uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yogesh Kalyani

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S. (Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Towards the Use of Mobile Agents for Privacy Negotiation

TITRE DE LA THÈSE / TITLE OF THESIS

Carlisle Adams

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Liam Peyton

Michael Weiss

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

PERFORMANCE OF LOW-DENSITY GENERATOR MATRIX CODES AT SHORT BLOCK LENGTHS

LEI JIN

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of Master of Applied Science, Electrical Engineering

September 2006

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-25791-3
Our file Notre référence
ISBN: 978-0-494-25791-3

To my parents

Performance of Low-Density Generator Matrix Codes at Short Block Lengths

Master of Applied Science, Electrical Engineering Thesis
School of Information Technology and Engineering
University of Ottawa

by Lei Jin
September 2006

Abstract

It is often necessary to apply codes with short block lengths in many delay-sensitive applications. In this thesis, primarily driven by their complexity advantages, we investigate and explore the performance of low-density generator matrix (LDGM) codes [1] at short block lengths. We show that with proper constructions, LDGM codes may perform no worse than the state-of-the-art, low-density-parity-check codes (LDPC) [2] but with lower complexity. Our construction of LDGM codes uses the Progressive Edge-Growth (PEG) algorithm [3], originally proposed for LDPC codes.

Acknowledgments

I would first like to thank my supervisor, Professor Yongyi Mao, for his guidance and mentorship throughout my two years of study as a graduate student at the University of Ottawa. He has shown us a very positive attitude toward research, work and life. I will never forget what he told me: Never avoid learning something new. Learning to express myself well is also an important part of my education under his supervision.

I would also express gratitude to my colleagues and fellows, Master's student Zhong Cheng and post-doctoral fellow Xueying Xie, for working closely with me throughout my thesis. I am grateful to PhD candidate, Jeff Castrua, for his help on Unix facilities on IBM clusters. I am thankful to Ketai Hu as well for his latex resources and other help with my thesis. I wish to thank Dafu Lou and Ronghui Tu in our research group for their good suggestions on my research. Finally, I would like to thank my parents and my sister for their continuous support and confidence on me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In delay-sensitive applications, for example, real-time communication of voice data, it is often necessary to use error correction codes at short block lengths. Although many algebraic codes (for examples, BCH codes [4, 5] or Reed-Solomn codes [6]) may be used for such purposes, modern revolution of coding theory in the past decade has pointed communication engineers to exciting new families of codes with superb performance, namely turbo codes, low-density parity-check (LDPC) codes, and their close relatives (*e.g.* [1, 7, 8]).

Turbo codes [9], invented in 1993, and LDPC codes [2], invented in 1962 and rediscovered in 1995, have demonstrated capacity-achieving performance [10] when they are used at long block lengths. At short block lengths ($500 \sim 5000$ bits), although these codes can no longer achieve the channel capacity, they, particularly LDPC codes, still offer a coding gain superior to the classical algebraic codes [6, 7].

In addition to performance, the designer of a communication system must also consider the complexity of the encoder and the decoder for any candidate coding scheme. Essentially concatenated convolutional codes implementable with shift registers, turbo codes have low encoding complexity. But the decoding complexity of turbo codes is rather high, particularly when compared with LDPC codes. Characterized by sparse parity-check matrices, on the other hand, LDPC codes have low decoding complexity.

However, their encoding complexity is quite high as LDPC codes typically do not have a sparse generator matrix representation.

Recently, a new family of codes, known as low-density generator-matrix (LDGM) codes [1, 11], have been invented. It is shown that at asymptotically long block lengths, they can perform as well as LDPC codes but have low complexity both at the encoder and at the decoder. This motivates us to consider using LDGM codes at short block lengths for delay-sensitive applications, and to investigate whether LDGM codes can perform comparably to LDPC codes at these block lengths. To the best of our knowledge, no results have been available on this subject.

1.2 Overview of research methods and thesis contribution

This work is largely inspired by the work of parallel concatenation LDGM codes [11] where the authors show that randomly-constructed asymptotically-long LDGM codes, upon parallel concatenation, have a capacity-achieving performance while maintaining low complexity both at the encoder and at the decoder.

A brute-force extension of such codes to short block length is to simply apply the random constructions suggested in [11]. However, this scheme is very unlikely to work due to the following reasons.

Like LDPC codes, LDGM codes are represented and decoded using what is known as a factor graph [12]. Random construction of LDGM codes at short block lengths can easily generate many short cycles in the corresponding factor graph representations. Such graph structures are unfavorable for the factor graph based decoding algorithm (*i.e.* the sum-product algorithm), and correspondingly the codes have little chance to perform well.

Realizing this, we see a necessity of optimizing graph structure in the construction of short LDGM codes. We note that such necessity diminishes as the block length increases since it is possible to show that at long block lengths, a randomly constructed code gives rise to a factor graph that is nearly cycle-free locally.

For graph optimization purpose, we adapt an algorithm, Progressive Edge-growth (PEG) [3], originally developed for constructing LDPC codes, in the design of short LDGM codes.

As such, we construct several families of short LDGM codes and compare their performance with LDPC codes at the same rate and block length. The comparison is carried out via Monte-Carlo simulations for Binary symmetric channel(BSC) and AWGN channels. Our major conclusion is that LDGM codes constructed in this manner may have comparable or ever better performance.

The main contributions of the thesis are summarized as follows:

- We show that short LDGM codes can perform as well as LDPC codes at the same lengths and rate, and their low encoding and decoding complexity make them better candidates for delay-sensitive applications in practice.

- We adapt the PEG algorithm to the construction of short LDGM codes and our results suggest that this approach allows us to find good LDGM codes.

1.3 Thesis organization

This thesis contributes to the short block length of LDGM codes at which can achieve the performance close to Shannon limit. I investigate that LDGM codes can perform no worse than LDPC codes at short block lengths.

The rest of the thesis is organized as follows.

Chapter 2 briefly describes the graphical model used to present a linear block code C corresponding to parity-check matrix H and generator matrix G. An efficient algorithm, known as the sum-product algorithm, operating on the graphical model for decoding purpose, is also given. Also included in Chapter 2 is a review of LDPC codes and LDGM codes with serial and parallel concatenation schemes.

Chapter 3 proposes a PEG-based construction of LDGM codes for a given code rate at short block lengths and analyze the characteristic of error floors for short irregular LDGM codes.

Chapter 4 contains simulation results for constructed irregular LDGM codes over AWGN channel and Binary Symmetric channle (BSC), respectively.

The thesis is briefly concluded in Chapter 5.

Chapter 2

Background

2.1 Factor graph and the sum-product algorithm

2.1.1 Factor graph

A *factor graph* represents the factorization of the global function into local functions. Synthesized from Tanner graph [13], a factor graph is a bipartite graph that consists of two types of nodes, *variable nodes* in one set and *function nodes* in another set. Each variable node represents a variable while each function node represents a *local function*. Factor graph then expresses which variables are arguments of which local functions [12] if and only if a function node is connected to a variable node. The global function, the product of all the local functions, is represented by the factor graph. Figure 2.1 shows that the factor graph represents the global function of (2.1).

$$f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_4, x_5, x_6) \tag{2.1}$$

2.1.2 The sum-product algorithm

As shown above, a global function, which is a product of many local functions, can be represented by a factor graph. The sum-product algorithm is an efficient means of using

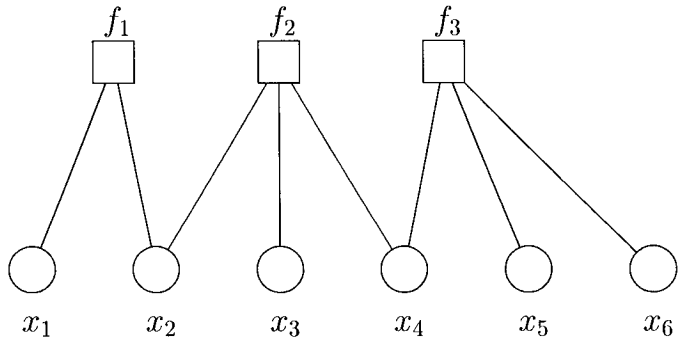

Figure 2.1: A factor graph representing product $f_1(x_1, x_2)f_2(x_2, x_3, x_4)f_3(x_4, x_5, x_6)$, where squares represent functions nodes, and circles represent variable nodes.

the distributive law between multiplication and addition [14] to simultaneously compute all "marginals" of the global function. The following description details the sum-product algorithm.

The factorization of function can be represented by factor graph $\mathcal{G}$ as follows:

$$y(x_1, \ldots, x_n) := \prod_{j=1}^{m} f_j(X_j), \tag{2.2}$$

where $X_j \subseteq \{x_1, x_2, \ldots, x_n\}$ is the set of all variables contained in the argument of local function f_j. Given each local function of the factor graph, the procedure of the sum-product algorithm is in fact the procedure of computing the following *marginal* functions of y:

$$y_i(x_i) := \sum_{\{x_1, x_2, \ldots, x_n\} \setminus \{x_i\}} y(x_1, x_2, \ldots, x_n) \tag{2.3}$$

for every $i \in \{1, 2, \ldots, n\}$.

Example 2. 1 Given the factor graph shown in Figure 2.1, the objective of the sum-product algorithm is to compute

$$
\begin{aligned}
y_1(x_1) &:= \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5,x_6),\\
y_2(x_2) &:= \sum_{x_1}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5,x_6),\\
y_3(x_3) &:= \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5,x_6),\\
y_4(x_4) &:= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_5} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5,x_6)\\
y_5(x_5) &:= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5,x_6) \text{ and}\\
y_6(x_6) &:= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5,x_6).
\end{aligned}
$$

The sum-product algorithm [12], which operates on the factor graph representation of a global function, iteratively passes messages between every pair of connected function node and variable node. There are two types of messages, namely the message sent from a variable node v to a function node f — denoted by $\mu_{v\to f}$, and the message sent from a function node f to a variable node v — denoted by $\mu_{f\to v}$. Notice that both message $\mu_{v\to f}$ and message $\mu_{f\to v}$ are functions of variable v.

We denote the set of neighbors of a given node u in the factor graph as $\mathcal{N}(u)$. By this notation, a message passed from a given node u to a given node w, where u and w are connected, will be computed only using the received messages from nodes $\mathcal{N}(u)\setminus\{w\}$. More specifically, the following examples illustrate the message passing rules involved in the sum-product algorithm: function node update, variable node update and summary messages.

Function node update

$$
\mu_{f\to v}(v) := \sum_{\mathcal{N}(f)\setminus\{v\}} f(\mathcal{N}(f)) \prod_{v'\in\mathcal{N}(f)\setminus\{v\}} \mu_{v'\to f}(v') \tag{2.4}
$$

Example 2. 2 Figure 2.2 shows the portion of a factor graph, that is, the message passed from function f to variable v is computed as

$$\mu_{f\to v}(v) = \sum_{u,u'u''} f(u,u',u'',v)\mu_{u\to f}(u)\mu_{u'\to f}(u')\mu_{u''\to f}(u'').$$

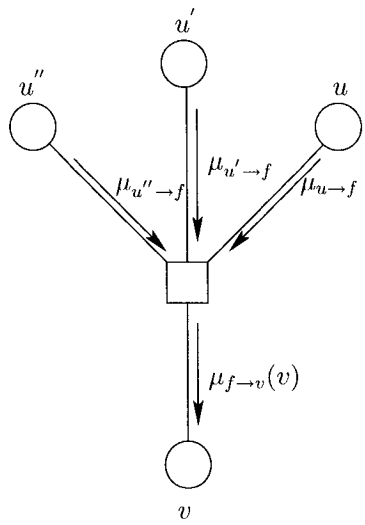

Figure 2.2: Function node f passes a message according to (2.4).

Variable node update

$$\mu_{v\to f}(v) := \prod_{f'\in\mathcal{N}(v)\setminus\{f\}} \mu_{f'\to v}(v) \tag{2.5}$$

Example 2. 3 Figure 2.3 illustrates that the message passed from function v to variable f is computed as

$$\mu_{v\to f}(v) = \mu_{h\to v}(v)\mu_{h'\to v}(v)\mu_{h''\to v}(v)$$

In addition, it is also required to compute the *summary messages* in the procedure of the sum-product algorithm. The summary message μ_v of a variable node v, again a

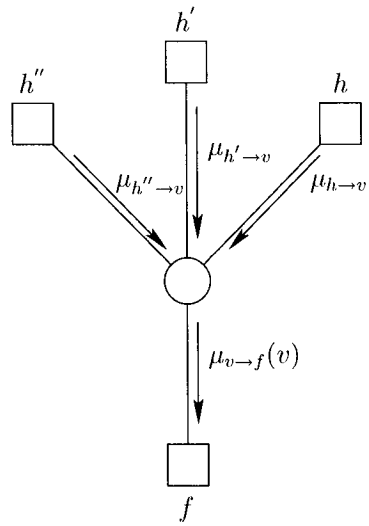

Figure 2.3: Variable node v passes a message according to (2.5).

function of variable v, is computed using the current incoming messages sent to node v, defined as follows.

Summary message

$$\mu_v(v) := \prod_{f\in\mathcal{N}(f)} \mu_{f\to v}(v). \tag{2.6}$$

Example 2. 4 Figure 2.4 demonstrates that the summary message of variable v is computed as

$$\mu_v(v) = \mu_{h\to v}(v)\mu_{h'\to v}(v)\mu_{h''\to v}(v)\mu_{f\to v}(v).$$

By definition of these essential computation rules, those local message-passing rules can be arranged by the sum-product algorithm based on the structure of the graph in different ways. Basically, the following three steps are involved in any variant of the sum-product algorithm:

- initialization step, where a set of messages are initialized according to certain choice,
- propagation step, where messages are passed on the graph according to some order, and

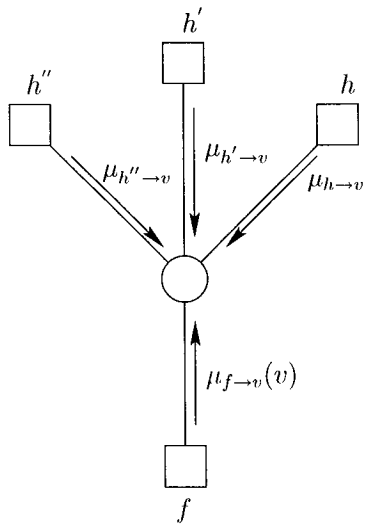

Figure 2.4: The computation of summary message μ_v for variable node v according to (2.6).

- termination step, where the passing of messages is terminated following some criterion.

If the factor graph is cycle-free, namely a tree, then the three steps are typically chosen as follows:

Sum-product algorithm on tree graphs

Initialization: Leaf nodes pass messages first. If a leaf node is variable node v, set $\mu_{v\to f}(v) := 1$, where f is the only neighbor of v; if a leaf node is a function node f, then set $\mu_{f\to v}(v) := f(v)$, where v is the only neighbor of f.

Propagation: A node only passes messages to a neighbor if the messages from all other neighbors have arrived.

Termination: When each variable node receives messages from all neighbors, it computes its summary messages and the algorithm terminates.

The following result has been proved [12, 15]: Let $\mathcal{G}$ be the factor graph representing

the factorization of $y(x_1, x_2, x_n)$ in (2.2), and suppose that $\mathcal{G}$ is a tree. When the sum-product algorithm described above is then applied on $\mathcal{G}$, the resulting summary message $\mu_{x_i}(x_i)$ for each $x_i, i \in \{1, 2, \ldots, n\}$ is $y_i(x_i)$ in (2.3).

In principle, the sum-product algorithm is an efficient means of using the distributive law between multiplication and addition [14]. By postponing multiplication to as late as possible, while the intermediate terms, which can be computed for calculating one y_i, are efficiently saved and shared by the computation of other y_j's.

Example 2. 5 The sum-product algorithm operating on the factor graph in Figure 2.1 for the computing y_1 and y_2 is shown as follows:

$$\begin{aligned} y_1(x_1) &= \sum_{x_2} f_1(x_1, x_2) \sum_{x_3, x_4} f_2(x_2, x_3, x_4) \sum_{x_5, x_6} f_3(x_4, x_5, x_6) \\ y_2(x_2) &= \sum_{x_1} f_1(x_1, x_2) \sum_{x_3, x_4} f_2(x_2, x_3, x_4) \sum_{x_5, x_6} f_3(x_4, x_5, x_6) \end{aligned}$$

There is subtle point in the procedure of sum-product algorithm that needs further comment. We only need to compute the terms $\sum_{x_3,x_4} f_2(x_2, x_3, x_4)$ and $\sum_{x_5,x_6} f_3(x_4, x_5, x_6)$ once while calculating both y_1 and y_2.

We will demonstrate one standard implementation of the sum-product algorithm for decoding codes represented by factor graphs with cycles in the following section of this chapter.

2.2 Codes on graphs and iterative decoding

2.2.1 Codes on graph

Codes on graphs have recently received a lot of attention because of their excellent performance and low-complexity iterative decoding. LDPC codes, rediscovered in [16,17], is an example of codes on graphs. Factor graph is a means of representing codes on graphs in this thesis.

In this context, a widely used function in factor graphs is the *indicator function* $\delta[P]$, where for any boolean proposition P, $\delta[P]$ is evaluated to 1 if P is `true` and evaluated

to 0 otherwise.

Given a linear block code C of length n and dimension k, there are two kinds of fundamental representations of C, namely, the generator matrix representations and the parity-check matrix representations. In fact, both representations have a corresponding factor graph. Either the generator matrix or the parity-check matrix defines the constraints of the code, and then these constraints can be translated into a set of indicator functions. Consequently, the product of these indicator functions defines the constraints of code C. The following example will demonstrate both representations in details.

Example 2. 6 Let the following matrices H and G be respectively a parity-check matrix and generator matrix of a binary linear block code C.

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \tag{2.7}$$

and

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.8}$$

Based on H, each codeword $(c_1, c_2, \ldots, c_7)$ satisfies the following constraints.

$$\begin{aligned} c_1 \oplus c_2 \oplus c_3 &= 0, \\ c_1 \oplus c_4 \oplus c_5 &= 0, \\ c_1 \oplus c_6 \oplus c_7 &= 0, \end{aligned}$$

where $\oplus$ denotes addition modulo 2.

We may then translate these constraints to a set of indicator functions as follows.

$$\begin{aligned} f_1(c_1, c_2, c_3) &:= \delta[c_1 \oplus c_2 \oplus c_3 = 0], \\ f_2(c_1, c_4, c_5) &:= \delta[c_1 \oplus c_4 \oplus c_5 = 0], \\ f_3(c_1, c_6, c_7) &:= \delta[c_1 \oplus c_6 \oplus c_7 = 0]. \end{aligned}$$

The product

$$f_1(c_1, c_2, c_3) f_2(c_1, c_4, c_5) f_3(c_1, c_6, c_7)$$

of these indicator functions then defines the code constraint of C. Thus this product can be represented using a factor graph as shown in Figure 2.5.

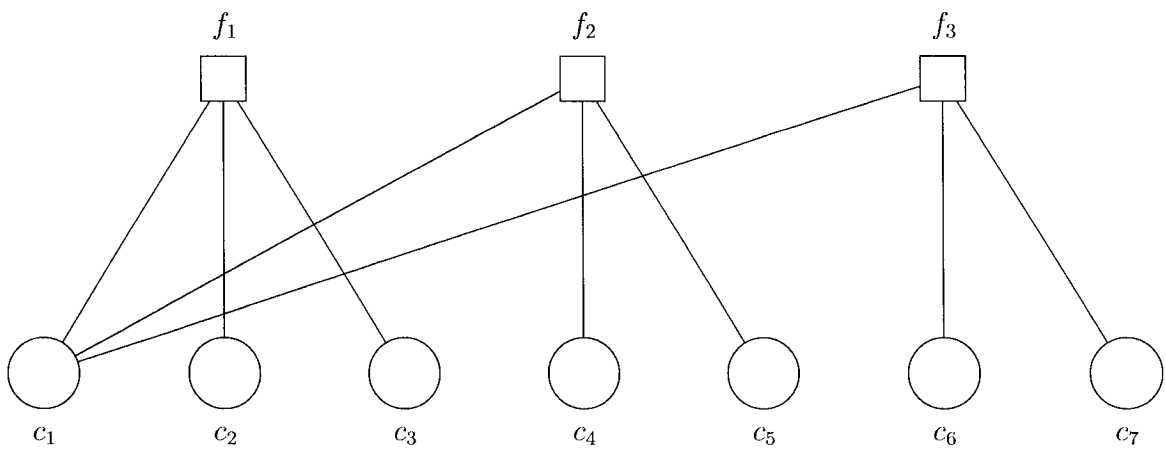

Figure 2.5: Factor graph corresponding to parity-check matrix H in (2.7).

On the other hand, based on G, each codeword $(c_1, c_2, \ldots, c_7)$ satisfies the following constraints.

$$\begin{aligned} m_1 &= c_1, \\ m_1 \oplus m_3 &= c_2, \\ m_3 &= c_3, \\ m_1 \oplus m_2 &= c_4, \\ m_2 &= c_5, \\ m_4 &= c_6, \\ m_1 \oplus m_4 &= c_7, \end{aligned}$$

for any binary vector (m_1, m_2, m_3, m_4). — In fact, if the encoding is carried out using generator matrix G, then (m_1, m_2, m_3, m_4) is precisely the information vector to be transmitted.

We may then translate these constraints to a set of indicator functions as follows.

$$\begin{aligned} h_1(m_1, c_1) &:= \delta[m_1 \oplus c_1 = 0], \\ h_2(m_1, m_3, c_2) &:= \delta[m_1 \oplus m_3 \oplus c_2 = 0], \\ h_3(m_3, c_3) &:= \delta[m_3 \oplus c_3 = 0], \\ h_4(m_1, m_2, c_4) &:= \delta[m_1 \oplus m_2 \oplus c_4 = 0] \\ h_5(m_2, c_5) &:= \delta[m_2 \oplus c_5 = 0], \\ h_6(m_4, c_6) &:= \delta[m_4 \oplus c_6 = 0], \\ h_7(m_1, m_4, c_7) &:= \delta[m_1 \oplus m_4 \oplus c_7 = 0] \end{aligned}$$

The product

$$h_1(m_1, c_1)h_2(m_1, m_3, c_2)h_3(m_3, c_3)h_4(m_1, m_2, c_4)h_5(m_2, c_5)h_6(m_4, c_6)h_7(m_1, m_4, c_7)$$

of these indicator functions also defines the code constraint of C. Then this product can be represented using a factor graph as shown in Figure 2.6.

Codes on graphs, such as LDPC codes which are constructed randomly, are specified

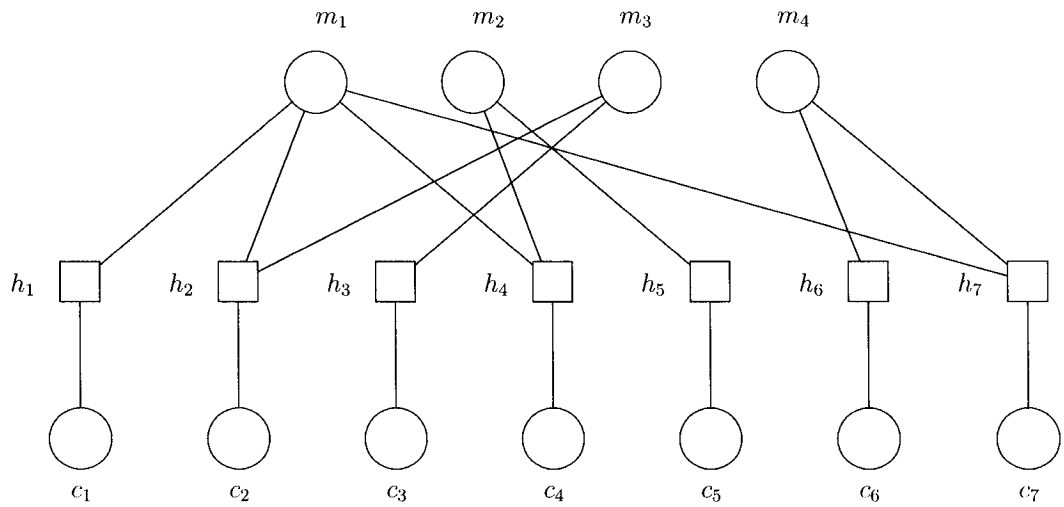

Figure 2.6: Factor graph corresponding to generator matrix G in (2.8).

by the notion of *degree distribution* $(\{\lambda_i\}, \{\rho_j\})$ of factor graphs representing parity-check matrices. In the context of factor graph, λ_i is denoted as the fraction of edges connected to variable nodes of degree i, and ρ_j as the fraction of edges connected to function nodes of degree j. Obviously, $\sum_i \lambda_i = 1$ and $\sum_j \rho_j = 1$. In general, an pair of degree distribution sequences, $\lambda_1, \lambda_2, \ldots$ and $\rho_1, \rho_2, \ldots$, can characterize an ensemble of factor graphs.

2.2.2 Iterative decoding on graphs

It is well known that the sum-product algorithm can be used for decoding of codes on graph. The procedure of iterative decoding codes on graph is in fact the procedure of finding many marginals of a product function simultaneously.

We will use the toy code C in this subsection to illustrate this principle.

Example 2. 7 Assume that BPSK modulated codewords are transmitted through a memoryless channel. Each codeword $(c_1, c_2, \ldots, c_7)$ is the information message which is intended for transmission, and a real vector $(x_1, x_2, \ldots, x_7)$ is in fact transmitted. For each $i \in \{1, 2, \ldots, 7\}$, $x_i = 1$ indicates that $c_i = 0$ and $x_i = -1$ indicates that $c_i = 1$. Each transmitted x_i is associated with conditional probability distribution $p(y_i|x_i)$, where y_i is the received symbol for the transmitted symbol x_i. It is attainable to decode and

to obtain an estimate $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ of the transmitted codeword $(c_1, c_2, \ldots, c_7)$ in an iterative manner, and then declare the transmitted (four) information bits. Notice that BPSK modulation, a bijective function, is denoted by ϕ. By introducing channel $p(y_i|x_i)$, we may formulate an equivalent channel characterized by conditional distribution $p(y_i|c_i)$ given by

$$p(y_i|c_i) = \sum_{x_i \in \{1,-1\}} p(x_i|c_i)p(y_i|x_i),$$

where

$$p(x_i|c_i) = \delta[\phi(c_i) = x_i].$$

Figure 2.7 demonstrates this communication diagram.

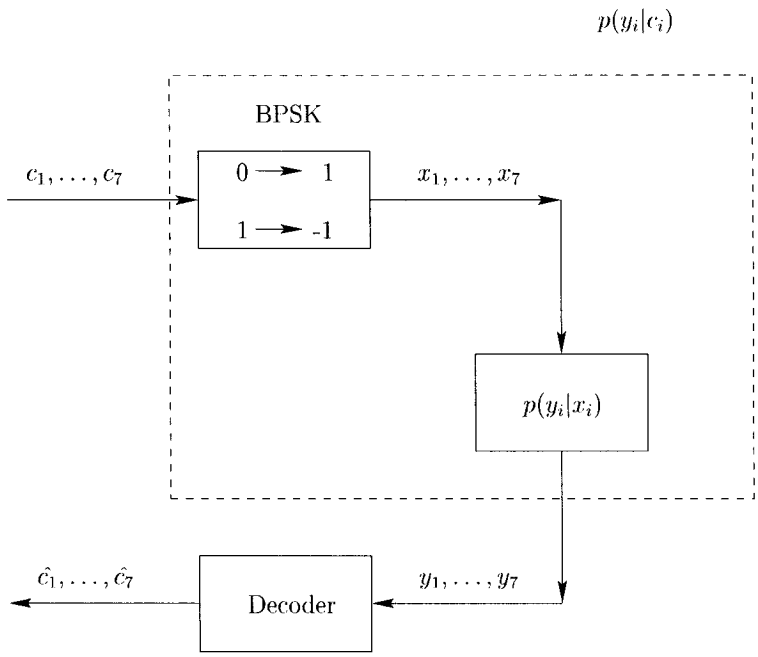

Figure 2.7: Example of communication diagram

The estimation of the vector $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ is in reality the inverse of encoding procedure in that the intended information bits can be obtained. The procedure of the estimation can be discussed as follows.

Under the long-established maximum *a posteriori* probability (MAP) criterion [18], the estimate $(\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_7)$ may be obtained as

$$\begin{aligned}
\hat{c}_1 &:= \arg\max_{c_1} p(c_1|y_1,\ldots,y_7) \\
\hat{c}_2 &:= \arg\max_{c_2} p(c_2|y_1,\ldots,y_7) \\
&\ldots \\
\hat{c}_7 &:= \arg\max_{c_7} p(c_7|y_1,\ldots,y_7)
\end{aligned}$$

Let us take the formulation of MAP estimate $\hat{c}_1$ and derive it further.

$$\begin{aligned}
\hat{c}_1 &= \arg\max_{c_1} p(c_1|y_1,\ldots,y_7) \\
&= \arg\max_{c_1} \frac{p(c_1,y_1,\ldots,y_7)}{p(y_1,\ldots,y_7)} \\
&= \arg\max_{c_1} P(c_1,y_1,\ldots,y_7) \\
&= \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(c_1,\ldots,c_7,y_1,\ldots,y_7) \\
&= \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(y_1,\ldots,y_7|c_1,\ldots,c_7)p(c_1,\ldots,c_7) \\
&\overset{\text{(memoryless channel)}}{=} \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(y_1|c_1)\ldots p(y_7|c_7)p(c_1,\ldots,c_7) \\
&= \arg\max_{c_1} \sum_{c_2,\ldots,c_7} p(c_1,c_2,\ldots c_7)\prod_{i=1}^{7} p(y_i|c_i)
\end{aligned}$$

The term $\sum_{c_2,\ldots,c_7} p(c_1,c_2,\ldots c_7)\prod_{i=1}^{7} p(y_i|c_i)$ shown above is indicated as a marginal of a function represented by a factor graph.

In other words, under the usual assumption that each codeword is equally likely to be transmitted, $p(c_1,c_2,\ldots,c_7)$ is precisely the global function $f_1(c_1,c_2,c_3)f_2(c_1,c_4,c_5)f_3(c_1,c_6,c_7)$ of the codes on graphs in terms of the parity-check matrix H in (2.7) shown in Figure 2.8. Moreover, computing a marginal of the global function represented by the factor graph

is equivalent to computing

$$\sum_{c_2,\ldots,c_7} p(c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)$$

.

Similarly, obtaining the MAP estimate $\hat{c}_i$ for any other i can be reduced to computing a corresponding marginal of the same global function. Consequently, these marginals can be computed simultaneously using the sum-product algorithm.

In addition, the factor graph of the code can be constructed based on the generator matrix G in (2.8), then $p(c_1, c_2, \ldots, c_7)$ is equivalent to function

$$\sum_{m_1,m_2,m_3,m_4} h_1(m_1, c_1)h_2(m_1, m_3, c_2)h_3(m_3, c_3)h_4(m_1, m_2, c_4)h_5(m_2, c_5)h_6(m_4, c_6)h_7(m_1, m_4, c_7)$$

up to scale, again under the assumption that each codeword is equally likely. Denote product

$$h_1(m_1, c_1)h_2(m_1, m_3, c_2)h_3(m_3, c_3)h_4(m_1, m_2, c_4)h_5(m_2, c_5)h_6(m_4, c_6)h_7(m_1, m_4, c_7)$$

by $h(m_1, \ldots, m_4, c_1, \ldots, c_7)$. Then

$$\sum_{c_2,\ldots,c_7} p(c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i) = \sum_{c_2,\ldots,c_7,m_1,\ldots,m_4} h(m_1, \ldots, m_4, c_1, \ldots, c_7) \prod_{i=1}^{7} p(y_i|c_i).$$

In the above equation, the product

$$h(m_1, \ldots, m_4, c_1, \ldots, c_7) \prod_{i=1}^{7} p(y_i|c_i)$$

is the global function represented by the factor graph in Figure 2.9. Thus, computing the MAP estimate for c_1 and similarly for any other c_i reduces to determining a corresponding marginal of this global function, and then the sum-product algorithm applies.

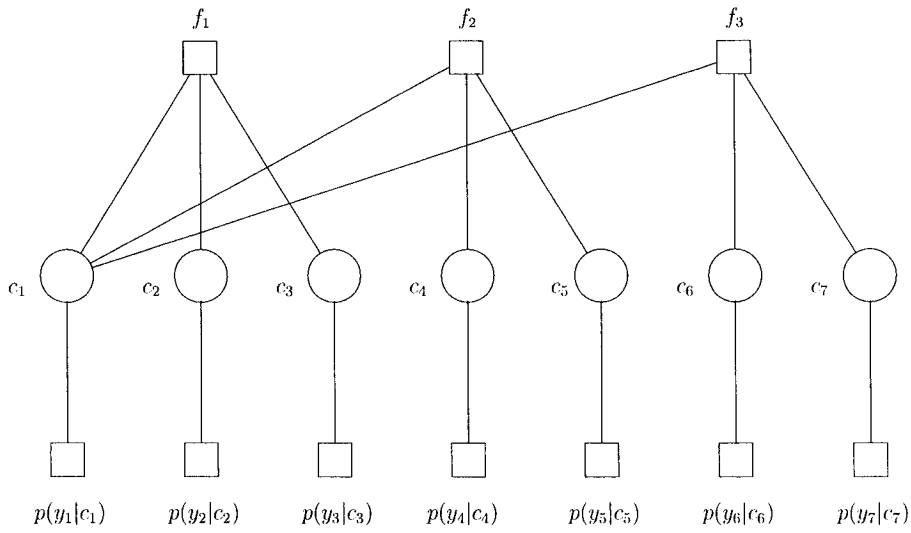

Figure 2.8: A factor graph representing $p(c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)$ constructed from parity-check matrix H in (2.7).

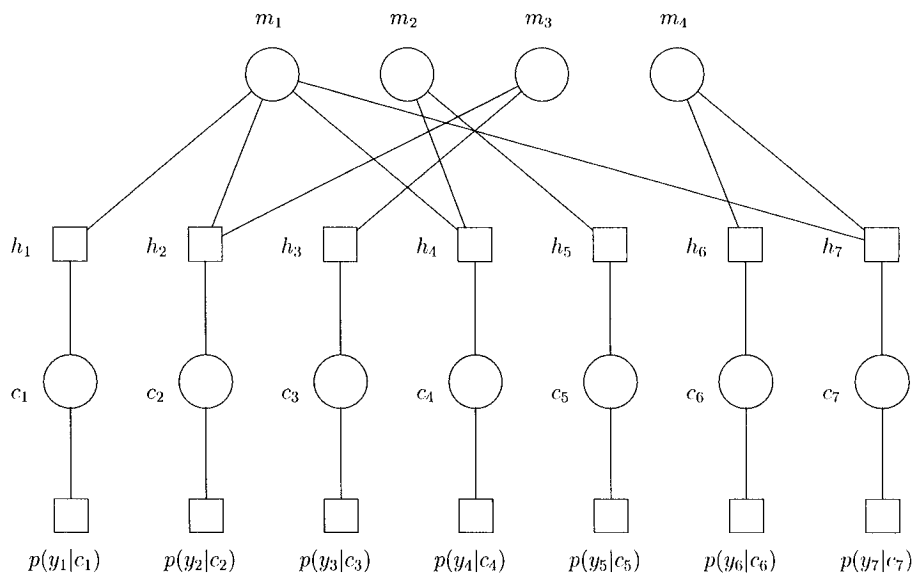

Figure 2.9: A factor graph representing $h(m_1, \ldots, m_4, c_1, c_2, \ldots c_7) \prod_{i=1}^{7} p(y_i|c_i)$ constructed from generator matrix G in (2.8).

For codes defined on graphs, it is known that cycle-free graphs correspond to poor codes [19]. A standard implementation of the sum-product algorithm is illustrated as follows for decoding codes represented by graphs with cycles. However, the illustration

is shown only for the factor graph which is constructed from the parity-check matrix of the code.

Sum-product decoding on graphs with cycles (constructed from a parity-check matrix)

Initialization: Initialize all messages to constant 1 except those sent from the leaf function nodes representing function $p(y_i|c_i)$, which are set to the function $p(y_i|c_i)$ itself. These messages are often referred to as the *intrinsic information.*

Propagation: All variable nodes representing a codeword symbol send messages; then all function nodes representing a parity-check constraint send messages. Iterate this process.

Termination: When every message converge to a fixed point (up to scale) or when a pre-determined number of iterations is reached, compute the summary messages for all variable nodes, and terminate the algorithm.

2.2.3 Decoding algorithm

To avoid overflow and underflow, it is necessary to represent message using their logarithmic values. We call this method as Log-sum-product algorithm, which is explained next.

We first review some definitions and notations for decoding algorithm as follows: An $M \times N$ parity-check matrix is denoted by H and the entry of H by $H_{i,j}$. Let the set of coded bits in a codeword associated with parity check m be denoted by $\mathcal{N}(m) = \{n : H_{m,n} = 1\}$, the set of parity checks associated with bit n be denoted by $\mathcal{M}(n) = \{m : H_{m,n} = 1\}$. The log-sum-product algorithm are summarized [20] as follows:

Iterative Log-sum-product Decoding Algorithm

- **Input:** The *prior* probabilities $p_n^0 = P(x_n = 0)$ and $p_n^1 = P(x_n = 1) = 1 - p_n^0$, $n = 1, \cdots, N$;
- **Output:** Hard decision $\hat{x} = \{\hat{x}_1, \cdots, \hat{x}_N\}$;
- **Procedure:**

1. *Initialization:* For each a, compute the intrinsic information $\gamma_n = \log \frac{p_n^0}{p_n^1}$ and for each $(m, n) \in \{(i, j) | H_{i,j} = 1\}$, compute

$$\alpha_{m,n} = sign(\gamma_n) \log(\frac{1 + e^{-|\gamma_n|}}{1 - e^{-|\gamma_n|}}), \tag{2.9}$$

where $sign(\gamma_n) = \begin{cases} +1, & \gamma_n \geq 0; \\ -1, & \gamma_n < 0. \end{cases}$

2. *Iterative Decoding:*

   (a) Check node processing step: For each m, n, compute

$$\beta_{m,n} = \log(\frac{1 + e^{-\alpha}}{1 - e^{-\alpha}}) \prod_{n' \in \mathcal{N}(m) \backslash \{n\}} sign(\alpha_{m,n'}), \tag{2.10}$$

   where $\alpha = \sum_{n' \in \mathcal{N}(m) \backslash n} |\alpha_{m,n'}|$

   (b) Variable node processing step: For each m, n, compute

$$\alpha_{m,n} = sign(\gamma_n) \log(\frac{1 + e^{-|\gamma_{m,n}|}}{1 - e^{-|\gamma_{m,n}|}}), \tag{2.11}$$

   where $\gamma_{m,n} = \gamma_n + \sum_{m' \in \mathcal{M}(n) \backslash m} \beta_{m',n}$

   (c) Update *LLR* step: For each n, update the log-likelihood ratio (LLR) λ_n as

$$\lambda_n = \gamma_n + \sum_{m \in \mathcal{M}(n)} \beta_{m,n}. \tag{2.12}$$

   (d) Decision step:

      i. Perform hard decision on $(\lambda_1, \ldots, \lambda_n)$ to obtain $\hat{x} = \{\hat{x}_1, \ldots, \hat{x}_N\}$ such that $\hat{x}_n = 0$ if $\lambda_n > 0$ and $\hat{x}_n = 1$ if $\lambda \leq 0$.

      ii. Algorithm is terminated upon reaching a pre-set number of iterations.

Concisely, it is known that the implementation of the sum-product algorithm over cycle-free factor graphs provides optimum decoding and has been the standard approach for decoding graph-based codes, such as the LDPC codes. Furthermore, the performance of these codes has been shown very close to Shannon's theoretical limit.

2.3 LDPC codes

Low-density parity check (LDPC) codes were invented by Gallager [2] in 1962, it had been ignored for over 30 years due to requirement of high complexity computation. It was rediscovered by Mackay(1997) [16], Richardson and Urbanke(2001) [17,21]. In Gallager's PhD thesis, he introduced two innovative ideas for LDPC codes: iterative decoding and constrained random code construction.

It has been shown that LDPC codes can approach Shannon capacity with good block error correcting performance and low error floor. Also, LDPC codes have linear decoding complexity in time and are suitable for parallel implementation.

Typically, a linear block code is represented by a generator matrix G or a parity check matrix H. An LDPC code is a linear block code defined by parity check matrix H which is sparse and has uniform or non-uniform degree distribution. Also, a bipartite graph can represent H matrix with symbol nodes and check nodes. There is an edge in the graph between symbol nodes and check nodes if and only if the value of the entry of H is a 1. In contrast with *irregular* LDPC codes, we often call (d_r, d_c)-*regular* LDPC code whose parity check matrix has uniform degree distribution for each row and column equal to d_r and d_c, respectively.

A family of LDPC codes are specified by a given degree distribution $(\{\lambda_i\}, \{\rho_j\})$ of factor graphs representing parity-check matrices. In LDPC literature, $\{\lambda_i\}$ is often referred to as the *left-degree distribution* and $\{\rho_j\}$ referred to as the *right-degree distribution*. In general, an pair of degree distributions can characterize an ensemble of bipartite or Tanner graphs. Figure 2.10 shows an example of irregular LDPC code of length 8.

Specifically, the degree distribution is defined as

$$\lambda(x) := \sum_{i \geq 1} \lambda_i x^{i-1} \tag{2.13}$$

where λ_i is the fraction of variable nodes connected to i check nodes; and

$$\rho(x) := \sum_{i \geq 1} \rho_i x^{i-1} \tag{2.14}$$

where ρ_i is the fraction of check nodes connected to i variable nodes.

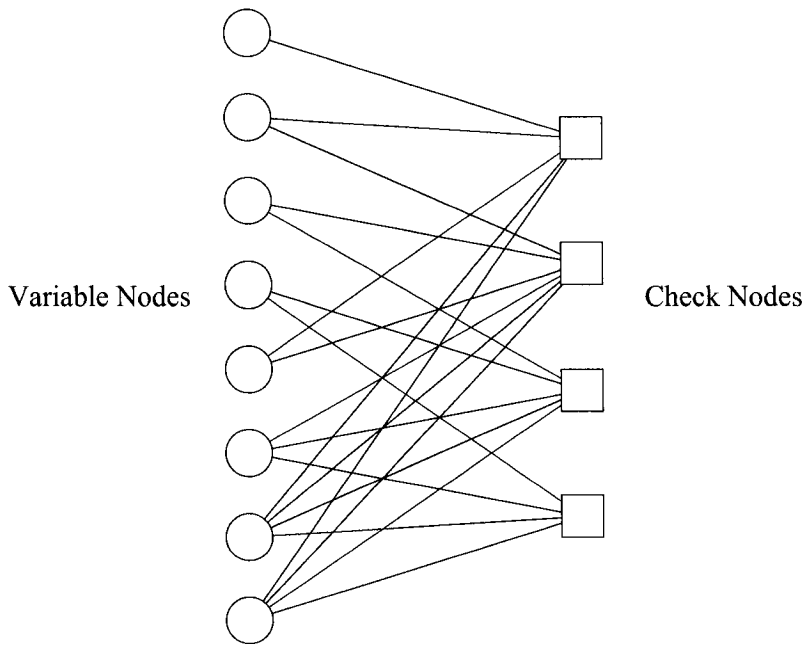

Figure 2.10: Graph representation of an irregular LDPC codes at length 8. The left nodes represent the variable nodes whereas the right nodes represent the check nodes.

Provided that the left and right degree distributions with the assumption that all rows of the parity check matrices are linearly independent, the rate of the family LDPC codes can be determined as a constant, independent of the length of the code. LDPC codes are randomly constructed usually. To date, the sum-product algorithm is known as the best-performing decoding algorithm for LDPC codes.

For long block lengths no smaller than $10,000$ [16], LDPC codes have been shown to

achieve performance very close to Shannon limit. Irregular LDPC codes seem to perform better than regular ones [17]. For codes at short block lengths, LDPC codes have been investigated and achieved competitive performance [22] as well.

2.4 LDGM codes

LDPC codes are shown to achieve Shannon limit. Another main advantage of LDPC codes over turbo codes is fully parallelized decoder which allows fast decoding. Since the encoding of LDPC codes operates the matrix multiplication at large size, the encoding complexity is much higher than that of turbo codes. Consequently, it is required to find a subset of LDPC code which has efficient encoding scheme as well as efficient decoding algorithm.

2.4.1 Overview of LDGM codes

Systematic LDGM code is a linear code with a sparse generator matrix, and therefore the corresponding parity check matrix is sparse, the decoding of LDGM codes has lower complexity than that of Turbo codes accordingly. Since LDGM codes are subset of LDPC codes and have sparse generator matrix, the encoding of LDGM codes has lower complexity than that of standard LDPC. However, regular LDGM codes are asymptotically bad [23] since they present error floors that are independent of the block length. Concatenated scheme of LDGM code [1] was shown to approach near Shannon limit at large block lengths.

LDGM codes are linear block codes with generator matrix, $G = [IP]$, where I is a $k \times k$ identity matrix and P is $k \times (n-k)$ sparse matrix [1]. Here, k denotes the number of information bits and n the number of bits in a codeword. Hence, the parity check matrix of LDGM codes is $H = [P^T I]$. The information message is denoted by $\underline{u} = [u_1, \cdots u_k]$, a real vector $\underline{c} = [c_1, \ldots, c_n]$, that is in fact transmitted, is generated by $\underline{c} = \underline{u}G$. Therefore, the transmitted sequence consists of systematic bits ($c_i = u_i, i = 1 \ldots k$) and coded bits ($c_j, j = k+1 \ldots n$). LDGM codes can be represented by bipartite graph shown in Figure 2.11.

As illustrated in Figure 2.11, there exists $(n-k)$ coded bit nodes of degree-1 and k bit nodes corresponding to the systematic bits. As a result, the messages propagated from the degree-1 coded bit nodes to the corresponding check nodes will always be the same, thus LDGM codes have high error floors.

At decoder, finally, each transmitted c_i suffers from Gaussian noise N_i according to

$$c_i' = c_i + N_i$$

where c_i' is the received number for c_i and N_i is a zero-mean Gaussian random variable with variance σ^2. The sum-product algorithm can be then applied to decode LDGM codes.

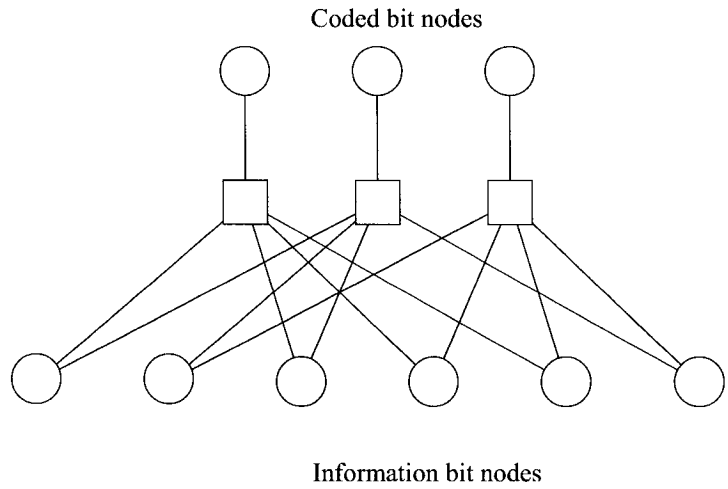

Figure 2.11: Bipartite graph representation of a LDGM code.

2.4.2 Serial concatenation of irregular LDGM codes

As mentioned in [23], regular LDGM codes are asymptotically bad. In [1], an analytical method was introduced to calculate the error floors of LDGM codes over Binary Symmetric channel (BSC). Equation (1) and (2) in [1] show that the use of single LDGM codes always leads to error floors that are independent of the block length.

In order to reduce error floors existing in regular LDGM codes, a simple serial concatenation scheme of two LDGM codes can be used as shown in Figure 2.12. The simulation performance of serial concatenation LDGM codes over AWGN channel and BSC are shown in [1] comparable to that of irregular LDPC and turbo codes but having lower encoding and decoding complexity. The graph representation of serial concatenation is

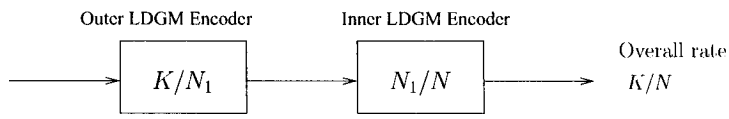

Figure 2.12: Serial concatenated scheme to reduce the error floor of LDGM codes. First the information bits are encoded by a high rate LDGM outer code. Then, the output of the outer code is encoded by an inner code to produce a rate k/N overall code

shown in Figure 2.13. Simulation results illustrated in [1] Fig.3 show the good performance of this concatenation LDGM codes.

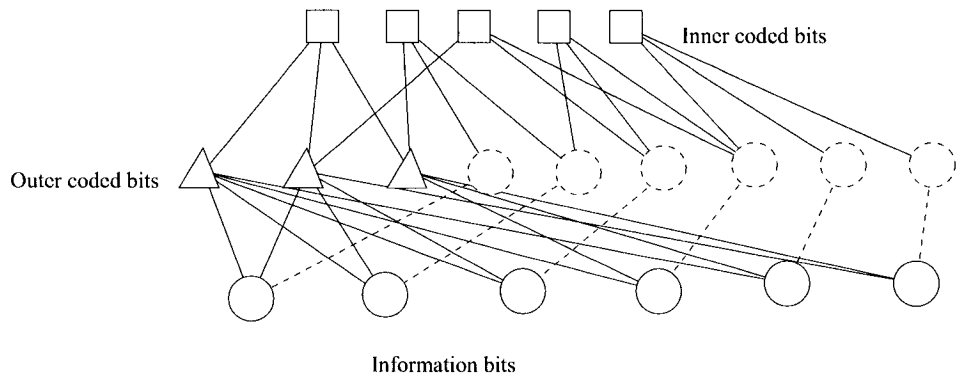

Figure 2.13: Graph associated with serial concatenated scheme with the inner coded bit nodes and the outer coded bit nodes, and the information bit nodes

The objective of the serial concatenated LDGM codes presented in [1] is to design two constituent codes, one with the high rate as outer code that takes care of the error floors, and the other with the desired rate as inner code that determines its *threshold*, a value of the channel parameter which defines the asymptotic performance of a given ensemble of a code. However, since the generator matrix G_{whole} of the whole code associated with its graph representation is the product of the generator matrices of the two constituent

codes, it results in the performance degradation such that the specific cycles appear in the graph [24]. It does not appear in the parallel concatenation scheme.

2.4.3 Parallel concatenation of irregular LDGM Codes

In [11], parallel concatenation LDGM codes were proposed instead of serial concatenation scheme. In fact, the whole code of this concatenation LDGM scheme is a single irregular LDGM code. Figure 2.14 shows that information bits are encoded by a rate-$k/(k+m)$ regular LDGM code (encoder 1) and by a rate-$k/(k+n)$ regular LDGM code (encoder 2) to construct a rate-$k/(k+m+n)$ overall code. Thus, the generator matrix of a single irregular LDGM codes is expressed by $G_{whole} = [IP_1P_2]$, where $G_1 = [IP_1]$ and $G_2 = [IP_2]$. Consequently, the transmitted message can be obtained as $[\underline{uc^1c^2}] = \underline{u}G_{whole}$. The graph representation of parallel concatenation is shown in Figure 2.15.

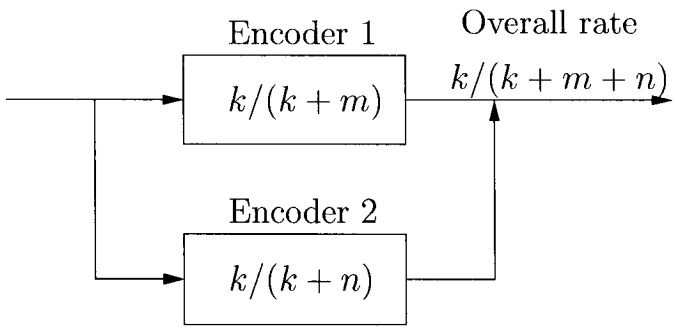

Figure 2.14: Parallel concatenated scheme to reduce the error floor of LDGM codes. Information bits are encoded by a rate-$k/(k+m)$ regular LDGM code (encoder 1) and by a rate-$k/(k+n)$ regular LDGM code (encoder 2) to construct a rate-$k/(k+m+n)$ overall code

For the case of the parallel concatenation LDGM codes, the first encoder will determine the thresholds while the second encoder will reduce error floors [11]. By observation shown in Figure 2.15, the parallel concatenation scheme is in fact a single irregular LDGM code. This leads to the implementation of iterative decoding using the graph associated with the whole code easily.

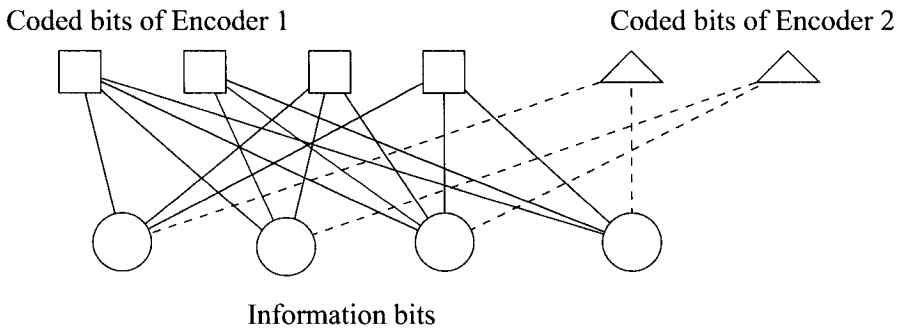

Figure 2.15: Graph associated with parallel concatenated scheme with the coded bit nodes of encoder 1 and the coded bit nodes of encoder 2, and the information bit nodes

Chapter 3

Code Construction

3.1 Progressive Edge-growth algorithm

In Chapter 2.2, we reviewed the approach to describing the decoding algorithm of LDPC codes over cycle-free factor graph. The sum-product algorithm provides optimum decoding based on factor graph for rather long block lengths.

For very large block lengths, it is possible to construct graph randomly. Unfortunately, random construction is not efficient for short block length. In this chapter, we investigate that LDGM codes can perform no worse than LDPC codes at short block length using Progressive Edge-Growth algorithm (PEG). In the beginning of this chapter, we briefly recall the PEG algorithm by introducing definitions and notations.

3.1.1 Definitions and notations

A LDPC code of length n is defined by a sparse parity check matrix H which has dimension $m \times n$ represented by factor graph. Factor graph is denoted as (V, E), where $V = V_c \cup V_s$, $V_c = \{c_0, c_1, \ldots, c_m\}$ is the set of check nodes, and $V_s = \{s_0, s_1, \ldots, s_{n-1}\}$ is the set of symbol nodes, E is the set of edges such that $E \in V_c \times V_s$. The n symbol nodes s_j represent the n columns of the parity-check matrix H, and the $m \geq n - k$ check nodes c_j associated with the m rows of H. The symbol node s_j is connected to the check node c_j by an edge, if and only if there is a "1" in the i-th column and in the j-th row

of the parity-check matrix H. The number of edges incident to s_j is the number of "1"s in column i and is called the symbol node degree $d(s_i)$. Similarly, the number of edges incident to c_j is the number of "1"s in row j and is called the check node degree $d(c_j)$.

A graph is then *simple* if 1)at most one edge connects each pair of nodes, 2)it contains no loop or multiple edges, 3)all edges are non-directed. A graph G' is a subgraph G if it's edges and vertices are subsets of the vertex and edge sets of G. Then G is the supergraph of G'. The *neighborhood* of a vertex is all the vertices that it is adjacent to. A *path* is a sequence of consecutive edges in a graph and the *length* is the number of edges traversed. In a *closed* path, $x_0 = x_n$. A *cycle* is a simple closed path. Girth g denote the length of the shortest cycle in a graph. For each symbol node s_j or check node c_j, local girth g_j is defined as the length of the shortest cycle passing through that symbol or check node.

Figure 3.1 shows the factor graph representing the LDPC code given by **H**. The following example demonstrate a regular factor graph.

Example 3. 1 Assume the parity-check matrix H will serve as an example as follows.

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

Also, let E_{s_j} be the all edges incident on node s_j, and $E_{s_j}^k$ be the k-th edge of the symbol node s_j. For the symbol node s_j, we define $\mathcal{N}_{s_j}^l$ as the set consisting of all check nodes reached by a tree spreading from symbol node s_j within depth l. The complementary set $\bar{\mathcal{N}}_{s_j}^l$ is the set of check nodes that are not reached by a tree from symbol node s_j within l.

In Figure 3.1, every symbol node has the same degree $d(s_j) = 2$ and every check node has the same degree $d(c_j) = 4$. We call such codes a $(2, 4)$-regular LDPC code. It is known that irregular LDPC codes outperform than regular LDPC code [17]. Moreover, we denote $d_{s_{\max}}$ and $d_{c_{\max}}$ as maximum symbol node degree and maximum check node degree, respectively, then the degree distribution is defined as

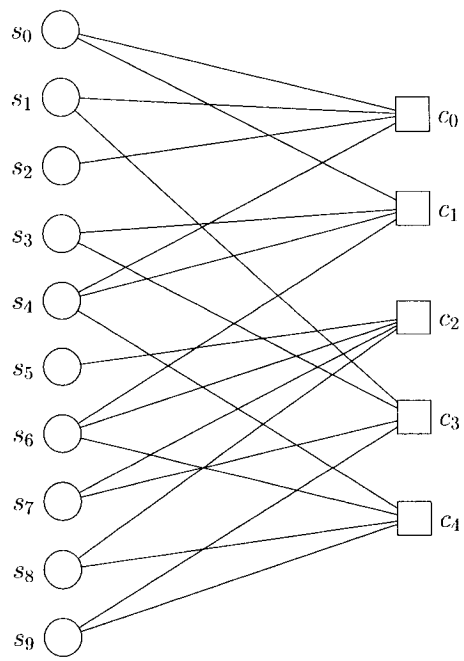

Figure 3.1: A (2,4)-regular Tanner graph

$$\lambda(x) := \sum_{\ell \geq 2}^{d_{s_max}} \lambda_\ell x^\ell \tag{3.1}$$

where λ_ℓ is the fraction of symbol nodes connected to ℓ check nodes; and

$$\rho(x) := \sum_{j \geq 2}^{d_{c_{\max}}} \rho_j x^j \tag{3.2}$$

where ρ_j is the fraction of check nodes connected to j symbol nodes.

3.1.2 PEG algorithm

The PEG algorithm was presented in [3] for efficiently constructing factor graphs with large local girth. In [3], when constructing a factor graph with a given symbol node degree distribution or check node degree distribution, an edge-selection procedure is started so that the placement of the new edge has an impact on the girth as small as possible. This

can be done by expanding a tree from one symbol node s_j up to the depth l, so that $\bar{\mathcal{N}}_{s_j}^{l} \neq \emptyset$ but $\bar{\mathcal{N}}_{s_j}^{l+1} = \emptyset$ or the cardinality stops increasing but is smaller than m. Then the symbol node s_j is connected to a check node from the set $\bar{\mathcal{N}}_{s_j}^{l}$ that has the lowest $d(c_j)$ under the current setting. This maximizes the length of the shortest cycle through this new edge.

Similarly, for the parity-check node, we define $\mathcal{N}_{c_j}^{l}$ as the set consisting of all symbol nodes reached by a tree spreading from check node c_j within depth l. The complementary set $\bar{\mathcal{N}}_{c_j}^{l}$ is the set of symbol nodes that are not reached by a tree from check node c_j within depth l.

Figure 3.2 demonstrates that a tree is spread by means of unfolding the Tanner graph in a breadth-first way. The procedure starts from root c_j, and traverse all edges incident on c_j; we denote these edges as $(c_j, s_{i_1}), (c_j, s_{i_2}), \ldots, (c_j, s_{i_{d_{c_j}}})$. Then, we traverse all other edges incident on vertices $s_{i_1}, s_{i_2}, \ldots, s_{i_{d_{c_j}}}$, excluding $(c_j, s_{i_1}), (c_j, s_{i_2}), \ldots, (c_j, s_{i_{d_{c_j}}})$. The procedure will not stop until the desired depth l is reached.

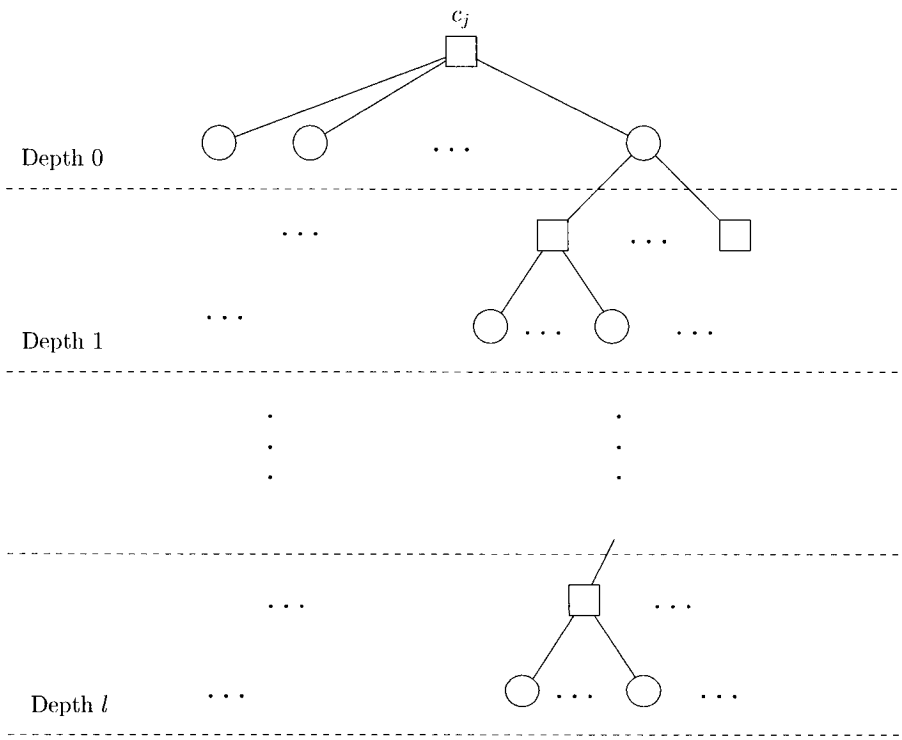

Figure 3.2: A subgraph spreading from check node c_j

For long block lengths, random construction of LDPC codes works very well. For short block lengths, however, the PEG algorithm presented in [3] deals with maximizing the local girth of a symbol or check node whenever a new edge is added to this node. Typically, the PEG algorithm is operated in order to find the most distant check node and afterwords to put a new edge for connecting the symbol node and this most distant check node.

The algorithm can be summarized [3] as follows:

Progressive Edge-Growth Algorithm:

for $j = 0$ to $n - 1$ **do**
  for $k = 0$ to $d_{c_j} - 1$ **do**
    if $k = 0$ **then**
      $E_{c_j}^0 \leftarrow \text{edge}(s_i, c_j)$, where $E_{c_j}^0$ is the first edge incident to c_j and s_i is a symbol node such that it has the lowest symbol node degree under the current graph setting $E_{c_0} \cup E_{c_1} \cup \ldots \cup E_{c_{j-1}}$.
    else
      expand a subgraph from symbol node c_j up to depth l under the current graph setting such that the cardinality of $\mathcal{N}_{c_j}^l$ stops increasing but is less than m, or $\bar{\mathcal{N}}_{c_j}^l \neq \emptyset$ but $\bar{\mathcal{N}}_{c_j}^{l+1} = \emptyset$, then $E_{c_j}^k \leftarrow edge(s_i, c_j)$, where $E_{c_j}^k$ is the kth edge incident to c_j and s_i is a symbol node picked from the set $\bar{\mathcal{N}}_{c_j}^l$ having the lowest symbol node degree.
    end if
  end for
end for

The PEG algorithm is very powerful in that it can generate good regular and irregular LDPC codes at short to moderate block lengths. Simulation results show that a significant improvement can be made compared to randomly constructed codes using the PEG algorithm [3]. For details, the reader is referred to [3].

3.2 Code construction and analysis

In previous section, we briefly described how the PEG algorithm generates LDPC codes at short to moderate block lengths. In [11], they made effort to design parallel concatenation LDGM codes combined with density evolution for long block lengths. Previous work has shown that LDGM codes are dual to LDPC codes with low complexity of encoding. In this section, we investigate that LDGM codes constructed by the PEG algorithm can perform no worse than LDPC codes at short block lengths.

3.2.1 Construction of irregular LDGM codes

In Chapter 2.4.3, we explained the parallel LDGM scheme and illustrated the construction of parallel scheme. In [11], the generator matrices of the whole code is $G_{whole} = [IP_1P_2]$, and matrices P_1 and P_2 are generated separately in a pseudorandom way. In fact, the whole code rate of parallel concatenation LDGM scheme is a single irregular LDGM code.

In this section, we may show how the construction of LDGM codes can be made in order to explore the performance of the constructed code at short block lengths for different code rates. In general, the generator matrix of the code is $G = [IP]$, where we will generate P matrix using the PEG algorithm described in the previous section. For a given code rate R, we want to construct P matrix, such that

1. The number of 1's in a row is r;

2. The number of 1's in a column is either c_1 or c_2 associated with matrices P_1 and P_2, respectively.

Let f_1 be the fraction of columns with c_1 1's, f_2 be the fraction of columns with c_2 1's, *i.e.*

$$f_1 = \frac{n_1}{n_1 + n_2} \tag{3.3}$$

$$f_2 = \frac{n_2}{n_1 + n_2}. \tag{3.4}$$

$$n = n_1 + n_2 \tag{3.5}$$

Given r, c_1, c_2, f_1, f_2, we have

$$k \cdot r = n_1 \cdot c_1 + n_2 \cdot c_2 \tag{3.6}$$

We substitute (3.3) to (3.5) into (3.6) and then we get

$$R = \frac{k}{n} = \frac{f_1 \cdot c_1 + f_2 \cdot c_2}{r + f_1 \cdot c_1 + f_2 \cdot c_2} \tag{3.7}$$

For a given code rate R, there are three of four parameters free to choose and determine the last one such that

$$f_1 = \frac{Rr + c_2 R - c_2}{c_1 - c_2 - c_1 R + c_2 R} \tag{3.8}$$

where $0 < f_1 < 1$, and

$$f_2 = 1 - f_1 \tag{3.9}$$

In order to construct P matrix of LDGM codes shown in Figure 3.3, the parity-check degree sequence with parameters f_1, f_2, c_1, c_2 can be determined as follows:

$$\rho(x) = f_1 x^{c_1} + f_2 x^{c_2} \tag{3.10}$$

As shown in Figure 3.3, given the k symbol nodes, the $(n - k)$ check nodes and the check-node-degree distribution of the graph, we can construct the graph associated with P matrix using the PEG algorithm. The procedure starts so that the placement of the new edge has an impact on the girth as small as possible.

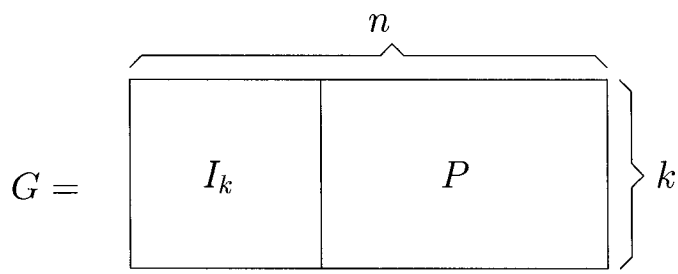

Figure 3.3: Structure of generator matrix G

The decoding algorithm can be used by first transforming P matrix to H matrix illustrated in Figure 3.4 and then apply sum-product over the graph described in Figure 3.5

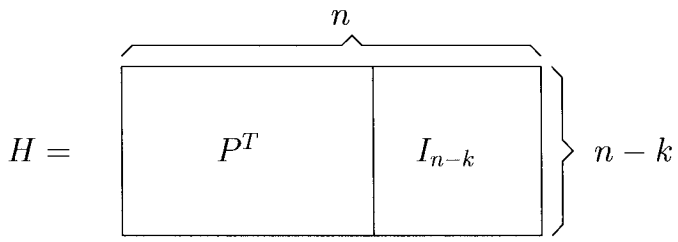

Figure 3.4: Structure of parity-check matrix H

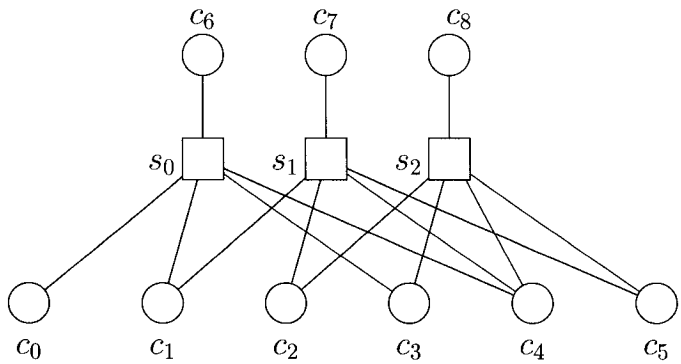

Figure 3.5: Factor graph(or bipartite graph) representation of a systematic LDGM code. $\{c_0, c_1, \ldots, c_k\}$ represent the systematic bits and $\{c_{k+1}, \ldots, c_n\}$ represent the coded bits. $\{s_0, s_1, \ldots, s_{n-k}\}$ represent the check nodes. Note that all the coded bits have degree 1 and are associated with their corresponding check nodes

3.2.2 Code discussion

Concatenated LDGM schemes

As explained in [1], and first realized by MacKay [23], LDGM codes are bad and present high error floors which do not improve with increasing block length. LDGM codes, however, have good convergence thresholds while high error floor for the ones with small degrees. An analytical method can be used to explain for BSC why the use of single LDGM codes always leads to error floors [1]. For BSC, the number of errors of the blocks

in error decays very fast, provided that the crossover probability is small enough. Moreover, the residual errors can be explored by decoder. As result, a significant reduction of error floors can be achieved by using concatenation schemes of two LDGM codes, such as the ones shown in Figure 2.12 and Figure 2.14, respectively. The resulting performance of parallel and serial concatenation LDGM codes over AWGN and BSC channels shown in [1, 11] are comparable to that of irregular LDPC and turbo codes with lower encoding and decoding complexity.

The whole code of two concatenation LDGM codes are in fact a single irregular LDGM code. For the case of serial concatenation scheme, the generator matrix is the product of the generator matrices of two constituent codes, that is $G_{whole} = G_{outer}G_{inner}$ which results in the performance degradation due to the specific cycle structures in G_{whole} [24]. For the case of parallel concatenation scheme, on the other hand, the generator matrix $G_{whole} = [IP_1P_2]$, where $G_1 = [IP_1]$ and $G_2 = [IP_2]$. Therefore, the construction of parallel concatenation scheme is more simpler than that of serial concatenation one, and decoding algorithm can be operated directly on the graph of the overall code. Thus, the parallel concatenation scheme was used to construct and investigate LDGM codes at short block length in this thesis.

The PEG algorithm

Since LDGM codes are subset of LDPC codes, which are usually decoded by the sum-product algorithm, LDGM codes can be decoded in the same way. However, the existence of cycles in the factor graph of the LDPC codes leads to the performance degradation of the decoding algorithm. The progressive edge-growth (PEG) algorithm [3] deals with this problem, and proposes a solution by maximizing the local girth with a greedy algorithm. It places the edges between symbol and check nodes progressively, trying to maximize the local girth in each step.

Although only local optimization is made, the PEG algorithm can generate LDGM codes with very good performance at short to moderate block lengths. This algorithm is to be known as one of the best code construction methods, so we apply this algorithm for the construction of P matrix and form the G matrix for short LDGM codes, then transform it into H matrix for decoding.

There is a subtlety for the construction of P matrix that needs clarification here. The subgraph is spread from check node c_j rather than symbol node illustrated in [3]. That is, as presented in (3.10), given the check-node degree distribution and the number of symbol nodes and the number of check nodes, P matrix can be constructed using the PEG algorithm.

Short LDGM codes

In [11], the authors made effort to design a set of parallel concatenation LDGM codes combined with discrete *density evolution* analysis, which is used to determine the asymptotic performance of ensembles of LDPC codes. Although density evolution is currently the best known technique to determine thresholds and to design irregular LDPC codes, and LDGM codes are a subset of LDPC codes, it can be used only under the assumption that the block length is theoretically infinite. Furthermore, density evolution ignores dependence between bits so that it is inefficient for evaluating codes of short block length. Thus in this thesis, we choose not to use density evolution optimized degree sequence and simply apply the several degree sequences provided in [1] in the construction of short LDGM codes.

Chapter 4

Simulation Results and Discussion

4.1 Simulation Setup

In this chapter, we present simulation results for transmissions over Binary Symmetric Channel (BSC) and over AWGN channel. Simulations are performed using two families of LDGM codes with parameter $(1268, 456)$ and $(504, 252)$, respectively.

For the case of BSC shown in Figure 4.1, codewords are directly transmitted through BSC and the sum-product decoder is applied for decoding.

For the case of AWGN channel illustrated in Figure 2.7, BPSK modulated codewords are transmitted through an AWGN channel. Each codeword $(x_1, x_2, \ldots, x_n)$ is intended for transmission, and a real vector $(y_1, y_2, \ldots, y_n)$ is in fact transmitted. For each $i \in \{1, 2, \ldots, n\}$, $y_i = 1$ indicates that $x_i = 0$ and $y_i = -1$ indicates that $x_i = 1$. Each transmitted y_i suffers from Gaussian noise N_i according to

$$z_i = y_i + N_i$$

where z_i is the received symbol for y_i and N_i is a zero-mean Gaussian random variable with variance σ^2, and all N_i's are independent. The sum-product algorithm is applied again at the decoder.

In our simulations, the P matrices of LDGM codes are determined by the check-node

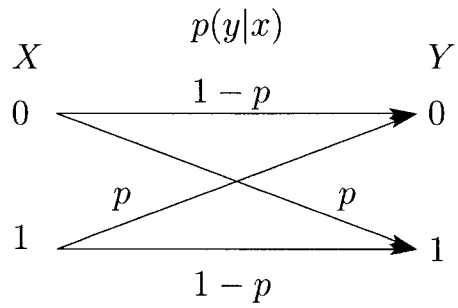

Figure 4.1: Binary Symmetric Channel (BSC)

degree distribution with parameters f_1, f_2, c_1 and c_2 illustrated in (3.10) and generated using the PEG algorithm. The iterative decoder is implemented using the graph representation associated with the corresponding parity-check matrix.

At the decision step during the execution of the decoding algorithm presented in Chapter 2.2.3, hard decision is performed on $(\lambda_1, \ldots, \lambda_k)$ instead. Moreover, the *bit-error rate* must be defined as follows:

$$Pe = \frac{\sum_{i=1}^{M} m_i}{kM}$$

where M is the total number of simulated codewords for a given level of σ^2, m_i is the number of bits that are incorrectly decoded in the first k bits of the i^{th} codeword, and k is the number of information bits in a codeword.

For each channel condition, simulation stops whenever at least 30 codewords are decoded incorrectly. For each block, the iterative decoding does not stop until 100 iterations are completed.

4.2 Simulation Results

Figure 4.2 to 4.4 and Figure 4.5 to 4.7 show the simulation results of irregular $(504, 252)$ LDGM codes over BSC and AWGN channel, respectively. Figure 4.8 to 4.10 and Figure 4.11 to 4.13 illustrate the simulation results of irregular $(1268, 456)$ LDGM codes over BSC and AWGN channel.

Table 4.1 to 4.3 list check-node-degree distribution, average symbol node degree and maximum check-node degree with various values of c_1 for irregular $(504, 252)$ LDGM codes. Table 4.4 to 4.6 list the same parametre settings for irregular $(1268, 456)$ LDGM codes with different values of c_1 as well.

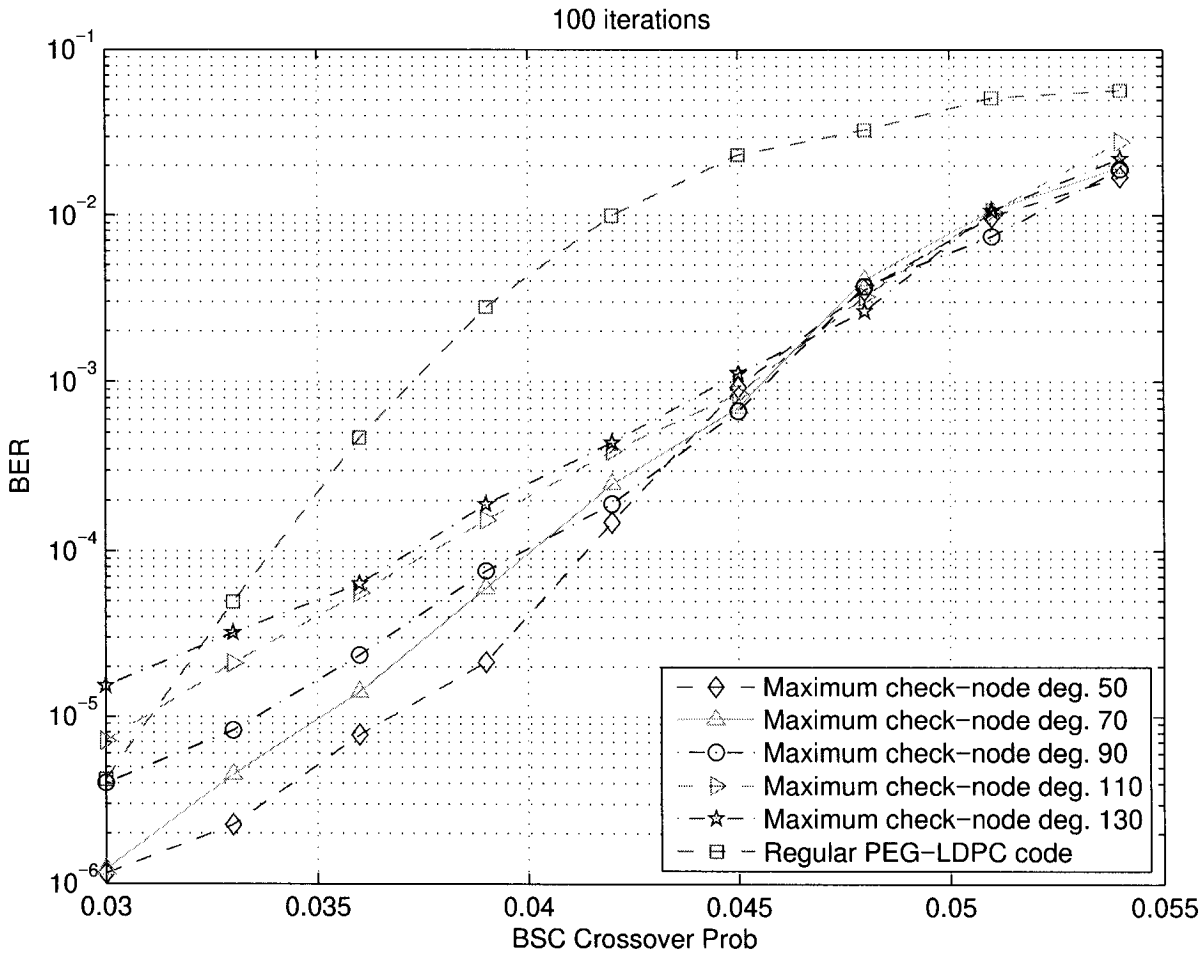

Figure 4.2: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over BSC, with $n = 504, m = 252, c_1 = 5$.

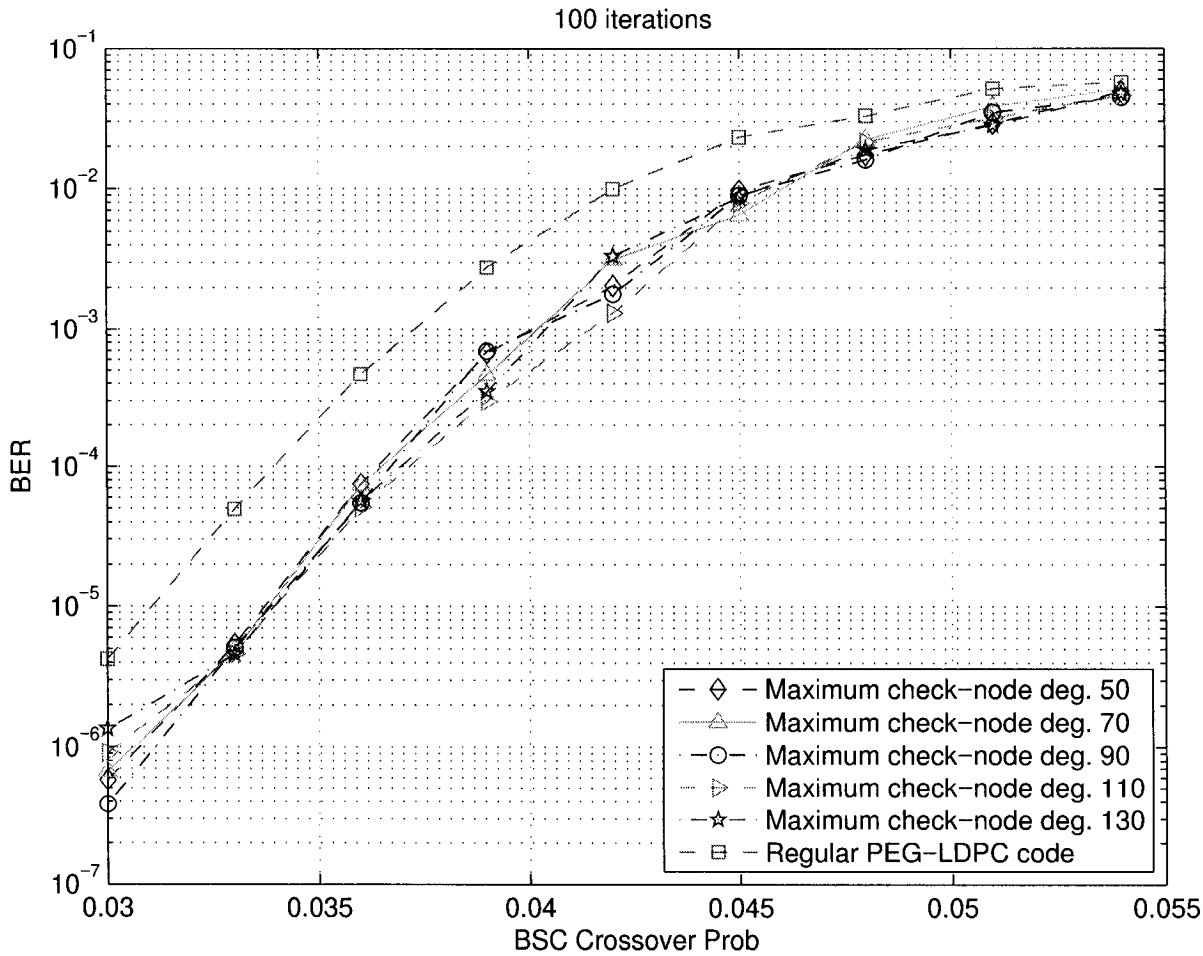

Figure 4.3: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over BSC, with $n = 504, m = 252, c_1 = 6$.

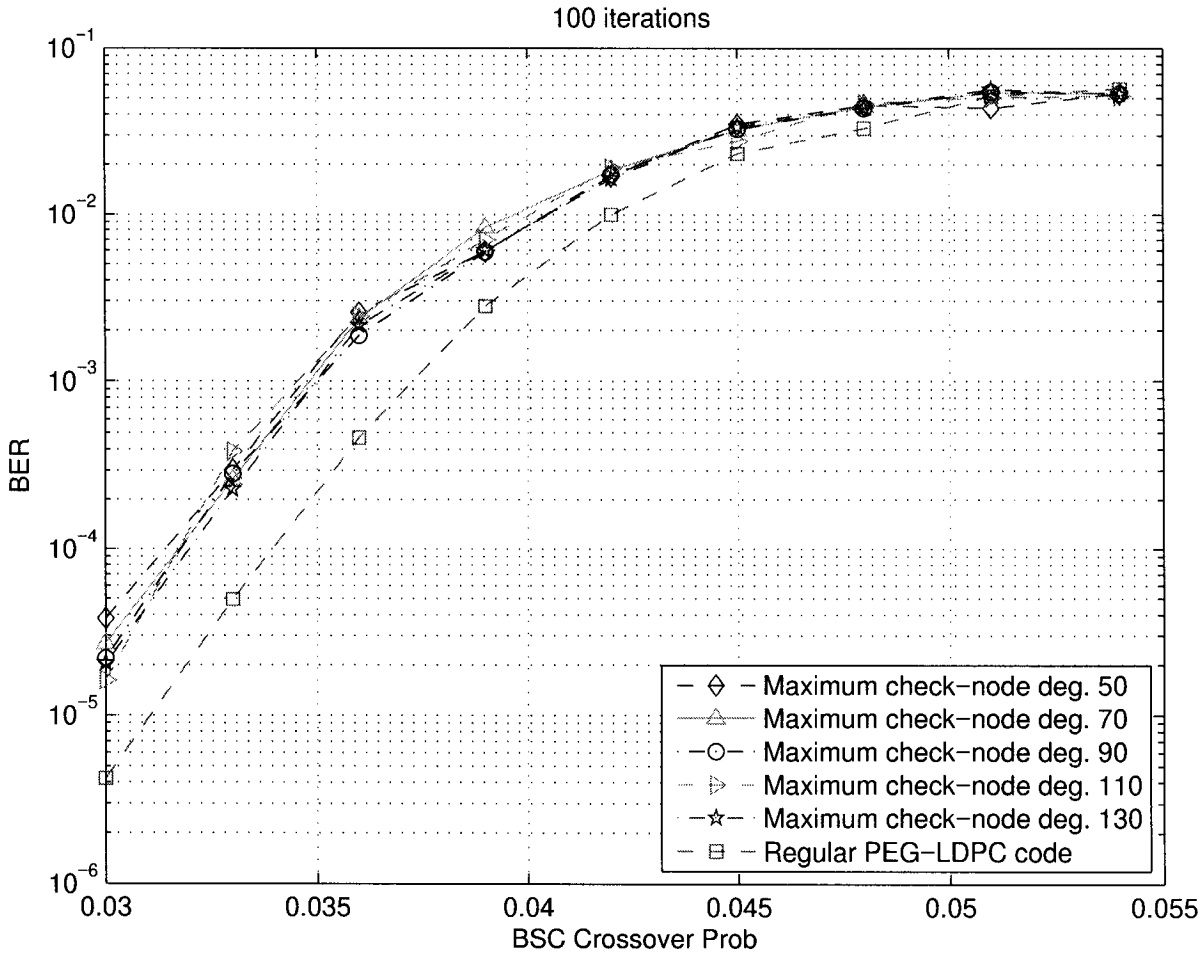

Figure 4.4: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over BSC, with $n = 504, m = 252, c_1 = 7$.

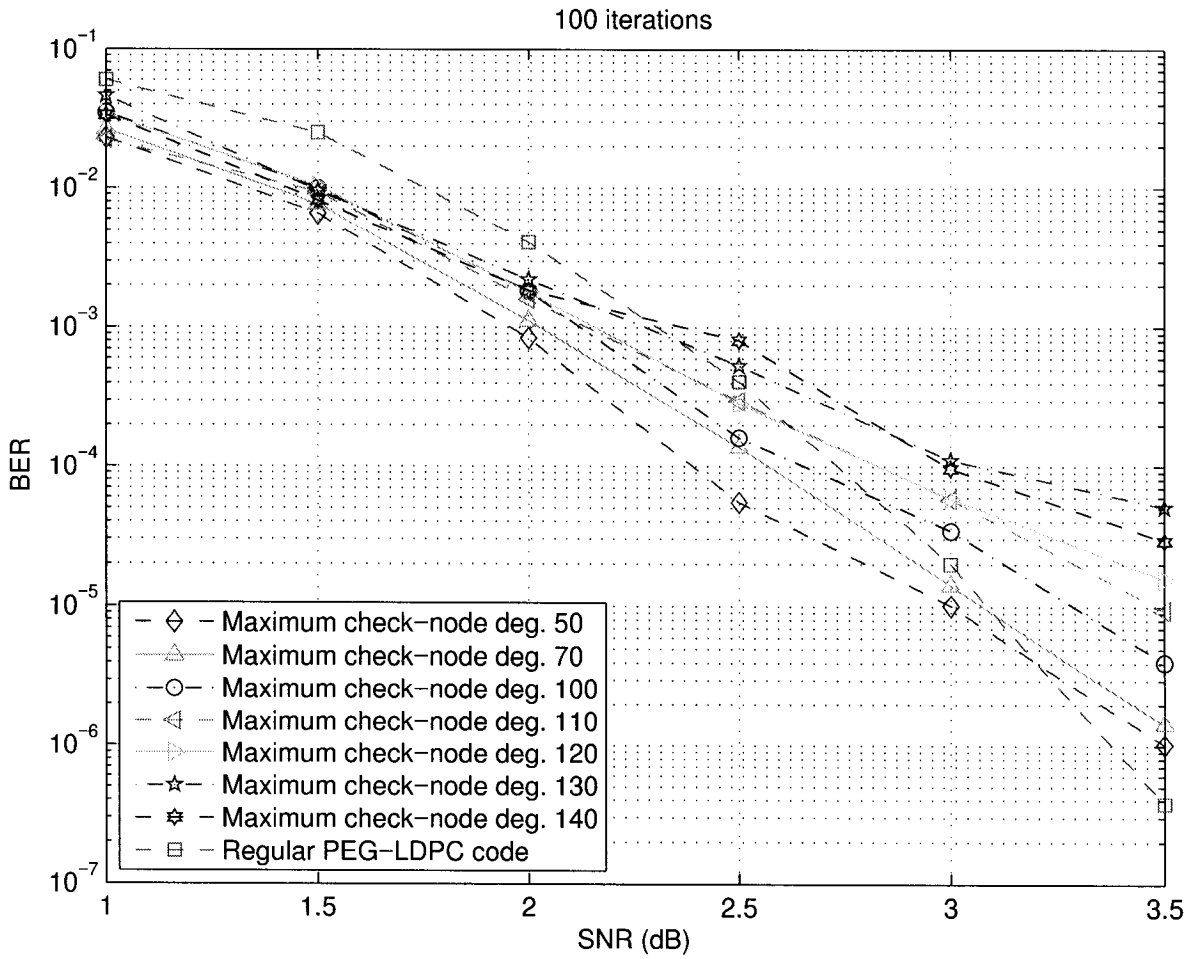

Figure 4.5: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over AWGN channel, with $n = 504, m = 252, c_1 = 5$.

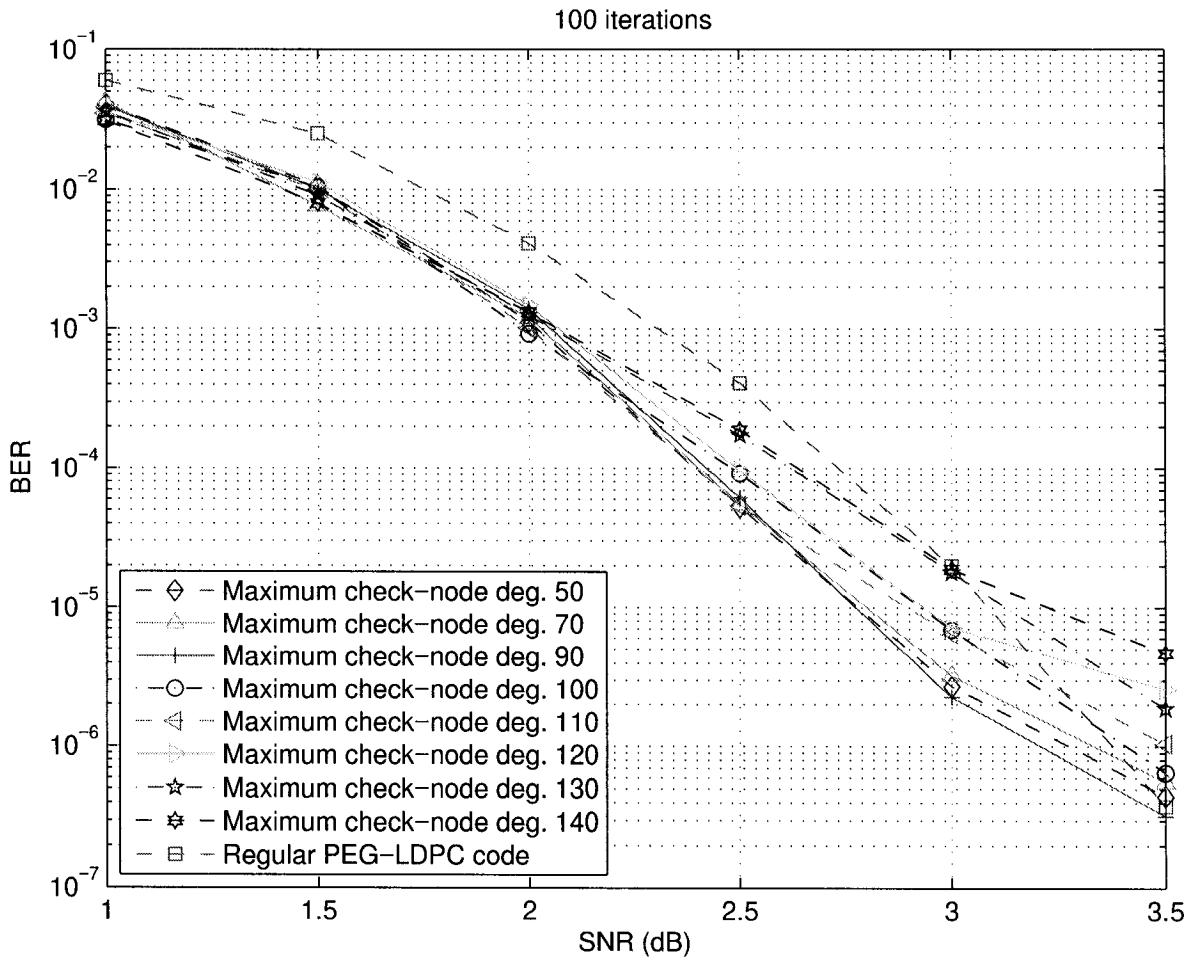

Figure 4.6: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over AWGN channel, with $n = 504, m = 252, c_1 = 6$.

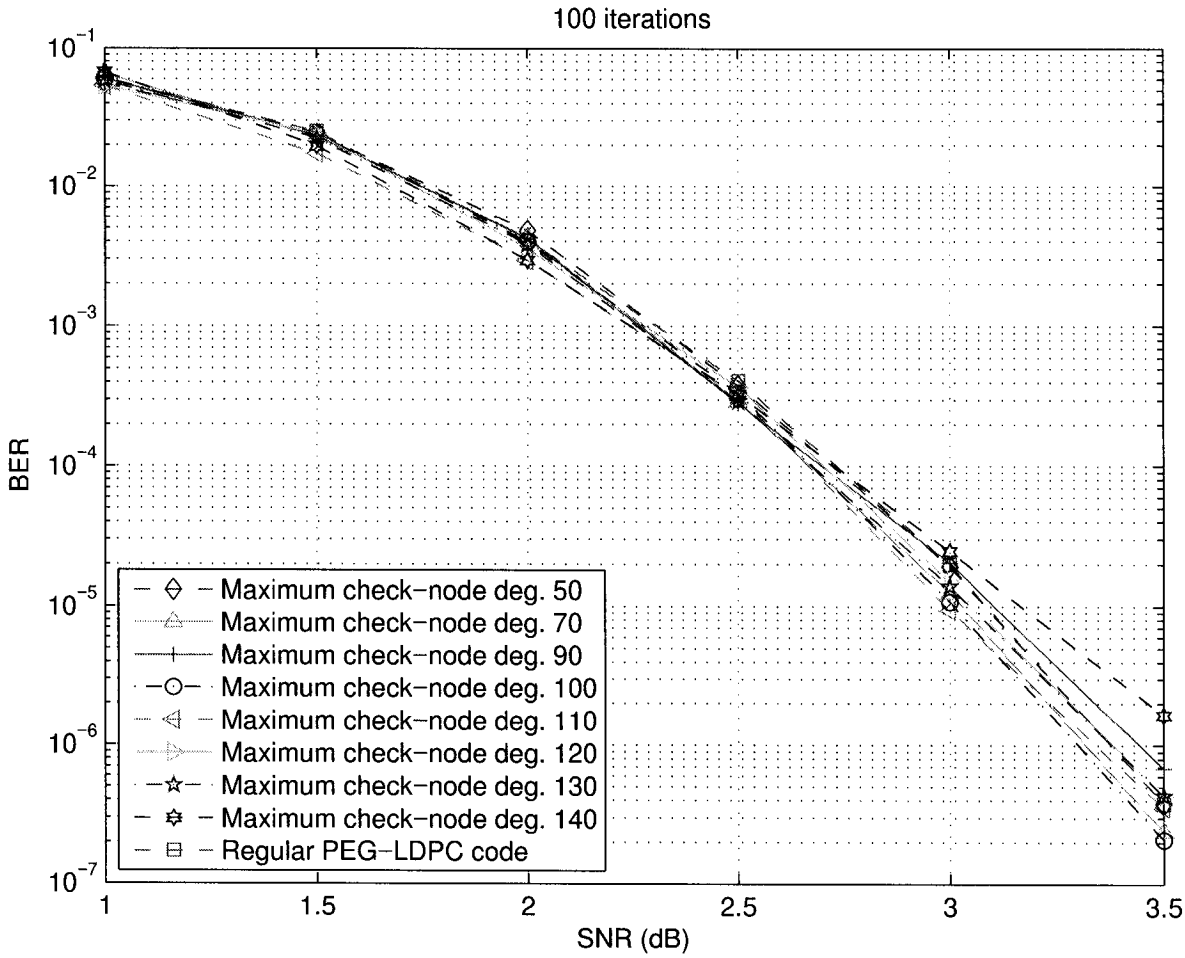

Figure 4.7: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over AWGN channel, with $n = 504, m = 252, c_1 = 7$.

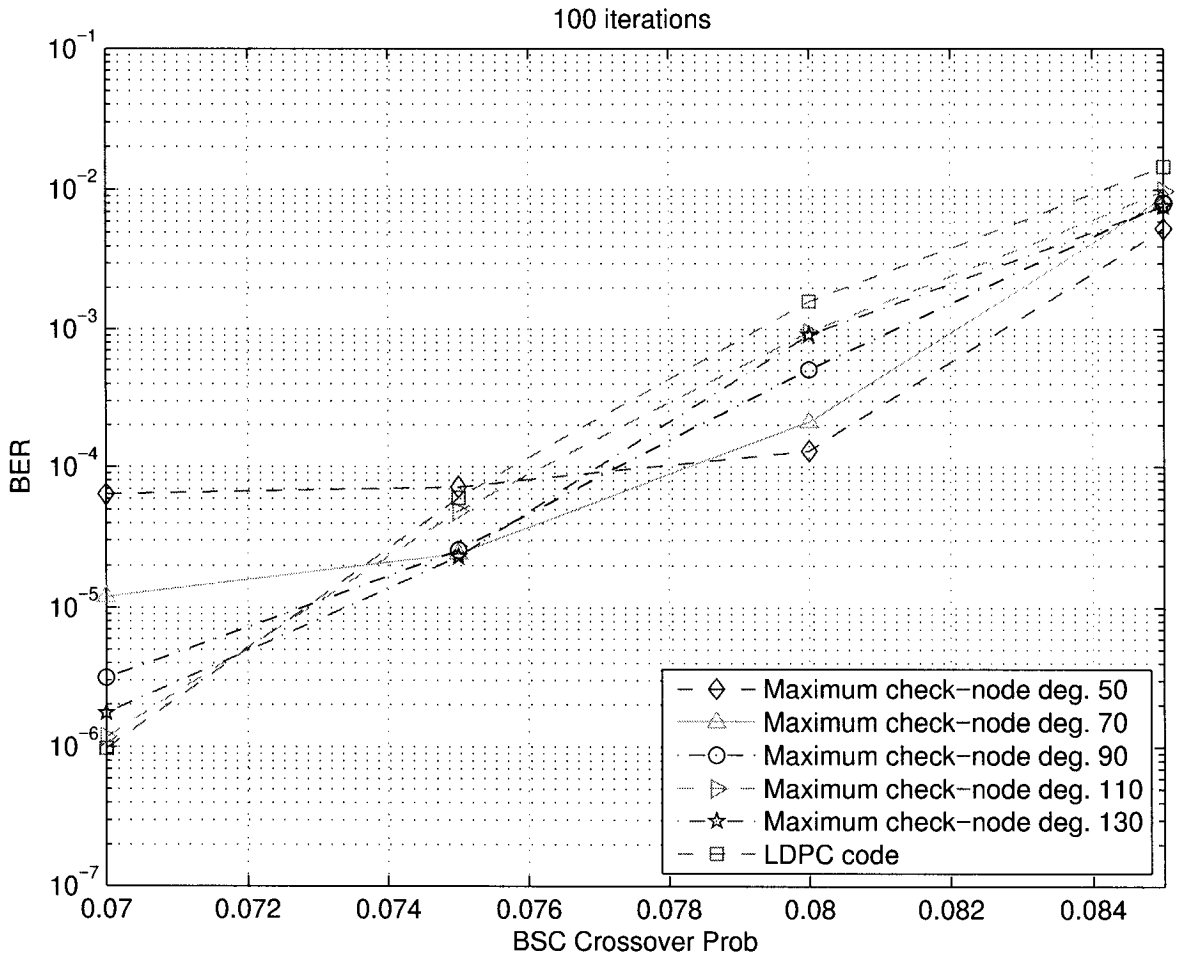

Figure 4.8: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over BSC, with $n = 1268, m = 812, c_1 = 5$.

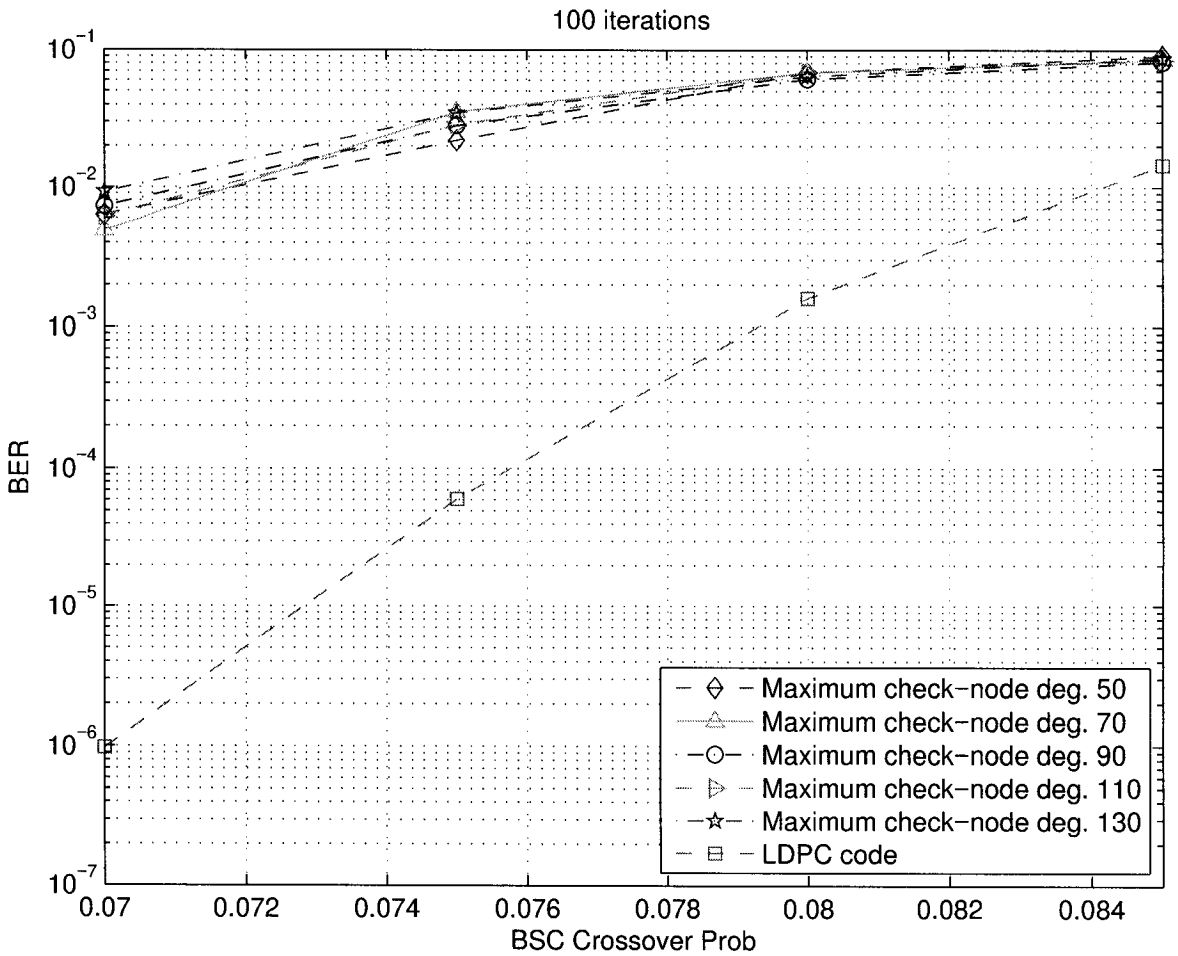

Figure 4.9: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over BSC, with $n = 1268, m = 812, c_1 = 6$.

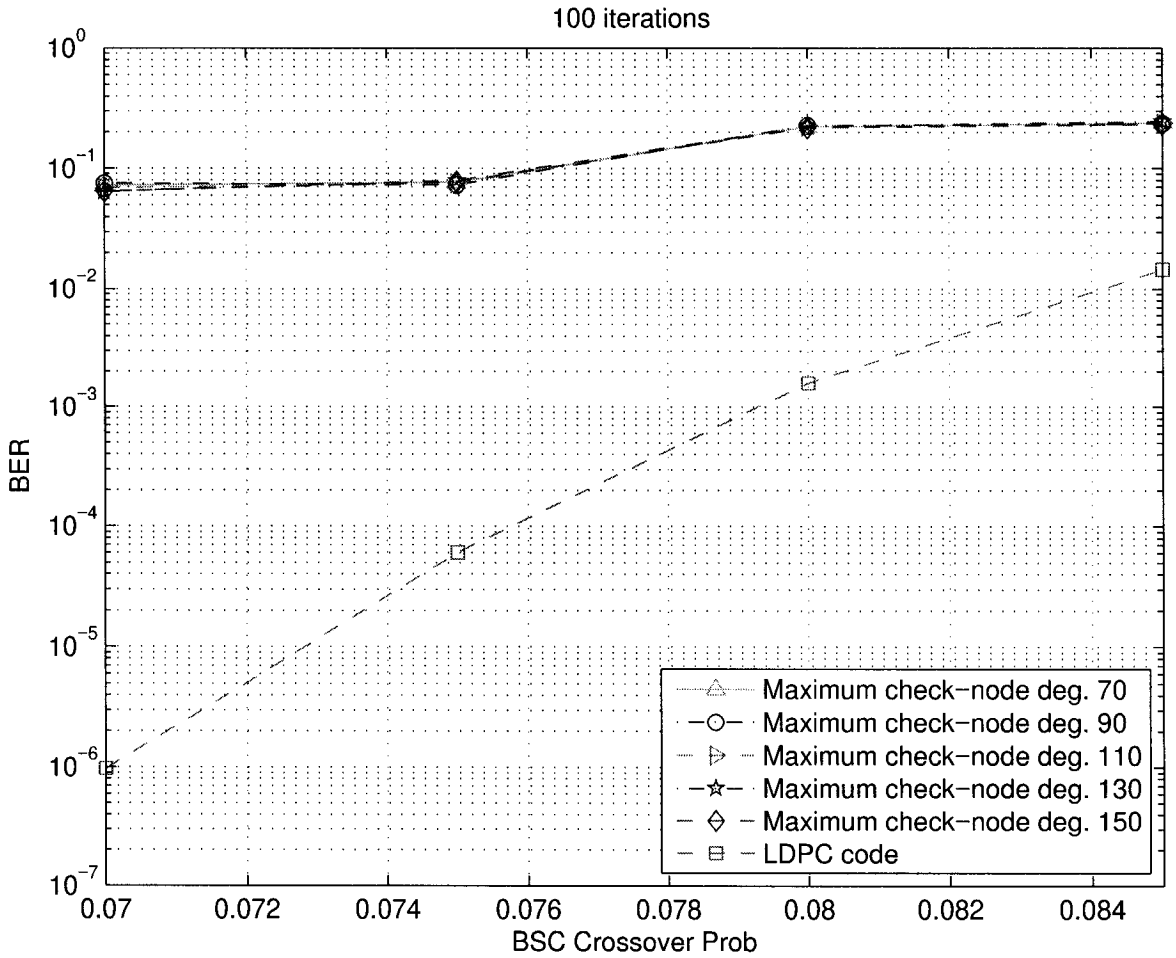

Figure 4.10: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over BSC, with $n = 1268, m = 812, c_1 = 7$.

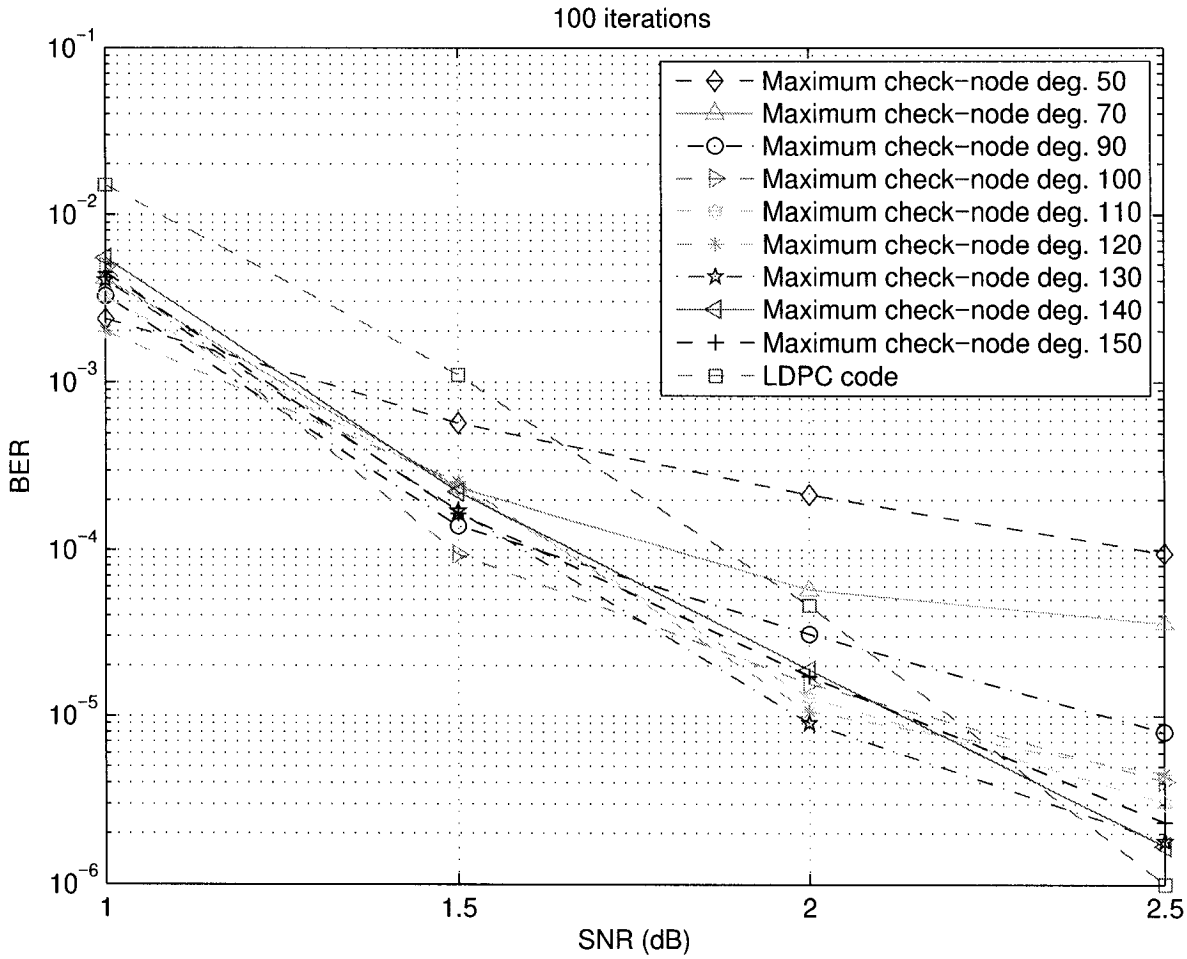

Figure 4.11: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over AWGN channel, with $n = 1268, m = 812, c_1 = 5$.

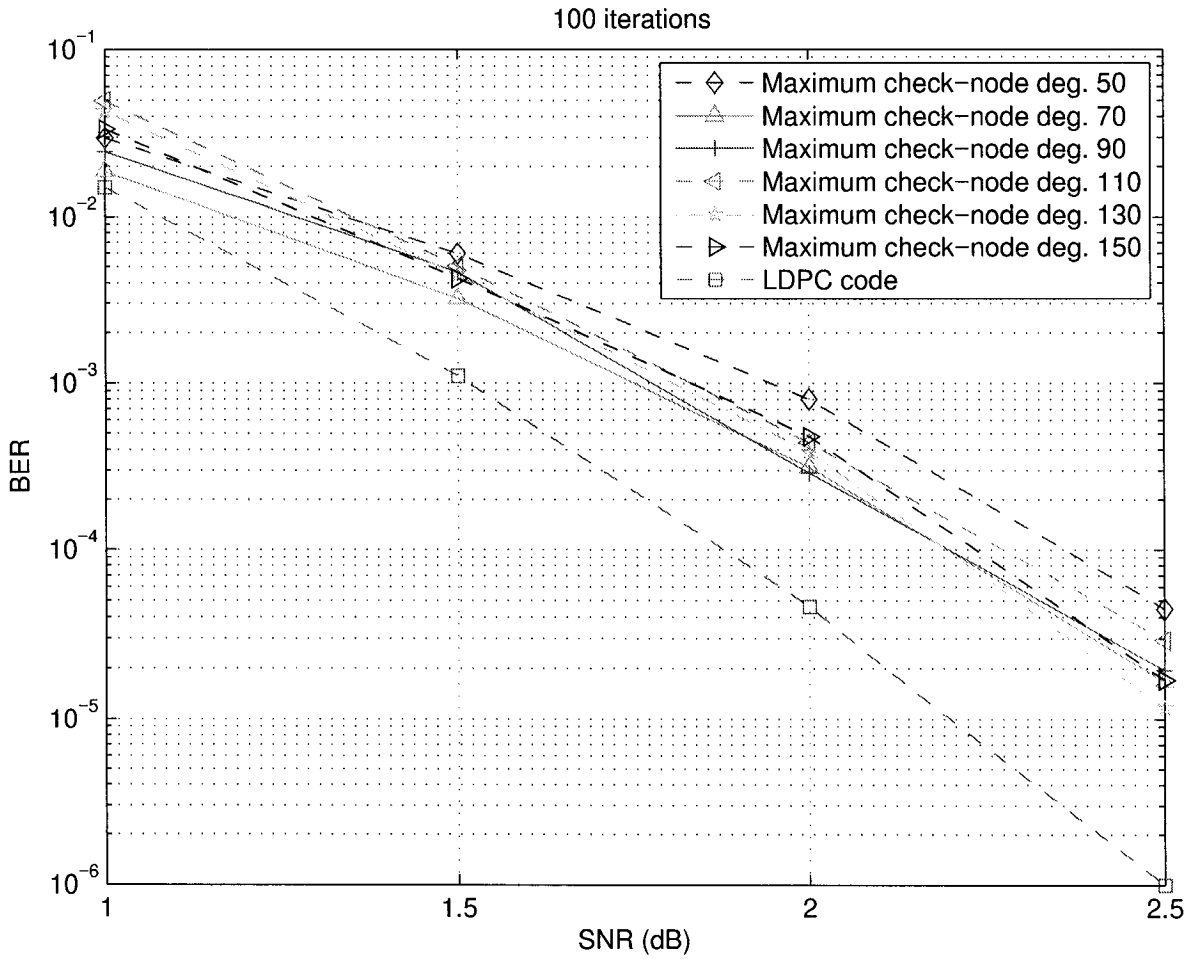

Figure 4.12: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over AWGN channel, with $n = 1268, m = 812, c_1 = 6$.

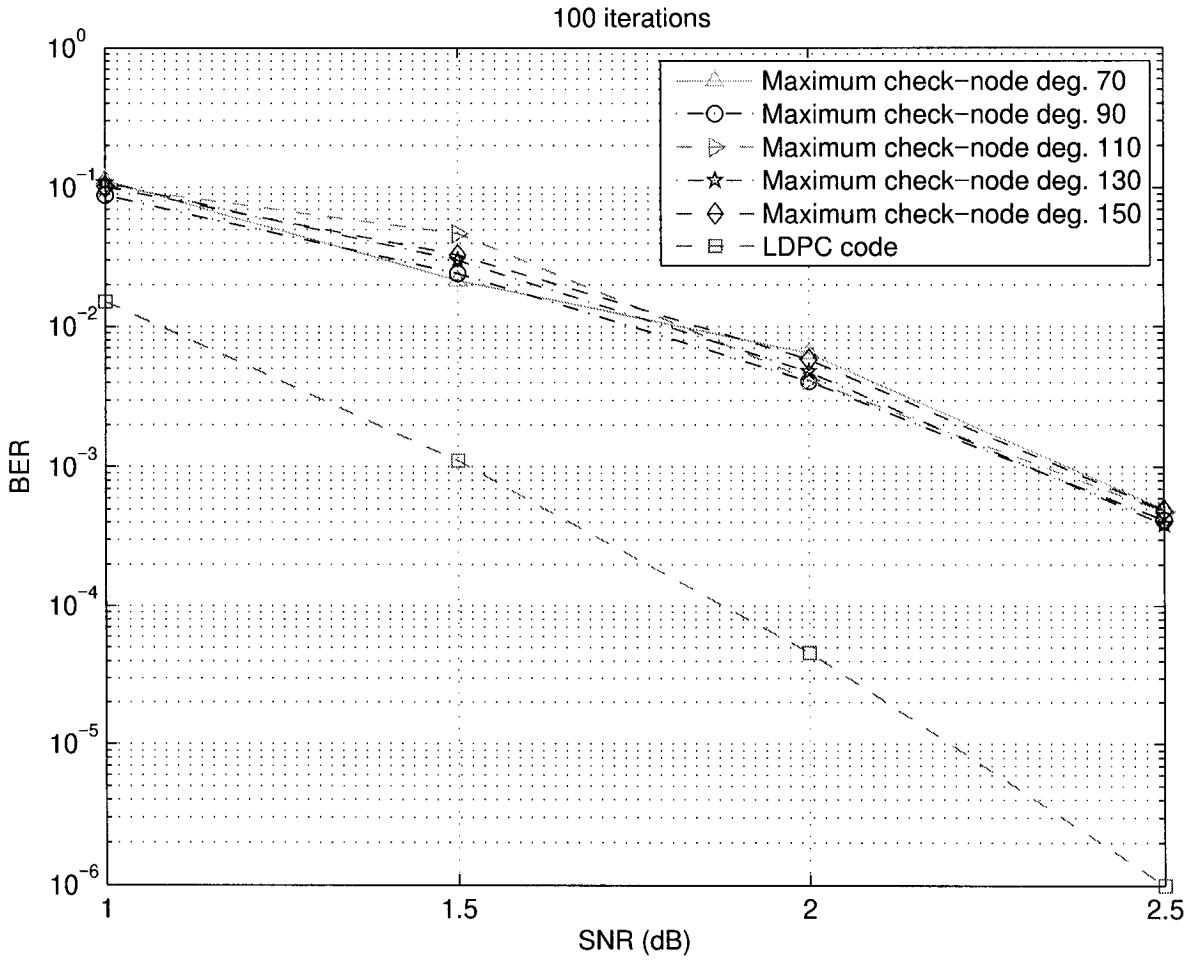

Figure 4.13: Simulation results showing the performance of the short irregular LDGM codes and LDPC code over AWGN channel, with $n = 1268, m = 812, c_1 = 7$.

From these figures, it is easily seen that the construction of LDGM codes using the PEG algorithm can be performed no worse than LDPC codes at short block length. But there are also some slightly variation among the different code rates. We will discuss this issue in the following subsections.

Check-node-degree distribution $\rho(x)$	Ave. symbol deg.	Max. check-node deg.
$0.888889x^5 + 0.111111x^{50}$	9.821428	50
$0.892308x^5 + 0.107692x^{70}$	11.964286	70
$0.894118x^5 + 0.105882x^{90}$	13.769841	90
$0.894737x^5 + 0.105263x^{100}$	14.801587	100
$0.895238x^5 + 0.104762x^{110}$	15.833333	110
$0.895652x^5 + 0.104348x^{120}$	16.865080	120
$0.896000x^5 + 0.104000x^{130}$	17.896826	130
$0.896296x^5 + 0.103704x^{140}$	18.928572	140

Table 4.1: PEG-LDGM Check-node-degree distribution, average symbol node degree and maximum check-node degree with $n = 504, m = 252, c_1 = 5$.

Check-node-degree distribution $\rho(x)$	Ave. symbol deg.	Max. check-node deg.
$0.886364x^6 + 0.113636x^{50}$	10.888889	50
$0.890625x^6 + 0.109375x^{70}$	12.857142	70
$0.892857x^6 + 0.107143x^{90}$	15.00000	90
$0.893617x^6 + 0.106383x^{100}$	15.698413	100
$0.894231x^6 + 0.105769x^{110}$	16.730158	110
$0.894737x^6 + 0.105263x^{120}$	17.761906	120
$0.895161x^6 + 0.104839x^{130}$	18.793652	130
$0.895522x^6 + 0.104478x^{140}$	19.825397	140

Table 4.2: PEG-LDGM Check-node-degree distribution, average symbol node degree and maximum check-node degree with $n = 504, m = 252, c_1 = 6$.

4.2.1 Short irregular $(504, 252)$ LDGM codes over BSC and AWGN channel

It has been shown in Figure 4.2 that the performance of short irregular $(504, 252)$ LDGM code over BSC is much better than that of $(504, 252)$ LDPC code in the waterfall region.

Check-node-degree distribution $\rho(x)$	Ave. symbol deg.	Max. check-node deg.
$0.883721x^7 + 0.116279x^{50}$	9.821428	50
$0.888889x^7 + 0.111111x^{70}$	11.964286	70
$0.891566x^7 + 0.108434x^{90}$	14.801587	90
$0.892473x^7 + 0.107527x^{100}$	15.833333	100
$0.893204x^7 + 0.106796x^{110}$	16.865080	110
$0.893805x^7 + 0.106195x^{120}$	17.896826	120
$0.894309x^7 + 0.105691x^{130}$	18.928572	130
$0.894737x^7 + 0.105263x^{140}$	18.928572	140

Table 4.3: PEG-LDGM Check-node-degree distribution, average symbol node degree and maximum check-node degree with $n = 504, m = 252, c_1 = 7$.

Check-node-degree distribution $\rho(x)$	Ave. symbol deg.	Max. check-node deg.
$0.888889x^5 + 0.111111x^{50}$	9.989035	50
$0.892308x^5 + 0.107692x^{70}$	11.896930	70
$0.894737x^5 + 0.105263x^{90}$	13.936403	90
$0.894737x^5 + 0.105263x^{100}$	14.945175	100
$0.895238x^5 + 0.104762x^{110}$	15.811403	110
$0.895652x^5 + 0.104348x^{120}$	16.973684	120
$0.896000x^5 + 0.104000x^{130}$	17.949562	130
$0.896296x^5 + 0.103704x^{140}$	18.969297	140
$0.896296x^5 + 0.103704x^{150}$	19.714912	150

Table 4.4: PEG-LDGM Check-node-degree distribution, average symbol node degree and maximum check-node degree with $n = 1268, m = 456, c_1 = 5$.

Moreover, the performance of these LDGM codes tends to give high error floors with increasing the maximum check-node degree at lower crossover probability. Among the various settings of c_1, shown from Figure 4.2 to 4.4, irregular $(504, 252)$ LDGM codes having higher value of c_1 lead to worse performance in the waterfall regions.

For $(504, 252)$ LDGM codes over AWGN channel, it is suggested by Figure 4.5 that the performance in the waterfall region is also better than that of LDPC code. More specifically, with 100 iterations and bit-error rate of 10^{-5}, the irregular LDGM code with maximum check-node degree 50 outperforms LDPC code by 0.1 dB, and the code with maximum check-node degree 70 performs close to LDPC code. Table 4.7 gives the performance of irregular $(504, 252)$ LDGM codes in comparison with that of LDPC code

Check-node-degree distribution $\rho(x)$	Ave. symbol deg.	Max. check-node deg.
$0.995970x^6 + 0.004030x^{50}$	10.973684	50
$0.979680x^6 + 0.020320x^{70}$	12.929825	70
$0.971147x^6 + 0.028853x^{90}$	14.921053	90
$0.965896x^6 + 0.034104x^{110}$	16.842106	110
$0.962339x^6 + 0.037661x^{130}$	18.842106	130
$0.959770x^6 + 0.040230x^{150}$	20.789474	150

Table 4.5: PEG-LDGM Check-node-degree distribution, average symbol node degree and maximum check-node degree with $n = 1268, m = 456, c_1 = 6$.

Check-node-degree distribution $\rho(x)$	Ave. symbol deg.	Max. check-node deg.
$0.986316x^7 + 0.013684x^{70}$	13.984649	70
$0.976082x^7 + 0.023918x^{90}$	15.923245	90
$0.969822x^7 + 0.030178x^{110}$	17.885965	110
$0.965597x^7 + 0.034403x^{130}$	19.747807	130
$0.962555x^7 + 0.037445x^{150}$	21.872807	150

Table 4.6: PEG-LDGM Check-node-degree distribution, average symbol node degree and maximum check-node degree with $n = 1268, m = 456, c_1 = 7$.

over BSC and AWGN channel.

c_1	BSC	AWGN channel
5	+ + + 0 0	+ 0 - - - - - -
6	+ + + + +	+ + + + + + + - -
7	- - - - -	0 0 0 0 0 0 0 -

Table 4.7: Irregular $(504, 252)$ LDGM codes comparing with LDPC code summarized from Figure 4.2 to 4.7. Each symbol "+", "-", or "0" corresponds to one simulated LDGM code, symbol "+" indicates that the code performs mostly better than the corresponding LDPC code up to bit-error rate of 10^{-5}, symbol "-" indicates that the code performs mostly worse than the corresponding LDPC code in that bit-error rate range, and symbol "0" indicates that the code performs similarly to the corresponding LDPC code in that bit-error rate range.

Apparently, the performance of these LDGM codes at high *SNR* or low crossover probability presents a different behavior for different maximum check-node degrees. For $c_1 \leq 6$, the performance decreases with maximum check-node degree in the high *SNR* region or low crossover probability region, while for $c_1 > 6$ the performance does not

show significant variation with the maximum check-node degree. On the other hand, for $c_1 \leq 6$, decreasing value of c_1 leads to performance degradation in the waterfall regions. For $c_1 > 6$, however, the performance of irregular $(504, 252)$ LDGM is no better than that of LDPC code. Obviously, the performance over AWGN channel demonstrates similar trend as that over BSC.

In other words, irregular $(504, 252)$ LDGM codes having higher maximum check-node degree result in higher error floor, and the values of c_1 have effect on the performance of irregular $(504, 252)$ LDGM codes in the waterfall regions. Therefore, the optimum combination of irregular $(504, 252)$ LDGM codes depends on the value of c_1 and maximum check-node degree.

4.2.2 Short irregular $(1268, 456)$ LDGM codes over BSC and AWGN channel

It has been shown in Figure 4.8 that the performance of short irregular $(1268, 456)$ LDGM codes over BSC with $c_1 = 5$ gives relatively lower error floors as maximum check-node degree increases. Specifically, irregular $(1268, 456)$ LDGM codes having maximum check-node degree greater than 90 can perform no worse than that of LDPC code. Moreover, the performance in the waterfall region has been improved compared with that of LDPC code. It is also observed that the performance of short irregular LDGM codes with the values of $c_1 = 6$ and 7 perform worse than LDPC code.

In Figure 4.11, for AWGN channel, we have seen that the performance for LDPC codes in the waterfall region has been improved with $c_1 = 5$, in comparison with LDPC code. Particularly, with 100 iterations and bit-error rate of 10^{-5}, irregular LDGM code with maximum check-node degree 130 outperforms LDPC code by 0.2 dB, and the irregular LDGM code with maximum check-node degree 100 outperforms LDPC by 0.06 dB, the codes having maximum symbol degree less than 90 present worse performance than that of LDPC. Table 4.8 presents the performance of irregular $(1268, 456)$ LDGM codes in comparison with that of LDPC code over BSC and AWGN channel. It is obvious that the performance over AWGN channel is consistent with that over BSC.

For Figure 4.12 and 4.13, it is shown that the irregular $(1268, 456)$ LDGM codes with

c_1	BSC	AWGN channel
5	- - 0 0 +	- - - + + + + + +
6	- - - - -	- - - - - -
7	- - - - -	- - - - -

Table 4.8: Irregular (1268, 456) LDGM codes comparing with LDPC code summarized from Figure 4.8 to 4.13. Each symbol "+", "-", or "0" corresponds to one simulated LDGM code, symbol "+" indicates that the code performs mostly better than the corresponding LDPC code up to bit-error rate of 10^{-5}, symbol "-" indicates that the code performs mostly worse than the corresponding LDPC code in that bit-error rate range, and symbol "0" indicates that the code performs similarly to the corresponding LDPC code in that bit-error rate range.

the values of $c_1 = 6$ and 7 could not make better performance than LDPC codes.

4.3 Summary

As observed above, it is easily seen that the performances of the short irregular LDGM codes over BSC and AWGN channel can perform no worse than LDPC codes. For a given value of c_1, the performances of the irregular LDGM (504, 252) codes present lower error floors as the maximum check-node degree decreases. For $c_1 = 5$, on the contrary, the performance of the irregular (1268, 456) LDGM codes suggest that lower maximum check-node degree leads to higher error floors, and the higher maximum check-node degrees leads to no worse performance than LDPC code. For $c_1 > 5$, however, irregular (1268, 456) LDGM codes perform poorly compared to LDPC code.

In a word, the performance of irregular LDGM codes depends on the value of c_1 and the maximum check-node degree.

Chapter 5

Conclusion and Future Work

This chapter discusses the conclusions of this thesis and potential future directions of research.

5.1 Conclusion

In this dissertation, we investigate whether the performance of irregular LDGM codes is comparable to that LDPC codes at short block lengths. We are motived by the fact that the complexity of encoding and decoding of short irregular LDGM codes is much lower comparing to LDPC codes and turbo codes, making them an appealing candidates for delay-sensitive applications.

In this work, we construct short irregular LDGM codes using the PEG algorithm and simulate these codes over BSC and AWGN channels. Simulation results show that some of our constructed irregular LDGM codes perform no worse than the best LDPC codes with the same parametre settings.

During the construction of P matrix for the generator matrices, symbol node degree distribution is not needed since the symbol node degree distribution is made as uniform as possible by the PEG algorithm. For a given value of c_1, two families of LDGM codes with parameter $(1268, 456)$ and $(504, 252)$ exhibit the different behavior as maximum check-node degree increases. More specifically, irregular $(504, 252)$ LDGM codes tend to give higher error floors with increasing maximum check-node degree while irregular $(1268, 456)$

LDGM codes give relatively lower error floors. From our observations, for various settings of c_1, irregular $(504, 252)$ LDGM codes with $c_1 = 6$ and irregular $(1268, 456)$ LDGM with $c_1 = 5$ achieve the best performance, significantly outperforming the corresponding LDPC codes. Irregular $(504, 252)$ LDGM codes with $c_1 = 5$ and 7 perform no worse than the corresponding regular LDPC codes. However, irregular $(1268, 456)$ LDGM codes with $c_1 = 6$ and 7 perform rather worse than their LDPC code counterparts. As is expected, the performance of our codes over BSC demonstrates similar trend to that over AWGN. Although both short irregular LDGM codes can achieve the performance close to their corresponding LDPC codes, they present error floors in the high *SNR* regions and in the low crossover probability regions. We note that, however, the codes with error floors lower than bit-error rate of 10^{-5} are of major concerns in practice as there are upper-layer protocols taking care of them.

Many of our observations match results from existing literature. For example, in parallelly-concatenated LDGM codes presented in [11], the first encoder determines the threshold while the second reduces the error floor. As observed from simulations in this work, the values of c_1 have effects on the performance of short irregular LDGM codes in the waterfall regions. Maximum check-node degree, on the other hand, determines the performance of short irregular LDGM codes in the high *SNR* regions over AWGN channel and in the low crossover probability regions over BSC.

5.2 Future work

As presented in [11], concatenated LDGM codes at long block lengths were designed using degree distribution optimized by discrete density evolution analysis. On the other hand, it was suggested in [3] that the PEG algorithm can generate good regular and irregular LDPC codes at short to moderate length when density evolution techniques are used to optimize the symbol-node degree distributions. In fact, density evolution technique is an efficient means of designing and analyzing of randomly constructed LDPC codes at long block lengths. Although density evolution is currently the best standard technique to determine thresholds and to design irregular LDPC codes, it can be used only under the assumption that the block lengths are asymptotically long. Thus, density evolution

is less effective for designing and analyzing for the purpose of constructing LDGM codes at short block lengths. Therefore, new analytic methods need to be discovered for the design of degree distribution for short LDGM codes.

As observed in [3], the PEG algorithm can reduce error floors to a level comparable to Mackay's codes [16]. However, in some cases, this construction results in a performance loss in the waterfall regions. Moreover, PEG construction only takes as input one degree distribution. Provided that both degree distributions are available, it is then necessary to modify the PEG algorithm for graph optimization based on both distributions.

Related to this work are the use of serially concatenated LDGM codes. Although the complexity of encoding of LDGM codes with serial concatenation is higher than that of the proposed parallel schemes, they still possess complexity advantages over LDPC codes. A future direction is then to adapt the PEG algorithm for constructing serially concatenated LDGM codes.

Finally, to make LDGM codes truly useful in practice, one is encouraged to investigate hardware architecturing and design of the encoder and decoder for LDGM codes at short block lengths.

References

[1] W. Zhong and J. Garcia-Frias, "Approaching shannon performance by iterative decoding of linear codes with low-density generator matrix," *IEEE Commun. Lett.*, vol. 7, pp. 266–268, June 2003.

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

[3] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Trans. Inform. Theory*, vol. 51, pp. 386–398, Jan. 2005.

[4] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, Sept. 1959.

[5] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Inform. Control*, vol. 3, pp. 68–79, March 1960.

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

[7] H. Jin, A. Khandekar, and R. McElience, "Irregular repeat-accumulate codes," in *Proc.2nd Intl. Symp. on Turbo codes and Related Topics, Brest, France*, Sept. 2000.

[8] L. Ping and K. Y. Wu, "Concatenated tree codes:a low complexity, high-performance approach," *IEEE Trans. Inform. Theory*, vol. 47, pp. 791–799, Feb. 2001.

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

[10] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, Jul. and Oct. 1948.

[11] W. Zhong and J. Garcia-Frias, "Approaching the shannon limit through parallel concatenation of regular LDGM codes," *Proc. ISIT'05*, pp. 942–953, September 2005.

[12] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[13] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[14] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, pp. 325–343, Mar. 2000.

[15] G. D. Forney, "On iterative decoding and the two-way algorithm," *Int. Symp. on Turbo Codes and Related Topics, Brest, France*, pp. 12–25, Sept. 1997.

[16] D. J. C. Mackay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *IEE Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

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

[18] J. G. Proakis, *Digital Communications, Fourth Edition.* MCGraw-Hill College, Aug. 2000.

[19] T. Etzion, A. Trachtenbeg, and A. Vardy, "Which codes have cycle-free tanner graphs?" *IEEE Trans. Inform. Theory*, vol. 45, pp. 2173–2181, Sep. 1999.

[20] T. Zhang and K. Parhi, "Joint (3,k)-regular ldpc code and decoder/encoder design," *IEEE Trans. on Signal Processing*, vol. 52, no. 4, pp. 1065–1079, Apr. 2004.

[21] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under messages-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[22] Y. Mao, A. Banihashemi, and M. Landolsi, "Comparison between low-density parity-check codes and turbo product codes for delay and complexity sensitive applications," in *Proc. 20th Biennial Symp. Comm. Kingston, Canada*, May 2000, pp. 151–153.

[23] D. J. C. Mackay and R. M. Neal, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[24] W. Zhong and J. Garcia-Frias, "LDGM codes for channel coding and joint source-channel coding of correlated sources," *EURASIP Journal on Applied Signal Processing*, pp. 942–953, May 2005.