In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

Low-Density Parity-Check Code Optimization for Burst Error Channels at Short Block Lengths

by

Qi Hong, B. Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute For Electrical and Computer Engineering
Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario

August, 2003

© Copyright
Qi Hong, 2003

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88874-6

Our file Notre référence

ISBN: 0-612-88874-6

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

Abstract

Many communication channels such as multipath fading channel exhibit bursty error characteristics. Interleaving techniques are generally employed for such channels to spread out the error bursts, so that an error-correcting code designed for random error channels can be used for bursty channels. Block interleavers, convolutional interleavers and random interleavers are among the most common interleaving techniques. For low-density parity-check (LDPC) codes, the interleaver can be incorporated in the structure of the code as the columns of the parity-check matrix can be arbitrarily permuted without affecting the distance properties of the code. This is in fact one of the advantages of LDPC codes over their fierce competitor turbo codes. In this work, the relationship between the performance of LDPC codes on bursty channels and the Tanner graph (TG) structure of LDPC codes is investigated. Using the TG structure of LDPC codes, we design interleavers (permutations of the columns of the parity-check matrix) that improve the performance of LDPC codes over bursty channels. To design such interleavers, we develop a set of optimization algorithms.

Different optimization criteria are used and in some cases the designed interleavers appear to outperform the existing ones. The proposed design methods for LDPC codes are universal in the sense that they do not depend on burst characteristics of the channel. Correlated Rayleigh fading channels and inter-symbol interference (ISI) channels are examined for the application of the optimized LDPC codes. Computer simulations have been performed on a number of regular and irregular LDPC codes. Simulation results suggest that the error performance of LDPC codes can be greatly improved without introducing additional complexity.

Acknowledgements

First, I would like to sincerely thank my thesis supervisor Dr. Amir. H. Banihashemi for his supervision and encouragement during the course of this work, as well as his financial support. His passion for research and valuable suggestions has made the completion of this thesis an enjoyable process. In addition, I thankfully acknowledge the financial support provided by National Capital Institute of Telecommunications (NCIT) and Ontario Graduate Scholarship in Science and Technology (OGSST), which made it possible for me to finish my Master studies at Carleton University.

Moreover, I am especially grateful to my professors at Carleton University and University of Ottawa for the strong technical background that I gained from the courses they taught. I would like to thank Dr. Ian D. Marsland for the fruitful discussions on turbo equalization and knowledgeable suggestions, as well as many helpful hints on the implementation of the algorithms.

All my friends and colleagues in our research group and laboratory have supported me in providing an intellectually stimulating and enjoyable atmosphere, which made my life at Carleton University easier.

Finally, I would like to thank my parents and parents-in-law for their unconditional love and support. I would like to truthfully thank my wife for her great sacrifices throughout the years of study and research.

Table of Contents

ABSTRACT.....	III
ACKNOWLEDGEMENTS	IV
TABLE OF CONTENTS	V
LIST OF FIGURES.....	VII
LIST OF TABLES.....	X
LIST OF ACRONYMS	XI
CHAPTER 1 - INTRODUCTION	1
1.1 THESIS MOTIVATION.....	2
1.2 THESIS CONTRIBUTION	3
1.3 THESIS ORGANIZATION	5
CHAPTER 2 – LOW-DENSITY PARITY-CHECK (LDPC) CODES	6
2.1 LDPC CODE PRINCIPLES.....	6
2.1.1 <i>Generator matrix and parity check matrix</i>	6
2.1.2 <i>LDPC Codes and Tanner Graph</i>	10
2.2 CONSTRUCTIONS OF LDPC CODES	13
2.2.1 <i>Some Definitions and Notations</i>	13
2.2.2 <i>Different Constructions of LDPC Codes</i>	17
2.3 ENCODING OF LDPC CODES.....	22
2.3.1 <i>Systematic encoding using generator matrix G</i>	22
2.3.2 <i>Linear-time encoding using parity-check matrix H</i>	22
2.4 ITERATIVE DECODING ALGORITHMS	26
2.4.1 <i>Sum-Product (SP) algorithm</i>	28
2.5 LDPC CODE PERFORMANCE ON AWGN CHANNEL.....	32
2.5.1 <i>Regular Codes</i>	33
2.5.2 <i>Irregular Codes</i>	35
2.5.3 <i>Linear-time Encodable Irregular LDPC Codes</i>	37
CHAPTER 3 – LDPC CODE OPTIMIZATION FOR BURST ERROR CHANNELS	39

3.1 PRINCIPLE OF INTERLEAVING	41
3.1.1 Interleaver	41
3.1.2 Interleaving techniques	43
3.2 TG STRUCTURE AND DISTANCE PROFILE.....	48
3.2.1 Cycles and girth.....	48
3.2.2 Distance Profile.....	49
3.3 OPTIMIZATION ALGORITHMS	54
3.3.1 Greedy algorithm 1A.....	55
3.3.2 Greedy algorithm 2A.....	56
3.3.3 Greedy algorithms 1B & 2B.....	57
CHAPTER 4 – LDPC CODE OPTIMIZATION FOR MULTIPATH FADING	
CHANNEL	58
4.1 MULTIPATH FADING CHANNEL	58
4.1.1 Modified Jakes model.....	61
4.2 SUM-PRODUCT ALGORITHM ON RAYLEIGH FADING CHANNEL	62
4.3 SIMULATION RESULTS	64
4.3.1 (3072,1024) irregular LDPC code.....	65
4.3.2 (3072,1024) PEG LE LDPC code	76
4.3.3 (4000,2000) (3,6) regular LDPC code.....	82
4.4 DISCUSSION	88
CHAPTER 5 - LDPC CODE OPTIMIZATION FOR ISI CHANNELS	91
5.1 ISI CHANNEL MODEL.....	93
5.2 MAP (BCJR) ALGORITHM	94
5.3 MESSAGE PASSING BETWEEN THE MAP EQUALIZER AND THE SP DECODER.....	97
5.4 LDPC CODE PERFORMANCE ON ISI CHANNEL	100
5.5 LDPC CODE OPTIMIZATION FOR ISI CHANNEL	105
CHAPTER 6 - CONCLUSIONS	113
REFERENCES	116

List of Figures

<i>Figure 2-1: Parity check matrix of a (3,5) regular LDPC code with block length 10</i>	<i>12</i>
<i>Figure 2-2: Tanner graph of the (3, 5)-regular LDPC code</i>	<i>12</i>
<i>Figure 2-3: An irregular Tanner graph with $D_s=(2, 2, 2, 2, 2, 3, 3, 3)$ and $D_c=(5, 5, 5, 4)$</i>	<i>14</i>
<i>Figure 2-4: The tree spreading from symbol node S_0</i>	<i>17</i>
<i>Figure 2-5: (a) Check node C_0 sending a new message to symbol node S_0 (b) Symbol node S_7 sending back updated message to check node C_0</i>	<i>28</i>
<i>Figure 2-6: (1008,504) regular LDPC code performance on AWGN channel</i>	<i>34</i>
<i>Figure 2-7: (1008,504) irregular/regular LDPC code performance on AWGN channel</i>	<i>37</i>
<i>Figure 2-8: (1008,504) irregular LDPC code performance on AWGN channel</i>	<i>38</i>
<i>Figure 3-1: Block diagram of a communication system with interleaver for bursty channel</i>	<i>40</i>
<i>Figure 3-2: An interleaver</i>	<i>42</i>
<i>Figure 3-3: An interleaver mapping</i>	<i>42</i>
<i>Figure 3-4: A block interleaver</i>	<i>44</i>
<i>Figure 3-5: A convolutional interleaver and deinterleaver</i>	<i>45</i>
<i>Figure 3-6: A convolutional interleaver with $L = 3, B = 2$</i>	<i>45</i>
<i>Figure 3-7: A cyclic shift interleaver</i>	<i>46</i>
<i>Figure 3-8: The tree expanding from symbol node S_0</i>	<i>49</i>
<i>Figure 4-1: (3072,1024) irregular LDPC code performance- No interleaver vs. with interleaver, for different interleavers</i>	<i>69</i>
<i>Figure 4-2: (3072,1024) irregular LDPC code performance- optimized vs. other interleavers</i>	<i>69</i>
<i>Figure 4-3: Distance distribution of (3072,1024) irregular LDPC code (with block interleaver)</i>	<i>70</i>
<i>Figure 4-4: Distance distribution of (3072,1024) irregular LDPC code (with conv. int.)</i>	<i>70</i>
<i>Figure 4-5: Distance distribution of (3072,1024) irregular LDPC code (with rand. int.)</i>	<i>71</i>
<i>Figure 4-6: Distance distribution of (3072,1024) irregular LDPC code (with s-rand. int.)</i>	<i>71</i>

<i>Figure 4-7: Distance distribution of (3072,1024) irregular LDPC code (algo.1A).....</i>	<i>72</i>
<i>Figure 4-8: Distance distribution of (3072,1024) irregular LDPC code (algo.2A).....</i>	<i>72</i>
<i>Figure 4-9: BER curves for (3072,1024) irregular LDPC code- for different channel parameters</i>	<i>73</i>
<i>Figure 4-10: WER curves for (3072,1024) irregular LDPC code- for different channel parameters</i>	<i>73</i>
<i>Figure 4-11: (3072,1024) irregular LDPC code performance without SI- for different interleavers</i>	<i>74</i>
<i>Figure 4-12: (3072,1024) PEG LE code performance on correlated Rayleigh fading channel with SI- for different interleavers</i>	<i>78</i>
<i>Figure 4-13: Distance distribution of (3072,1024) PEG LE LDPC code (algo.1A).....</i>	<i>78</i>
<i>Figure 4-14: Distance distribution of (3072,1024) PEG LE LDPC code (algo.2A).....</i>	<i>79</i>
<i>Figure 4-15: Distance distribution of (3072,1024) PEG LE LDPC code (blk. int.)</i>	<i>79</i>
<i>Figure 4-16: Distance distribution of (3072,1024) PEG LE LDPC code (rand. int.).....</i>	<i>80</i>
<i>Figure 4-17: (3072,1024) PEG LE code performance on correlated Rayleigh fading channel without SI- for different interleavers</i>	<i>80</i>
<i>Figure 4-18: Distance distribution of (4000,2000) regular LDPC code (algo.1A)</i>	<i>84</i>
<i>Figure 4-19: Distance distribution of (4000,2000) regular LDPC code (algo.2A)</i>	<i>84</i>
<i>Figure 4-20: Distance distribution of (4000,2000) regular LDPC code (algo.1B)</i>	<i>85</i>
<i>Figure 4-21: Distance distribution of (4000,2000) regular LDPC code (algo.2B)</i>	<i>85</i>
<i>Figure 4-22: Distance distribution of (4000,2000) regular LDPC code (blk. int.).....</i>	<i>86</i>
<i>Figure 4-23: Distance distribution of (4000,2000) regular LDPC code (rand. int.)</i>	<i>86</i>
<i>Figure 4-24: (4000,2000) regular code performance on correlated Rayleigh fading channel with SI- for different interleavers</i>	<i>87</i>
<i>Figure 4-25: (3072,1024) code performance- optimized interleavers</i>	<i>90</i>
<i>Figure 5-1: System Block Diagram</i>	<i>92</i>
<i>Figure 5-2: Discrete time channel model of ISI with AWGN</i>	<i>94</i>
<i>Figure 5-3: A trellis section and its associated factor graph</i>	<i>95</i>
<i>Figure 5-4: Message passing between the MAP equalizer and the SP decoder</i>	<i>98</i>
<i>Figure 5-5: ISI channel models</i>	<i>101</i>
<i>Figure 5-6: (1920,640) regular LDPC code performance on Proakis B ISI channel....</i>	<i>104</i>

<i>Figure 5-7: (1400,800) irregular LDPC code performance on 2-tap ISI channel.....</i>	<i>104</i>
<i>Figure 5-8: (504,252) PEG LDPC code performance on Proakis B ISI channel</i>	<i>105</i>
<i>Figure 5-9: (1024,512) PEG LDPC code performance on Proakis B ISI channel- for different channel interleavers</i>	<i>107</i>
<i>Figure 5-10: Distance distribution of (1024,512) PEG LDPC code (algo.1A).....</i>	<i>108</i>
<i>Figure 5-11: Distance distribution of (1024,512) PEG LDPC code (algo.2A).....</i>	<i>108</i>
<i>Figure 5-12: Distance distribution of (1024,512) PEG LDPC code (blk. int.)</i>	<i>109</i>
<i>Figure 5-13: Distance distribution of (1024,512) PEG LDPC code (conv. int.)</i>	<i>109</i>
<i>Figure 5-14: Distance distribution of (1024,512) PEG LDPC code (rand. int.)</i>	<i>110</i>
<i>Figure 5-15: Distance distribution of (1024,512) PEG LDPC code (s-rand. int.)</i>	<i>110</i>

List of Tables

<i>Table 4-1: Statistics of the number of decoding iterations for (3072,1024) irregular code with SI (converged case).....</i>	<i>74</i>
<i>Table 4-2: Breakdown of error events for (3072,1024) irregular code with SI.....</i>	<i>75</i>
<i>Table 4-3: Statistics of the number of decoding iterations for (3072,1024) PEG LE code with SI (converged case).....</i>	<i>81</i>
<i>Table 4-4: Breakdown of error events for (3072,1024) PEG LE code with SI</i>	<i>82</i>
<i>Table 4-5: Statistics of the number of decoding iterations for (4000,2000) regular code with SI (converged case).....</i>	<i>87</i>
<i>Table 5-1: Statistics of the number of channel iterations for (1024,512) PEG code (converged case)</i>	<i>111</i>
<i>Table 5-2: Statistics of the number of decoding iterations for (1024,512) PEG code (converged case)</i>	<i>112</i>

List of Acronyms

<i>APP</i>	A Posteriori Probability
<i>AWGN</i>	Additive White Gaussian Noise
<i>BEC</i>	Binary Erasure Channel
<i>BER</i>	Bit Error Rate
<i>BF</i>	Bit Flipping
<i>BPSK</i>	Binary Phase Shift Keying
<i>BSC</i>	Binary Symmetric Channel
<i>DFE</i>	Decision Feedback Equalizer
<i>ECC</i>	Error Correction Code
<i>ISI</i>	Inter-Symbol Interference
<i>LDPC</i>	Low-Density Parity-Check
<i>LE</i>	Linear Equalizer/Linear-time Encoding
<i>LLR</i>	Log-Likelihood Ratio
<i>MAP</i>	Maximum A Posteriori
<i>MLSE</i>	Maximum-Likelihood Sequence Estimation
<i>MS</i>	Min-Sum
<i>PEG</i>	Progressive-Edge Growth
<i>SI</i>	Side Information
<i>SNR</i>	Signal-to-Noise Ratio
<i>SP</i>	Sum-Product
<i>TG</i>	Tanner Graph
<i>WER</i>	Word Error Rate

Chapter 1 - Introduction

Low-Density Parity-Check (LDPC) codes were first introduced by Gallager in the early 1960's [1]-[2]. They were however, almost completely forgotten after their invention due to the limited computational capabilities of the computers at that time which made the associated iterative decoding algorithms look impractical. Following the invention of turbo codes [3], LDPC codes were rediscovered in the late 1990's [4]-[5]. It has been shown that these codes achieve a remarkable performance with iterative decoding algorithms that is very close to the Shannon limit [4], and in particular, irregular LDPC codes have been shown to perform better than turbo codes [6]. Their near Shannon limit error performance and low decoding complexity have attracted a great amount of research (see, e.g. [7],[8]). In particular, for the past few years, there have been a large number of articles dealing with the LDPC code error performance on memoryless channels, i.e. additive white Gaussian noise (AWGN) channel, binary symmetric channel (BSC), and binary erasure channel (BEC). For a list of references, please see the related papers in [9] and the references therein.

In this thesis, we study the LDPC code performance on the channels with memory, in particular, burst error channels. By exploring the structure of underlying Tanner graph (TG) [10] defined by the parity-check matrix of LDPC codes, we introduce a set of optimization algorithms to improve the LDPC code performance on burst error channels. We show by computer simulation that optimized LDPC codes have, in general, a better error performance compared to non-optimized ones, and a similar or better error per-

formance compared to those incorporating existing interleavers, such as random interleaver, in their structure.

1.1 Thesis Motivation

Many communication channels exhibit burst error characteristics. One example is a mobile radio channel, where the transmission errors can be caused by variations in received signal strength referred to as fading.

Also many practical communication systems such as mobile communication systems encounter the problem of data transmission over a channel with inter-symbol interference (ISI), where the transmitted signal experiences non-trivial impairment and the received sample depends not only on the transmitted signal, but also on the delayed versions of the transmitted signal. It is well known that ISI channel can be modeled as a finite-state Markov source. As pointed out in [11], Markov sources (e.g. ISI channels), when corrupted with memoryless noise and decoded by a symbol or sequence maximum *a posteriori* (MAP) detector, generate burst errors.

Interleaving is an effective method to cope with burst errors and has been widely used in conjunction with error control coding for bursty channels. An interleaver is inserted between the channel encoder and the channel and the deinterleaver is located between the output of the channel and the decoder. The interleaver and deinterleaver spread out the error bursts in time so that errors within a codeword appear to be random, thus the burst error channel is transformed into a random error channel and a code designed for independent error channels can be used for burst error channels.

In [12], it is pointed out that there is a built-in “interleaver” in the parity-check matrix H of LDPC codes, and in principle, it is possible to design this “interleaver”, i.e. to de-

sign the matrix H such that all the nonzero entries in H are distributed so as to achieve the maximal “interleaver” gain. This observation raises one interesting question: can we modify a given parity-check matrix H to maximize the interleaving gain?

Reference [68] also states: “In theory, since a random interleaver is equivalent to a permutation matrix, it can be included in the design of the parity-check matrix; in practice, optimization of the parity-check matrix for ISI channel is difficult.” We may ask the similar question: can we optimize the parity-check matrix for ISI channel?

Iterative decoding algorithms can be naturally described by the underlying TG which is constructed based on the parity-check matrix H of the LDPC code. For a given TG, iterative decoding algorithms operate by passing messages (or information) between two types of nodes in the graph: symbol nodes and check nodes. Intuitively, by exploring the TG structure, we should be able to achieve improved error performance. Motivated by this intuition and the above questions, we develop a set of optimization algorithms to improve the LDPC code performance on burst error channels.

1.2 Thesis Contribution

The major contribution of the research presented in this thesis is to investigate and compare the performance of different interleavers on burst error channels in conjunction with LDPC codes. We also devise built-in interleavers for a given parity check matrix H which achieve similar performance over existing interleavers. To do this, we develop a set of optimization algorithms that achieve improved system performance without increasing the decoding complexity. The proposed optimization algorithms are designed based on the structural properties of TG, and different optimization measures have been considered in this work.

In general, short cycles (please see section 2.2.1 for the definition of cycle) have a negative impact on the iterative decoding algorithms for LDPC codes [14]. To make sure that there are not too many symbol nodes with low reliability in the same short cycle, we introduce a new concept, *distance profile*, as a measure to design the built-in interleaver for LDPC codes. Two types of optimization algorithms are then developed: random optimization and deterministic optimization.

Using the argument that symbol nodes with high degrees have better protection from their connected check nodes, we design the optimized interleavers based on the symbol node degree distribution (please see section 2.1.2 for the description of degree distribution). To make sure that there are not too many high-degree symbol nodes to be in error or to be unreliable at the same time, we distribute the high-degree symbol nodes uniformly among the low-degree symbol nodes, thus the distance between high-degree symbol nodes is maximized, and we would expect better performance intuitively.

Two types of burst error channels are considered in this thesis, namely, frequency flat, time-correlated Rayleigh fading channels, and inter-symbol interference (ISI) channels. In fact, the design methodology presented in this research is universal for bursty channels in the sense that it can readily be applied to other burst error channels such as Gilbert-Elliott channel [15]-[16], frequency selective fading channel, etc.

We show by computer simulation that the optimized interleavers achieve great improvement in error performance over no interleaver case, and provide almost the same gain over existing interleaving techniques. Our observations indicate that different interleavers, including the ones designed in this thesis, provide more or less the same amount of improvement. By simulating a large number of LDPC codes, we observed that optimi-

zation algorithms and existing interleaving techniques do provide some “interleaving gain” for LDPC codes on bursty channels. The result of this work shows the importance of interleaving techniques in improving the performance of LDPC codes with short and medium block length on burst error channels. The results of this work also suggest that there may exist better performance measures for the design of built-in interleavers for LDPC codes.

1.3 Thesis Organization

The contents of this thesis are organized as follows:

Chapter 2 provides a general review of LDPC codes. Different methods of constructing LDPC codes together with encoding and decoding are examined. LDPC code performance on AWGN channel is then presented.

In chapter 3, the concept of distance profile is introduced and four optimization algorithms for LDPC codes on burst error channels are developed based on distance profile.

Chapter 4 describes multipath fading channel, in particular, correlated Rayleigh fading channel and the decoding algorithm for such a channel. Simulation results for the application of optimization algorithms to correlated Rayleigh fading channel are presented along with related discussion and analysis.

In chapter 5, we describe ISI channel model and the message passing between the equalizer and the decoder. LDPC code performance on ISI channels is presented followed by the simulation results for the application of optimization algorithms to ISI channels.

Finally, chapter 6 is devoted to the conclusions and suggestions for future research.

Chapter 2 – Low-Density Parity-Check (LDPC) Codes

Error control codes introduce controlled amounts of redundancy into a transmitted data stream, providing the receiver with the ability to detect and possibly correct errors caused by the noise on the communication channel. A binary low-density parity-check (LDPC) code is a binary linear block code defined by a sparse parity check matrix H . In this chapter, we first review the useful representations of linear block codes: generator matrix and parity check matrix, then we describe the graphical representation of LDPC codes: Tanner Graph (TG) [10]. Next we describe different methods of constructing LDPC codes and the encoding complexity of LDPC codes. We also explain iterative decoding algorithms as efficient algorithms for decoding LDPC codes. Finally, the performance of LDPC codes constructed with different constructions is compared on memoryless AWGN channel.

2.1 LDPC code principles

2.1.1 Generator matrix and parity check matrix

An (n, k) linear block code C can be generated by a set of k linearly independent binary n -tuples $g_0, g_1, \dots, g_{k-1}$. The codewords are obtained by linearly combining these n -tuples. Consequently, the codeword for a message $\mathbf{m} = (m_0, m_1, \dots, m_{k-1})$ can be represented as

$$\mathbf{v} = m_0 \mathbf{g}_0 + m_1 \mathbf{g}_1 + \dots + m_{k-1} \mathbf{g}_{k-1}. \quad (2.1)$$

The *generator matrix* G of code C is constructed by taking the vectors $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$ as rows of a $k \times n$ matrix as follows

$$G = \begin{pmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{pmatrix} = \begin{pmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{pmatrix}. \quad (2.2)$$

The corresponding codeword $\mathbf{v}$ of the message $\mathbf{m}$ can therefore be written as

$$\mathbf{v} = \mathbf{m}G. \quad (2.3)$$

Note that any k linearly independent codewords can form a generator matrix and thus the generator matrix of a linear block code is not unique.

Example 2.1

The set of 4 linearly independent 7-tuples

$$\mathbf{g}_0 = (1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0), \mathbf{g}_1 = (0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0), \mathbf{g}_2 = (1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0), \text{ and } \mathbf{g}_3 = (1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1),$$

generates a $(7, 4)$ linear block code with the following generator matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The dual space of a linear block code C is called the dual code of C and is denoted by $C^\perp$. We know that $C^\perp$ is a vector space of dimension $(n-k)$. Let $\{\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{n-k-1}\}$ be a basis for $C^\perp$, i.e., every codeword of $C^\perp$ can be written as a linear combination of $\{\mathbf{h}_0, \mathbf{h}_1,$

$\dots, h_{n-k-1}\}$. The *parity check matrix* H is constructed by taking these $(n-k)$ linearly independent vectors as rows and arranged as in the following $(n-k) \times n$ matrix:

$$H = \begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{n-k-1} \end{pmatrix} = \begin{pmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1} \end{pmatrix}, \quad (2.4)$$

The parity check matrix H has the property that any vector in row space of G is orthogonal to any row of H and any vector that is orthogonal to all the rows of H is in row space of G . Thus we have the following parity check theorem:

Theorem 2.1 – The Parity Check Theorem

A vector v is a codeword in C if and only if it satisfies that following equation:

$$vH^T = 0. \quad (2.5)$$

The parity check matrix offers a convenient means for determining the minimum distance of the code as explained in the following theorem:

Theorem 2.2

Let C have parity check matrix H . The minimum distance of C is equal to the minimum nonzero number of columns in H for which a nontrivial linear combination sums to zero.

Now we will turn to the systematic generator matrix G and systematic parity check matrix H , which will greatly simplify the encoding at the transmitter and error detection at the receiver. We will use these results when we examine the encoding process of LDPC codes in section 2.3.

Consider a linear block code C with generator matrix G . Using Gaussian elimination and column reordering, it is always possible to obtain a generator matrix with the following form

$$G_{k \times n} = (P_{k \times (n-k)} | I_k), \quad (2.6)$$

where I_k is the $k \times k$ identity matrix and P is a $k \times (n-k)$ binary matrix of the form

$$P = \begin{pmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{pmatrix}. \quad (2.7)$$

We can easily see that when a data block is encoded using a systematic generator matrix G , it is embedded without any modification in the last k coordinates of the resulting codeword:

$$\begin{aligned} \mathbf{v} &= \mathbf{m} G \\ &= (m_0 \ m_1 \ \cdots \ m_{k-1}) (P | I) \\ &= (v_0 \ v_1 \ \cdots \ v_{n-k-1} | m_0 \ m_1 \ \cdots \ m_{k-1}). \end{aligned} \quad (2.8)$$

Also, after the decoding, the last k bits of the estimated codeword are passed along to the data sink. Notice that the generator matrix given in Example 2.1 is in systematic form.

Given the generator matrix of a code is in the systematic form of (2.6), the parity-check matrix can be easily obtained as follows:

$$H_{(n-k) \times n} = (I_{n-k} | P^T), \quad (2.9)$$

where P^T denotes the transpose of the matrix P .

Example 2.2

Here is the parity-check matrix of the code given in Example 2.1

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

It can easily be verified that $GH^T = 0$, which implies that the row spaces of generator and parity-check matrices are orthogonal.

We point out here that performing Gaussian elimination on a generator matrix does not alter the codeword set for the associated code. Column reordering, on the other hand, may generate codewords that are not in the original code. If a given application requires that a particular codeword set be used and thus not allow for column reordering, it is always possible to use some set of coordinates, which may not be the last k for the message positions. Unfortunately this will complicate encoder design for LDPC codes as explained in section 2.3.

2.1.2 LDPC Codes and Tanner Graph

An LDPC code C [1]-[2] is a linear block code and therefore can be fully described by its parity-check matrix H , through the parity-check equations $\mathbf{v}H^T = \mathbf{0}$, $\mathbf{v} \in C$. However, parity-check matrices of LDPC codes are sparse; that is, they consist mainly of 0's and a small number of 1's. For a code C with parameters (n, k) , H has typically n columns and $n-k$ rows, where n and k being the block length and the dimension of the code, respectively. In parity-check matrix H , the number of nonzero elements per column (row) is referred as the column (row) weight.

Definition 2.1.2a An LDPC code is called *regular* if its parity check matrix H has the same weight (number of 1's) per row and the same weight per column. Otherwise, it is an *irregular* LDPC code.

If H has the same weight per column (row) and the weight per row (column) is as uniform as possible, we will still call it *regular*, or we may sometimes call it *semi-regular*.

We use a two tuple (d_s, d_c) to represent a regular LDPC code whose column weight is d_s and row weight is d_c . For an (d_s, d_c) -regular LDPC code C of length n , it has a parity-check matrix H with n columns of weight d_s and $m = \frac{nd_s}{d_c}$ rows of weight d_c .

LDPC codes, as any other linear block code, can be represented by Tanner graphs (TG) [10] which is constructed based on its parity check matrix H .

Definition 2.1.2b A Tanner graph (TG) is a bipartite graph that contains two types of nodes, *symbol nodes* (*bit nodes*) and *check nodes*, corresponding to the columns and the rows of the parity check matrix H , respectively. A '1' located at position (i, j) of H corresponds to an edge between symbol node s_j and check node c_i .

From the TG, the number of edges connected to any symbol node or check node is referred to as symbol or check node degree, respectively. For a (d_s, d_c) -regular LDPC code of length n , the Tanner graph has $nd_s = md_c$ edges, each symbol node has degree d_s and each check node has degree d_c .

As an example, Figure 2-1 shows the parity-check matrix for a (3, 5)-regular LDPC code of length 10.

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Figure 2-1: Parity check matrix of a (3,5) regular LDPC code with block length 10

The Tanner graph corresponding to the above parity-check matrix is shown in Figure 2-2. In this example, every symbol node (represented by circles) has degree 3 and every check node (represented by squares) has degree 5.

Figure 2-2: Tanner graph of the (3, 5)-regular LDPC code

As we can see from the above example, there is a one-to-one correspondence between a parity check matrix and a TG. We thus use the terms TG and code interchangeably throughout the thesis if no confusion arises.

The original Gallager's codes introduced in [1]-[2] are regular LDPC codes. Irregular LDPC codes were introduced in [17], and were further studied in [6],[18]-[20]. It has been shown in [19] that irregular LDPC codes usually have better performance than regu-

lar LDPC codes. It has also been observed that irregular LDPC codes with long block length even outperform the best known turbo codes [6]. An irregular LDPC code is characterized by a degree distribution pair (λ, ρ) or its corresponding generating functions. We can use two forms of degree distribution: edge-based or node-based. Edge-based degree distribution is presented below and we will discuss node-based degree distribution in next section. Suppose we have an irregular LDPC code with maximum symbol node degree $d_s^{\max}$ and maximum check node degree $d_c^{\max}$, the symbol (check) node degree distribution [6] is specified by the following polynomial:

$$\lambda(x) = \sum_{i=1}^{d_s^{\max}} \lambda_i x^{i-1} \quad (\rho(x) = \sum_{i=2}^{d_c^{\max}} \rho_i x^{i-1}) \quad (2.10)$$

where $\lambda_i(\rho_i)$ is the fraction of edges emanating from symbol (check) nodes of degree i .

(Note: $\sum_i \lambda_i = \sum_i \rho_i = 1$).

Later we will see that *iterative decoding algorithms* for LDPC codes can naturally be described using the Tanner graph representation of these codes.

2.2 Constructions of LDPC codes

In this section we first introduce some definitions and notations on the graph. In this part we largely follow the notations and definitions in [22]. Then different ways of constructing LDPC codes are explained.

2.2.1 Some Definitions and Notations

Suppose that an LDPC code is defined by a sparse parity check matrix H with dimensions $m \times n$. The corresponding TG has m check nodes and n symbol nodes. Formally, a Tanner graph is defined as $G = (V, E)$, Where $V = V_c \cup V_s$ is the set of vertices with $V_c =$

$\{c_0, c_1, \dots, c_{m-1}\}$ being the set of check nodes and $V_s = \{s_0, s_1, \dots, s_{n-1}\}$ the set of symbol nodes. Set E is the set of edges in $V_c \times V_s$, with edge $(c_i, s_j) \in E$ if and only if $h_{i,j} \neq 0$, $0 \leq i \leq m-1$, $0 \leq j \leq n-1$, where $h_{i,j}$ is the element in the i th row and j th column of H . In a (d_s, d_c) -regular Tanner graph every symbol node is connected to d_s check nodes and every check node is connected to d_c symbol nodes. Suppose that $D_s = (d_{s_0}, d_{s_1}, \dots, d_{s_{n-1}})$ is the symbol degree sequence in which d_{s_j} is the degree of symbol node s_j , $0 \leq j \leq n-1$, in non-decreasing order, i.e., $d_{s_0} \leq d_{s_1} \leq \dots \leq d_{s_{n-1}}$, and $D_c = (d_{c_0}, d_{c_1}, \dots, d_{c_{m-1}})$ is the check degree sequence in which d_{c_i} is the degree of parity-check node c_i , $0 \leq i \leq m-1$, and $d_{c_0} \leq d_{c_1} \leq \dots \leq d_{c_{m-1}}$. Also the set of edges E can be partitioned in terms of V_s as $E = E_{s_0} \cup E_{s_1} \cup \dots \cup E_{s_{n-1}}$, in which E_{s_j} , $0 \leq j \leq n-1$, contains all edges that are connected to symbol node s_j . Further, denote the k -th edge connected to s_j by $E_{s_j}^k$, $0 \leq k \leq d_{s_j}-1$. Figure 2-3 shows an example of an irregular Tanner graph with $D_s = (2, 2, 2, 2, 2, 3, 3, 3)$ and $D_c = (5, 5, 5, 4)$.

Figure 2-3: An irregular Tanner graph with $D_s = (2, 2, 2, 2, 2, 3, 3, 3)$ and $D_c = (5, 5, 5, 4)$

In general, an ensemble of Tanner graphs, and correspondingly an ensemble of LDPC codes, is determined by its degree distribution pairs; the degree distribution of symbol

nodes is defined as $\lambda(x) = \sum_{i \geq 1} \lambda_i x^i$, where λ_i is the fraction of symbol nodes with degree

i , d_s^{max} is the largest entry in $D_s = (d_{s_0}, d_{s_1}, \dots, d_{s_{n-1}})$ and $\sum_{i \geq 1} \lambda_i = 1$. Similarly the degree

distribution of parity-check nodes is defined as $\rho(x) = \sum_{i \geq 2} \rho_i x^i$, where ρ_i is the fraction

of parity-check nodes with degree i , d_c^{max} is the largest entry in $D_c = (d_{c_0}, d_{c_1}, \dots, d_{c_{m-1}})$,

and $\sum_{i \geq 2} \rho_i = 1$.

Two important closely related graph properties are *cycle* and *girth*, we will give the definitions here and discuss more about them in the next chapter.

Definition 2.2.1a A *cycle* in a TG is defined as a sequence of connected edges which starts and ends at the same node (symbol or check) and satisfies the condition that no node appears more than once, except for the initial node.

Definition 2.2.1b The *girth* g of a TG is defined as the length of the shortest cycle in this graph. For each symbol node s_j , we define the *local girth* g_{s_j} as the length of the shortest cycle passing through that symbol node.

It follows, by definition, $g = \min_j \{g_{s_j}\}$.

Definition 2.2.1c The *girth distribution* of a TG is defined as the function $g(l)$, $l = 4, 6, 8, \dots, l_{max}$, which gives the fraction of the symbol nodes with girth l , where l_{max} is the maximum girth in the graph [14].

Definition 2.2.1d In a Tanner graph, the *neighborhood within depth l* of a given symbol node s_j , $\mathcal{N}_{s_j}^l$, is defined as the set of all check nodes that can be reached by traversing at most $2l+1$ edges, on a tree spreading from symbol node s_j . The complementary set of $\mathcal{N}_{s_j}^l$, $\overline{\mathcal{N}_{s_j}^l}$, is defined as $V_c \setminus \mathcal{N}_{s_j}^l$, i.e. $\mathcal{N}_{s_j}^l \cup \overline{\mathcal{N}_{s_j}^l} = V_c$ and $\mathcal{N}_{s_j}^l \cap \overline{\mathcal{N}_{s_j}^l} = \emptyset$. Similarly the neighborhood within depth l of a given check node c_i , $\mathcal{N}_{c_i}^l$, is defined as the set of all parity-check nodes that can be reached by traversing at most $2l$ edges, on a tree spreading from c_i .

Figure 2-4 shows the tree spreading from symbol node s_0 by unfolding the Tanner graph shown in Figure 2-3 in a breadth-first way.

Figure 2-4: The tree spreading from symbol node S_0

2.2.2 Different Constructions of LDPC Codes

There are different methods which can be used for constructing LDPC codes. These construction methods can be divided into two main categories: algebraic methods and random methods. The algebraic category includes systematic methods such as *finite geometry* based constructions [23] and *group-theoretic* based constructions [24], [25]. For

example, the finite geometry based constructions allow us to construct LDPC codes, with relatively good minimum distances, based on the lines and points of a finite geometry. However, applications of algebraic constructed codes are usually limited to specified code rates and block sizes, which is not the case for random construction methods. In this subsection we first explain a well-known random construction and its four variations. Then we will describe a new general method which can be used to generate LDPC codes with very good performances at short block length [21], [22]. The version of algorithm that we describe here is completely deterministic. However, as it will be described, the method also has a random version which is slightly different from the deterministic version.

In random construction [20], we define an LDPC code in two steps. First a *profile* that describes the desired number of columns of each weight and the desired number of rows of each weight is selected. As it was explained, each nonzero entry in the parity-check matrix corresponds to an edge connecting a symbol node to a check node of the Tanner graph. Thus, this profile specifies the degrees of vertices (nodes) in this graph.

Second we choose a *construction method*, i.e. a pseudo-random algorithm for putting edges between the vertices of Tanner graph in a way that satisfies the constraints.

Poisson Construction- In this method we choose the edges of Tanner graph completely at random, subject to the profile constraints. Obviously, we can not place two edges between one pair of vertices, since this would correspond to a double entry in the parity-check matrix. One way for implementation of a Poisson construction is to make a list of all columns, containing each column with multiplicity equal to its degree and a

similar list of all rows, containing each row with a multiplicity equal to its degree and then use a random permutation to have a one-to-one map from one list to the other.

There are four variations of this construction [4], which are summarized as follows:

Construction 1A: An $m \times n$ matrix is created at random with weight per column t (e.g. $t = 3$), and weight per row as uniform as possible, and overlap between any two columns no greater than 1.

Construction 2A: Up to $m/2$ of the columns are designated weight-2 columns, and these are constructed such that there is zero overlap between any pair of columns. The remaining columns are made at random with weight-3, with the weight per row as uniform as possible, and overlap between any two columns of the entire matrix no greater than 1.

Construction 1B and 2B: A small number of columns are deleted from a matrix produced by construction 1A and 2A, respectively, so that the bipartite graph corresponding to the matrix has no short cycles of length less than some length l .

Progressive Edge-Growth (PEG) construction is a new general method proposed in [21] and [22] for constructing Tanner graphs with large girth by progressively establishing connections between symbol and check nodes in an edge-by-edge manner.

Progressive Edge-Growth (PEG) Construction [21], [22] - It is known that the sum-product (SP) algorithm over cycle-free graph provides optimum *a posteriori* probability (APP) [10], hence it is natural to try to minimize the influence of the cycles on the iterative decoding process. One way for achieving this goal is to construct LDPC codes that have larger girth distribution, or in other words, have more symbol nodes with larger girths [14]. In random construction methods as shown above, the Tanner graph is ran-

domly constructed, usually by avoiding cycles of length 4. Although random graphs have been used to construct LDPC codes with impressive performance [5], [26], there is no guarantee that at short block lengths any given random graph defines a good code having a suitable girth distribution that facilitates iterative decoding. An efficient method for constructing Tanner graphs with a large average girth, in a greedy sense, is PEG construction, in which the edges are progressively established between symbol and check nodes of the Tanner graph in an edge-by-edge manner [21], [22].

Given the number of symbol nodes n , the number of check nodes m , and the symbol node degree sequence of the graph, an edge-selection procedure is started such that the placement of a new edge has an impact as small as possible on the graph. After a best-effort edge has been determined, the graph with this new edge is updated, and the procedure continues with the placement of the next edge. For the sake of brevity, we summarize the deterministic version of the algorithm as follows:

Deterministic PEG Algorithm:

For $j = 0$ to $n-1$ **do**

begin

for $k = 0$ to $d_{s_j} - 1$ **do**

begin

if $k = 0$

$E_{s_j}^0 \leftarrow \text{edge}(c_i, s_j)$, where $E_{s_j}^0$ is the first edge incident to s_j and c_i is the check

node which has the smallest check node degree under the current graph setting

$E_{s_0} \cup E_{s_1} \cup \dots \cup E_{s_{j-1}}$. If they are multiple choices for c_i , we select the one with the smallest subscript.

else

Expand a tree from symbol node s_j up to depth l under the current graph setting such that $\overline{\mathcal{N}_{s_j}^l} \neq \emptyset$ but $\overline{\mathcal{N}_{s_j}^{l+1}} = \emptyset$, or the cardinality of $\mathcal{N}_{s_j}^l$ stops increasing but is less than m , then $E_{s_j}^k \leftarrow \text{edge}(c_i, s_j)$, where $E_{s_j}^k$ is the k -th edge incident to s_j and c_i is the check node picked from the set $\overline{\mathcal{N}_{s_j}^l}$ which has the smallest check degree. If they are multiple choices for c_i , we select the one with the smallest subscript.

end

end

Here we make two comments on the PEG algorithm. Whenever we have multiple choices for connecting to s_j , i.e. multiple check nodes exist in $\overline{\mathcal{N}_{s_j}^l}$, we select one having the smallest number of incident edges under the current graph setting. This selection strategy renders a TG with check node degree distribution as uniform as possible. It is important to note that a concentrated check-node-degree degree is optimum in some weak sense [27], [28].

Multiple choices may still exist because multiple check nodes in $\overline{\mathcal{N}_{s_j}^l}$ might have the same lowest degree, particularly during the start-up period. There are mainly two approaches to resolve this issue. The first is to randomly select one of these check nodes as adopted by [21], [22]. We here use the second approach: to always select the one with the

smallest subscript in the sequence of $c_0, c_1, \dots, c_{m-1}$. This approach makes the algorithm completely deterministic.

2.3 Encoding of LDPC codes

In this work, two approaches of encoding LDPC codes are considered as follows.

2.3.1 Systematic encoding using generator matrix G

When we discuss the encoding of LDPC codes, one natural approach is to use generator matrix G with the application of Equation (2.8). In this approach, we have to transfer the parity-check matrix H into equivalent systematic form H_{sys} unless special methods are used during the construction of the parity-check matrix H . To obtain H_{sys} we perform Gaussian elimination on H without column swapping to a reduced row echelon form. We will see in the next chapter that allowing column permutation here will ruin our optimization algorithms. In general, H_{sys} and G_{sys} take the form of (2.9) and (2.6), respectively. It is necessary for the encoder to store $P_{k \times (n-k)}$ to facilitate the encoding process. As mentioned in section 2.2.1, message positions may not be the last k positions because we do not consider column reordering, thus we have to record the message bit and parity bit positions in H , this clearly makes the encoder design more complex.

2.3.2 Linear-time encoding using parity-check matrix H

As can be seen, the first approach needs to store G_{sys} (or $P_{k \times (n-k)}$) corresponding to H_{sys} in the encoder, thus a considerable amount of memory is required because H_{sys} is not sparse even though H is usually designed so. Also, since the parity bits have to be com-

puted using non-sparse operations, the encoding complexity is quadratic with respect to the block length. In fact the only major drawback of LDPC codes compared to turbo codes appears to be their greater encoding complexity. Therefore a considerable amount of research has been conducted to take care of the encoding complexity during construction of LDPC codes. The use of a cascade of subcodes to accomplish linear-time encoding was proposed in [17], [29]. A performance loss is induced because each subcode is considerably smaller than the overall code in general. Another approach, proposed in [20], is to force the parity-check matrix to have a lower or upper triangular form. Similar approach is to incorporate a pattern, referred to as zig-zag pattern, in the parity-check matrix [30], [31].

One of the most important results on encoding of LDPC codes is given in [32] where it is proved that optimized LDPC codes can be encoded in linear time. In particular, one can choose a randomly-constructed code with an optimized irregular degree sequence pair, and then the parity-check matrix of this code can be rearranged to have an (almost) upper-triangular structure that allows linear-time encoding. The significance of [32] lies in the existence proof of optimal LDPC codes with (almost) upper-triangular structure. Note that the rearrangement of a parity-check matrix does not change the LDPC code itself and the complexity of rearrangement is generally quadratic, thus it is preferable to construct such LDPC codes directly.

The PEG algorithm can be used to construct LDPC codes with (almost) triangular structure for H meanwhile having good girth properties and with optimum irregular degree sequence. Following [30], we decompose the codeword $\mathbf{v}$ as $\mathbf{v} = [\mathbf{p}, \mathbf{d}]$, where $\mathbf{p}$

and d contain the parity and information bits respectively. Accordingly H is decomposed into $H = [H^p, H^d]$. So we have

$$[H^p, H^d]v^T = [H^p, H^d] \begin{bmatrix} p \\ d \end{bmatrix} = 0. \quad (2.11)$$

For the sake of clarity we focus on a parity-check matrix $H = [H^p, H^d]$ with upper triangular structure. The $m \times m$ square matrix $H^p = \{h_{ij}^p\}$ of the parity-check matrix can be written as:

$$H^p = \begin{pmatrix} 1 & h_{1,2}^p & \cdots & h_{1,m}^p \\ 0 & 1 & h_{2,3}^p & \vdots \\ \vdots & \ddots & \ddots & h_{m-1,m}^p \\ 0 & \cdots & 0 & 1 \end{pmatrix}, \quad (2.12)$$

It is easy to see that the parity bits can be computed recursively by

$$p_i = \sum_{j=i+1}^m h_{i,j}^p p_j + \sum_{j=1}^{n-m} h_{i,j}^d d_j \pmod{2},$$

from $i = m$ to $i = 1$.

By combining the PEG construction with the constraint that the resulting H matrix can be of upper triangular form, we obtain powerful, linear-time encodable, irregular LPDC codes with large girth and very good performance. To facilitate linear-time encoding, we partition the symbol node set V_s into two disjoint subsets: a redundant subset V_s^p and an information subset V_s^d . The redundant subset V_s^p contains the first m symbol nodes, which are redundant bits in a codeword. The information subset V_s^d contains the other $n-m$ symbol nodes, which corresponds to information bits (systematic) bits in a

codeword. Then the PEG algorithm is used to establish the edges of symbol nodes while maintaining the special pattern in (2.12). The procedure of establishing the edges of information subset V_s^d is the same as before (Deterministic PEG algorithm in section 2.2.2). For the edges of V_s^p the following modified PEG algorithm can be used:

Deterministic PEG Algorithm for Establishing the Edges of V_s^p :

For $j = 0$ to $m-1$ do

begin

for $k = 0$ to $d_{s_j} - 1$ do

begin

if $k = 0$

$E_{s_j}^0 \leftarrow \text{edge}(c_j, s_j)$, where $E_{s_j}^0$ is the first edge incident to s_j . This edge corresponds to the '1' in the diagonal line of matrix H^p .

else

Expand a tree from symbol node s_j up to depth l under the current graph setting

$E_{s_0} \cup E_{s_1} \cup \dots \cup E_{s_{j-1}}$, such that $\overline{\mathcal{N}_{s_j}^l} \cap \{c_0, c_1, \dots, c_{j-1}\} \neq \emptyset$ but $\overline{\mathcal{N}_{s_j}^{l+1}} \cap \{c_0, c_1, \dots, c_{j-1}\} = \emptyset$, or the cardinality of $\mathcal{N}_{s_j}^l$ stops increasing. Then $E_{s_j}^k \leftarrow \text{edge}(c_i, s_j)$,

where $E_{s_j}^k$ is the k -th edge incident to s_j and c_i is one check node picked from

the set $\overline{\mathcal{N}_{s_j}^l} \cap \{c_0, c_1, \dots, c_{j-1}\}$ having the lowest check node degree. If they are

multiple choices for c_i , we select the one with the smallest subscript.

end

end

2.4 Iterative decoding algorithms

LDPC codes are efficiently decodable by different iterative decoding algorithms, which are also known as “message-passing algorithms”. They are naturally described using the TGs of the codes, i.e. they are operated by exchanging messages (or information) between symbol nodes and check nodes through the edges of the graph, in both directions and iteratively. Since the number of edges of the Tanner graph is linear in the length of the code it follows that for a fixed degree distribution, the decoding complexity per iteration grows only linearly with the block length. For a fixed number of iterations, the total decoding complexity is then a linear function of block length.

An iterative decoding algorithm first generates initial messages at symbol nodes from the observation at the output of the channel and the channel characteristics. The check node to symbol node message vectors start off with a uniform distribution. Then iterations start by passing messages to check nodes through the edges of the graph. Each iteration consists of two steps: 1) Check nodes process the information received from the symbol nodes and pass new messages to symbol nodes. These messages reflect the estimate of check nodes from the value of symbol node and the associated reliability. 2) Symbol nodes process the messages received from check nodes and send back updated messages (information) about their value and the associated reliability to check nodes. In each iteration, hard decision on the value of each bit is made at symbol nodes and the algorithm stops if the hard decision assignment for bits satisfies all the check constraints, or the maximum number of iterations is reached. The messages passing through the graph could be “hard” and/or “soft” information, and the operations performed in symbol and check nodes to generate the messages depend on the nature of the decoding algorithm and

the message type(s). As an example, Figure 2-5 shows instances of message passing between symbol nodes and check nodes in the graph defined by Figure 2-3.

It is useful to review some terminology before we describe some iterative decoding algorithms. If $p = P(x=0)$ is the probability that a bit x is equal to 0, then with respect to an observation y , the *prior* probability $p = P(x=0)$ refers to what was known before the observation (we will call it *intrinsic* information). The *posteriori* probability $P(x=0 | y)$ is the conditional probability given the observation y . Finally, the *extrinsic* information is what has been learned from the observation y , and is the difference of the posterior and prior information. It is convenient to use *log-likelihood ratio (LLR)* for messages (information):

$$L(x) = \log \frac{P(x=0)}{P(x=1)} = \log \frac{p}{1-p} \quad (2.13)$$

Then we have the following expression [13]:

$$L^{\text{prior}}(x) + L^{\text{extrinsic}}(x) = L^{\text{posterior}}(x) \quad (2.14)$$

In the context of Tanner graph, the message that is sent from a symbol node s or a check node c along an edge e , does not depend on the message previously received along edge e . Excluding the incoming message along edge e , in computing the outgoing message along edge e , guarantees that only *extrinsic* information is passed along and is an important property of good iterative decoders [7]. This property is clearly illustrated on Figure 2-5. In Figure 2-5(a) the outgoing message sent from check node C_0 to symbol node S_0 , r_{00} , is just a function of the incoming messages from all other symbol nodes connected to C_0 , $q_{02}, q_{04}, q_{05}, q_{07}$ (Note that check node C_0 has degree 5). Also in Figure 2-5 (b) the outgoing message sent from symbol node S_7 to check node C_0 , is a function of the ini-

initial message p_7 and the incoming messages from other check nodes, r_{17}, r_{27} (Note that symbol node S_7 has degree 3).

Figure 2-5: (a) Check node C_0 sending a new message to symbol node S_0 (b) Symbol node S_7 sending back updated message to check node C_0

There are a wide variety of iterative algorithms for decoding LDPC codes. There are hard decision algorithms, also referred to as bit-flipping (BF) iterative decoding algorithms such as Gallager's algorithms A and B [2], etc. There are also soft decision algorithms such as max-sum algorithm, sum-product algorithm [10], etc. Also there are combined hard/soft decision algorithms such as weighted bit-flipping algorithm in [23] and [33]. The sum-product (SP) algorithm (also known as "belief-propagation algorithm") is shown to have the best performance among iterative decoding algorithms.

For the sake of brevity, in the following subsection, we explain the SP algorithm that has been used throughout this thesis.

2.4.1 Sum-Product (SP) algorithm

The SP algorithm was first introduced by Gallager in his thesis to decode LDPC codes [2]. Later, Tanner [10] generalized Gallager's LDPC codes to codes on Tanner

graphs, developed SP for decoding these codes, and proved their convergence on cycle-free graphs. Wiberg [34] proved in his thesis that if the Tanner graph is cycle-free, SP algorithm computes the exact marginal posterior probabilities. Fortunately, for graphs containing cycles, as is the case for LDPC codes, SP algorithm still delivers impressive results. This observation along with the fact that SP is practically implementable, have instigated a lot of research in both theoretical and practical aspects of this algorithm.

Let us consider a binary LDPC code of length n defined by its parity-check matrix H with dimension $m \times n$. Suppose that each bit v_i of a codeword $\mathbf{v} = (v_1, v_2, \dots, v_n)$ is mapped into the bipolar signal $x_i = (-1)^{v_i}$ (Note that the bit value $\{0,1\}$ and the corresponding signal value $\{+1,-1\}$ are used interchangeably throughout this thesis). And then $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is transmitted over a binary input additive white Gaussian noise (AWGN) channel. At the output of the channel, we have $\mathbf{y} = \mathbf{x} + \mathbf{n}$, where $\mathbf{n}$ is a Gaussian random vector independent of $\mathbf{x}$, whose elements are independent zero-mean Gaussian random variables with variance $\sigma^2 = N_0 / (2RE_b)$, in which R is the rate of the code, and E_b and $\frac{N_0}{2}$ are the average energy per information bit and the power spectral density of AWGN, respectively.

For such a communication scenario, we use the following notations:

S_i : symbol nodes, $i = 1, 2, \dots, n$

C_j : check nodes, $j = 1, 2, \dots, m$

$p_i^b = \mu_{X_i \rightarrow S_i}(v_i = b)$: the message passed from node X_i (the channel) to symbol node

S_i for $b = 0, 1$

$q_{ji}^b = \mu_{S_i \rightarrow C_j}(v_i = b)$: the message passed from symbol node S_i to check node C_j

$r_{ji}^b = \mu_{C_j \rightarrow S_i}(v_i = b)$: the message passed from check node C_j to symbol node S_i

q_i^b : the estimate of *a posteriori* probability (APP)

$\mathbf{M}(i)$: the set of parity checks in which bit v_i is involved, i.e. the set of row locations in the i -th column of the parity check matrix that contain a '1'.

$\mathbf{M}(i) \setminus \{j\}$: the set $\mathbf{M}(i)$ with the element j omitted.

$\mathbf{N}(j)$: the set of symbol nodes connected to the j -th parity check, i.e. the set of columns in the j -th row that contain a '1'.

$\mathbf{N}(j) \setminus \{i\}$: the set $\mathbf{N}(j)$ with the element i omitted.

We also define the following LLR's:

$$L(p_i) = \log \frac{p_i^0}{p_i^1} = \log \frac{P(v_i = 0 | y_i)}{P(v_i = 1 | y_i)} \quad (2.15)$$

$$L(q_{ji}) = \log \frac{q_{ji}^0}{q_{ji}^1} \quad (2.16)$$

$$L(r_{ji}) = \log \frac{r_{ji}^0}{r_{ji}^1} \quad (2.17)$$

$$L(q_i) = \log \frac{q_i^0}{q_i^1} \quad (2.18)$$

The message passing between symbol nodes and check nodes are summarized as follows [13]:

Step 0: Initialization. The input to the decoding algorithm is the intrinsic probability with respect to the decoder for each bit v_i , i.e. $L(p_i) = LLR^{\text{int}}(v_i)$. For an AWGN channel with noise variance σ^2 and received signal y_i , this intrinsic information

is $LLR^{\text{int}}(v_i) = \frac{2}{\sigma^2} \cdot y_i$. In addition, the check-to-symbol message vectors start off with a

uniform distribution, i.e. $L^{(0)}(r_{ji}) = \log \frac{r_{ji}^0}{r_{ji}^1} = 0$. The iteration number l starts at 1.

Step 1: Symbol-to-check messages. The messages from symbol nodes to check nodes are given by:

$$L^{(l)}(q_{ji}) = \sum_{j' \in M(i) \setminus \{j\}} L^{(l-1)}(r_{ji'}) + L(p_i) \quad (2.19)$$

Step 2: Check-to-symbol messages. The messages from check nodes to symbol nodes are given by:

$$L^{(l)}(r_{ji}) = \left(\prod_{j' \in N(j) \setminus \{i\}} \text{sgn}(L^{(l)}(q_{ji'})) \right) \cdot \Phi \left(\sum_{j' \in N(j) \setminus \{i\}} \Phi(|L^{(l)}(q_{ji'})|) \right) \quad (2.20)$$

where $\Phi(x) = -\log \left(\tanh \left(\frac{x}{2} \right) \right) = \log \frac{e^x + 1}{e^x - 1}$

Step 3: Compute output. The output message from the decoder is the extrinsic information (in the context of turbo equalization in chapter 4):

$$LLR_{LDPC}^{\text{ext}}(v_i) = \sum_{j' \in M(i)} L^{(l)}(r_{ji'}) \quad (2.21)$$

and the estimate of the posterior information which is given by:

$$L^{(l)}(q_i) = \sum_{j' \in M(i)} L^{(l)}(r_{ji'}) + L(p_i) \quad (2.22)$$

This is used to compute the bit-by-bit estimate of the codeword:

$$\hat{v}_i^{(l)} = \begin{cases} 0, & \text{if } L^{(l)}(q_i) \geq 0 \\ 1, & \text{if } L^{(l)}(q_i) < 0 \end{cases} \quad (2.23)$$

Step 4: Repeat until done. Check if the stopping condition has been reached (e.g. maximum number of iterations has been reached, or the decision word consisting of $\hat{v}_i^{(l)}$ satisfies the parity check equations). If not, then increment $l \leftarrow l + 1$, and repeat Steps 1, 2, and 3.

Before we leave this section, we would like to make a comment on the decoding of irregular LDPC codes. There exists a phenomenon called the “wave effect” [18] in the decoding process of irregular LDPC codes, whereby the symbol nodes with high degrees tend to approach their correct values very quickly, and in turn, they provide more reliable information to the check nodes and subsequently to the symbol nodes with lower degrees. Thus it is better to have high degree for the symbol nodes, since the more information a symbol node gets from its connected check nodes, the more reliably it can provide its own correct value. On the other hand, it is better for a check node to have low degree in order to provide more accurate information to its connected symbol nodes.

2.5 LDPC code performance on AWGN channel

In this section we study the performance of regular and irregular LDPC codes using SP algorithm by computer simulation and compare the performance of codes obtained by random Poisson construction and PEG construction. Although the performance of some of these codes using SP algorithm, has been reported in other references, all of our comparisons are made based on our simulation results to make sure that the comparisons are

fair in terms of using the same simulation program and the same confidence interval for simulations. In fact, if there are minor differences between our simulation results and other reported results, they are due to slight differences between the simulation programs and confidence intervals. For LDPC codes constructed using PEG algorithm, we have used the deterministic versions of PEG algorithm that were described in the previous section, thus the generated codes may have slightly different girth distribution from the codes that have been generated in [22] by using a non-deterministic version of PEG algorithm and their performance may also have small differences.

Since AWGN channel along with SP decoding is symmetric [7], we always assume in our simulations that the transmitted codeword is the all-zero codeword. The maximum number of iterations is chosen to be 80 and enough noisy codewords are simulated to generate at least 100 codeword errors for each signal to noise ratio (SNR). Also at low SNR, a minimum of 1000 codewords are simulated.

2.5.1 Regular Codes

Now we compare two LDPC codes of rate-1/2 with block length $n=1008$. The first code is the MacKay's (3, 6)-regular LDPC code of length 1008 which has been produced using Poisson construction by avoiding cycles of length 4 [35], and is one of the best codes with these parameters. The second code has a PEG-constructed Tanner graph with 1008 symbol nodes of degree 3, 30 check nodes of degree 7, 30 check nodes of degree 5, and 444 check nodes of degree 6. In the PEG-constructed Tanner graph, 7% of symbol node have a local girth of 8 and 93% of 10, while in the MacKay's code, 39.5% of the symbol nodes have a local girth of 6 and 60.3% of 8. Clearly PEG code has a superior girth distribution. Figure 2-6 shows the curves of bit error rate (BER) and word error rate

(WER) for these two codes on AWGN channel. We can see that PEG code performs better than Mackay's code. At WER of 10^{-5} , PEG code outperforms Mackay's code by about 0.2dB. Since these two codes have the same rate, same block length and very similar degree distributions, they have essentially the same decoding complexity and thus this gain is achieved without any increase in the decoding complexity. The significance of this difference would be clearer if we note that MacKay's codes are among strong codes for short and medium block length.

Figure 2-6: (1008,504) regular LDPC code performance on AWGN channel

2.5.2 Irregular Codes

It was shown in [17]-[19] that properly designed irregular LDPC codes can outperform regular ones. Having large girth is only one condition for an irregular LDPC code to have a good performance; design of good irregular degree sequence for symbol nodes and check nodes is equally or even more important. For very long codes, a concentration theorem has been proved in [7] which states that the performance of an ensemble of LDPC codes is concentrated around its mean which itself is the performance of a cycle-free graph with the same parameters. This allows us to find optimum degree sequences by *density evolution* techniques. Density evolution is basically a technique for obtaining, through deterministic algorithms, expressions for the probability density functions of the messages which are passed in each iteration of an iterative decoding algorithm, provided that the Tanner graph of the code is cycle free. The irregular LDPC codes which have the optimum degree sequences, obtained by using density evolution techniques, and have been constructed based on random Poisson construction, give performances extremely close to Shannon limit on AWGN channel at large block lengths.

However for short LDPC codes the cycle-free assumption is no longer valid and thus the density evolution technique cannot be used to obtain optimum degree sequences. In [14] the average of girth distribution is used as a heuristic tool to select good codes among random codes at short block lengths. An empirical Monte-Carlo approach has been introduced in [22] which uses a variant of the “downhill simplex” optimization technique [36] to design symbol node-degree distribution for short codes with fewer than a thousand bits. Here, like regular codes, the degree of check nodes is made as uniform as

possible. It is also shown in [22] that irregular degree distributions optimized by the density evolution approach match well with the PEG construction for short block lengths.

Here we will compare two codes of rate-1/2 with block length $n=1008$. The first code is constructed based on the PEG algorithm using the following optimized degree distribution [22]:

$$\lambda(x) = 0.47532x^2 + 0.279537x^3 + 0.0348672x^4 + 0.108891x^5 + 0.101385x^{15}$$

This code has 479, 282, 35, 110, and 102 symbol nodes with degrees 2, 3, 4, 5, and 15, respectively. Also it has 31, 450, and 23 check nodes with degrees 7, 8, and 9, respectively. The second code is constructed based on random Poisson construction by avoiding cycles of length 4 with the same degree distributions. Simulation results for these two codes are presented in Figure 2-7. Mackay's (3,6) regular code performance is also plotted in the same figure. We observe that the PEG code outperforms both random graph code and Mackay's code by about 0.3dB at BER of 10^{-4} , and by about 0.4dB at WER of 10^{-3} . This clearly shows that at short block lengths, using PEG construction for constructing LDPC codes has a significant advantage over the random Poisson construction.

Figure 2-7: (1008,504) irregular/regular LDPC code performance on AWGN channel

2.5.3 Linear-time Encodable Irregular LDPC Codes

As it was mentioned in section 2.3, linear-time encoding property can be naturally combined with the PEG construction and as we will see in the following, this comes without noticeable degradation in the performance of the code. Let us consider an irregular PEG code whose parity-check matrix is forced to be an upper-triangular matrix and has been constructed by deterministic modified-PEG algorithm described in subsection 2.3.2, with parameters $n=1008$, $k=504$, and based on the symbol node degree distribution function $\lambda(x) = 0.47532x^2 + 0.279537x^3 + 0.0348672x^4 + 0.108891x^5 + 0.101385x^{15}$. As the parity-check matrix is forced to have an upper triangular form, the Tanner graph of

this LDPC code has one symbol node with degree 1 and the remaining symbol nodes will be of degree 2, 3, 4, 5, and 15 with 478, 282, 35, 110, and 102 symbols with each degree, respectively. Also the check nodes have degrees 7, 8, and 9 with 31, 451, and 22 check nodes with each degree, respectively. Figure 2-8 compares the performance of this PEG LE code with the PEG code of Figure 2-7 which has the same characteristics except that it does not have an upper-triangular parity-check matrix. As we can see, these two irregular codes have essentially the same performance, thus we conclude that linear-time-encoding property can be achieved without noticeable degradation in the performance of the code, under PEG construction.

Figure 2-8: (1008,504) irregular LDPC code performance on AWGN channel

Chapter 3 – LDPC Code Optimization for

Burst Error Channels

Most of the well-known codes that have been devised for increasing the reliability in the transmission of information are effective when the errors caused by the channel are statistically independent, as is the case for AWGN channel. However, there are channels that exhibit bursty error characteristics [37]. One example is a multipath fading channel, which we are going to discuss in the next chapter. Signal fading due to time-variant multipath propagation often causes the signal to fall below the noise level, thus resulting in a large number of errors. A second example is a magnetic recording channel (tape or disk) in which defects in the recording media result in clusters of errors. Such errors are not usually correctable by codes that are optimally designed for statistically independent errors.

An effective method to cope with burst errors is to insert an interleaver between the channel encoder and the channel. Figure 3-1 shows a block diagram of a system that employs interleaving. The coded data is reordered by the interleaver and then transmitted over the channel. At the receiver, the deinterleaver performs the reverse operation to restore the data to its original order and passes it to the decoder. As a result of the interleaving/deinterleaving operation, error bursts are spread out in time so that errors within a codeword appear to be independent. Thus the burst error channel is transformed into a

random error channel at the input of the decoder and a code designed for independent error channels could be used for burst error channels.

In this chapter, we will first review the principle of interleaving and examine several types of interleavers. Then we will study the TG structure in more details and introduce *distance profile* as a measure of the performance of LDPC codes over channels with burst errors. Using distance profile, we will then develop a set of optimization algorithms for achieving better performance on bursty channels. In sequel, the application of optimization algorithms to multipath fading channels is presented in the next chapter along with the comparisons of different interleaving techniques. The application of optimization algorithms to another bursty channel – Inter-Symbol Interference (ISI) channel is presented in chapter 5.

Figure 3-1: Block diagram of a communication system with interleaver for bursty channel

3.1 Principle of Interleaving

Interleaving is a process of rearranging the order of a data sequence in a one-to-one deterministic format. The inverse of the process is deinterleaving which restores the received sequence to its original order. Interleaving is a practical technique to enhance the error correcting capability of codes over burst error channels. Also in powerful turbo codes introduced in [3], interleaving plays an important role in achieving capacity-approaching performance of these codes. We will first formally define an interleaver, and then discuss three types of interleaving techniques.

3.1.1 Interleaver

An interleaver with size N is shown in Figure 3-2. For simplicity, we assume that the data sequence at the input of the interleaver IL is binary and is given by:

$$c = (c_1, c_2, \dots, c_N) \quad (3.1)$$

where $c_i \in \{0,1\}, 1 \leq i \leq N$. The interleaver permutes the sequence c to a binary sequence:

$$\tilde{c} = (\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_N) \quad (3.2)$$

where $\tilde{c}_j \in \{0,1\}, 1 \leq j \leq N$. The sequence $\tilde{c}$ has all the elements of c but in a different order. If we consider the input sequence c and the output sequence $\tilde{c}$ as a pair of sets with N elements, there is a one-to-one mapping $c_i \rightarrow \tilde{c}_j$ between each element of c and each element of $\tilde{c}$.

If we define a set A as $A = \{1, 2, \dots, N\}$, then the interleaver can be defined by a one-to-one index mapping function:

$$\pi(A \rightarrow A): j = \pi(i), i, j \in A \quad (3.3)$$

where i and j are the index of an element in the original sequence c and the interleaved sequence $\tilde{c}$, respectively. The interleaving function can be represented by an interleaving vector:

$$\pi_N = (\pi(1), \pi(2), \dots, \pi(N)) \quad (3.4)$$

Figure 3-2: An interleaver

Example 3.1

Let us consider a pseudo-random interleaver with size $N=10$. The input sequence is $c = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10})$, the output interleaved sequence is given by: $\tilde{c} = (\tilde{c}_1, \tilde{c}_2, \tilde{c}_3, \tilde{c}_4, \tilde{c}_5, \tilde{c}_6, \tilde{c}_7, \tilde{c}_8, \tilde{c}_9, \tilde{c}_{10}) = (c_2, c_4, c_1, c_{10}, c_6, c_3, c_8, c_7, c_5, c_9)$. The mapping function is illustrated in Figure 3-3. The interleaving vector can be expressed as:

$$\begin{aligned} \pi_{10} &= (\pi(1), \pi(2), \pi(3), \pi(4), \pi(5), \pi(6), \pi(7), \pi(8), \pi(9), \pi(10)) \\ &= (3, 1, 6, 2, 9, 5, 8, 7, 10, 4) \end{aligned}$$

Figure 3-3: An interleaver mapping

An interleaver or a deinterleaver can be characterized by its delay and storage capacity. The interleaving or deinterleaving delay is the maximum delay encountered by any symbol before it is inserted into the output sequence. The storage capacity is the number of symbols stored by the interleaver or deinterleaver.

3.1.2 Interleaving techniques

In this subsection, we will look at three types of interleaving techniques: block interleavers, convolutional interleavers, and random interleavers.

Block Interleavers A block interleaver formats the input sequence in a matrix of N_r rows and N_c columns, such that $N = N_r \times N_c$. The input sequence is written into the matrix row-wise and read out column-wise as illustrated in Figure 3-4. The deinterleaver stores the data into a matrix with the same format as the interleaver, but the data is read out row-wise. The interleaving function of the block interleaver can be expressed as:

$$\pi(i) = [(i-1) \bmod N_c] \times N_r + \lfloor (i-1) / N_c \rfloor + 1, i \in A \quad (3.5)$$

The end-to-end interleaver-deinterleaver delay is $2N_rN_c$ bit interval. The memory requirement is N_rN_c for both the interleaver and the deinterleaver. We call the number of rows in the interleaver matrix as the interleaver degree (or depth) and the number of columns is called the interleaver span [37]. The block interleavers and deinterleavers are easy to implement. However, if they used conjunction with turbo codes they may fail to break certain low weight input patterns, such as weight 4, 6, 9, etc, square or rectangular input patterns [39].

Figure 3-4: A block interleaver

Convolutional Interleavers Convolutional interleavers have been proposed in [40] and [41]. Figure 3-5 shows the structure proposed in [41].

Both the interleaver and deinterleaver consist of an input and output commutator and a bank of L shift registers. The data sequence to be interleaved is arranged in blocks of L bits. The input commutator cyclically inserts each block of L bits into the bank of L registers. The i th bit in each block is delayed by the i th shift register and the delay of the shift register is $(i-1)B$, where B is the delay interval. The output commutator cyclically samples the bank of L registers in the same order as the input one. The deinterleaver performs the inverse operation, i.e. the i th bit in each block is delayed by the i th shift register, where the delay of the shift register is $(L-i)B$. If the initial state of the shift registers is zero state, the convolutional interleaving function can be expressed as:

$$\pi(i) = i + [(i-1) \bmod L] \times LB, i \in A \quad (3.6)$$

This interleaver is also called $LB \times L$ interleaver. The end-to-end interleaver-deinterleaver delay is $(L-1)LB$. The memory requirement is $(L-1)LB/2$ for either the interleaver or the deinterleaver.

Figure 3-5: A convolutional interleaver and deinterleaver

Figure 3-6 shows an example of convolutional interleaver with $L = 3$, and $B = 2$. The interleaver size is 21 and the interleaved sequence is given by:

$$\tilde{c} = \{c_1, 0, 0, c_4, 0, 0, c_7, c_2, 0, c_{10}, c_5, 0, c_{13}, c_8, c_3, c_{16}, c_{11}, c_6, c_{19}, c_{14}, c_9, 0, c_{17}, c_{12}, 0, c_{20}, c_{15}, 0, 0, c_{18}, 0, 0, c_{21}\}$$

Figure 3-6: A convolutional interleaver with $L = 3$, $B = 2$

We can see that the output sequence is expanded because of the extra zeros inserted in the output sequence. Now we will describe an alternative way of generating convolutional interleavers, which is based on cyclic shifts [38]. In the interleaver, the data sequence is written into a matrix column-wise. The matrix has N_r rows and N_c columns such that $N = N_r \times N_c$ and $N_r \leq N_c$. Then N_r row sequences of size N_c are applied to the bank of N_r registers, where the i th register shifts the i th row sequence in the matrix cyclically to the right by $(i-1)B$, where B is a small integer such that $B \leq N_c / N_r$. The shifted sequences form a new matrix and the data are read out from the matrix column-wise. The deinterleaver performs the similar operation, whereby, the i th row sequence in the matrix is cyclically shifted to the left by $(i-1)B$. An example of cyclic shift interleaver with $N_r = 3$, $N_c = 7$, $B = 2$ is shown in Figure 3-7. The interleaved sequence is given by: $\tilde{c} = \{c_1, c_{17}, c_{12}, c_4, c_{20}, c_{15}, c_7, c_2, c_{18}, c_{10}, c_5, c_{21}, c_{13}, c_8, c_3, c_{16}, c_{11}, c_6, c_{19}, c_{13}, c_9\}$. Clearly, the cyclic shift interleaver overcomes the disadvantage of expanding the output sequence.

Figure 3-7: A cyclic shift interleaver

Random Interleavers In the random interleaver a block of N input bits is read into the interleaver and read out randomly. The interleaving vector $\pi(i), i \in 1, 2, \dots, N$, can be generated according to the following algorithm, which requires N steps:

Step 1: Choose randomly an integer i_1 from the set $A = \{1, 2, \dots, N\}$ according to a uniform distribution between 1 and N with the probability of $p(i_1) = 1/N$. The chosen integer i_1 is set to be $\pi(1)$.

Step k ($k > 1$): Choose randomly an integer i_k from the set $A_k = \{i \in A, i \neq i_1, i_2, \dots, i_{k-1}\}$ according to a uniform distribution with the probability of $p(i_k) = \frac{1}{N-k+1}$. The chosen integer i_k is set to $\pi(k)$.

When $k = N$, the last integer i_N is set to $\pi(N)$.

We will also describe S -random interleaver next.

S -random Interleavers Divsalar and Pollara proposed S -random interleavers in [42]. They are pseudo-random interleavers based on the random generation of N integers from 1 to N with an S -constraint, where S is defined as the minimum interleaving distance. S -random interleavers are also called “spread interleavers”.

We can define an S -random interleaver as follows [42]. Each randomly selected integer is compared to S previously selected integers. If the current selection is equal to any S previous selections within a distance of $\pm S$, then the current selection is rejected. This process is repeated until all N integers are selected. In general, an S -random interleaver can be described as:

$$|\pi(i_1) - \pi(i_2)| \geq S \quad (3.7)$$

whenever

$$|i_1 - i_2| \leq S, i_1, i_2 \in A \quad (3.8)$$

where S is an integer smaller than N . As the searching time increased with S , choosing $S < \sqrt{N/2}$ usually produces a solution in reasonable time (For $S = 1$, we have a purely

random interleaver) [42]. This interleaver assures that short cycle events are avoided, where a short cycle event occurs when two bits are close to each other both before and after interleaving.

As we will see later, the optimization algorithms developed in this work are similar to S -random interleaving in spirit.

3.2 TG structure and distance profile

In this subsection, we will study the Tanner graph structure and introduce a new concept *distance profile* for LPDC codes.

3.2.1 Cycles and girth

From the Definition 2.2.1a and 2.2.1b, we will now look at how to find the cycles and the girth for a TG. Similar to the neighborhood definition in Definition 2.2.1d, we expand a tree step by step starting from the “root” node u (symbol node S_i or check node C_j). At step k , all the nodes at distance k from u are included into the tree, and all the nodes on the tree are considered as the neighbors of node u . This process is repeated until, two nodes with the same label appear at the same level of the tree and they are originating from different upper nodes except the root. This identifies the formation of the first cycle. Integer $2k$ is the local girth of node u . Note that the cycle passing through node u is not unique as can be seen in the following example.

Example 3.2

Let us consider the TG defined by Figure 2-3, Figure 3-8 shows the tree expanding from the symbol node S_0 up to step $k = 3$. At step $k = 2$, we can find a cycle of length 4: S_0 - C_0 - S_5 - C_1 - S_0 , thus the local girth of S_0 is 4. We also see that there is another cycle of

length 4 passing through S_0 : $S_0-C_0-S_7-C_1-S_0$. At step $k = 3$, we can find the following 9 cycles of length 6:

$S_0-C_0-S_2-C_2-S_5-C_1-S_0$, $S_0-C_0-S_2-C_2-S_6-C_1-S_0$, $S_0-C_0-S_2-C_2-S_7-C_1-S_0$,
 $S_0-C_0-S_4-C_3-S_3-C_1-S_0$, $S_0-C_0-S_4-C_3-S_6-C_1-S_0$, $S_0-C_0-S_5-C_2-S_6-C_1-S_0$,
 $S_0-C_0-S_5-C_2-S_7-C_1-S_0$, $S_0-C_0-S_7-C_2-S_5-C_1-S_0$, $S_0-C_0-S_7-C_2-S_6-C_1-S_0$.

Figure 3-8: The tree expanding from symbol node S_0

We will point out here that the complexity of finding cycles and computing girth is low and quite manageable for short block lengths.

3.2.2 Distance Profile

Now we can associate a *distance profile* to each symbol node.

Definition 3.2.2a *Distance profile* of cycles of length L for a symbol node u is defined as the profile indicating the distances between u and all the symbol nodes involved in all the cycles of length L passing through u . The minimum of all such distances is referred to as “shortest L -cycle distance” of u . The minimum of all shortest L -cycle distances of u for

all possible values of L is called “shortest cycle distance” of u . The distance here is defined as the absolute value of the difference between two node numbers.

To illustrate this concept, let us look at one example.

Example 3.2 (Cont’d)

We find that two cycles of length 4 are passing through symbol node S_0 if we expand the tree up to step $k = 2$. Considering only the symbol nodes in the cycles, these two cycles are (only node numbers are shown): $\{0, 5\}$, $\{0, 7\}$. If the tree growing stops at this step, we can see that:

- 1) there are 2 cycles of length-4 passing through symbol node S_0
- 2) these two cycles are: $\{0, 5\}$, $\{0, 7\}$
- 3) there are 2 involved symbol nodes other than symbol node S_0
- 4) these two symbol nodes are: $\{5, 7\}$

Thus the shortest 4-cycle distance of symbol node S_0 is 5.

Spreading the tree further to step $k = 3$, we find the cycles of length 4 and 6 passing through symbol node S_0 as shown in the following cycle list (only node numbers are shown and numbers are sorted in ascending order): $\{0, 5\}$, $\{0, 7\}$, $\{0, 2, 5\}$, $\{0, 2, 6\}$, $\{0, 2, 7\}$, $\{0, 3, 4\}$, $\{0, 4, 6\}$, $\{0, 5, 6\}$, $\{0, 5, 7\}$, $\{0, 5, 7\}$, $\{0, 6, 7\}$.

We notice that although $S_0-C_0-S_5-C_2-S_7-C_1-S_0$ and $S_0-C_0-S_7-C_2-S_5-C_1-S_0$ are different cycles, they can be treated as one cycle in the context of distance profile. Similarly, we can see that:

- 1) there are 10 involved cycles which are passing through symbol node S_0
- 2) the involved cycles are: $\{0, 5\}$, $\{0, 7\}$, $\{0, 2, 5\}$, $\{0, 2, 6\}$, $\{0, 2, 7\}$, $\{0, 3, 4\}$, $\{0, 4, 6\}$, $\{0, 5, 6\}$, $\{0, 5, 7\}$, $\{0, 6, 7\}$.

3) there are 6 involved symbol nodes

4) the involved symbol nodes are: {2, 3, 4, 5, 6, 7}

Thus the shortest 6-cycle distance is 2. If we only consider cycles of length-4 and 6, the shortest cycle distance of symbol node S_0 is 2.

In describing the optimization algorithms we also need the following definitions:

Definition 3.2.2b If a TG has a girth distribution (refer to Definition 2.2.1c) $g(l)$, $l = 4, 6, \dots, l_{\max}$, we define *shortest cycle set* $\mathfrak{R}_s$ as the set consisting of all the cycles of length g_u passing through u , for every symbol node u , where g_u is the local girth of node u . And we order the cycles in cycle set $\mathfrak{R}_s$ according to node labels in an ascending order.

We use the following procedure to find this cycle set: 1) the tree rooted from a symbol node u with local girth g_u is growing up to step $g_u/2$; 2) all cycles of length g_u are found at this step, and symbol node labels in the cycles are recorded; 3) repeat 1) and 2) for all the symbol nodes; 4) sort the symbol node labels in an ascending order, and remove the redundant (repeated) cycles.

Definition 3.2.2c If a TG has a girth distribution $g(l)$, $l = 4, 6, \dots, l_{\max}$, we define *all cycle set* $\mathfrak{R}_a$ as the set consisting of all the cycles of length $4, 6, \dots, l_{\max}$ passing through u , for every symbol node u . And we order the cycles in cycle set $\mathfrak{R}_a$ according to node labels in an ascending order.

We use the following procedure to find this cycle set: 1) the tree rooted from a symbol node u with girth g_u is growing up to step $l_{\max}/2$; 2) all cycles of length $g_u, g_u+2, \dots, l_{\max}$ are found at this step, and symbol node labels in the cycles are recorded; 3) repeat 1)

and 2) for all the symbol nodes; 4) sort the symbol node labels in an ascending order, and remove the redundant (repeated) cycles.

Since the symbol node labels in the cycle set are sorted in an ascending order and we are interested in the shortest distance, there are $(l-1)$ distances for a cycle of length l . We define the distance distribution as follows:

Definition 3.2.2d The *distance distribution* of the shortest cycle set $\mathfrak{R}_s$ or the all cycle set $\mathfrak{R}_a$ is defined as the function $h(d)$, $d = 1, 2, \dots, n-1$, which gives the probability of the distance between the adjacent symbol nodes in the cycle set being d . We also define the *average distance* for the shortest cycle set $\mathfrak{R}_s$ or the all cycle set $\mathfrak{R}_a$ as $\sum_{d=1}^{n-1} h(d) \cdot d$.

To make the above definitions clear, we will look at one example.

Example 3.3

Let us consider the TG defined by Figure 2-3 again. Following the procedure outlined in section 3.2.1, we find that all the symbol nodes have local girth 4 except symbol node S_4 has local girth 6. We use the following procedure to find the shortest cycle set $\mathfrak{R}_s$:

We expand the tree rooted from symbol nodes S_0 - S_3 , S_5 - S_7 up to step $k = 2$, find all the cycles of length-4, record the symbol node numbers in the cycles, sort the symbol node numbers and remove the redundant cycles. For symbol node S_4 we expand the tree up to step $k = 3$, find all the cycles of length-6. After sorting and removing the redundant cycles, we find that there are totally 19 cycles in the shortest cycle set $\mathfrak{R}_s$, which are: $\{0,5\}$, $\{0,7\}$, $\{1,6\}$, $\{2,5\}$, $\{2,7\}$, $\{3,6\}$, $\{5,7\}$, $\{5,6\}$, $\{6,7\}$, $\{0,3,4\}$, $\{0,4,6\}$, $\{1,2,4\}$, $\{2,4,6\}$, $\{3,4,5\}$, $\{4,5,6\}$, $\{1,4,5\}$, $\{3,4,7\}$, $\{4,6,7\}$, $\{1,4,7\}$.

We use the following procedure to find the all cycle set $\mathfrak{R}_a$:

We expand the tree rooted from all the symbol nodes S_0 - S_7 up to step $k = 3$, and find all the cycles of length-4 and 6. After sorting and removing the redundant cycles, we find that there are totally 36 cycles in the all cycle set $\mathfrak{R}_a$, which are listed below: $\{0,5\}$, $\{0,7\}$, $\{1,6\}$, $\{2,5\}$, $\{2,7\}$, $\{3,6\}$, $\{5,7\}$, $\{5,6\}$, $\{6,7\}$, $\{0,2,5\}$, $\{0,2,6\}$, $\{0,2,7\}$, $\{0,3,4\}$, $\{0,4,6\}$, $\{0,5,6\}$, $\{0,5,7\}$, $\{0,6,7\}$, $\{1,2,4\}$, $\{1,4,5\}$, $\{1,3,5\}$, $\{1,5,6\}$, $\{1,3,6\}$, $\{1,4,7\}$, $\{1,3,7\}$, $\{1,6,7\}$, $\{2,4,6\}$, $\{2,5,6\}$, $\{2,5,7\}$, $\{2,6,7\}$, $\{3,4,5\}$, $\{3,5,6\}$, $\{3,4,7\}$, $\{3,6,7\}$, $\{4,5,6\}$, $\{4,6,7\}$, $\{5,6,7\}$.

We can see that the shortest cycle set $\mathfrak{R}_s$ is a subset of the all cycle set $\mathfrak{R}_a$.

There are totally 29 (= $9+10 \times 2$) distances in the shortest cycle set $\mathfrak{R}_s$, we find 11,6,7,1,3,0,1 instance(s) of distance $d = 1$ to 7, respectively, thus the distance distribution is: $h(1) = 11/29 = 0.3793$, $h(2) = 6/29 = 0.2069$, $h(3) = 7/29 = 0.2414$, $h(4) = 1/29 = 0.0345$, $h(5) = 3/29 = 0.1034$, $h(6) = 0$, $h(7) = 1/29 = 0.0345$. The average distance is $\sum_{d=1}^7 h(d) \cdot d = 2.4138$.

There are totally 63 (= $9+27 \times 2$) distances in the all cycle set $\mathfrak{R}_a$, we find 21,16,12,5,7,1,1 instance(s) of distance $d = 1$ to 7, respectively, thus the distance distribution is: $h(1) = 21/63 = 0.3333$, $h(2) = 16/63 = 0.2540$, $h(3) = 12/63 = 0.1905$, $h(4) = 5/63 = 0.0794$, $h(5) = 7/63 = 0.1111$, $h(6) = 1/63 = 0.0159$, $h(7) = 1/63 = 0.0159$. The average distance is $\sum_{d=1}^7 h(d) \cdot d = 2.4921$.

We will describe a set of optimization algorithms next.

3.3 Optimization algorithms

As discussed in section 2.4, LDPC codes are decoded by different iterative message-passing algorithms such as sum-product algorithm or belief propagation algorithm. In general, short cycles are problematic in the decoding algorithms, and we expect them to be even more problematic when they include many unreliable symbol nodes feeding each other information with low-reliability through the cycle. Thus, what we are trying to do is to make sure that there are not too many bits with low reliability in the same short cycle, and the goal of the optimization algorithms is to minimize the chance of this happening. The details follow.

For channels that exhibit burst error characteristic, interleaving techniques are generally employed to spread out the error bursts, and transform the channel to a random error channel. One example is a multipath fading channel which we will discuss in next chapter. As shown in Figure 3-1, a channel interleaver is introduced to break up the correlation of channel fades among the consecutive bits. The random construction of the parity-check matrix H and its sparseness property suggest that the LDPC codes have a built-in “interleaver”. In principle, it is possible to design this “interleaver”, i.e. to design the matrix H such that all the nonzero entries in H are distributed so as to achieve the maximal “interleaver” gain. In this work, we take another approach to achieve the “interleaver” gain. For a given TG, we find the shortest cycle set $\mathfrak{R}_s$ or the all cycle set $\mathfrak{R}_a$, and separate the symbol nodes in the cycles in these sets as far as possible through swapping the columns of H matrix. Please bear in mind that after swapping the columns of the parity-check matrix H , we are still dealing with the same LDPC code.

For a given TG, we find the shortest cycle set $\mathcal{R}_s$ or the all cycle set $\mathcal{R}_a$, and then we are interested in finding a column permutation with maximum separation between symbol nodes in the cycles. Given that at short block lengths, a TG contains quite a few short cycles, finding the optimal column permutation which results in the best separation among symbol nodes involved in cycles is a tedious task. We thus use the following *greedy* algorithms to find suboptimal solutions.

3.3.1 Greedy algorithm 1A

Step 1: For a given TG, find the shortest cycle set $\mathcal{R}_s$.

Step 2: Build the distance profile for all symbol nodes from the cycle set $\mathcal{R}_s$.

Step 3: Begin with the smallest distance d in the current cycle set $\mathcal{R}_s$, usually $d = 1$ initially.

Step 4: Searching from the first cycle in the cycle set $\mathcal{R}_s$, if the distance between two nodes, say node1 and node2 (with larger index) in the cycle is less than or equal to d , find the swapping candidates (all the symbol nodes with node1 and node2 excluded).

Step 5: Select one of the nodes from the set of swapping candidates, say node3, and temporarily swap node2 and node3. If the distance profiles of both node2 and node3 are improving (the shortest cycle distance is greater than the distance before swapping), record the updated the shortest cycle distance (the sum of the shortest cycle distance of node2 and node3) in an array. Repeat this step for all the swapping candidates. If the array is empty, it indicates that the distance profile can not be improved and the algorithm stops.

Step 6: Find the largest entry from the array and the corresponding symbol node index, say node4, and swap node2 with node4 (exchange their labels).

Step 7: Update the cycle set $\mathcal{R}_s$ and the distance profile.

Step 8: Increment the cycle index and go to the next cycle.

Step 9: Increase the distance by 1 and repeat from step 4.

It is easy to see that the algorithm is completely deterministic, it is easy to implement, and our simulation also shows that it has low complexity.

Please note that at the end of Step 8, the distance d may not be improved, thus we go back to Step 4 and check the distance of less than d . In most of our simulations, we observed that the distance d was indeed improving at the end of Step 8.

3.3.2 Greedy algorithm 2A

Now we will describe a random version of the optimization algorithm 1A.

Step 1: For a given TG, find the shortest cycle set $\mathcal{R}_s$.

Step 2: Build the distance profile for all symbol nodes from the cycle set $\mathcal{R}_s$.

Step 3: Begin with the smallest distance d in the current cycle set $\mathcal{R}_s$, usually $d = 1$ initially.

Step 4: Searching from the first cycle in the cycle set $\mathcal{R}_s$, if the distance between two nodes, say node1 and node2 (with larger index) in the cycle is less than or equal to d , randomly select another node other than node1 and node2, say, node3, and swap node2 with node3.

Step 5: Update the cycle set $\mathcal{R}_s$ and the distance profile.

Step 6: If the distance profiles of both node2 and node3 are improving, i.e., the shortest cycle distance is greater than the distance before swapping, confirm this swapping, otherwise, discard this swapping, randomly select another node, say node4 (node1 and node2 are excluded), swap node2 with node4 and go back to step 5. If we have tried a predetermined number of times (a number much greater than the number of symbol nodes n), and we still can not improve the distance profile, the algorithm stops.

Step 7: Increment the cycle index and go to the next cycle.

Step 8: Increase the distance by 1 and repeat from step 4.

3.3.3 Greedy algorithms 1B & 2B

The algorithms 1B and 2B are the same as algorithm 1A and 2A, respectively, except that at step 1, the input to the algorithm is the all cycle set $\mathcal{R}_a$.

The inputs to the above four algorithms are the shortest cycle set $\mathcal{R}_s$ or the all cycle set $\mathcal{R}_a$ and the original TG. The outputs of the algorithms are: 1) the interleaving array; 2) the new interleaved TG; 3) the updated cycle set, and this cycle set is distance-optimized.

Chapter 4 – LDPC Code Optimization for Multipath Fading Channel

It has been shown that using iterative message-passing decoding algorithm (i.e. [6], [7]), LDPC codes can achieve reliable transmission at SNR extremely close to Shannon limit on the AWGN channel. However, on more realistic channels such as radio, satellite and mobile channels, transmission errors are also caused by variations in received signal strength referred to as fading. This severely degrades the digital transmission performance and powerful error control coding techniques are needed to reduce the SNR penalty.

In this chapter, we will describe a multipath fading channel for mobile communications with a focus on correlated fading channel, which is a bursty channel. Then we will describe the sum-product algorithm on fading channel, both independent and correlated fading. The simulation results for the application of the optimization algorithms to correlated Rayleigh fading channel are presented following by a discussion on error performance.

4.1 Multipath fading channel

In a cellular mobile radio environment, the surrounding objects such as houses, buildings, and trees, act as reflectors of radio waves. These obstacles produce reflected waves with attenuated amplitude and different phase. If a modulated signal is transmitted, multiple reflected waves of the transmitted signal will arrive at the receiving antenna from

different directions with different propagation delays. These reflected waves are called *multipath waves* [47]. Due to the different arrival angles and times, the multipath waves at the receiver will have different phases. When they are collected by the receiver antenna at some point in space, they may combine either in a constructive or a destructive way, depending on the random phase. The sum of these multipath components forms a spatially varying wave field. The mobile station moving through the multipath field will receive a signal which can vary widely in amplitude and phase. When the mobile station is stationary, the amplitude variations in the received signal are due to the movement of surrounding objects in the radio channel. The amplitude fluctuation of the received signal is called *signal fading*. It is caused by the time-variant multipath characteristics of the channel.

Due to relative motion between the transmitter and the receiver, each multipath wave is subject to a shift in frequency. The shift in received signal frequency due to relative motion is called *Doppler shift*, and is directly proportional to the speed and direction of motion of the mobile station with respect to the direction of arrival of the received multipath wave. Consider a situation where only a single tone of frequency f_c is transmitted and a received signal consists of only one wave coming at an incident angle θ with respect to the direction of the vehicle motion. The Doppler shift of the received signal, denoted by f_d , is given by $f_d = \frac{vf_c}{c} \cos \theta$, where v is the vehicle speed and c is the speed of light. The Doppler shift effect in multipath propagation environment spreads the bandwidth of the multipath waves into the range of $f_c \pm f_m$, where f_m is the maximum Doppler shift and is given by $f_m = \frac{vf_c}{c}$. We also call $f_m T_s$ the normalized Doppler frequency,

where T_s is the coded symbol duration. It determines the correlation between the fading envelopes of two consecutive symbols.

A widely used statistical model for fading channel is Rayleigh fading model, where the envelope of the channel impulse response is Rayleigh distributed and the phase of the channel response is uniformly distributed between $-\pi$ and π . The probability density function of Rayleigh distribution is given by:

$$p(a) = \begin{cases} 2ae^{-a^2} & a \geq 0 \\ 0 & a < 0 \end{cases} \quad (4.1)$$

where the average signal power is normalized to unity, i.e. $E[a^2] = 1$. The mean and the variance are: $E[a] = 0.8862$, $\sigma_a^2 = 0.2146$.

In this thesis, we are considering coherent BPSK signaling over a flat fading channel. Clarke's fading model [47] and Jakes fading model [48] are among the most widely used models for wireless fading channels. In this thesis, we will use the modified Jakes model [49] which is described in subsection 4.1.1.

The *level crossing rate* and *average fade duration* of Rayleigh fading signal are two important statistics which are useful for designing error control codes to be used in mobile communication systems [47]. The level crossing rate is defined as the expected rate at which the Rayleigh fading envelope, normalized to the local rms signal level, crosses a specified level in a positive-going direction. For Clarke's fading model, the number of level crossings per second is given by [47]:

$$N_R = \sqrt{2\pi} f_m \rho e^{-\rho^2} \quad (4.2)$$

where ρ is the threshold level which is the value of specified level normalized to the local rms amplitude of the fading envelope. There are few crossings at both high and low levels, with the maximum rate occurring at $\rho = 1/\sqrt{2}$.

The average fade duration is defined as the average period of time for which the received signal is below a specified level ρ . It can be expressed as [47]:

$$\bar{\tau} = \frac{e^{\rho^2} - 1}{\rho f_m \sqrt{2\pi}} \quad (4.3)$$

We can determine on average how many successive symbols (bits) are experiencing fading for some threshold level ρ using:

$$\frac{\bar{\tau}}{T_s} = \frac{e^{\rho^2} - 1}{\sqrt{2\pi} \rho f_m T_s} \quad (4.4)$$

4.1.1 Modified Jakes model

The Jakes fading model is a deterministic method for simulating time-correlated Rayleigh fading waveforms [48]. The model assumes that N equal-strength rays arrive at a moving receiver with uniformly distributed arrival angles α_n , such that ray n experiences a Doppler shift $w_n = w_m \cos(\alpha_n)$, where $w_m = 2\pi f_m$. The modified Jakes model [49] uses a slightly different arrival angle: $\alpha_n = 2\pi(n - 0.5)/N$. The fading waveform is modeled as $N_c + 1$ complex oscillators, and can be expressed as:

$$T(t) = \sqrt{\frac{2}{N_c}} \sum_{n=1}^{N_c} [\cos(\beta_n) + j \sin(\beta_n)] \cos(\omega_n t + \theta_n) \quad (4.5)$$

where $N_c = N/4$ and the normalization factor $\sqrt{2/N_c}$ results in $E\{T(t)T^*(t)\} = 1$. By using $\beta_n = \pi n / N_c$, the real and imaginary part of $T(t)$ have equal power and are uncorrelated. Parameters θ_n are initial phases, and assigning different values to θ_n provides different waveform realizations. The amplitude of $T(t)$ (envelope) is Rayleigh distributed. In this work, we use $N_c = 16$.

4.2 Sum-Product algorithm on Rayleigh fading channel

Let us consider the uncorrelated (independent) Rayleigh fading channel first. Similar to the communication scenario on AWGN channel, each bit v_i of a codeword $\mathbf{v} = (v_1, v_2, \dots, v_n)$ is mapped into the bipolar signal $x_i = (-1)^{v_i}$. And then $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is transmitted over an uncorrelated Rayleigh fading channel. At the output of the channel, we have $\mathbf{y} = \mathbf{a}^T \cdot \mathbf{x} + \mathbf{n}$, where $\mathbf{n}$ is a Gaussian random vector independent of $\mathbf{x}$, whose elements are independent zero-mean Gaussian random variables with variance $\sigma^2 = N_0 / (2RE_b)$, in which R is the rate of the code, and E_b and $\frac{N_0}{2}$ are the average energy per information bit and the power spectral density of white noise, respectively. Vector $\mathbf{a}$ is independent of vectors $\mathbf{x}$ and $\mathbf{n}$, and each of its elements is a normalized Rayleigh fading factor and is independent and identically distributed with the pdf of (4.1).

We can apply the same sum-product algorithm described in section 2.4, except that initialization step is different [12]. We should consider two cases: the first case is that we have ideal Side Information (SI) at the receiver; the second case is no SI is available.

1) *Ideal Side Information*: When we have ideal SI, the input intrinsic information to SP decoder can be expressed as:

$$LLR_{LDPC}^{int}(v_i) = \frac{2}{\sigma^2} y_i \cdot a_i \quad (4.6)$$

2) *No Side Information*: When no SI is available, we can assume that the conditional probability density distribution of the matched filter output y is Gaussian distributed in the region of the most probable y [12], and we approximate the intrinsic information input of SP decoder as:

$$LLR_{LDPC}^{int}(v_i) = \frac{2}{\sigma^2} y_i \cdot E[a] \quad (4.7)$$

where $E[a] = 0.8862$.

Now we will see how to apply the sum-product algorithm to correlated Rayleigh fading channel. After matched filtering and proper sampling, the discrete representation of the received signal at time i is:

$$y_i = a_i \cdot x_i + n_i, \quad i = 0, 1, 2, \dots \quad (4.8)$$

where

x_i transmitted constellation point, which in this work is $\{+1, -1\}$;

n_i i.i.d (white) Gaussian noise process with zero mean, variance $\sigma^2 = N_0/2$, independent of $\{x_i\}$ and $\{a_i\}$;

a_i correlated channel fading process, generated by modified Jakes model using (4.5):

$$a_i = |T(i)| = \left| \sqrt{\frac{2}{N_c}} \sum_{n=1}^{N_c} [\cos(\beta_n + j \sin(\beta_n))] \cos(\omega_n i + \theta_n) \right|, \text{ independent of } \{x_i\} \text{ and } \{n_i\}, \text{ and}$$

its average power is unity.

To break up the correlation of channel fades, an interleaving technique is generally employed to transform the burst error channel into random error channel, so that a code designed for random error channel can be used for burst error channel. For correlated Rayleigh fading channel, as an LDPC code has a built-in interleaver, we can apply the same sum-product algorithm as on uncorrelated Rayleigh fading channel to decode LDPC codes.

We will present the simulation results in next section.

4.3 Simulation results

In this subsection, we will study the LDPC code performance on frequency flat, time correlated Rayleigh fading channel by computer simulation. We assume the carrier frequency is 900 MHz, the source rate is 9.6 kbps, and the mobile speed is 70 m/h (112km/h). Modified Jakes model described in section 4.1.1 is used to generate the time-correlated Rayleigh fading waveform.

Simulations are performed on a number of LDPC codes. For all the simulations in this subsection, we send the all-zero codeword because of the symmetry of the channel and the decoding algorithm [7]. The maximum number of iterations is chosen to be 200, and enough codewords are simulated at each SNR to generate at least 100 word errors. For low SNR, at least 1000 codewords are simulated.

The results are presented for the following three codes:

- (1) (3072,1024) irregular LDPC code
- (2) (3072, 1024) PEG LE (Linear-time encodable) LDPC code
- (3) (4000,2000) (3,6) regular LDPC code

The BER results are given for all bits (message bits and parity bits).

4.3.1 (3072,1024) irregular LDPC code

The first code used in our simulation is a (3072,1024) irregular LDPC code with rate $1/3$, which is constructed by random construction method using the degree sequence from [12], and is similar to “ir1” code in [12]. This code appears to perform very well on both AWGN channel and uncorrelated Rayleigh fading channel [12]. The TG of this code has no cycle of length 4, and it is constructed such that all degree-2 symbol nodes are cycle-free.

Figure 4-1 compares the code performance on correlated Rayleigh fading channel without using the channel interleaver to the performance with different channel interleavers. All the interleavers have size $N = 3072$. For the block interleaver, the interleaving degree is 64, and the interleaver span is 48. We use the cyclic shift to generate the convolutional interleaver, and the matrix we used has 24 rows and 128 columns, the delay of the shift register is 10. Also for S -random interleaver, to find a solution in reasonable time, the minimum interleaving distance S is chosen to be 18. We can see from the figure that different interleaving techniques provide almost the same gain, which is about 0.5dB at BER of 10^{-5} and about 0.8dB at WER of 10^{-4} . Thus we can say that applying any type of interleaver (any permutation of columns of H) results in more or less the same improvement over the original LDPC code. The reason is that the originally built LDPC code, has a certain order for columns (according to the weight of the columns), i.e., first from left to right, all the columns with the highest weight are filled, then all the columns with the second highest weight and so on. So in the original construction, all the symbol nodes of the same degree are adjacent.

There are totally 20,868 cycles in the shortest cycle set $\mathfrak{R}$, for the original TG (repeated cycles are removed), 16.8%, 74.45% and 8.75% of them have length 6, 8 and 10 cycles, respectively. The BER and WER curves with the optimized interleavers are shown in Figure 4-2. The code performance with block and random interleavers is also plotted in the same figure. It can be seen that both algorithms 1A and 2A have achieved about 0.5dB gain at BER of 10^{-5} . Algorithm 1A has achieved about 0.8dB at WER of 10^{-4} while algorithm 2A performs slightly worse. We also see that with the application of optimization algorithms, we are able to achieve almost the same performance as the application of other interleaving techniques. We may note that after applying the optimization algorithms or other interleaving techniques, we still have the same number of cycles, and the symbol nodes participating in the cycles are the same but are labeled differently (have been permuted), thus the distance distribution is different. Figure 4-3 to Figure 4-8 show the distance distribution for this code with block interleaver, convolutional interleaver, random interleaver, S -random interleaver, and with the application of algorithms 1A and 2A, respectively. From these figures, we can see that the distance distribution is improved, and the average distance is increased both with the optimized interleavers and other interleavers.

It is interesting to notice that by applying the optimization algorithms 1A and 2A to this code, the shortest cycle distance of any symbol node is greater than the minimum interleaving distance S for S -random interleaver.

The statistics of the number of required decoding iterations for convergence is given in Table 4-1. The percentage of undetected and detected errors is also listed in Table 4-2. Since swapping the columns of parity-check matrix H does not change the strength of the

LDPC code, it can be seen that by applying the optimization and other interleaving techniques, we have slightly lower decoding complexity. Table 4-2 also suggests that with the application of interleavers, the percentage of undetected errors remains more or less the same.

As we mention in the beginning of this subsection, we assume that the carrier frequency is 900MHz, the source rate is 9.6kbps, and the mobile speed is 70 m/h. Because the code rate is 1/3 and the codeword block size is 3072, the corresponding normalized Doppler frequency $f_m T_s$ is 3.24×10^{-3} . To investigate the “interleaving gain” under different channel parameters, we also performed simulations using different values of $f_m T_s$ and the simulation results are presented in Figure 4-9 and Figure 4-10. The code performance on uncorrelated (independent) Rayleigh fading channel is also shown in the same figures. Although we adopt modified Jakes model in this work, we use equation (4.2) to compute the level crossing rate approximately, the number of level crossings for $\rho = 1/\sqrt{2}$ is 107.5 crossings per second if f_m is 100Hz. And using equation (4.4) we can obtain the average number of coded symbols (bits) that are in fade: 3.7, 37, 370, 3700 successive symbols (bits) are in fade for $f_m T_s = 0.1, 0.01, 0.001, 0.0001$, respectively. If the value of $f_m T_s$ becomes infinite, the correlation between the channel fades is zero, and the channel is equivalent to the uncorrelated Rayleigh fading channel. At the other extreme, if the value of $f_m T_s$ becomes zero, the correlation between channel fades is one ($a_i = 1$, for every i). In both cases, interleaving techniques fail to provide any “interleaving gain”. However, if the value of $f_m T_s$ is between zero and infinity, interleaving techniques would be able to achieve some “interleaving gain” as it can be seen from Figure 4-9 and Figure 4-10. For the curves in Figure 4-9 and Figure 4-10, as the value of $f_m T_s$ becomes smaller,

the channel correlation becomes greater, the code performs worse, and the interleaving gain increases, e.g. at $f_m T_s = 0.1$, optimized interleaver and block interleaver provide no gain while at $f_m T_s = 0.001$, both optimized interleaver and block interleaver can provide about 0.3dB gain at BER of 10^{-4} , and about 0.5dB gain at WER of 10^{-3} (Note that this trend cannot go on forever because as $f_m T_s$ goes to zero the interleaving gain should go to zero too!). We also see that the optimized interleaver obtained by algorithm 1A and block interleaver provide almost the same gain for different $f_m T_s$ values. Interestingly, this code performs almost the same on correlated Rayleigh fading channel with $f_m T_s = 0.1$ as it does on uncorrelated Rayleigh fading channel.

In the above simulations, we assume that the ideal side information (SI) is available. Figure 4-11 shows the code performance on correlated Rayleigh fading channel when no SI is available. It can also be seen that optimized interleaver obtained by algorithm 1A provides a significant improvement in performance over no interleaver case, i.e., we can observe about 0.6dB gain at BER of 10^{-4} and 0.75dB gain at WER of 10^{-3} with the application of algorithm 1A. However, the optimized interleaver provides almost the same gain as block interleaver and random interleaver.

Figure 4-1: (3072,1024) irregular LDPC code performance- No interleaver vs. with interleaver, for different interleavers

Figure 4-2: (3072,1024) irregular LDPC code performance- optimized vs. other interleavers

Figure 4-3: Distance distribution of (3072,1024) irregular LDPC code (with block interleaver)

Figure 4-4: Distance distribution of (3072,1024) irregular LDPC code (with conv. int.)

Figure 4-5: Distance distribution of (3072,1024) irregular LDPC code (with rand. int.)

Figure 4-6: Distance distribution of (3072,1024) irregular LDPC code (with s-rand. int.)

Figure 4-7: Distance distribution of (3072,1024) irregular LDPC code (algo.1A)

Figure 4-8: Distance distribution of (3072,1024) irregular LDPC code (algo.2A)

Figure 4-9: BER curves for (3072,1024) irregular LDPC code- for different channel parameters

Figure 4-10: WER curves for (3072,1024) irregular LDPC code- for different channel parameters

Figure 4-11: (3072,1024) irregular LDPC code performance without SI- for different interleavers

Table 4-1: Statistics of the number of decoding iterations for (3072,1024) irregular code with SI (converged case)

SNR (dB)	no int.		blk. int.		conv. int.		rand. int.	
	mean	std. dev.	mean	Std. dev.	mean	std. dev.	mean	std. dev.
2.0	28.8	20.1	25.4	18.8	24.6	16.2	25.2	17.3
3.0	21.5	18.9	17.2	12.7	16.9	11.4	17.3	13.2
4.0	14.0	14.9	11.5	6.5	11.5	6.6	11.5	6.5
5.0	8.0	4.9	8.3	3.5	8.3	3.5	8.3	3.6
6.0	5.6	2.0	6.5	2.2	6.5	2.2	6.5	2.2
7.0	4.3	1.1	5.2	1.6	5.2	1.6	5.2	1.6

SNR (dB)	s-rand. int.		algo.1A		algo.2A	
	mean	std. dev.	mean	std. dev.	mean	std. dev.
2.0	25.5	18.2	26.4	19.7	26.8	20.2
3.0	17.4	13.2	17.5	11.6	18.0	13.0
4.0	11.5	6.4	11.9	7.5	12.0	7.4
5.0	8.3	3.6	8.6	3.9	8.7	3.9
6.0	6.5	2.3	6.7	2.4	6.7	2.5
7.0	5.2	1.6	5.4	1.7	5.5	1.8

Table 4-2: Breakdown of error events for (3072,1024) irregular code with SI

SNR (dB)	no int.		blk. int.		conv. int.		rand. int.	
	det.	undet.	det.	undet.	det.	undet.	det.	undet.
2.0	100%	0%	100%	0%	100%	0%	100%	0%
3.0	100%	0%	100%	0%	100%	0%	100%	0%
4.0	100%	0%	100%	0%	100%	0%	100%	0%
5.0	99%	1%	98%	2%	100%	0%	100%	0%
6.0	99%	1%	98%	2%	100%	0%	100%	0%
7.0	100%	0%	98%	2%	99%	1%	100%	0%

SNR (dB)	s-rand. int.		algo.1A		algo.2A	
	detected	undetected	detected	undetected	detected	undetected
2.0	100%	0%	99.5%	0.5%	99.5%	0.5%
3.0	100%	0%	100%	0%	100%	0%
4.0	100%	0%	100%	0%	99%	1%
5.0	100%	0%	99%	1%	100%	0%
6.0	100%	0%	99%	1%	99%	1%
7.0	99%	1%	99%	1%	98%	2%

(Note: det. = detected, undet. = undetected)

4.3.2 (3072,1024) PEG LE LDPC code

We also performed simulations on a (3072,1024) irregular LDPC code constructed by deterministic PEG algorithm. This code has linear-time encoding property, the parity-check matrix H is an upper-triangular matrix, and the symbol node degree sequence is the same as the previous irregular code except that one symbol node has degree one. The TG of this code has no cycle of length 4 and 6.

We point out here that with the application of optimization algorithms, the columns of the parity-check matrix H are swapped, however, the linear-time encoding property is preserved while the encoder design is more complicated.

There are totally 54,087 cycles in the shortest cycle set $\mathfrak{R}$, in the original TG (repeated cycles are removed), 98.3% of them have length 8 while 1.7% of them have length 10. The BER and WER curves with optimized interleavers on correlated Rayleigh fading channel with SI are shown in Figure 4-12. The BER and WER curves for this code with block interleaver and random interleaver are also plotted in the same figure. Without interleaver, this code performs badly on correlated Rayleigh fading channel. The performance is improved significantly with the application of interleaving techniques. Interleavers obtained by algorithm 1A, 2A and random interleaver perform almost the same, there is about 1.5dB gain at BER of 10^{-4} and about 2.5dB at WER of 10^{-2} , while block interleaver provides additional 0.1dB gain. We can see that by performing optimization, this code performs almost the same as the previous irregular LDPC code. Note that this code has the additional advantage of being linear-time encodable.

Figure 4-13 to Figure 4-16 show the distance distribution for this code after the application of algorithm 1A, 2A, block interleaver and random interleaver, respectively. From

these figures, we can see that the distance distribution is significantly improved, and the average distance is increased drastically.

The statistics of the number of required decoding iterations for convergence is given in Table 4-3. This table indicates that the decoding complexity is significantly reduced with the application of interleaving techniques both using optimization algorithms and existing interleavers. The percentage of undetected and detected errors is listed in Table 4-4. It can be seen from the table that by applying interleaving techniques, we have significantly reduced undetected errors.

Figure 4-17 shows the code performance on correlated Rayleigh fading channel when no SI is available. The same trend can be observed as in the SI is available case. Without interleaver, this code performs badly on correlated Rayleigh fading channel. The performance is improved significantly with the application of interleaving techniques. Interleaver obtained by Algorithm 2A and random interleaver perform almost the same, there is about 1.5dB gain at BER of 10^{-3} and about 3.0dB at WER of 2×10^{-2} , while block interleaver provides additional 0.2dB gain. We can see that by performing optimization, this code performs almost the same as the previous irregular LDPC code while this code has additional advantage of being linear-time encodable.

Figure 4-12: (3072,1024) PEG LE code performance on correlated Rayleigh fading channel with SI-
for different interleavers

Figure 4-13: Distance distribution of (3072,1024) PEG LE LDPC code (algo.1A)

Figure 4-14: Distance distribution of (3072,1024) PEG LE LDPC code (algo.2A)

Figure 4-15: Distance distribution of (3072,1024) PEG LE LDPC code (blk. int.)

Figure 4-16: Distance distribution of (3072,1024) PEG LE LDPC code (rand. int.)

Figure 4-17: (3072,1024) PEG LE code performance on correlated Rayleigh fading channel without SI- for different interleavers

Table 4-3: Statistics of the number of decoding iterations for (3072,1024) PEG LE code with SI (converged case)

SNR (dB)	no int.		algo.1A		algo.2A	
	mean	std. dev.	mean	std. dev.	mean	std. dev.
2.0	43.2	26.1	25.7	18.9	26.6	18.5
3.0	35.5	23.8	17.3	12.2	18.2	12.2
4.0	28.4	19.7	11.6	6.2	12.5	6.9
5.0	22.6	13.7	8.6	3.6	9.2	4.0
6.0	19.3	11.8	6.7	2.3	7.2	2.6
7.0	16.6	10.0	5.4	1.7	5.9	1.9

SNR (dB)	blk. int.		rand. int.	
	mean	std. dev.	mean	std. dev.
2.0	26.7	20.6	24.7	17.3
3.0	18.2	12.9	16.7	11.7
4.0	12.2	7.2	11.2	6.2
5.0	8.9	4.3	8.1	3.5
6.0	6.9	2.8	6.3	2.2
7.0	5.6	2.0	5.1	1.6

Table 4-4: Breakdown of error events for (3072,1024) PEG LE code with SI

SNR (dB)	no int.		algo.1A		algo.2A	
	detected	undetected	detected	undetected	detected	undetected
2.0	99%	1%	100%	0%	100%	0%
3.0	97.5%	2.5%	100%	0%	100%	0%
4.0	92%	8%	100%	0%	100%	0%
5.0	91%	9%	100%	0%	100%	0%
6.0	94%	6%	100%	0%	99%	1%
7.0	95%	5%	100%	0%	98%	2%

SNR (dB)	blk. int.		rand. int.	
	detected	undetected	detected	undetected
2.0	100%	0%	100%	0%
3.0	100%	0%	100%	0%
4.0	100%	0%	100%	0%
5.0	100%	0%	100%	0%
6.0	100%	0%	100%	0%
7.0	96%	4%	100%	0%

4.3.3 (4000,2000) (3,6) regular LDPC code

Simulations are also performed on a (4000,2000) (3,6) regular LDPC code constructed by Poisson random construction [35]. The TG of this code has no cycles of length 4.

There are totally 8,823 cycles in the shortest cycle set $\mathcal{R}_s$ on the original TG (repeated cycles are removed), 1.9%, 13.8%, 84.3% of them have length 6, 8, and 10, respectively. Figure 4-18 and Figure 4-19 show the distance distribution for this code after the application of algorithms 1A and 2A, respectively. There are totally 11,358 cycles in the all cycle set $\mathcal{R}_a$ on the original TG (repeated cycles are removed), 1.5%, 10.7%, and 87.8% of them have length 6, 8, and 10, respectively. Figure 4-20 and Figure 4-21 show the distance distribution for this code after the application of algorithms 1B and 2B, respectively. We also show the distance distribution for this code with block interleaver and random interleaver in Figure 4-22 and Figure 4-23, respectively. From these figures, we can see that the distance distribution is improved, and the average distance is increased. We would expect some gain with the application of optimization algorithms. Figure 4-24 shows the BER and WER curves for this code on correlated Rayleigh fading channel with SI. The BER and WER curves for this code with block interleaver and random interleaver are also plotted in the same figure. It can be seen that the application of interleaving techniques provides negligible gain. One main reason for that may come from the fact the original TG has already have a good distance distribution as evident from the figures. We also notice that all the symbol nodes have the same degree and therefore are equally protected by the check nodes. Thus, unlike the case for irregular codes, the permutation of the symbol nodes would not make any difference in terms of spreading burst errors among nodes with different degrees.

The statistics of the number of required decoding iterations for convergence is given in Table 4-5. This table indicates that the decoding complexity is almost the same with

the application of interleaving techniques. No undetected errors are observed in all cases for this code.

Figure 4-18: Distance distribution of (4000,2000) regular LDPC code (algo.1A)

Figure 4-19: Distance distribution of (4000,2000) regular LDPC code (algo.2A)

Figure 4-20: Distance distribution of (4000,2000) regular LDPC code (algo.1B)

Figure 4-21: Distance distribution of (4000,2000) regular LDPC code (algo.2B)

Figure 4-22: Distance distribution of (4000,2000) regular LDPC code (blk. int.)

Figure 4-23: Distance distribution of (4000,2000) regular LDPC code (rand. int.)

**Figure 4-24: (4000,2000) regular code performance on correlated Rayleigh fading channel with SI-
for different interleavers**

**Table 4-5: Statistics of the number of decoding iterations for (4000,2000) regular code with SI (con-
verged case)**

SNR (dB)	no int.		algo.1A		algo.2A		algo.1B	
	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.
2.0	28.8	20.1	31.1	23.1	32.0	29.2	30.0	20.7
3.0	21.5	18.9	20.3	14.6	22.6	21.5	21.2	19.2
4.0	14.0	14.9	13.0	9.3	13.3	11.6	13.1	10.5
5.0	8.0	4.9	7.8	4.3	7.9	5.1	7.9	4.7
6.0	5.6	2.0	5.5	2.1	5.5	2.0	5.5	1.8
7.0	4.3	1.1	4.2	1.1	4.3	1.1	4.2	1.0

SNR (dB)	algo.2B		blk. int.		rand. int.	
	mean	std. dev.	mean	std. dev.	mean	std. dev.
2.0	27.6	18.8	30.3	24.5	32.4	29.0
3.0	21.2	17.5	22.7	20.8	21.7	18.7
4.0	13.2	11.2	13.4	11.4	13.3	10.6
5.0	7.9	4.4	8.0	4.7	8.0	5.1
6.0	5.5	1.9	5.6	1.9	5.6	2.0
7.0	4.2	1.1	4.3	1.2	4.3	1.1

4.4 Discussion

From the simulation results in section 4.3, we observe that there are cases where an interleaver with smaller shortest cycle distance has a better performance compared to another interleaver with larger shortest cycle distance. This indicates that the measure of shortest cycle distance may not be a very good measure for designing interleavers for LDPC codes on correlated Rayleigh fading channel. We also see that the application of interleaving techniques, both the optimized interleavers developed in this work and existing interleavers, to regular LDPC codes provides negligible gain, while for irregular LDPC codes, different interleavers provide some noticeable “interleaving gain”. This tempts us to believe that degree distribution would be a good measure for designing interleavers. As reference [18] pointed out, there is a “wave effect” in the decoding process of irregular LDPC codes, i.e. the symbol nodes with high degrees tend to converge very quickly, and in turn, they provide more reliable information to the check nodes and subsequently to the symbol nodes with lower degrees. If many high-degree nodes (we call them “elites”) go to fade at the same time, they generate many unreliable messages in the graph at the beginning of the decoding algorithm. Since only a single unreliable extrinsic

message at the input of a check node makes the corresponding output unreliable, this could have a devastating effect on the performance of iterative decoding, thus we want to separate the high-degree symbol nodes far apart such that they are not in error or unreliable at the same time. One way to do this is to distribute them uniformly or almost uniformly among low-degree symbol nodes, thus the distance between high-degree symbol nodes is maximized, and we would expect better performance.

To verify this conjecture, we performed optimization on the (3072,1024) irregular code described in section 4.3.1 as follows. There are 210, 26, 180 symbol nodes with degrees 16, 8, 6, respectively. We first uniformly distribute 210 symbol nodes with degree 16 over 3072 positions, i.e., we make the distance between adjacent nodes with degree 16 almost equal. Then uniformly distribute 26 symbol nodes with degree 8 over the 2862 (= 3072-210) remaining positions while taking care that no overlaps with the previous assignments occur. We repeat this procedure for 180 symbol nodes of degree 6. Simulation results are presented in Figure 4-25. It can be seen that this code with this optimized interleaver has almost the same WER performance as with the optimized interleavers obtained by algorithm 1A and 2A, while it has a slightly better BER performance. We may conclude that degree distribution is not a very good measure for designing interleavers for LDPC codes either or one may think that all these “non-trivial” interleavers are almost equally good and one cannot find any interleaver that performs much better than the ones studied in this thesis. This is a topic that requires more investigation.

Another measure we have considered is the distance among symbol nodes connected to the same check node. As we know in the decoding process of LDPC codes, when symbol node S_i (represents a bit of the codeword) updates its messages from check node C_j , it

only makes use of the information from those symbol nodes participating in check node C_j . Because the parity-check matrix H of the LDPC code is usually constructed by random construction, the probability that these symbol nodes will be in close adjacency of symbol node S_i is small. However, a randomly constructed code may still have problems if neighboring symbol nodes of a check node are closely located. To make sure that neighboring symbol nodes of a check node are not closely located, we need to maximize the distances among them. We also performed simulations on the (3072,1024) irregular code to verify this idea, our results suggest that this measure is not able to provide improvements over optimization algorithms and existing interleaving techniques.

Finding more appropriate measures for interleaver design for LDPC codes or demonstrating that any other measure is almost as good as the one discussed here remain as a topic for future research.

Figure 4-25: (3072,1024) code performance- optimized interleavers

Chapter 5 - LDPC Code Optimization for

ISI channels

Many practical communication systems (e.g. mobile communication systems) encounter the problem of data transmission over a channel with inter-symbol interference (ISI). With ISI, the transmitted signal experiences non-trivial impairment and the received sample depends not only on the transmitted signal, but also on the delayed versions of the transmitted signal. As a consequence, it is necessary for the optimal receiver to deal with this phenomenon in order to achieve acceptable performance.

To combat the detrimental effect of ISI, equalization techniques such as linear equalizer (LE), decision feedback equalizer (DFE), maximum likelihood sequence estimation (MLSE), maximum *a posteriori* (MAP) equalizer, etc. are generally employed [37]. On the other hand, to protect the integrity of the data to be transmitted, a controlled amount of redundancy is introduced (encoding) using an error correction code (ECC). Conventionally, equalization and channel coding are done separately; however, this is suboptimal because the knowledge about the structure on the transmitted symbols imposed by the ECC is not exploited by the equalizer. As a result, it introduces a substantial ISI penalty compared to data transmission over an ISI-free channel. Turbo equalization [50] was proposed by Douillard et al. in 1995 to compensate for this penalty. It was demonstrated that the turbo equalizer was capable of mitigating the effect of ISI, provided that the channel impulse response was known.

In analogy to turbo decoding, soft information is iterated between the equalizer and the decoder, which results in a joint operation of equalization and channel decoding and significant improvements in bit error rate (BER) performance.

In this chapter, we will study the principle of turbo equalization with LDPC codes and apply the message-passing algorithm to describe the iteration between the equalizer and the decoder. The system block diagram is shown in Figure 5-1.

Figure 5-1: System Block Diagram

The transmitter consists of an encoder, an optional interleaver and a signal mapper. The major components of the receiver (turbo equalizer) are equalizer and decoder. The channel sample is corrupted by Additive White Gaussian Noise (AWGN). In this chapter, we will also restrict our focus on binary LDPC codes, and we consider Binary Phase Shift

Keying (BPSK) mapping. To establish turbo equalizer performance benchmark, ISI channel is equalized using symbol-by-symbol Maximum a posteriori (MAP) algorithm (also known as BCJR algorithm) [51], [52], and we apply sum-product (SP) algorithm for the decoder. Please note that the interleaver and the de-interleaver blocks are optional considering the “built-in interleaver” in the parity-check matrix H . The principle of turbo equalization will be studied in section 4.3. Here is a brief summary:

The MAP equalizer receives channel samples (no prior information for the bits at this iteration) and produces output probabilities that are then passed as the prior information onto the LDPC decoder. The LDPC decoder operates on this data using sum-product algorithm, generating extrinsic information that can be passed back to the MAP equalizer as prior information. In this way, soft information is iterated between the MAP equalizer and the LDPC decoder, which results in a joint decoding of channel and code but with much less complexity than a maximum-likelihood decoder.

5.1 ISI channel model

Following the analysis in [37], the ISI channel can be represented by a linear filter for the ISI, followed by additive white Gaussian noise (AWGN), as shown in Figure 5-2. The discrete representation of the received signal y_i can be expressed as:

$$y_i = \sum_{l=0}^L f_l x_{i-l} + n_i, \quad i = 0, 1, 2, \dots \quad (4.1)$$

where x_i is the transmitted symbol, which in this work is $\{+1, -1\}$; n_i is the AWGN sample with zero mean, variance σ^2 . We assume that we have perfect knowledge of channel taps

$f_0, f_1, f_2, \dots, f_L$, and channel taps have been normalized such that $\sum_{l=0}^L |f_l|^2 = 1$, where L is

the channel ISI length. The ISI channel can be represented by a trellis, which we will describe in next section.

Figure 5-2: Discrete time channel model of ISI with AWGN

5.2 MAP (BCJR) algorithm

The MAP (BCJR) algorithm [51], [52], also known as forward-backward algorithm, is a specific application of the message-passing algorithm when the underlying code structure can be represented by a trellis. It involves the passing of messages in the forward and backward directions along the linear graph (trellis) to calculate extrinsic probabilities based on intrinsic probabilities and the constraint that the sequence corresponds to a path through the trellis. The BCJR algorithm can be applied to a wide variety of decoding problems, such as the decoding of convolutional codes and turbo codes, the decoding of ISI channel, etc. We will see how to apply the BCJR algorithm to decode ISI channels in this subsection (we may also refer to this as MAP equalization). A trellis has its generalized factor graph [53] representation as shown in Figure 5-3. The figure only show a single trellis section, where s_i is the state variable, u_i is the output variable, x_i is

the input variable, and T_i is the trellis check node. We use α and β to describe the forward and backward messages respectively, and γ is used to represent the intrinsic message for each section of the trellis, which are defined as follows:

$$\alpha_i(s_i) = P(s = s_i, y_1^i) \quad (4.2)$$

$$\beta_i(s_i) = P(y_{i+1}^{N-1} | s = s_i) \quad (4.3)$$

where $N = n+L$, and n is the block length of the LDPC code.

$$\gamma_i(s_{i-1}, s_i) = P(s = s_i, y_i | s' = s_{i-1}) \quad (4.4)$$

Figure 5-3: A trellis section and its associated factor graph

For the sake of brevity, we summarize the BCJR algorithm for decoding the ISI channel as follows [54]:

Step 0: Initialization. Using additional L bits to terminate the trellis at zero state, we know that the trellis starts and ends at zero state, thus the forward and backward messages α and β are initialized to:

$$\alpha_0(0) = 1 \text{ and } \alpha_0(s) = 0 \quad \forall s \neq 0 \quad (4.5)$$

$$\beta_{N-1}(0) = 1 \text{ and } \beta_{N-1}(s) = 0 \quad \forall s \neq 0 \quad (4.6)$$

Step 1: Calculate the intrinsic information (branch metric). The intrinsic information input to the MAP equalizer can be expressed as:

$$\gamma_i(s_{i-1}, s_i) = \gamma'_i(s_{i-1}, s_i) \cdot \exp\left(\frac{1}{2} \cdot x_i \cdot L(x_i)\right) \quad (4.7)$$

where

$$\gamma'_i(s_{i-1}, s_i) = \exp\left(-\frac{1}{2\sigma^2} \cdot \left|y_i - \sum_{l=0}^L f_l \cdot x_{i-l}\right|^2\right) \quad (4.8)$$

and $L(x_i)$ is the a prior information for LLR of x_i , $i = 0, 1, 2, \dots, N-1$. It is first set to zero and then to the messages passed from the decoder.

Step 2: Forward/backward message passing. The forward/backward message passing can be expressed naturally as recursive formulae:

$$\alpha_i(s_i) = \sum_{s_{i-1}} \gamma_i(s_{i-1}, s_i) \cdot \alpha_{i-1}(s_{i-1}) \text{ and } i = 1, 2, \dots, N-1 \quad (4.9)$$

$$\beta_{i-1}(s_{i-1}) = \sum_{s_i} \gamma_i(s_{i-1}, s_i) \cdot \beta_i(s_i) \text{ and } i = N-1, N-2, \dots, 1 \quad (4.10)$$

Step 3: Compute output. The output of the MAP equalizer is the *a posteriori* probability which can be expressed as LLR based on:

$$L(\hat{x}_i) = L(x_i) + L^E(\hat{x}_i) \quad (4.11)$$

where

$$L^E(\hat{x}_i) = \log \frac{\sum_{\substack{(s_{i-1}, s_i) \\ x_i = +1}} \gamma_i'(s_{i-1}, s_i) \cdot \alpha_{i-1}(s_{i-1}) \cdot \beta_i(s_i)}{\sum_{\substack{(s_{i-1}, s_i) \\ x_i = -1}} \gamma_i'(s_{i-1}, s_i) \cdot \alpha_{i-1}(s_{i-1}) \cdot \beta_i(s_i)} \quad (4.12)$$

$L^E(\hat{x}_i)$ is the extrinsic information output of the MAP equalizer, which can be passed onto the SP decoder as the prior information.

5.3 Message passing between the MAP equalizer and the SP decoder

Now we can describe the message passing between the MAP equalizer and the SP decoder as shown in Figure 5-4. The figure illustrates the main concept of iterative equalizing/decoding system, i.e., passing information between modules: the input to each module is the prior information, the output is the extrinsic information, and this extrinsic information is used as the prior information for the next module. The prior information for a given module is equal to the sum of the extrinsic information from other modules.

As shown in the figure, we use LeD to represent the extrinsic messages from the decoder in LLR format, and LeE represents the extrinsic messages from the equalizer from the decoder's view.

The extrinsic information from the SP decoder to the MAP equalizer is initially set to a uniform distribution, because we are using log-likelihood ratios (LLRs), the messages are set to $LeD^{(0)} = \mu_{s_i \rightarrow T_i} = 0$, where the superscript indicates the iteration number. Using the messages $LeD^{(0)}$ as the intrinsic information, along with the received channel samples y_i , and the perfect knowledge of the channel taps $f_0, f_1, f_2, \dots, f_L$, the MAP equalizer ap-

plies the BCJR algorithm as described in section 5.2 to obtain the first estimate of the extrinsic information $LeE^{(1)} = \mu_{T_i \rightarrow S_i}$, which are passed onto the SP decoder.

Figure 5-4: Message passing between the MAP equalizer and the SP decoder

Taking these messages $LeE^{(1)}$ as the intrinsic information for the codeword bits v_i , i.e., $LLR_{LDPC}^{int}(v_i) = LeE^{(1)}$, the SP decoder applies the message-passing algorithm. This can be a single iteration (from symbol to check nodes and then from check to symbol nodes), or it can be multiple iterations of the LDPC decoding algorithm. Based on the iteration(s), the decoder obtains extrinsic messages $LeD^{(1)}$, which it passes back to the MAP equalizer.

Taking these messages $LeD^{(1)}$ as the intrinsic information, the MAP equalizer applies the BCJR algorithm again to produce the updated output messages $LeE^{(2)}$. These messages are then used by the SP decoder as the new intrinsic information. For the k -th iteration, the SP decoder takes the input messages $LeE^{(k)}$ and generates the output messages $LeD^{(k)}$. The MAP equalizer then takes these messages as input, and generates the updated $LeE^{(k+1)}$ as output, and so on. In this way, the MAP equalizer and the SP decoder cooperate to produce a joint decoding of both the channel and the code with much less complexity than the joint maximum-likelihood decoder.

At any point in this process (e.g., after the first iteration, or after a reasonable number of iterations), it is possible to estimate the posterior information for the codeword bits:

$$L(\hat{v}_i) = LeE^{(k)}(v_i) + LeD^{(k)}(v_i) \quad (4.10)$$

The sign of the LLR value gives a bit-by-bit decision for the codeword bits $\{v_i\}$.

$$\hat{v}_i = \begin{cases} 0, & \text{if } L(\hat{v}_i) \geq 0 \\ 1, & \text{if } L(\hat{v}_i) < 0 \end{cases} \quad (4.11)$$

We summarize the message passing between the MAP equalizer and the SP decoder as follows.

Step 0: Initialization. The message from the SP decoder to the MAP equalizer is set to a uniform distribution $LeD^{(0)} = 0$ (LLR format). The received signal is y_i . Set the iteration number $k = 1$.

Step 1: MAP equalizer. The inputs are messages $LeD^{(k)}$ from the SP decoder, the received signals $\{y_i\}$, and perfect knowledge of the channel taps $f_0, f_1, \dots, f_L$; the outputs are messages $LeE^{(k)}$ to the LDPC decoder.

Step 2: LDPC decoder. The inputs are messages $LeE^{(k)}$ from the MAP equalizer and the outputs are messages $LeD^{(k+1)}$ to the MAP equalizer.

Step 3: Repeat until done. If the stopping criteria have not been satisfied, increment $k \leftarrow k + 1$, and go to step 1. The stopping criteria are the same as the sum-product algorithm on AWGN channel: if the decoded word $\hat{v}$ is a codeword or the maximum number of iteration has been reached.

5.4 LDPC code performance on ISI channel

In this section, we will study the LDPC code performance on ISI channel by computer simulation. There has been some previous work on using LDPC codes in turbo equalization framework on ISI channels, e.g. [56] - [59], where LDPC codes are applied to magnetic recording channels or partial response channels. In this work, we study some ISI channels encountered in mobile communication systems. The ISI channels considered in this work are shown in Figure 5-5.

Please note that ISI channel is not symmetric [7], thus we can not assume that the all-zero codeword is always sent in our simulations and encoding process must be considered.

The first code used in our simulation is a (1920,640) LDPC code, which is constructed by Poisson random construction 2A. This code has code rate $R = 1/3$, and it has uniform check node degree, i.e., all check nodes have degree 4, 1/3 of symbol nodes have degree 2 and 2/3 of symbol nodes have degree 3 [35]. The BER performance on Proakis B ISI channel (Figure 5-5 a) is shown in Figure 5-6. In our simulation, the maximum iteration number for the SP decoder is 200 and we performed fixed 5 iterations between the MAP equalizer and the SP decoder. The encoder is using systematic encoding using generator matrix G , and a random interleaver with size $N = 1920$ is used as the channel interleaver. From the figure, we can see that we have achieved about 1dB gain in the 2nd iteration, and about 1.7dB in the 5th iteration at BER of 10^{-4} . This is the “turbo processing gain”. However, there is still about 1.5 dB penalty introduced by ISI after 5th iteration, as indicated by the gap between the ISI-free curve and the 5th iteration curve at BER of 10^{-4} . Can we eliminate ISI completely using turbo equalization technique? Let us look at another example.

Figure 5-5: ISI channel models

The second code is a (1400,800) irregular LDPC code constructed by PEG algorithm using the following symbol node degree distribution from [22], which is optimized for AWGN channel:

$$\lambda(x) = 0.47532x^2 + 0.279537x^3 + 0.0348672x^4 + 0.108891x^5 + 0.101385x^{15}$$

This code has linear time encoding property, and its parity-check matrix H has an upper-triangular structure. Figure 5-7 shows the BER performance of this code on 2-tap ISI channel (Figure 5-5 b). The BER curves for MAP equalizer without coding and no ISI and no coding are also shown in the same figure, labeled by “uncoded ISI” and “uncoded AWGN”, respectively. These two curves are almost matched, which indicate that 2-tap ISI channel is a very mild ISI channel. In our simulation, the maximum iteration number for the SP decoder is 500 and we performed fixed 10 iterations between the MAP equalizer and the SP decoder and a random interleaver with size $N = 1400$ is used as the channel interleaver. From the figure, we can see that after the 10th iteration, turbo equalizer is able to almost completely eliminate ISI. This observation agrees with the results in [55], where convolutional codes are employed in the MAP-based turbo equalizer, and ISI penalty can be almost totally compensated by turbo processing.

We want to make a comment on a subtle point in the implementation of the turbo equalizer employing LDPC codes. As described in section 2.4.1, for the sum-product algorithm on AWGN channel, the initial messages sent from check nodes to symbol nodes are all set to 0 (LLR format), but on ISI channel, it's possible to preserve the check to symbol node messages from the previous iteration [56]. That is, on the first iteration between the MAP equalizer and the SP decoder, the initial messages from check to symbol nodes are all started with 0, but from 2nd iteration on, the initial messages from check to

symbol nodes are not 0, the values are taken from the most recent results in previous iteration. On the other hand, we can take the same approach as on AWGN channel case: reset the check to symbol messages to 0 whenever a BCJR is performed [57]. This is equivalent to resetting the LDPC part of the “supergraph”, the combined ISI channel trellis and LDPC code factor graph as shown in Figure 5-4. Thus, the memory that was used to store the initial messages can be reused during the BCJR step. In practice, this could result in a substantial saving in memory. Figure 5-8 compares the code performance using these two approaches on Proakis B ISI channel. The LDPC code used here is a (504,252) code, which is a PEG-constructed regular LDPC code, all symbol nodes have degree 3 and most of the check nodes have degree 6, except for 8 check nodes that have degree 7 and 8 check nodes that have degree 5. Label ‘P’ means that we preserve the initial messages. The figure is obtained by performing fixed 8 channel iterations (the iteration between the MAP equalizer and the SP decoder) and fixed 80 decoder iterations. We can see from the figure that these two approaches have almost the same performance.

As we know, a codeword of an LDPC code must satisfy the following parity check equations: $\mathbf{v}H^T = 0$, where $\mathbf{v}$ is the codeword vector and H is the parity check matrix. This gives us a natural stopping criterion for the SP decoder: as soon as the decoded word is a valid codeword, we stop the iteration and declare the success of the decoding, otherwise, we iterate until we reach the maximum number of iterations. Thus, we can reduce the decoding complexity. In the turbo equalization context, this may be considered as an advantage of LDPC codes over convolutional codes and turbo codes where the application of stopping criteria may be complex. Some stopping criteria for convolutional codes were presented in [54].

Figure 5-6: (1920,640) regular LDPC code performance on Proakis B ISI channel

Figure 5-7: (1400,800) irregular LDPC code performance on 2-tap ISI channel

Figure 5-8: (504,252) PEG LDPC code performance on Proakis B ISI channel

5.5 LDPC code optimization for ISI channel

It is well known that ISI channels can be modeled as a finite-state Markov source. As pointed out in [11], Markov sources (i.e. ISI channel), when corrupted with memoryless noise and decoded with a symbol or sequence MAP detector, generate burst errors. Previous turbo equalization schemes employ convolutional codes or turbo codes [60]–[64] together with an interleaver after the encoder and a deinterleaver before the decoder as shown in Figure 5-1. In this section, we will demonstrate that by using the optimized interleavers developed in the previous chapter, we are able to achieve better error performance than existing interleaving techniques with similar decoding complexity.

For all the simulations in this subsection, the maximum number of channel iterations is chosen to be 8, and maximum decoder iteration is 80. Enough codewords are simulated

at each SNR to generate at least 100 word errors. For low SNR, a minimum of 1000 codewords are simulated.

Simulations are performed on a (1024,512) regular LDPC code constructed by PEG algorithm. All the 1024 symbol nodes have degree 3, and most of the check nodes have degree 6 except that 32 check nodes have degree 5 and 32 check nodes have degree 7. The TG of this code has no cycle of length 4 and 6.

There are totally 11,945 cycles in the shortest cycle set $\mathfrak{R}_s$ on the original TG (repeated cycles are removed), most of the cycles have length 10 except that 6 cycles have length 8. Figure 5-9 compares the code performance on Proakis B ISI channel without using the channel interleaver to the performance with different channel interleavers. All the interleavers have size $N = 1024$. For the block interleaver, the interleaving degree is 16, and the interleaver span is 64. We use the cyclic shift to generate the convolutional interleaver, and the matrix we used has 16 rows and 64 columns, the delay of the shift register is 8. Also for S -random interleaver, to find a solution in reasonable time, the minimum interleaving distance S is chosen to be 7. It can be seen from the figure that with the optimized interleavers obtained by algorithm 1A and 2A, we have achieved about 0.2dB gain across different SNR regions both in BER and WER. We also can see that optimized interleavers perform slightly better than existing interleavers.

Figure 5-10 to Figure 5-15 show the distance distribution for this code with the application of algorithm 1A and 2A, block interleaver, convolutional interleaver, random interleaver, and S -random interleaver, respectively. We can see from these figures that with both optimized interleavers and existing interleavers, the average distance is almost the same as in no channel interleaver case.

The statistics of the number of required channel and decoding iterations for convergence is given in Table 5-1 and Table 5-2, respectively. Please note that the decoding iterations are the sum of the SP decoder iterations, thus the maximum decoding iteration is 640 ($= 8 \times 80$). These tables indicate that the decoding complexity is almost the same with the application of the optimized interleavers or existing interleaving techniques. No undetected errors are observed in all cases for this code.

Figure 5-9: (1024,512) PEG LDPC code performance on Proakis B ISI channel- for different channel interleavers

Figure 5-10: Distance distribution of (1024,512) PEG LDPC code (algo.1A)

Figure 5-11: Distance distribution of (1024,512) PEG LDPC code (algo.2A)

Figure 5-12: Distance distribution of (1024,512) PEG LDPC code (blk. int.)

Figure 5-13: Distance distribution of (1024,512) PEG LDPC code (conv. int.)

Figure 5-14: Distance distribution of (1024,512) PEG LDPC code (rand. int.)

Figure 5-15: Distance distribution of (1024,512) PEG LDPC code (s-rand. int.)

Table 5-1: Statistics of the number of channel iterations for (1024,512) PEG code (converged case)

SNR (dB)	no int.		algo.1A		algo.2A	
	mean	std. dev.	mean	std. dev.	mean	std. dev.
3.00	3.9	1.5	3.7	1.6	3.5	1.5
3.25	3.4	1.4	3.1	1.3	3.0	1.4
3.50	2.9	1.2	2.6	1.1	2.5	1.0
3.75	2.4	1.0	2.2	0.8	2.2	0.8
4.00	2.1	0.8	1.9	0.7	1.9	0.7
4.25	1.8	0.7	1.6	0.6	1.6	0.6
4.50	1.5	0.6	1.4	0.5	1.4	0.5

SNR (dB)	blk. int.		conv. int.		rand. int.		s-rand. int.	
	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.
3.00	3.9	1.5	3.8	1.6	3.9	1.5	3.2	1.4
3.25	3.4	1.4	3.2	1.5	3.4	1.5	2.8	1.3
3.50	2.8	1.2	2.8	1.2	2.9	1.2	2.5	1.2
3.75	2.4	0.9	2.3	1.0	2.4	1.0	2.1	0.9
4.00	2.0	0.7	2.0	0.8	2.1	0.7	1.8	0.8
4.25	1.7	0.6	1.7	0.7	1.8	0.6	1.5	0.7
4.50	1.5	0.6	1.4	0.6	1.5	0.6	1.3	0.5

Table 5-2: Statistics of the number of decoding iterations for (1024,512) PEG code (converged case)

SNR (dB)	no int.		algo.1A		algo.2A	
	mean	std. dev.	mean	std. dev.	mean	std. dev.
3.00	249.6	120.7	231.6	125.4	217.1	120.8
3.25	206.2	115.8	184.1	107.5	177.3	109.3
3.50	163.7	97.5	144.1	88.2	137.0	82.5
3.75	127.5	77.7	110.1	64.6	105.3	64.1
4.00	98.7	61.2	85.0	51.2	81.4	49.9
4.25	74.3	48.9	63.4	43.4	60.9	42.6
4.50	53.6	42.6	44.1	38.1	42.7	37.4

SNR (dB)	blk. int.		conv. int.		rand. int.		s-rand. int.	
	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.
3.00	252.2	123.8	238.6	126.0	253.8	120.8	196.0	116.4
3.25	204.2	113.1	194.7	117.0	209.4	117.3	160.6	104.4
3.50	159.1	93.8	156.3	99.7	164.4	97.3	132.3	94.8
3.75	122.9	72.3	118.1	75.8	128.9	79.0	101.0	75.0
4.00	94.4	54.5	91.0	60.4	99.2	58.0	75.5	62.1
4.25	72.3	45.1	68.4	49.7	75.7	46.7	54.6	49.9
4.50	52.4	40.0	49.0	42.0	55.0	41.3	39.0	40.5

Chapter 6 - Conclusions

In this work, we investigated the relationship between the performance of LDPC codes on burst error channels and the TG structure of LDPC codes. We also compared the performance of different interleavers on burst error channels in conjunction with LDPC codes. By exploring the TG structure of LDPC codes, we designed interleavers to improve the LDPC code performance on burst error channels. And we proposed two types of optimization algorithms, random optimization and deterministic optimization, based on the concept of distance profile. We also looked into other measures for designing interleavers for LDPC codes such as the symbol node degree distribution and the distance among symbol nodes connected to the same check node. In this work, the performance of different interleavers, both the optimized interleavers developed in this thesis and existing interleavers, was evaluated through computer simulations on a number of LDPC codes over correlated Rayleigh fading channels and ISI channels.

Based on the simulation results, we can have the following general conclusions:

- The optimized interleavers and existing interleaving techniques DO provide some “interleaving gain” for LDPC codes on burst error channels.
- The optimized interleavers achieve great improvement in error performance over no interleaver case while providing almost the same gain over existing interleaving techniques.
- The “interleaving gain” is particularly large for irregular LDPC codes.

- For regular or semi-regular LDPC codes, interleaving techniques provide rather small gain.
- With the application of interleaving techniques, the decoding complexity in terms of the average number of iterations required for convergence is almost the same or lower compared to no-interleaver case. In some cases, the decoding complexity is significantly reduced.
- The application of interleaving techniques has not introduced additional undetected errors, while in some cases, undetected errors are significantly reduced.

This work shows that interleaving techniques play an important role in improving the performance of irregular LDPC codes on burst error channels, particularly at short and medium block lengths. The results of this work also suggest that all non-trivial channel interleavers are almost equally good for LDPC codes. As an attempt to study and optimize the built-in interleaver of LDPC codes, we hope that this work will inspire more interest and research in this area.

Ideas for future research

There are a few directions in which the contribution of this work can be extended. Investigating other measure(s) for designing interleavers for LDPC codes on burst error channels would be of definite interest.

Investigating the performance and the complexity of optimized and existing interleavers on other burst error channels such as Gilbert-Elliott channel, and frequency selective fading channel etc., would be another interesting topic.

Justifying the homogeneous performance of different interleavers and whether this performance can in fact be improved or not would also be one direction for future research.

In this thesis, we focused on sum-product decoding algorithm, which is considered to be the best iterative decoding algorithm in term of performance but not in term of complexity. Another well-known soft iterative decoding algorithm is Min-Sum (MS), and recently, the quantized versions of Min-Sum algorithm and its minor variations have been shown to achieve very good error performance with much lower complexity than SP algorithm [69]. Thus it would be an interesting topic for future work to investigate the performance of LDPC codes on burst error channels under these algorithms.

References

- [1] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. on Inform. Theory*, vol.IT-8, pp. 21-28, Jan. 1962.
- [2] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
- [3] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes", *Proc. 1993 IEEE Int. Conf. Communications*, Geneva, Switzerland, pp.1064-1070, May 1993.
- [4] D. J. C. Mackay and R. M. Neal, "Near Shannon limit performance of low density parity check code," *Electron. Lett.*, vol.32, no.18, pp. 1645-1646, Aug. 1996. Reprinted *Electron. Lett.*, vol.33, no.6, pp. 457-458, Mar. 1997.
- [5] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices", *IEEE Trans. Inform. Theory*, vol.45, pp.399-432, Mar. 1999.
- [6] T. J. Richardson, A. Shokrollahi, R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol.47, pp.619-637, Feb. 2001.
- [7] T. Richardson, R. Urbanke, "The capacity of low density parity-check codes under message passing decoding," *IEEE Trans. Inform. Theory*, vol.47, pp.599- 618, Feb. 2001.
- [8] M.P.C. Fossorier, M. Mihaljevic, H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Commun.*, vol.47, no.5, pp. 673-679, May 1999.

- [9] "Codes on graphs and iterative algorithms," Special issue *IEEE Trans. Inform. Theory*, vol.47, no.2, Feb. 2001.
- [10] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533-547, Sept. 1981.
- [11] A. Patapoutian, B.-Z. Shen, and P.A. McEwen, "Event error control codes and their applications", *IEEE Trans. Inform. Theory*, vol.47, pp.2595-2603, Sept. 2001.
- [12] J. Hou, P.H. Siegel, L.B Milstein, "Performance analysis and code optimization of low density parity-check codes on Rayleigh fading channels", *IEEE J. Select. Areas Comm.*, vol. 19, pp. 924-934, May 2001.
- [13] J.L. Fan, *Constrained Coding and Soft Iterative Decoding*, Kluwer Academic Publishers, 2001.
- [14] Y. Mao and A.H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," *IEEE ICC2001*, Helsinki, Finland, June 2001.
- [15] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell Syst. Tech. J.*, vol. 39, pp. 1253-1265, Sept. 1960.
- [16] E. O. Elliott, "Estimates of error rates for codes on burst-noise channel," *Bell Syst. Tech. J.*, vol. 42, pp. 1977-1997, Sept. 1963.
- [17] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," *Proc. 29th ACM Symp. Theory of Computing*, El Paso, TX, May 1997, pp. 150-159.
- [18] M. Luby, M.Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low-density codes and improved designs using irregular graphs," *Proc. 30th ACM STOC*, Dallas, TX, 1998, pp. 249-258.

- [19] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585-598, Feb. 2001.
- [20] D.J.C. Mackay, S.T. Wilson, M.C. Davey, "Comparison of Constructions of Irregular Gallager Codes", *IEEE Trans. on Comm.*, Vol.47, No.10, pp.1449-1454, Oct. 1999.
- [21] X.-Y. Hu, E. Eleftheriou and D.-M. Arnold, "Progressive edge-growth Tanner graphs," *IEEE Proc. Globecom'2001*, San Antonio, TX, Nov. 2001.
- [22] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *Submitted to IEEE Trans. Inform. Theory*.
- [23] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, vol. 47, pp. 2711-2736, Nov. 2001.
- [24] G. A. Margulis, "Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators," *Problems of Information Transmission (USA)*, vol. 24, no. 1, pp. 39-46, Jan.-March 1988; translated from *Problemy Peredachi Informatsii (USSR)*, vol. 24, no. 1, pp. 51-60, Jan.-March 1988.
- [25] A. Lubotzky, R. Phillips, and P. Sarnak, "Ramanujan graphs," *Combinatorica*, vol. 8, no. 3, pp. 261-277, 1988.
- [26] S.-Y. Chung, G.D. Forney, Jr., and T.J. Richardson, R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Comm. Lett.*, vol. 5, no. 2, pp. 58-60, Feb 2001.

- [27] L. Bazzi, T. Richardson and R. Urbanke, "Exact thresholds and optimal codes for the binary symmetric channel and Gallager's decoding algorithm A," *IEEE Trans. Inform. Theory*, to appear.
- [28] M. A. Shokrollahi, "New sequence of linear time erasure codes approaching the channel capacity," *Proc. 13th Int. Sym. on Applied Algebra, Algebraic Algorithm and Error-correcting Codes*, 1999.
- [29] D. Spielman, "Linear-time encodable and decodable error-correcting codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1723-1731, Nov. 1996.
- [30] L. Ping, W. K. Leung, and N. Phamdo, "Low density parity check codes with semi-random parity check matrix," *IEEE Elect. Lett.*, vol. 35, pp. 38-39, Jan. 1999.
- [31] H. Jin, A. Khandekar, and R. J. McEliece, "Irregular repeat-accumulate codes," *Proc. of 2nd Int. Symp. on Turbo Codes & Related Topics*, Brest, France, Sept. 2000, pp. 1-8.
- [32] T. Richardson, R. Urbanke, "Efficient encoding of Low-Density Parity-Check codes," *IEEE Trans. Inform. Theory*, vol.47, pp.638- 656, Feb. 2001.
- [33] A. Nouh and A. H. Banihashemi, "Bootstrap decoding of low-density parity-check codes," *IEEE Comm. Letters*, vol. 6, no. 9, pp. 391-393, Sept. 2002.
- [34] N. Wiberg, "Codes and decoding on general graphs," Ph. D. dissertation, Dept. Elec. Eng., Univ. Linkoping, Sweden, Apr. 1996.
- [35] D.J.C. Mackay, "Online database of low-density parity-check codes," available at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html>.

- [36] S. L. Jacoby, J. S. Kowalik, J. T. Pizzo, and W. T. Veterling, *Iterative Methods for Nonlinear Optimization Problems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.
- [37] J. G. Proakis, *Digital Communications*, 4th ed, McGraw Hill, 2000.
- [38] B. Vuketic, and J. H. Yuan, *Turbo Codes: Principles and Applications*, Kluwer Academic Publishers, 2000.
- [39] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo codes," *IEEE Trans. Commun.*, vol.44, no.10, pp. 1261-1271, Oct. 1996.
- [40] J. L. Ramsey, "Realization of optimum interleavers," *IEEE Trans. Inform. Theory*, vol.16, no.3, pp.338-345, May 1970.
- [41] G. D. Forney, Jr., "Burst-correcting codes for the classic bursty channel," *IEEE Trans. Commun. Tech.*, vol.19, no.5, pp.772-781, Oct. 1971.
- [42] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. ICC'95*, Seattle, WA, pp.54-59, June 1995.
- [43] G. D. Forney, Jr., "On iterative decoding and the two-way algorithm," in *Proc. Int. Symp. on Turbo Codes and Related Topics*, Brest, France, pp.12-25, Sept. 1997.
- [44] T. Etzion, A. Trachtenbeg, and A. Vardy, "Which codes have cycle-free Tanner graphs?" *IEEE Trans. Inform. Theory*, vol.45, pp.2173-2181, Sept. 1999.
- [45] Y. Mao and A. H. Banihashemi, "Decoding low-density parity-check codes with probabilistic scheduling," *IEEE Comm. Lett.*, vol.5, no.10, pp.414-416, Oct.2001.
- [46] H. Xiao and A. H. Banihashemi, "New schedules for decoding LDPC codes," in *Proc. Biennial Symp. on Comm. Conf.*, Kingston, Canada, pp.87-81, May 2002.

- [47] T. S. Rappaport, *Wireless Communications: Principles and Practice*, Second Edition, Prentice-Hall, New Jersey, 2002
- [48] W. C. Jakes (ed.), *Microwave Mobile Communications*, Wiley, New York, 1974.
- [49] P. Dent, G. E. Bottomley, and T. Croft, "Jakes fading model revisited," *IEEE Electron. Lett.*, vol.29, no.13, pp.1162-1163, June 1993.
- [50] C. Douillard, M. Jezequel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of intersymbol interference: Turbo--Equalization," *Eur. Trans. on Telecommun.*, vol. 6, pp. 507--511, 1995.
- [51] L. R. Bahl, J. Cocke, F. Jelinek, J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Trans. Inform. Theory*, vol.IT-20, pp.284-287, Mar. 1974.
- [52] R.W. Chang, and J.C. Hancock, "On Receiver Structures for Channels Having Memory", *IEEE Trans. Inform. Theory*, vol.IT-12, pp.463-468, Oct. 1966.
- [53] F.R. Kschischang, B.J. Frey, H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol.47, pp.498-519, Feb. 2001.
- [54] G. Bauch, H. Khorram, and J. Hagenauer, "Iterative equalization and decoding in mobile communications systems", in *Proc. 2nd EPMCC'97 and 3rd ITG-Fachtagung "Mobile Kommunikation"*, Bonn, Germany, Oct. 1997.
- [55] N. Lin, "A Comparison of Turbo Equalization Techniques for Stationary Intersymbol Interference Channels," M.App.Sc. Thesis, Carleton University, 2002.
- [56] J. L. Fan, E. Kurtas, A. Friedmann, S.W. McLaughlin, "LDPC codes for magnetic recording," *Proc. 37th Allerton Conf. on Commun., Control, and Computing*, 1999.

- [57] A. Thangaraj, and S.W. McLaughlin, "Thresholds and scheduling for LDPC-coded partial response channels", *IEEE Trans. Magnetics*, vol.38, pp.2307-2309, Sept. 2002.
- [58] B.M. Kurkoski, P.H. Siegel, and J.K. Wolf, "Joint message-passing decoding of LDPC codes and partial-response channels", *IEEE Trans. Inform. Theory*, vol.48, pp.1410-1422, June 2002.
- [59] N. Varnics and A. Kavcic, "Optimized Low-Density Parity-Check Codes for Partial Response Channels", *IEEE Comm. Lett.*, vol.7, no.4, pp.168-170, Apr. 2003.
- [60] A. Glavieux, C. Laot, J. Labat, "Turbo equalization over a frequency selective channel," *Proc. Int. Symp. Turbo Codes*, Brest, France, pp.96-102, Sept. 1997.
- [61] M. Tuchler, R. Koetter, and A.C. Singer, "Turbo equalization: principles and new results," *IEEE Transactions on Communications*, Vol.50, No.5, pp.754-767, May 2002.
- [62] B.L. Yeap, T.H. Liew, J. Hamorsky, and L. Hanzo, "Comparative study of turbo equalization schemes using convolutional, convolutional turbo, and block-turbo codes, *IEEE Transactions on Wireless Communications*, Vol.1, No.2, pp. 266 - 273, April 2002.
- [63] D. Raphaeli, Y. Zarai, "Combined turbo equalization and turbo decoding," *IEEE Comm. Letters*, vol.2, pp.107-109, April 1998.
- [64] L. Hanzo, T.H. Liew, B.L. Yeap, *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels*, John Wiley & Sons, 2002.
- [65] J. Hagenauer, E. Offer, L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol.IT-42, pp.425-429, March 1996.

- [66] S. B. Wicker, *Error Control Systems for Digital Communications and Storage*.
Prentice- Hall, Inc., 1995.
- [67] C. Heegard, and S. B. Wicker, *Turbo Coding*, Kluwer Academic Publishers, 1999.
- [68] Z. Wu, *Coding and Iterative Detection for Magnetic Recording Channels*, Kluwer
Academic Publishers, 1999.
- [69] F. Zarkeshvari and A. H. Banihashemi, "On implementation of min-sum algorithm
for decoding low-density parity-check (LDPC) codes," *Proc. GLOBECOM2002*,
pp.1349-1353, 2002