-binary cases will be;

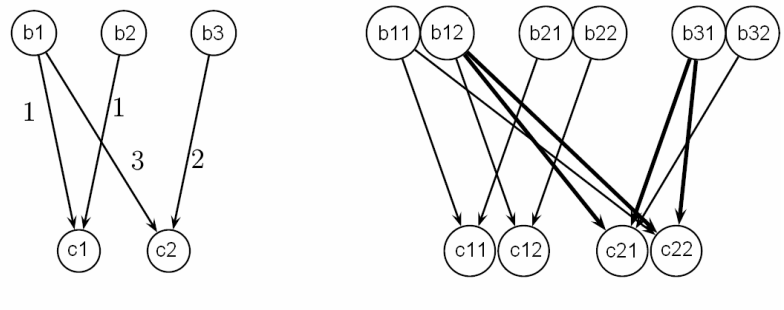


Figure 2.8 Comparison of GF(2) and GF(4) Tanner graphs of a 4-ary LDPC code

From the Tanner graph above, it can be seen that there is a length-4 cycle while there is not a cycle in non-binary graph.

Although non-binary alphabet reduces cycles, it has a drawback such that the decoding complexity increases with the increasing number of elements to be decided during the decoding steps.

2.3.2 Decoding over GF(q)

Decoding issue for non-binary LDPC codes is considered in the same way as in binary codes with a difference in message structure. For binary decoding there are two probabilities for decision, but decoding process is more complicated for non-binary decoding since the messages are probability vectors with q elements in GF(q).

The complexity of decoding and related approximation methods will be discussed in the following section with message passing decoding algorithms over GF(q).

2.3.3 Message-Passing Algorithms over GF(q)

The decoder's aim is to recover the codeword c that is an element of the LDPC code over GF(q). Decoding algorithm consists of alternating variable node messages and check node messages. In a variable node update process, messages are sent from variable nodes to check nodes. In a check node update process, the opposite occurs. Note that it is same as defined for binary LDPC codes.

In standard binary LDPC codes, messages are scalar values but in non-binary LDPC case the messages are q-dimensional probability vectors. Different decoding algorithms are mainly based on different types of the messages as defined for binary LDPC codes, A Posteriori Probability (APP) and Log-Likelihood (LL) decoders.

In the following sections we will mention the differences for APP and LL decoders over non-binary alphabet and describe another decoding algorithm called Fourier Transform (FT) decoder.

- **A Posteriori Probability (APP) Decoder over Non-binary Alphabets:**

For binary codes, in the message passing algorithm the messages that are passed along the edges are probabilities, i.e., real values representing conditional probabilities, likelihood ratios or log-likelihood ratios. For codes over non-binary alphabets, say q-ary codes, the messages that are passed along the edges are vectors of length q representing a probability distribution over $\{\alpha^0,...,\alpha^{q-1}\}$, where $\alpha^0,...,\alpha^{q-1}$ are the elements of the alphabet. With the notation above, the α^i's, $i = 0,...,q-1$, represent the elements a of GF(q). So the message from a variable node to a check node for APP decoder can be updated as;

$$I_{v\to c} = \left(P(x_i = 0 \mid y),..., P(x_i = q-1 \mid y)\right) \tag{2.38}$$

The algorithm has the same steps as defined over binary alphabet with the following differences for non-binary case.

1) Initialization

The algorithm starts with the initialization of iteration number as 1 and $I_{ij}(v)$ values with $P_i(v)$ for all i, j and for $v \in \mathrm{GF}(2^s)$.

$$I_{ij}(v) = P_i(v) = P(x_i = v \mid y_i) = \prod_{k=1}^{s} P(x_{i_k} = v_k \mid y_i) \tag{2.39}$$

where v_k and x_{i_k} are the k'th bit of the binary representations of v and x_i.

2) Check Node Update

The way of calculating the check node message $Q_{ji}(v)$ can also be updated as follows for non-binary alphabet;

$$Q_{ji}(v) = \sum_{x_i = v} P(\text{parity - check equation } c_j \text{ is satisfied} \mid x') \prod_{i' \in V_j / i} I_{i'j}(x'_{i'}) \tag{2.40}$$

where the i'th entry of x' is fixed at v ,i. e., we write $x' = (x'_{i1}, x'_{i2}, \ldots, x_i = v, \ldots, x'_{in})$, and $P(\text{parity-check equation } c_j \text{ is satisfied} \mid x') \in \{0, 1\}$ is an indicator function. It is 0 if x' does not satisfy parity-check c_j, and it is 1 if x' satisfies parity-check c_j. So, the probability is calculated by summing over all configurations of x' for which the parity check is satisfied and also the variable node is equal to symbol v.

3) Variable Node Update

The way of calculating the variable node message $I_{ij}(v)$ can also be updated for $q>2$ as follows;

$$I_{ij}(v) = k_{ij}(v) P_i(v) \prod_{j' \in C_i / j} Q_{j'i}(v) \tag{2.41}$$

where $k_{ij}(v)$ are the normalization factors chosen to satisfy $\sum_{v \in GF(q)} I_{ij}(v) = 1$.

4) Decision

Similarly, decision values can be computed with non-binary alphabet as;

$$D_i(v) = k_{ij}(v) P_i(v) \prod_{j' \in C_i} Q_{j'i}(v) \tag{2.42}$$

such that the constants $k_{ij}(v)$ are chosen to satisfy $\sum_{v \in GF(q)} D_i(v) = 1$.

Each bit is decided as:

$$z_i = \arg\max_{v \in GF(q)} D_i(v) \tag{2.43}$$

After decision for each bit is made, the check whether the parity check equations are satisfied is;

$$Hc^T = 0 \tag{2.44}$$

where multiplication is defined over GF(q). If all parity check equations are satisfied or if a certain maximum number of iterations is reached, the algorithm stops; else goes back to the second step, and the iteration number is incremented.

An important point about this algorithm is that it has a decoding complexity with the search of the combination that satisfies parity check equation for q length vector in the check node update step as in (2.40). This is why all the combinations that satisfy the parity check equation should be checked for the number of the column weight.

- **Log-Likelihood (LL) Decoder over Non-binary Alphabet**

LL decoder has the same steps as defined for the binary LL decoder, with some differences in the messages and their calculation methods. As for binary case, the LL-vector is of size q. The algorithm is as follows:

1) Initialization

The message calculation in the initialization step is similar to the binary LL decoder;

$$LL(x_i = j) = \ln\left(\frac{P(x_i = 0 \mid y)}{P(x_i = j \mid y)}\right) \quad where \quad j \in GF(q) \tag{2.45}$$

with only a change in the alphabet size.

2) Check Node Update

The check node update process is different since it first uses inverse log-likelihood function LL^{-1} of messages to convert them into probabilities. The LL^{-1} function is defined as

$$LL^{-1}(x) = \frac{e^{-x_i}}{1 + \sum_{k=1}^{q-1} e^{-x_k}} \quad where \quad i = 0,...,q-1 \tag{2.46}$$

Then LL of messages is taken to obtain $LL(Q_{ji}(v))$*;*

$$LL(Q_{ji}(v)) = LL\left[\sum_{x_i = v} P(\text{parity - check equation } c_j \text{ is satisfied} \mid x') \prod_{i' \in V_j / i} LL^{-1}\left[LL(I_{i'j}(x'_{i'}))\right]\right] \tag{2.47}$$

The check message sent from a check node c on an edge $e(v, c)$ is the product of all incoming messages from variable nodes other than v such that first LL^{-1} function is used to obtain probability values then LL function is used to obtain back the log-likelihood values. The reason of using probability values instead of likelihoods is that an easy simplification cannot be done as in (2.22) and (2.24).

3) Variable Node Update

Similar to the binary case, the variable node message $LL(I_{ij}(v))$ is;

$$LL\left(I_{ij}(v)\right) = LL\left(x_i = j\right) + \sum_{j' \in C_i / j} LL\left(Q_{j'i}(v)\right) \tag{2.48}$$

with only a change in the alphabet size.

4) Decision

$LL(D_i)$ again has the same definition as in binary case with a change in the alphabet size:

$$LL\left(D_i(v)\right) = LL\left(x_i = j\right) + \sum_{j' \in C_i} LL\left(Q_{j'i}(v)\right) \tag{2.49}$$

The decision on each bit can be obtained by using LL^{-1} function (2.46):

$$z_i = LL^{-1}\left[LL(D_i(v))\right] = \frac{e^{-LL(D_i(v))}}{1 + \sum_{k=1}^{q-1} e^{-LL(D_k(v))}} \qquad where \quad i = 0,...,q-1 \tag{2.50}$$

The rest of decision step is similar to the binary case.

The LL decoder has similar execution complexity with APP decoder since the LL and LL^{-1} functions are used together.

- **Fourier Transform (FT) Decoder**

To reduce the decoding complexity for both APP and LL decoders in non-binary case, a method called Fourier Transform can be used which is described in [Richardson-Urbanke-2001]. It brings another approach for the message definition as

$$F(x_i = j) = \sum_{k=0}^{q-1} r^{jk} P(x_i = k \mid y) \qquad where \quad r = e^{\frac{2\pi i}{p}}, \quad q = p^s, \qquad j \in GF(q) \quad (2.51)$$

where r is the primitive root of the corresponding finite field and multiplication is defined over GF(q).

For example, the Fourier transform taken over GF(4) is given by;

$$\begin{aligned}
F(x_i = 0) &= \left[P(x_i = 0 \mid y) + P(x_i = 1 \mid y)\right] + \left[P(x_i = 2 \mid y) + P(x_i = 3 \mid y)\right] \\
F(x_i = 1) &= \left[P(x_i = 0 \mid y) - P(x_i = 1 \mid y)\right] + \left[P(x_i = 2 \mid y) - P(x_i = 3 \mid y)\right] \\
F(x_i = 2) &= \left[P(x_i = 0 \mid y) + P(x_i = 1 \mid y)\right] - \left[P(x_i = 2 \mid y) + P(x_i = 3 \mid y)\right] \\
F(x_i = 3) &= \left[P(x_i = 0 \mid y) - P(x_i = 1 \mid y)\right] - \left[P(x_i = 2 \mid y) - P(x_i = 3 \mid y)\right]
\end{aligned} \quad (2.52)$$

since $r = e^{\frac{2\pi i}{2}} = e^{\pi i} = (-1)^i$ for $q = 2^2$.

The inverse transform F^{-1} is given by the same equation (2.52) followed by division by 1/q. As in LL decoder, FT-vector is of size q, where q is the order of the finite field.

The algorithm for the FT decoder has the same steps as APP and LL decoders with the differences given below:

1) Initialization

The algorithm starts with the initialization of the $F(I_{ij}(v))$ values with $F(x_i=j)$ in Equation (2.52), for all i, j and for $v \in$ GF(2^n).

$$F(I_{ij}(v)) = F(x_i = j) \tag{2.53}$$

2) Check Node Update

FT decoder algorithm is similar with the LL and APP decoders but improves the check node update method;

$$\begin{aligned} Q_{ji}(v) &= \sum_{x_i = v} P(\text{parity - check equation } c_j \text{ is satisfied}|x') \prod_{i' \in V_j / i} I_{i'j}(x'_{i'}) \\ &= \sum_{x_i = v} \delta\left(\sum_{i' \in V_j} H_{ij'} x'_{i'} = c_j \right) \prod_{i' \in V_j / i} I_{i'j}(x'_{i'}) \end{aligned} \tag{2.54}$$

which can be defined as a convolution of $I_{ij}(v)$. So this summation can be replaced by a product of Fourier Transforms such as:

$$Q_{ji}(v) = F^{-1}\left[\prod_{i' \in V_j / i} \bullet W\left[\overline{w}(v', c)\right] \cdot F(I_{i'j}(x'_{i'})) \right] \tag{2.55}$$

where we use • for component-wise multiplication between two column vectors, $\overline{w}(v', c)$ is the normalized edge weight satisfying $\overline{w}(v', c).w(v, c) = -w(v', c)$, and $W[w]$ is a $(q-1)\times(q-1)$ matrix (indexed from 1) with entries $W_{ij}[w] = \delta(wi\text{-}j)$ where multiplication and subtraction are defined over GF(q). Note that $W[w]$ is a permuted identity matrix used for satisfying the parity check equations by shuffling the components of $F(I_{ij}(v))$ to take into account the edge weight w.

As an example for GF(4) is

$$W[1] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad W[2] = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad W[3] = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.56}$$

3) Variable Node Update

This step can be followed as in APP or LL decoder. Then for variable node message, one may choose either $I_{ij}(v)$ as in APP decoder (2.41) or $LL(D_i)$ values as in LL decoder (2.48).

4) Decision

Decision step also follows as in APP or LL decoders. In order to give the decision, one may choose either D_i values as in (2.42) or $LL(D_i)$ values as in (2.49). Then the decision on each bit is made as in (2.43) or (2.50) according to the selection. The rest of the algorithm is similar to binary case.

With the FT decoder, the complexity of check node update process is reduced such that search process for the configuration in the parity check equation to be valid as in (2.40) is eliminated and number of calculation steps in multiplication and addition is also reduced by using Fourier transforms.

To sum up, in this chapter we reviewed LDPC codes and iterative decoding algorithms named as message passing decoding algorithms. We investigated different types of message passing decoding algorithms. We also mentioned non-binary LDPC codes and their decoding algorithms. An important point for all of these algorithms is that the computation per digit per iteration is independent of the block length n. Since the algorithm is followed through the edges in the graph, decoding time is proportional to the number of edges in the graph and the alphabet size. So, if the number of edges is of the same order as the number of variable nodes in the Tanner graph, the complexity of the decoding algorithm is linear with block length.

CHAPTER 3

DECODING PERFORMANCE COMPARISON OF BINARY AND NON-BINARY LDPC CODES

In the previous chapters we have investigated the decoding algorithms defined for binary and non-binary LDPC codes. To compare the decoding performances of these algorithms with each other we need to discuss some parameters which are important from the decoding side of view and decide on the criteria of comparison.

Firstly, in section 3.1 we investigate the parameters which are effective on the message passing decoding algorithms for LDPC codes that we have implemented over GF(2), GF(4) and GF(8) alphabets. Then, in section 3.2, simulation results are given and discussed.

3.1 Simulation Settings for Performance Comparison

The aim of our simulations is to compare the performance of binary and non-binary LDPC codes that we have implemented over GF(2), GF(4) and GF(8) alphabets. Hence, simulation parameter settings such as code properties, channel properties and decoder properties need to be specified as we do in the following sections.

3.1.1 Code Properties That Affect Decoding Performance

The configuration of our simulations related to the code properties can be described in terms of the following five items:

i) Alphabet

We design the pseudo-random LDPC codes used in this study over GF(2), GF(4) and GF(8) in order to test the effect of the alphabet on the performance.

ii) Code Length

The length of the codewords, i.e., the codelength is expected to affect the performance of binary and non-binary LDPC codes. In order to make a fair comparison among different Galois fields, we choose two different codeword lengths, 672 and 896 bits for GF(2), and adjust the codeword lengths for GF(4) and GF(8) accordingly.

iii) Code Rate

We choose two fixed rates, i.e., ½ and ¾, to test the effect of rate on the performance of the LDPC codes over GF(2), GF(4) and GF(8). Combining the parameters mentioned above, the implemented codes have the variety of parity check matrix sizes summarized in Table 3.1.

Table 3.1 Size of the parity check matrices used in the simulations

Rate / Field	½		¾	
GF(2)	336×672	448×896	168×672	224×896
GF(4)	168×336	224×448	84×336	112×448
GF(8)	84×168	112×224	42×168	56×224

iv) Regularity

Studies in literature show that the irregular structure of *H* matrix with a careful design has better performance over regular LDPC codes [Davey-Mackay-1998], [Luby-Mitzenmacher-Shokrollahi-Spielman-1998]. The advantage of irregular code design can be summarized as follows:

Ideally a variable node should be connected to many check nodes to gain the maximum amount of information about its error-free value. Conversely, a check node prefers few variable nodes, so that it can provide more confident estimates for the state of each variable node. So choosing a few specific variable nodes with many connections can determine their correct values with high probability, in an early stage of decoding iterations. Such “elite” variable nodes then become the source of accurate information. The design of irregular codes makes use of this strategy.

In the simulations, we mainly use regular LDPC codes in comparing different types of codes, but we also give an example to see the effect of moving from regular to irregular structure. For all the regular codes, we fix the variable node degree as 3.

v) Girth and Cycle

Short length cycles have a lowering effect on the decoding performance as shown in the Example 2 in Section 2.2.5.1.1. Since the message passing decoding algorithms assume no-cycles in the ideal case, the existence of cycles brings errors in decoding. Cycles introduce feedback in the flow of algorithmic messages, allowing bits that are part of a cycle to stimulate themselves with their own possibly erroneous state. This violates the message passing decoding principle that messages should contain only extrinsic information.

That's why code design with the largest possible girth is desired to achieve better performance. There is a convention on published works that the constraint of larger than 4-girth provides sufficient feedback protection [Mackay-1999], [Lin-Costello-2004], [Gallager-1962]. So we also generate LDPC codes with girth greater than 4 in our simulations. In order to construct these codes, we utilize the Mackay and Neal construction method mentioned in Section 2.2.3.

3.1.2 Channel Properties

We use the AWGN channel model with BPSK modulation for our simulations. For this model the channel output $y=x+n$, where the transmitted signal $x = \pm 1$, and n is the Gaussian distributed channel noise with zero mean and variance σ^2. Hence the conditional probability density function of $(y \mid x)$ is;

$$f\ (y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-x)^2}{2\sigma^2}} \tag{3.1}$$

Calling the transmitted symbol energy E_s, each information bit of the codeword uses a bit energy $E_b = E_s / R$, where R is the code rate. If one sided noise spectral density is N_0, the variance of noise is $\sigma^2 = N_0 / 2$. Hence,

$$\frac{\sigma^2}{E_s} = \frac{N_0 / 2}{RE_b} \tag{3.2}$$

and the variance of noise is found using (3.2) as;

$$\sigma^2 = E_s \left(2R \frac{E_b}{N_o} \right)^{-1} \tag{3.3}$$

Since the transmitted symbol is $x = \pm 1 = \pm\sqrt{E_s}$, the symbol energy $E_s = 1$, so;

$$\sigma^2 = \left(2R \frac{E_b}{N_o} \right)^{-1} \tag{3.4}$$

3.1.3 Decoder Properties

We have used three different forms of message passing decoders, which are differentiated by the soft information that is used in decoding. This soft information is chosen as one of the following:

i) a posteriori probabilities (APP)

ii) log likelihood ratio of APP's

iii) Fourier transform of APP's

Corresponding decoders are called,

i) APP decoder

ii) LL decoder

iii) FT decoder

in the remaining part of this work, for quick reference. We have then compared the performance of the APP and LL decoders for each of the constructed LDPC codes. The performance of FT decoder has been evaluated only for one of these codes.

A crucial parameter in decoding is the maximum number of iterations that the algorithm is allowed to do. Because, all message passing decoders stop either when all parity check equations are satisfied or a certain maximum number of iterations is reached.

So the maximum number of allowed iterations should be set very carefully considering its effect on the performance of the decoder. To determine the correct value of the maximum number of iterations, we have performed some experiments using the LDPC code with a parity check matrix H of size 448×896. We have used the APP decoder until 20 codeword errors are counted for each E_b / N_0 value in the set of {1, 1.5, 2, 2.5, 3} dB. The maximum iteration number I_{max} is set to 100 iterations, but especially at high SNR's, the algorithm decides on a correct codeword at a much smaller iteration step. To reach the 100^{th} iteration step is the indication of unsuccessful decoding, with no codeword at the decoder output. The histogram of performed iteration numbers are sketched in Figure 3.1, for different SNR values.

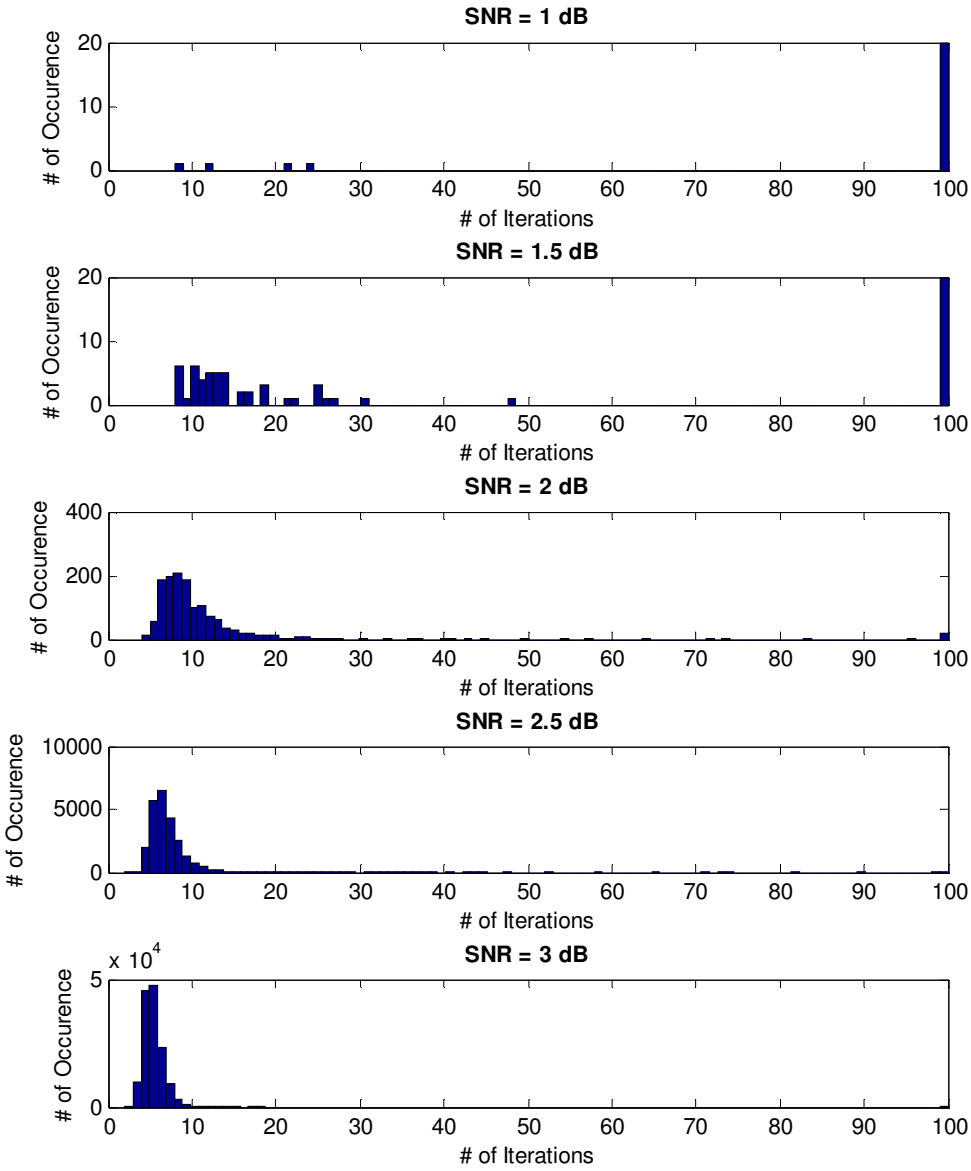


Figure 3.1 Iteration number histogram for different SNR values with counting 20 codeword errors with I_{max}=100

Figure 3.1 shows that I_{max}=100 is not sufficient for SNR=1dB, but it is more than sufficient for SNR≥2dB, since more than 90% of decoding trials end up successfully at an iteration step smaller than 50.

3.2 Simulation Results and Discussion

In this section, we give our simulation results and compare the performance of the constructed LDPC codes with respect to various parameters. We measure the performance in terms of the "bit error ratio versus signal to noise ratio", or shortly, BER versus SNR. Our simulations are made as follows:

1) We generate the LDPC code with desired properties, avoiding 4-cycles in H.

2) We set the maximum iteration number for all decoding methods to 50.

For each level of SNR in the set of {1, 1.5, 2, 2.5, 3, 3.5, 4} dB,

3) We calculate σ value as defined in Equation (3.3) and generate noise samples which are to be added to the sent codeword. The codeword is produced by encoding a random sequence of binary numbers.

4) To stop the simulation we count up to 20 codeword errors for each SNR level.

5) Finally we calculate the BER and note this value.

We apply these steps for all the LDPC codes described in the previous section, using a Matlab R2006a program on a computer with Pentium D 3.2 MHz CPU and 2 GB RAM. Results and related discussions are given in the following subsections.

3.2.1 Effect of Code Properties

Simulation results of this part will be given in the order of:

a) Performance of LDPC codes over GF(2)

b) Performance of LDPC codes over GF(4)

c) Performance of LDPC codes over GF(8)

d) Effect of field

e) Effect of irregularity

All performance curves are found in terms of "BER versus SNR" using an APP decoder, which uses a posteriori probabilities as the soft information.

a) For GF(2):

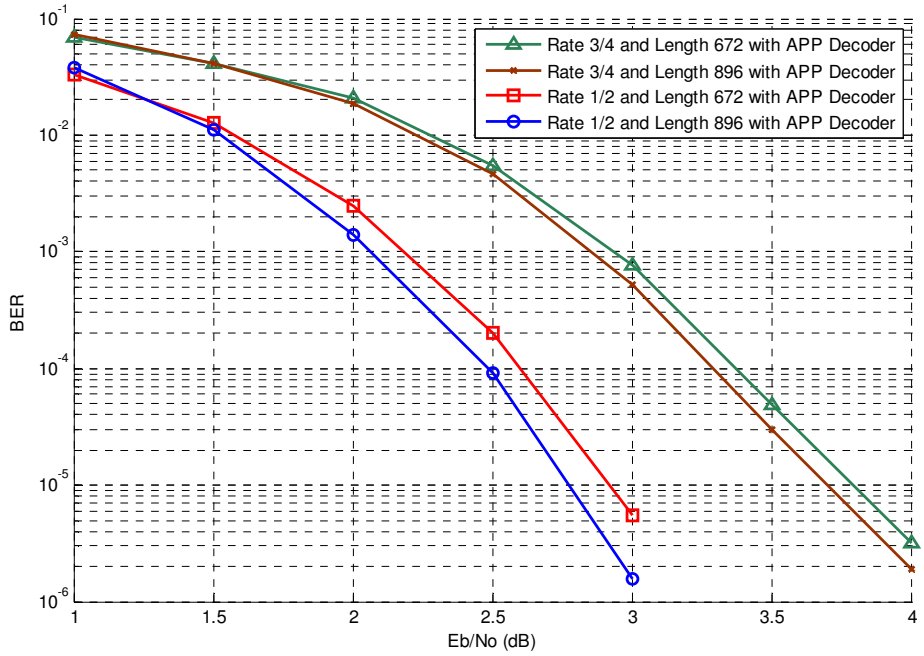


Figure 3.2 Performance comparison for rates ½ and ¾ and lengths 896 or 672 binary LDPC codes with APP decoder

One can compare the performances of the four LDPC codes designed over GF(2) with code rates of ½ and ¾ and code lengths of 896 and 672 bits, using Figure 3.2.

For code rate comparison, we have two different examples with ½ and ¾ code rates. When the code length is 896 bits, we observe from Figure 3.2 that the performance decreases by ~0.8dB at BER=10^{-4}, as the code rate increases. Similarly

for the code length of 672 bits, SNR loss is ~0.75dB at BER=10^{-4}, when we increase the code rate from ½ to ¾.

For code length comparison, we have two different examples with codeword lengths of 896 and 672 bits. At rate ½, the graph shows that as the code length increases, performance increases by ~0.15dB at BER=10^{-4}. Similarly for the code rate of ¾, increasing the codeword length from 672 to 896 results in an SNR gain of ~0.1dB at BER=10^{-4}.

Note that there is a contradiction with the result described above, when the SNR is below 1.25dB, probably because the maximum number of iterations is not sufficient for low SNR's.

b) For GF(4):

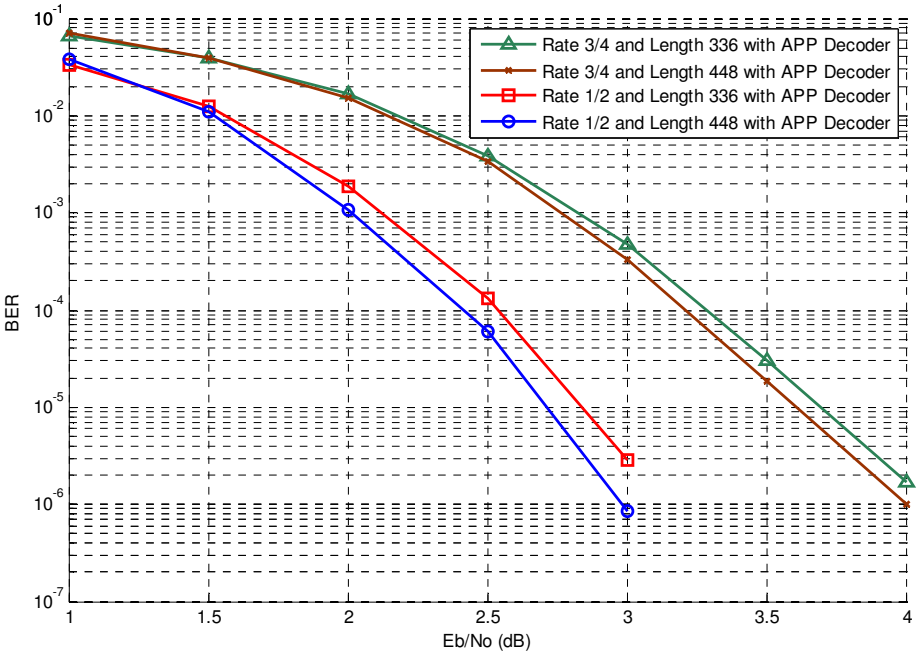


Figure 3.3 Performance comparison for rates ½ and ¾ and lengths 448 or 336 symbols LDPC codes for GF(4) with APP decoder

Figure 3.3, shows the performance curves of LDPC codes over GF(4) with code rates of ½ and ¾ and code lengths of 448 and 336 symbols.

For code rate comparison, when the code length is 448 symbols, Figure 3.3 shows that the decoding performance decreases by ~0.8dB at BER=10^{-4} as the code rate increases. Similarly for the code length of 336 symbols, the same result is obtained with a decrease of ~0.75dB at BER=10^{-4}, when we increase the code rate from ½ to ¾.

For code length comparison, we have two different examples with code lengths of 448 and 336 symbols. At rate ½, the graph shows that as the code length increases, SNR gain is ~0.12dB at BER=10^{-4}. Similarly at rate ¾, if the codeword length increases from 336 to 448 symbols, SNR gain of ~0.1dB at BER=10^{-4} is obtained.

c) For GF(8):

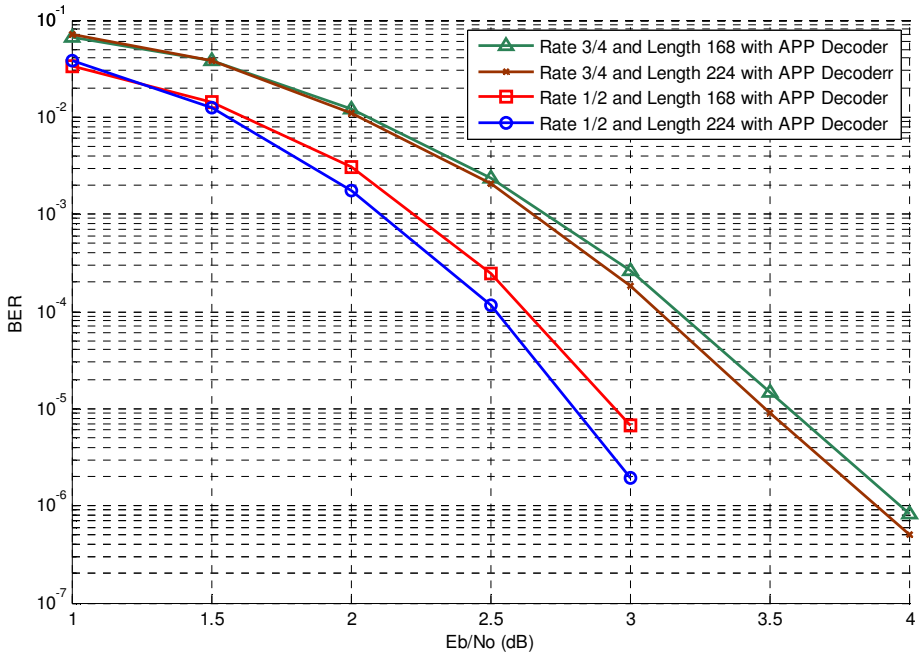


Figure 3.4 Performance comparison for rates ½ and ¾ and lengths 224 or 168 symbols LDPC codes for GF(8) with APP decoder

Using Figure 3.4, one can compare the performances of LDPC codes with code rates of ½ and ¾ and code lengths of 224 and 168 symbols.

For code rate comparison, we have the similar results as in GF(2) and GF(4) codes. When the code length is 224 symbols, we observe from Figure 3.4 that the decoding performance decreases by ~0.6dB at BER=10^{-4} as the code rate increases. Similarly for the code length of 168 symbols, the same result is obtained with a decrease of ~0.65dB at BER=10^{-4}, when we increase the code rate from ½ to ¾.

For code length comparison, we have two different examples with code lengths of 224 and 168 symbols. At rate ½, the graph shows that as the code length increases, decoding performance increases by ~0.1 dB at BER=10^{-4}. Similarly at rate ¾, codeword length increase results in an SNR gain of ~0.08dB at BER=10^{-4}.

The summary of the results given in this subsection is shown in Table 3.2.

Table 3.2 Performance comparison of LDPC codes versus code parameters

Field over which the LDPC code is generated	(*n* or *R*, change in *n* or *R*)	For BER=10^{-4}, SNR loss in dB
GF(2)	(*n*=896, *R*: ½ to ¾)	0.8
	(*n*=896, *R*: ½ to ¾)	0.75
	(*R*=½, *n*: 896 to 672)	0.15
	(*R*=½, *n*: 896 to 672)	0.1
GF(4)	(n=896, R: ½ to ¾)	0.8
	(n=896, R: ½ to ¾)	0.75
	(R=½, n: 896 to 672)	0.12
	(R=½, n: 896 to 672)	0.1
GF(8)	(n=896, R: ½ to ¾)	0.6
	(n=896, R: ½ to ¾)	0.65
	(R=½, n: 896 to 672)	0.1
	(R=½, n: 896 to 672)	0.08

The last column of the Table 3.2 is very similar for GF(2), GF(4) or GF(8) codes. So, one can say that if the code rate increases from ½ to ¾, code performance decreases by 0.6-0.8dB at BER=10^{-4}. On the other hand, if the codeword length is reduced to its ¾'th, corresponding loss in performance is ~0.1dB, independent of the field over which the code is designed.

The results given above are obtained by generating an LDPC code randomly with the properties given in Table 3.1. To show the performance difference between random LDPC codes having the same parameters, we generate many random LDPC codes with the same parameters over GF(2). Their performances are shown in Figure 3.5;

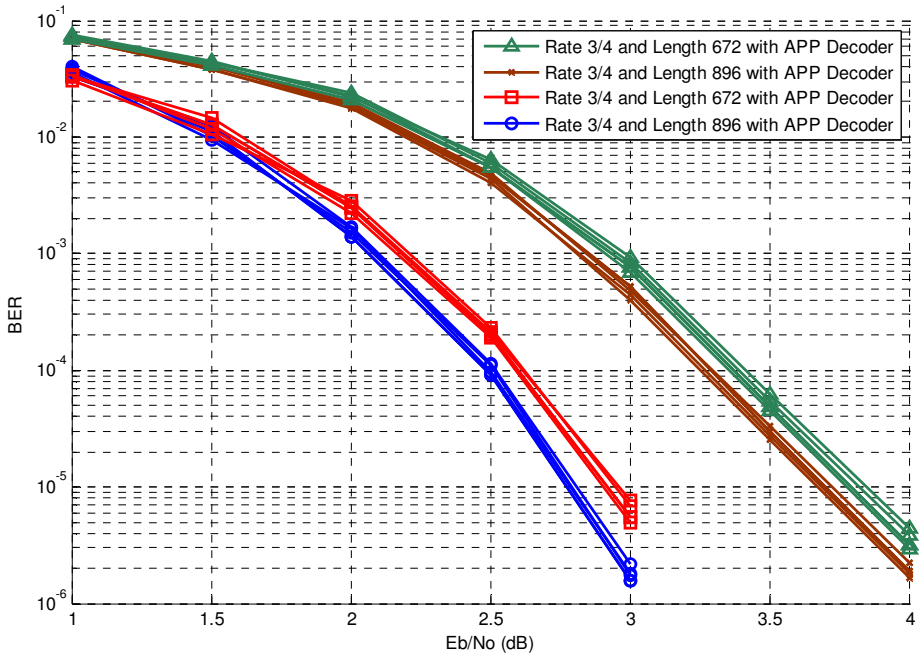


Figure 3.5 Performances of many rate ½, rate ¾, and lengths 896 or 672 binary LDPC codes for GF(2) with APP decoder

Using Figure 3.5, one can conclude that the performances of random LDPC codes with the same parameters do not differ very much. For code rate ½, both length-896 codes and length-672 codes have maximum ~0.04dB difference at BER=10^{-4}. At

rate ¾, length-896 codes differ by ~0.05dB and length-672 codes differ by ~0.06dB at BER=10^{-4}.

d) Effect of field:

We now compare the performance results with respect to the alphabet over which the codes are designed. To examine the difference between the alphabets GF(2), GF(4) and GF(8), we sketch the performance of the LDPC codes with rate ½ and ¾ decoded by APP decoder, in Figure 3.6, Figure 3.7 and Figure 3.8;

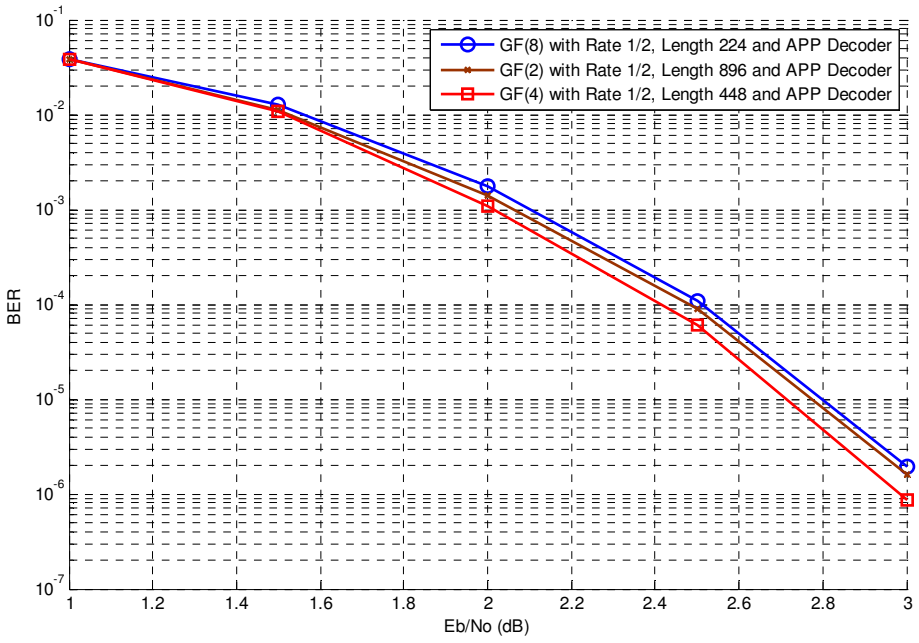


Figure 3.6 Performance comparison for rate ½ and length 896 regular (d_v=3) LDPC codes with APP Decoder over GF(2), GF(4) and GF(8) alphabets

Using Figure 3.6, one can conclude that that for rate ½ codes, the best performance is obtained by GF(4)-code. Second and third performances belong to GF(2) and GF(8) codes, which have respective SNR losses of ~0.07dB and ~0.1dB with respect to the GF(4)-code at BER=10^{-4}. This unexpected ordering can also be seen in [Davey-Mackay-1998], where it is shown that rate-½ and mean-column-weight-3

codes over GF(4) will perform the best followed by codes over GF(2) and then codes over GF(8), which is exactly what our results show.

The reason can be attributed to the dependence of the decoding algorithm on the matrix weight since for a given field order and rate, mean column weight defines the order of performances of different fields according to Davey's study. He uses Monte Carlo methods to simulate LDPC codes of infinite length whose associated graphs have a tree structure and hence the decoding algorithm is known to be exact and he compares the decoding performance on the basis of rate and matrix column weight. Davey shows the effect of changing the field order and mean column weight on the decoding algorithm. His rate ½ codes over GF(8) perform the best at mean column weight of 2.8, followed by GF (4) and then GF(2). In Figure 3.7 we will use the same configuration to examine this characteristics with an irregular structure of mean d_v=2.8 generated by the variable node distribution chosen as $\lambda(x) = 0.2x^2 + 0.8x^3$.

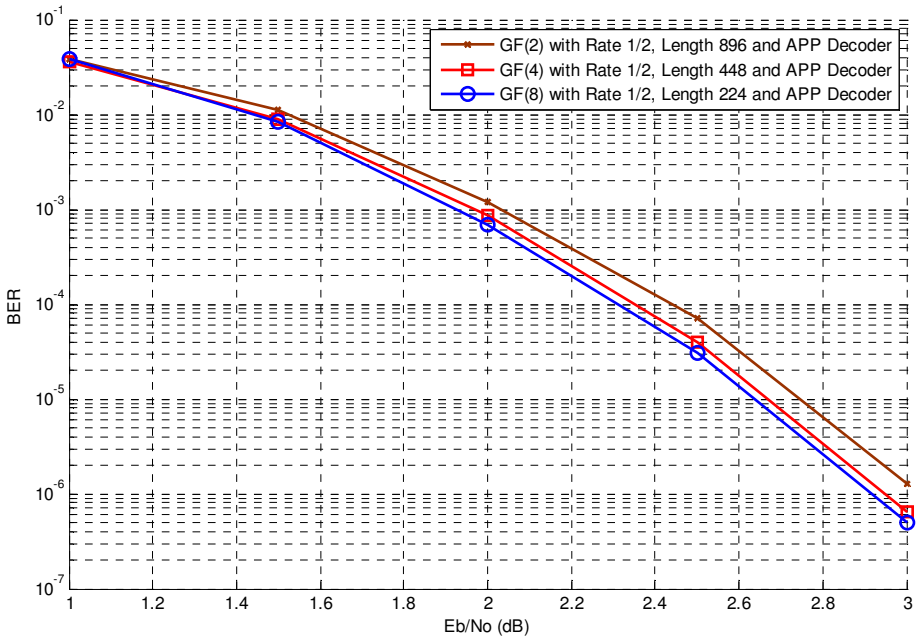


Figure 3.7 Performance comparison for rate ½ and length 896 irregular (mean d_v=2.8) LDPC codes with APP decoder over GF(2), GF(4) and GF(8) alphabets

From Figure 3.7, one can see an improvement of ~0.08dB at BER=10^{-4} when moving from GF(2) to GF(4) and an improvement of ~0.04dB at BER=10^{-4} when moving from GF(4) to GF(8). This result is in the expected ordering and confirms Davey's results.

To see the effect of further decrease in variable node degree, we select an irregular LDPC code of mean d_v=2.6 and variable node distribution $\lambda(x) = 0.4x^2 + 0.6x^3$.

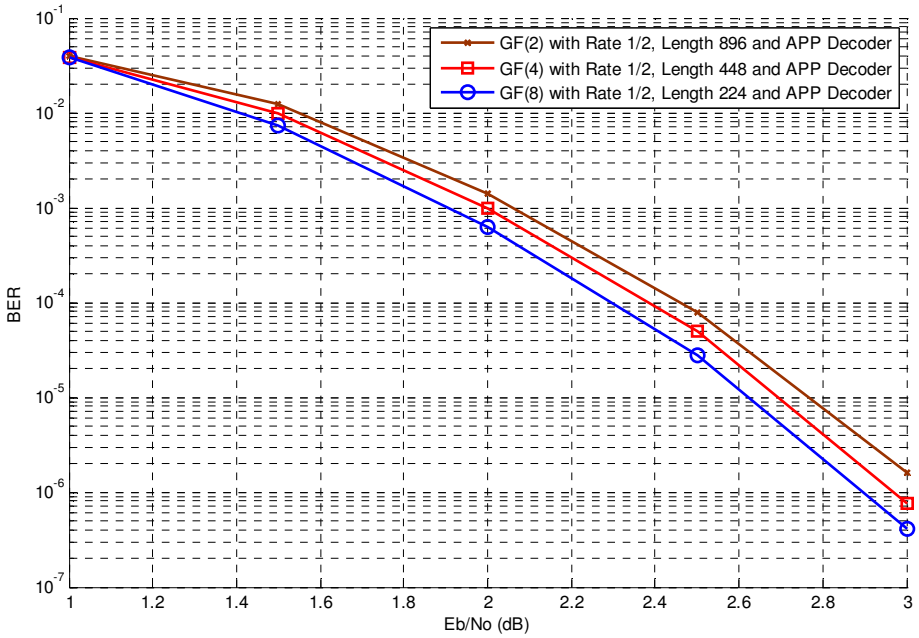


Figure 3.8 Performance comparison for rate ½ and length 896 irregular (mean d_v=2.6) LDPC codes with APP decoder over GF(2), GF(4) and GF(8) alphabets

From Figure 3.8, one observes that the ordering is similar to the previous case and an improvement of ~0.07 dB at BER=10^{-4} when moving from GF(2) to GF(4) and an improvement of ~0.09 dB at BER=10^{-4} when moving from GF(4) to GF(8) is obtained. This simulation shows that the performance of GF(8) is better for mean d_v=2.6 than mean d_v=2.8 and slightly worse for GF(2) and GF(4) alphabets when we compare the results of mean d_v=2.6 than mean d_v=2.8 cases. These results shown

in figures 3.6, 3.7 and 3.8 agree with Davey's Monte Carlo simulation results for d_v=3, mean d_v=2.8, and mean d_v=2.6.

In order to show the performance differences between random LDPC codes having the same parameters, we generate many random LDPC codes with the same parameters over GF(2), GF(4) and GF(8). Their bit error ratio versus bit signal to noise ratio performances are shown in Figure 3.9, Figure 3.10 and Figure 3.11, for regular codes with variable node degree means of 3, 2.8 and 2.6 respectively.

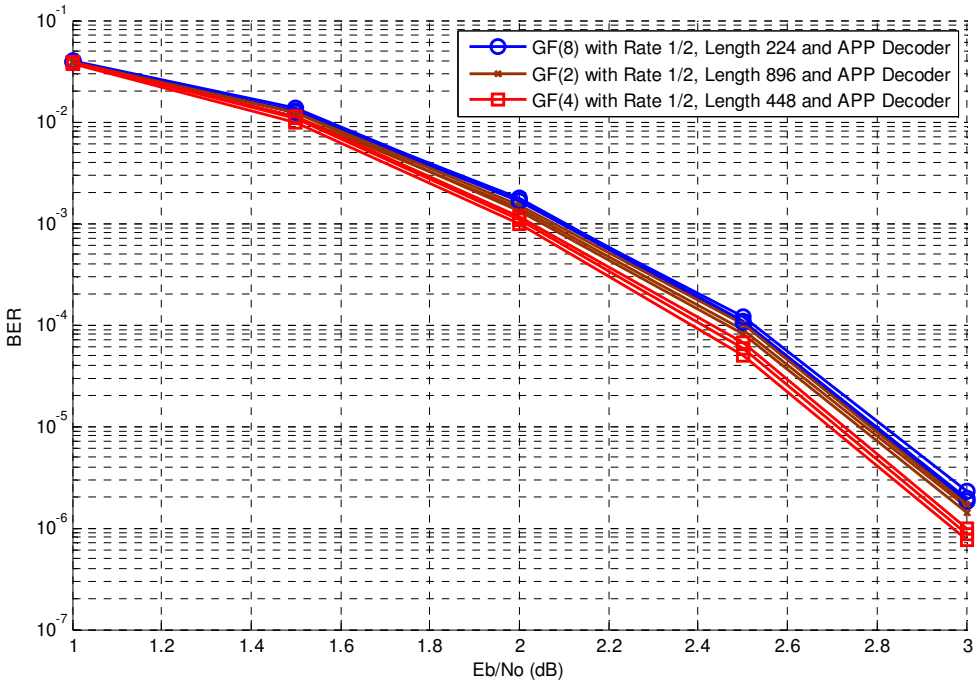


Figure 3.9 Performance comparison of many rate ½ and length 896 regular (d_v=3) LDPC codes with APP Decoder over GF(2), GF(4) and GF(8) alphabets

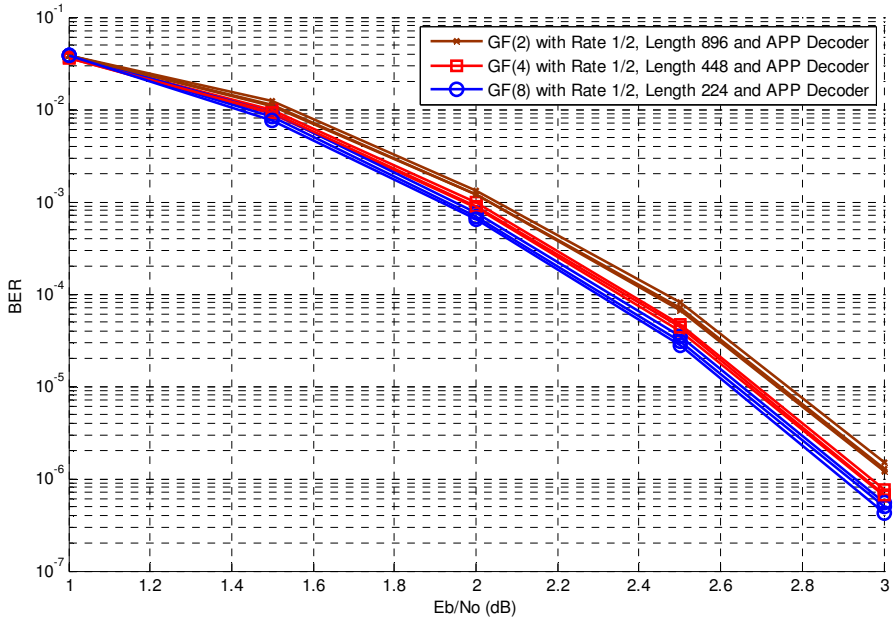


Figure 3.10 Performance comparison of many rate ½ and length 896 irregular (d_v=2.8) LDPC codes with APP Decoder over GF(2), GF(4) and GF(8) alphabets

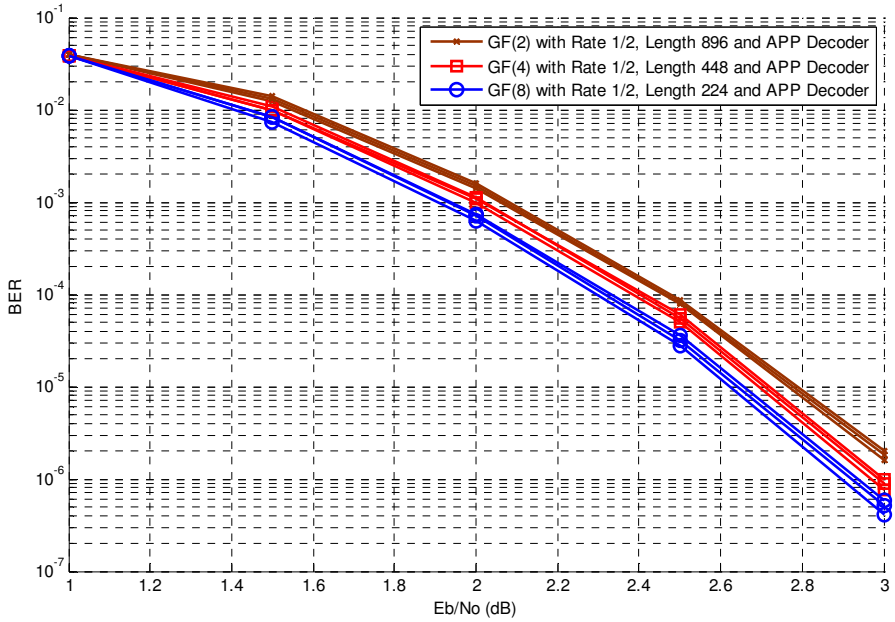


Figure 3.11 Performance comparison of many rate ½ and length 896 irregular (d_v=2.6) LDPC codes with APP Decoder over GF(2), GF(4) and GF(8) alphabets

Examining Figure 3.9, Figure 3.10 and Figure 3.11, one can say that the results are similar with those in Figure 3.6, Figure 3.7 and Figure 3.8 respectively. In Figure 3.9, the performance difference between GF(4) and GF(2) is at most ~0.12dB for BER=10^{-4} and the difference between GF(2) and GF(8) is at most ~0.06dB for BER=10^{-4}. From Figure 3.10, one can conclude that the performance difference between GF(8) and GF(4) is at most ~0.08dB for BER=10^{-4} and the performance difference between GF(4) and GF(2) is at most ~0.11dB for BER=10^{-4}. Examining Figure 3.11, one can observe that the performance difference between GF(8) and GF(4) is at most ~0.12dB for BER=10^{-4} and the difference between GF(4) and GF(2) is at most ~0.09dB for BER=10^{-4}

Another comparison can be made by looking the decoding durations for GF(2), GF(4) and GF(8) alphabets. To give an idea about the decoding complexity, in Table 3.3 we tabulate the average durations per iteration of the APP decoder by using a Matlab R2006a program on a computer with Pentium D 3.2 MHz CPU and 2 GB RAM.

Table 3.3 Average durations per iteration in seconds for rate ½ and length 896 LDPC codes over GF(2), GF(4) and GF(8)

Degree / Field	d_v=3	mean d_v=2.8	mean d_v=2.6
GF(2)	0.68	0.63	0.57
GF(4)	20.45	18.80	16.72
GF(8)	300.65	262.88	215.67

From Table 3.3, one can conclude that when we increase the alphabet size from GF(2) to GF(4) and GF(8), the decoding durations increases ~30 times and ~442 times for d_v=3 case, ~29 times and ~417 times for mean d_v=2.8 and ~29 times and ~392 times for mean d_v=2.6 respectively. When we compare the decoding durations with different variable node degree characteristic, for GF(2), GF(4) and GF(8) alphabets, while moving from d_v=3 to mean d_v=2.8 a decrease of ~7%, ~8% and ~13% is obtained and while moving from mean d_v=2.8 to mean d_v=2.6 a decrease of ~10%, ~11% and ~18% is obtained in decoding duration times respectively. Note that with this configuration one can measure a BER of 10^{-4} for GF(8) in about 92 hours for rate ½ but in about 204 hours for rate ¾ codes.

In Figure 3.12 we compare rate ¾ and regular (d_v=3) LDPC codes over GF(2), GF(4) and GF(8) and see that performance ordering changes as opposed to rate ½.

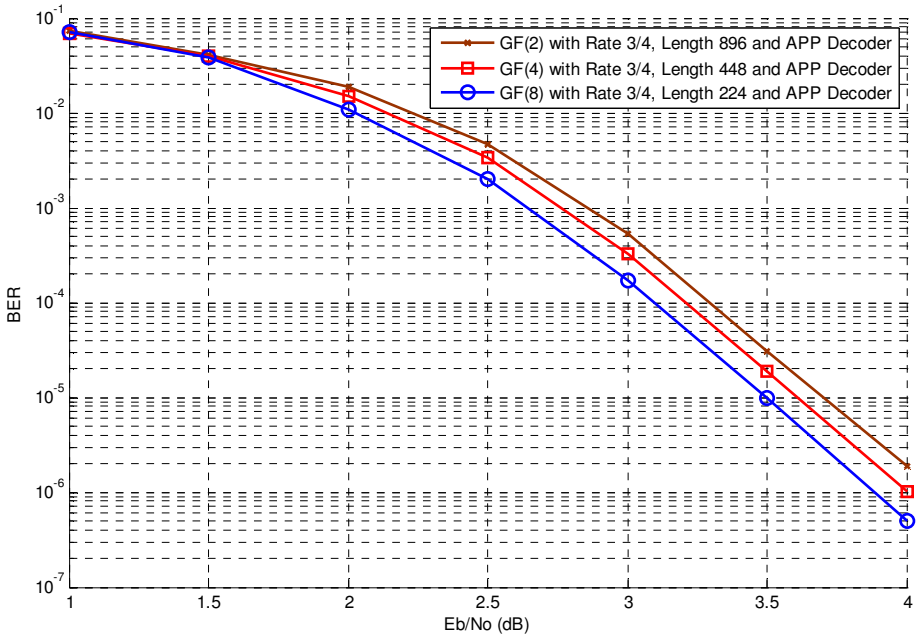


Figure 3.12 Performance comparison for rate ¾ and length 896 regular (d_v=3) LDPC codes with APP decoder over GF(2), GF(4) and GF(8) alphabets.

From Figure 3.12, one can see that an improvement of ~0.09 dB at BER=10^{-4} when moving from GF(2) to GF(4) and an improvement of ~0.11 dB at BER=10^{-4} when moving from GF(4) to GF(8).

e) Effect of irregularity:

After completing the configurations defined in Table 3.1, we will examine the effect of irregularity on the performance. We select an irregular LDPC code with variable node degree distribution polynomial $\lambda(x) = \frac{1}{3}x^2 + \frac{1}{3}x^3 + \frac{1}{3}x^4$, and compare the performance of this code with a regular LDPC code with node degrees (d_v=3, d_c=6). The result is as given in Figure 3.13.

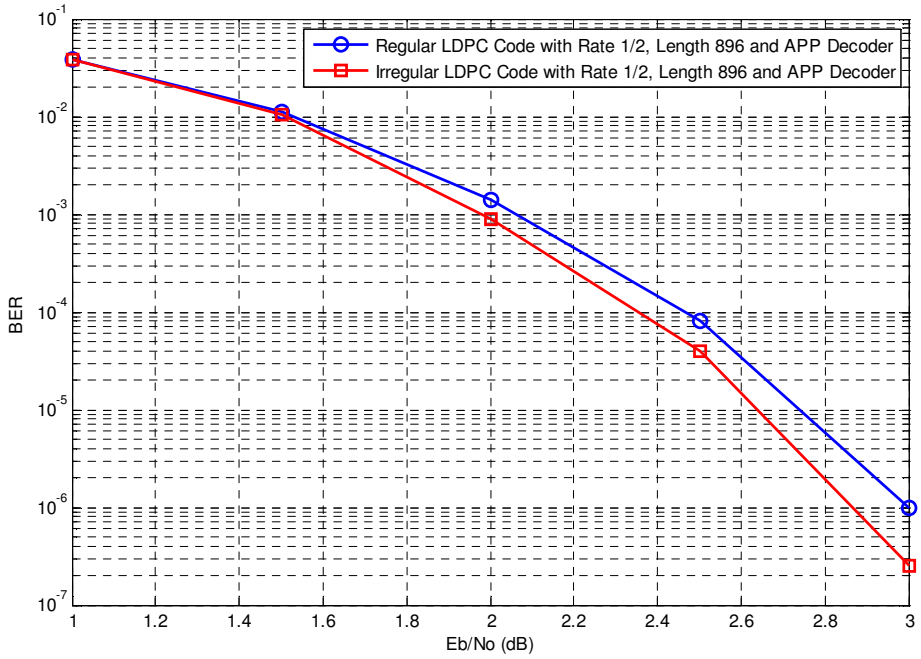


Figure 3.13 Performance comparison for rate ½ and length 896 regular and irregular LDPC codes for GF(2) with APP decoder

For both of the binary LDPC codes in Figure 3.13 the code rate is ½ and the code length is 896 bits. Performance is obtained using an APP decoder. As described in

section 3.1.4, irregular design, when carefully applied, helps to improve the performance. We observe an SNR gain of ~0.1dB at BER=10^{-4} in our simulation.

3.2.2 Effect of Decoding Algorithms

In this section, we will present decoding performances and give a brief comment on the decoding complexity for three different types of message passing algorithms.

a) APP and LL decoders:

Firstly we will compare APP decoders and LL decoders described in Section 2.2.5.1 and 2.3.3 for the LDPC codes constructed over GF(2), GF(4) and GF(8) alphabets. The results for GF(2) alphabet with 896 bits code length are given in Figure 3.14;

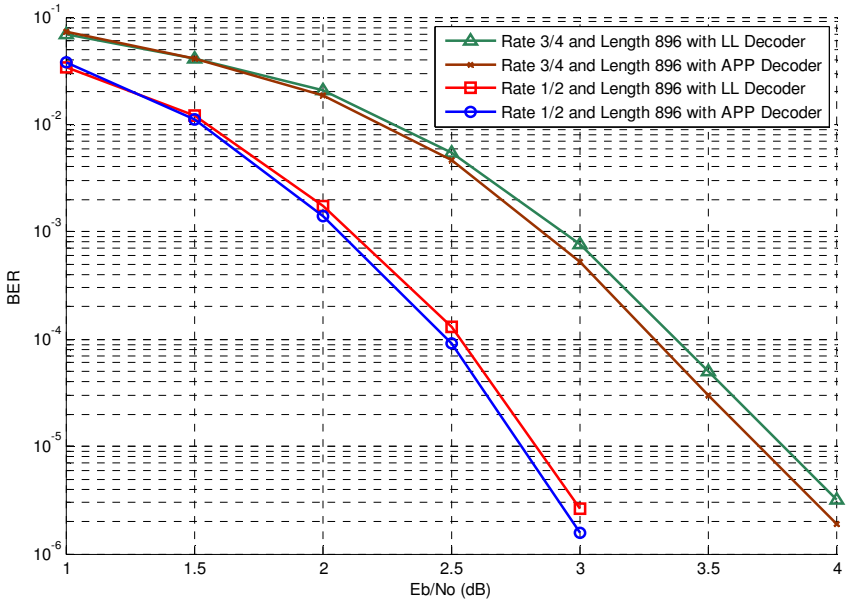


Figure 3.14 Performance comparison for rates ½ and ¾ and length 896 LDPC codes for GF(2) with APP and LL decoders

Performance of the LL decoder for LDPC codes with ½ and ¾ code rates is worse than APP decoder for both code rates. For code rate of ½, performance decrease is ~0.08 dB and for code rate of ¾ performance decrease is ~0.1 dB at BER=10^{-4}.

For similar codes over GF(4) and GF(8) alphabets, performances obtained by APP and LL decoders are sketched in Figure 3.15 and Figure 3.16.

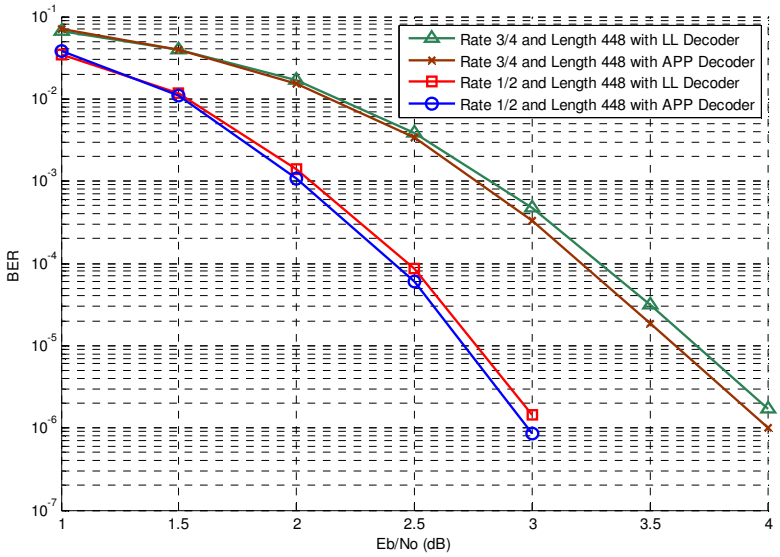


Figure 3.15 Performance comparison for rates ½ and ¾ and length 448 symbols LDPC codes for GF(4) with APP and LL decoders

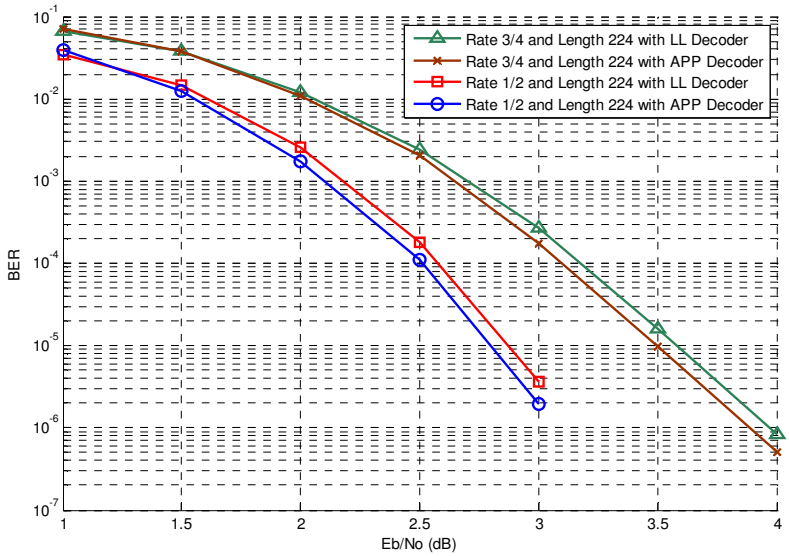


Figure 3.16 Performance comparison for rates ½ and ¾ and length 224 symbols LDPC codes for GF(8) with APP and LL decoders

One can draw very similar conclusions from both figures, which are also similar to those derived from Figure 3.14. LL decoder performs slightly worse than APP decoder, independent of the field over which the LDPC codes is designed. In the definitions of the APP and LL decoders, no assumptions have been made. So, both should have the same performance in the ideal case. The reason of the performance difference can be explained with the numerical rounding errors during the calculations in the implementation of LL decoder.

The results of this subsection are also given obtained by generating a single random LDPC code for each case given in Table 3.1. To see the performance difference of the decoding algorithms in the ensemble of codes with identical parameters, we generate many random LDPC codes with the properties given in Table 3.1 over GF(2) as in Section 3.2.1. The result is given in Figure 3.17**;**

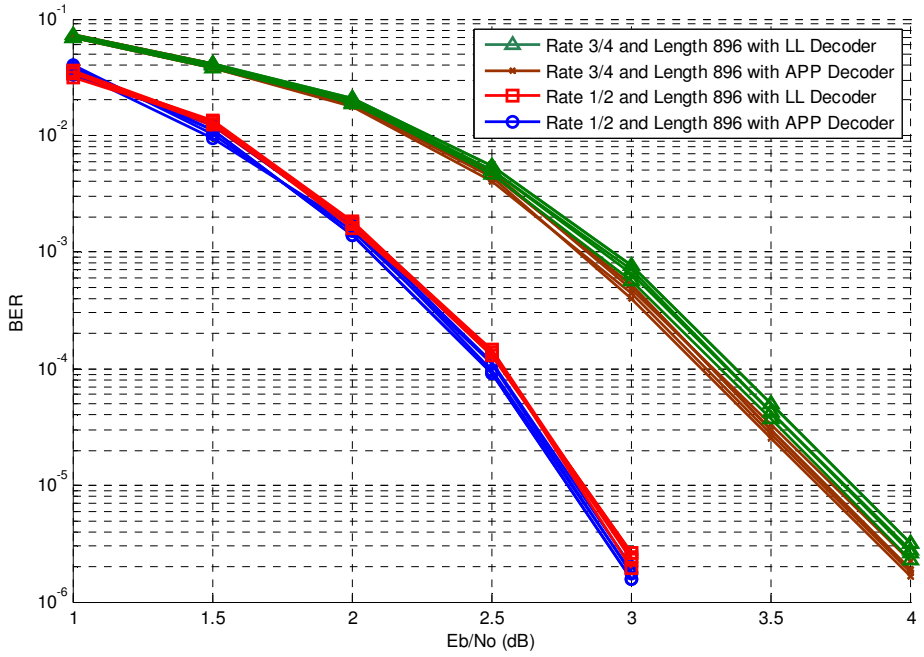


Figure 3.17 Performances of many rate ½, rate ¾, length 896 LDPC codes with APP and LL decoders

Examining Figure 3.17, one can say that the performance of APP and LL decoders with the same parameters do not differ very much. Rate-½ codes have maximum ~0.1dB and rate-¾ code have ~0.12dB difference between APP and LL decoders at BER=10^{-4}.

b) FT decoder:

After the discussion on APP and LL decoders, we now make a comparison for another decoding method, FT decoder, which was defined in Section 2.3.3. The comparison is made between the APP, FT and LL decoders applied to a rate- ½ and length-448 symbols LDPC code over GF(4).

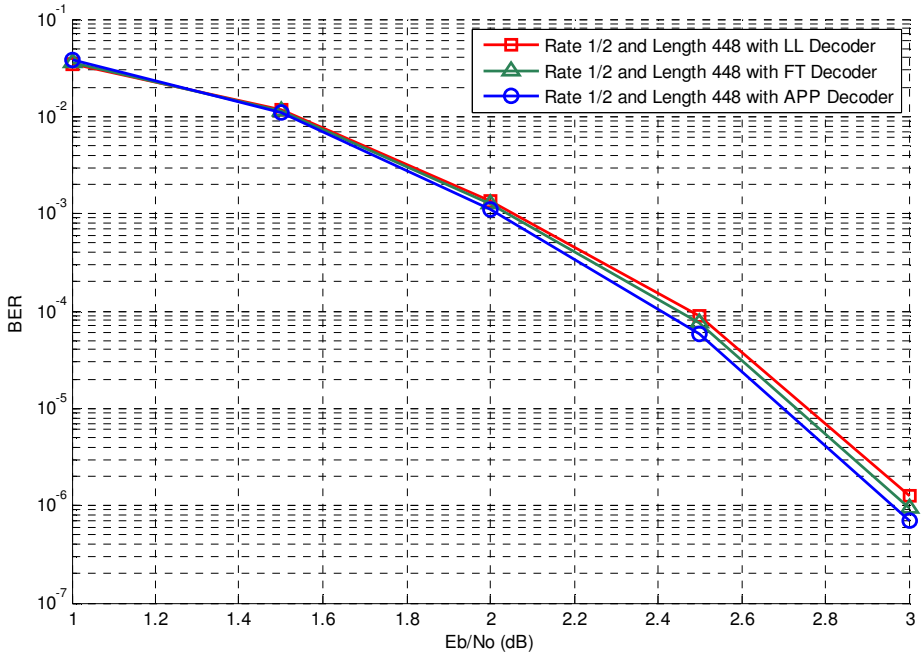


Figure 3.18 Performance comparison for rate ½ and length 448 symbols LDPC codes for GF(4) with) APP, FT and LL decoders

The performance graphs are shown in Figure 3.18, in which the best decoder is APP, followed by FT and followed by LL with SNR differences of 0.03-0.05dB

between curves. The performance difference is the numerical errors in the implementation of LL and FT decoders.

Finally, we will briefly comment on the decoding complexity. Only to give an idea, average decoding times for one iteration of APP, LL and FT decoders are measured for a rate ½ LDPC code, having a code length of 448 symbols, designed over GF(4).

A single iteration takes 20.45 sec for APP, 14.14 sec for LL and 11.56 sec for FT decoders for the mentioned LDPC code. These time values only give an idea about relative complexities of the decoders since it is dependent on the simulation computer, simulation code and simulation platform, for instance, one can say that the FT decoder needs roughly half of the time required by the APP decoder.

CHAPTER 4

CONCLUSIONS

4.1 Conclusions

In this work, performances of randomly generated binary and non-binary LDPC codes have been measured using different decoding algorithms. All codes are constructed by avoiding cycles of length four in the Tanner graph, and mainly as regular codes with variable node degree d_v=3. Simulation results are given in the form of "bit error ratio versus bit signal-to-noise ratio" curves. Observed differences in the performances of the codes with various different parameters are usually very small, when expressed as "bit SNR gain at constant BER". The main reason for such small differences is the relative shortness of the codeword lengths used in our simulations as opposed to those of practically meaningful LDPC codes.

For code properties comparison, there exists an increase of performance with decreasing code rate from ¾ to ½. This is reasonable since adding more parity bits provides more immunity to noise. Also, an improvement in performance can be achieved with an increase of code length, as randomness of the parity check matrix is increased.

As for the field size over which the LDPC codes are designed, there is an increase in the performance with increasing field size, provided that the variable node degree is chosen carefully. According to Davey's studies the reason of mean column

weight effect on performance can be explained such that as the matrix weight is increased, the number of neighbors for the check node increases and so the check node is less confident of its neighbors and the decoder performs worse. At the same time increasing the field order would produce similar effect because now the neighbor has more possible states. So, going to higher order field with high matrix weight should give worse performance. But intuitively we also see that producing more stringent conditions on the check node reduces error. As the simulation results show, for regular (d_v=3) LDPC codes with ½ rate, GF(4) is the best and GF(8) is the worst field that agrees with [Davey-Mackay-1998]. However, when we change the variable node degree slightly to construct an irregular LDPC code of mean d_v=2.8, GF(8) is the best and GF(2) is the worst as expected. Similarly when we decrease the variable node degree to mean d_v=2.6 value, again GF(8) has the best and GF(2) has the worst performances. We also observe an improvement in the performance while moving mean d_v from 2.8 to 2.6 for GF(8) and slight decreases for GF(2) and GF(4), which also agrees with Davey's results. For rate of ¾ and regular (d_v=3) LDPC code, the best performance is in GF(8) and the worst is in GF(2). So, for a given field order and rate the selection of mean column weight is crucial in the performance ordering of fields as emphasized by Davey. Another point to be taken into account is the decoding complexity with the increase in the alphabet. Our simulation results show that, in GF(4), APP decoding algorithm spends approximately 30 times of the duration required for GF(2). On the other hand, the duration of the same algorithm is 400 times longer in GF(8) as compared to GF(2). So, we can conclude that carefully constructed non-binary LDPC codes are likely to outperform their binary counterparts but decoding complexity should not be forgotten since it is an effective factor. Note also that an important parameter on construction of LDPC code is the column weight since we observe that it changes the order of GF(2), GF(4) and GF(8) performances for rate-½. This result also includes the irregularity comparison. When we move from regular to irregular LDPC codes, there exists an increase in the decoding performance according to our simulations.

As for decoder comparison, our simulation results show that there exists a decrease of performance while going from APP decoder to LL and FT decoders. The reason of this performance decrease can be the accumulation of numerical errors in the implementations of LL and FT decoders. On the other hand, when we select LL and FT decoders, decoding complexity is decreased with respect to APP decoder, which is an advantage to be chosen when the speed of decoding is important. FT decoder has the smallest and APP has the largest duration according to our simulation results, the largest to smallest duration ratio being less than 2.

To sum up, one can conclude that choosing non-binary alphabets with careful code design in terms of code rate, code length, irregularity, girth and variable node degree provides improvement in the performance. From decoder selection side of view, if the decoding performance is important, APP decoder can be the best candidate and if the decoding time is the main criterion, then FT Decoder can be the best choice according to our simulation results. Since none of the decoders used in this work are optimized in terms of programming, possible directions of future work may include an optimized hardware implementation to see the real performance of different types of LDPC codes and decoders.

REFERENCES

- **[Berrou-Glavieux-Thitimajshima-1993]** C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo-codes," in Proc. ICC '93, Geneva, Switzerland, May 1993, pp. 1064–1070.

- **[Bose-Chaudhuri-1960]** R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error-correcting binary group codes," Information and Control, vol. 3, no. 1, pp. 68–69, 1960.

- **[Brack-Alles-Lehnigk-Emden 2007]** T. Brack, M. Alles, T. Lehnigk-Emden, "Low Complexity LDPC Code Decoders for Next Generation Standards", Design, Automation and Test in Europe Conference and Exhibition, DATE '07.

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

- **[Davey-Mackay-1998]** M. Davey and D. Mackay, .Low-density parity check codes over GF(q). IEEE Communications Letters, vol. 2, 6, pp. 165 .167, June 1998.

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

- **[Hamming-1950]** R. W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147–160, Apr. 1950.

- **[Lin-Costello-2004]** Shu Lin and Jr. Daniel J. Costello. Error Control Coding. Pearson, Prentice Hall, 2004.

- **[Luby-Mitzenmacher-Shokrollahi-Spielman-1998]** M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low density parity check codes using irregular graphs and belief propagation," in Proc. ISIT '98, Cambridge, MA, USA, Aug. 1998, p. 117.

- **[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-1999]** D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. IEEE Trans. Inform. Theory, 45:399-431, 1999.

- **[Mackay-Neal-1996]** D. J. C. Mackay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," IEE Electronics Letters, vol. 32, no. 18, pp. 1645–1646, Aug. 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

- **[Mackay-2005]** D. J. C. MacKay. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, 9-21, 206-228, 557-573, March 2005.

- **[Nozawa-2005]** T. Nozawa. Ldpc adopted for use in comms, broadcasting, hdds. Nikkei Electronics Asia, 2005

- **[Reed-Solomon-1960]** I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," Journal of the Society for Industrial and Applied Mathematics (SIAM), vol. 8, no. 2, pp. 300–304, June 1960.

- **[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.Richardson and R. Urbanke, .The capacity of low-density parity-check codes under message-passing decoding. IEEE Transactions on Information Theory, vol. 47, 2, pp. 599-618, Feb. 2001.

- **[Robertson-Villebrun-Hoeher-1995]** P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," in Proc. ICC '95, vol. 2, Seattle, WA, USA, June 1995, pp. 1009–1013.

- **[Shannnon-1948]** C. E. Shannon. A mathematical theory of communication. Bell Systems Technical Journal, 27:379-423, 623-656, 1948**.**

- **[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.

- **[Wiberg-1996]** N. Wiberg, Codes and Decoding on General Graphs. PhD thesis, Dept. of Electrical Engineering, Linköping, Sweden, 1996. Linköping studies in Science and Technology. Dissertation No. 440.