

5085

UNIVERSITY OF HAWAII LIBRARY

GENERALIZED CONSTRUCTIONS, DECODING AND IMPLEMENTATION OF
LDPC CODES

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAII IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

ELECTRICAL ENGINEERING

MAY 2008

By
Yige Wang

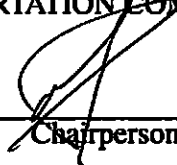
Dissertation Committee:

Marc P. C. Fossorier, Chairperson
N. Thomas Gaarder
Anthony Kuh
James B. Nation
James R. Yee

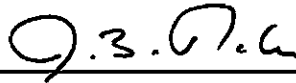
5085

We certify that we have read this dissertation and that, in our opinion, it is satisfactory in scope and quality as a dissertation for the degree of Doctor of Philosophy in Electrical Engineering.

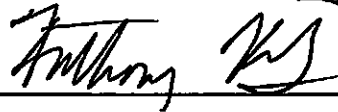
DISSERTATION COMMITTEE

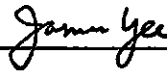


Chairperson









Copyright 2008

by

Yige Wang

iii

**To Guohua,
my mom,
and my dad.**

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor, Professor Marc Fossorier, for his guidance and support throughout my studies and research at the University of Hawaii. Professor Fossorier's insights and extensive knowledge in coding theory are always great resources for me and many new ideas in this dissertation were inspired by the discussions with him. His enthusiasm for high-quality research has made a very deep impression on me. Moreover, I am extremely grateful to him for his concern and support to my life. No words can express my deepest appreciation for his trust, his encouragement and his consideration all these years.

My heartfelt thanks go to Professor N. Thomas Gaarder, Professor Harry L. Van Trees and Professor James R. Yee. Without Professor Gaarder's help, my studies at UH could not start so smoothly. His wonderful lectures have greatly increased my interest in research. I have also benefited a lot from Professor Van Trees' class and carrying out the research project with him is a precious experience for me. Professor Yee is always ready to help me whenever I have difficulties and discussions with him are always informative.

I would like to thank my committee members, Professor Marc Fossorier, Professor N. Thomas Gaarder, Professor Anthony Kuh, Professor James B. Nation, and Professor James R. Yee. Their insightful comments and suggestions played a large part in this dissertation.

I also would like to thank Mitsubishi Electric Research Laboratories (MERL) at Cambridge, Massachusetts, where I worked as an intern. Particularly, I am very grateful to Dr. Jonathan Yedidia who guided me through all the challenging research projects which played a very important part in this dissertation. I am also grateful to Dr. Stark C. Draper, Jeff Proctor, Dr. Benjamin Vigoda and David Reynolds for their valuable support and collaborations.

My years at UH have been greatly enriched by my labmates in POST325: Jeff Battles, Nico Bernold, Frederic Bugey, Jinghu Chen, Wenyi Jin, Fabian Lim, Jianhan Liu, Nenad Miladinovic, Patrick Perry, Pascal Reymond, Marcos Vasconcelos, Chung-Li Wang, Zigui Yang, Juntan Zhang, and Qin Zhou. I want to thank all of them for their support, humor and friendship.

Finally, I want to dedicate this dissertation to my husband Guohua for his selfless love, support, and patience during the hard and joyful times in these years; and to my parents, who have taught me the true values of life. Their love, support, and encouragement have been with me every single moment.

ABSTRACT

Low-density parity-check (LDPC) codes have received significant attention due to their near-Shannon-limit error performance. The belief propagation (BP) algorithm is the best-known decoding algorithm for LDPC codes. However it often needs several tens or hundreds of iterations to converge. Although fast convergence BP algorithms have been proposed, there is neither an easy tool to analyze their performance nor a standard VLSI architecture to implement them. Moreover, there exists a trade-off between waterfall (i.e., threshold) and error floor (i.e., minimum pseudo distance) performances for LDPC codes. For generalized LDPC (GLDPC) codes, better minimum distance properties can be achieved, but at the expense of a rate loss.

This research investigates performance analysis and hardware implementation of LDPC codes using fast convergence algorithms. Closed-form extrinsic mutual information transfer (EXIT) functions for several fast convergence BP decoding algorithms are proposed. It is shown theoretically that these algorithms converge faster than standard BP, while the corresponding thresholds remain unchanged. Since these algorithms are of serial nature, their parallelization without compromising convergence speed and error performance is also investigated. This result is useful in hardware implementation because in practice partly parallel decoding schemes are often used. This result also sheds some light on how to group nodes in order to preserve performance. The VLSI architecture of a particular decoding algorithm for LDPC codes is proposed. This approach can be adopted in many standards to achieve a large throughput.

Doubly GLDPC (DGLDPC) codes are proposed to compensate the rate loss of GLDPC codes. EXIT charts are used to analyze DGLDPC codes and EXIT functions for variable component codes are thoroughly discussed. Based on EXIT charts, differential evolution is used to optimize their threshold. Ensemble weight enumerators are generalized to protograph-based DGLDPC codes. Simulation results show that DGLDPC codes constructed based on these results can improve the performance of their LDPC and GLDPC counterparts in both the waterfall region and the error floor region.

Contents

Acknowledgements	v
Abstract	vii
List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Introduction to Error Control Coding	1
1.2 Motivation of this Work	3
1.3 Outline of the Dissertation	4
2 LDPC Codes	6
2.1 Representations of LDPC Codes	6
2.2 Regular Codes	7
2.3 Irregular Codes	9
2.4 Constructions of LDPC Codes	10
2.4.1 Random Constructions	10
2.4.2 Pseudorandom Constructions	11
2.4.3 Finite Geometry LDPC Codes	14
2.4.4 Quasi-Cyclic LDPC Codes	16
3 Iterative Decoding of LDPC Codes	18
3.1 BP Algorithm	19
3.1.1 Probabilistic BP Algorithm	19
3.1.2 LLR BP Algorithm	21
3.2 Simplified BP Algorithms	23
3.2.1 BP-Based Algorithm	23
3.2.2 Normalized BP-Based Algorithm	24
3.2.3 Simulation Results	24
3.3 Fast Convergence BP Algorithms	26
3.3.1 Plain Shuffled BP Decoding of LDPC Codes	26

3.3.2	Replica Shuffled BP Decoding of LDPC Codes	27
3.3.3	Group Plain Shuffled BP and Group Replica Shuffled BP Decoding	30
3.3.4	Simulation Results	33
4	Analysis of Fast Convergence BP Algorithms by EXIT Charts	37
4.1	Introduction to EXIT Charts	37
4.2	EXIT Charts of LDPC Codes Using Standard BP Decoding	39
4.3	EXIT Charts of Plain Shuffled BP	41
4.4	EXIT Charts of Replica Shuffled BP	45
4.5	EXIT Charts of Group Plain Shuffled BP	49
4.6	EXIT Charts of Group Replica Shuffled BP	54
5	Hardware Implementation of Replica Shuffled Normalized BP-Based Decoder	57
5.1	Code Structure in the 802.16e Standard	58
5.2	Decoding Algorithm	58
5.3	Quantization and Parameter Selection	60
5.4	VLSI Architecture of Replica Shuffled Normalized BP-Based Decoder	61
5.4.1	Structure of Super Processor	63
5.4.2	Structure of Check Processor	64
5.4.3	Structure of Link Processor	68
5.4.4	Structure of Message Registers	68
5.4.5	Structure of Belief Registers	70
5.4.6	VLSI Architecture of Replica Shuffled Normalized BP-Based Decoder with Two Sub-Decoders	71
5.5	FPGA Testing Results	74
6	Doubly Generalized LDPC Codes	75
6.1	A Brief Review of GLDPC Codes	76
6.1.1	Structure of GLDPC Codes	76
6.1.2	Performance Example of a GLDPC Code	78
6.2	Structure of DGLDPC Codes	78
6.3	Iterative Decoding of DGLDPC Codes	82
6.4	Analysis by EXIT Charts	86
6.4.1	Closed Form EXIT Functions Obtained from Information Combining	86
6.4.2	EXIT Functions over the AWGN Channel Obtained from the BEC EXIT Functions	95
6.5	Threshold Optimization over the AWGN Channel	100
6.6	Ensemble Weight Enumerators for Protograph-Based DGLDPC Codes	102
6.7	Simulation Results	109
7	Conclusion	116
	Bibliography	118

List of Tables

3.1	Performance comparison of non-synchronous and synchronous replica shuffled BP decoding for the (273,191) PG-LDPC code.	30
3.2	Performance comparison of group plain and group replica shuffled BP decoding.	34
5.1	FPGA resource utilization statistics.	74
6.1	Threshold comparison of LDPC codes, a GLDPC code, and a DGLDPC code. (D): from density evolution (E): from EXIT charts.	110
6.2	Threshold comparison of an LDPC code and two DGLDPC codes with rate 3/4. (D): from density evolution (E): from EXIT charts.	112

List of Figures

2.1	An LDPC code represented in Tanner graph.	8
2.2	An example of an LDPC ensemble.	11
2.3	A tree spreading from bit n	12
3.1	The WER performance of the standard BP, BP-based and normalized BP-based algorithms for a (1056, 704) irregular LDPC code.	25
3.2	Illustration of the scheduling of group plain shuffled BP decoding with 2 groups and group replica shuffled BP decoding with 2 sub-decoders and 4 groups.	33
3.3	WER of a (8000, 4000)(3, 6) LDPC code with standard BP, vertical plain shuffled BP, vertical group plain shuffled BP for $G = 6$, vertical replica shuffled BP with four sub-decoders and synchronous updating and its group version with $G = 24$	34
3.4	WER of a (4000, 2000) irregular LDPC code with standard BP, horizontal plain shuffled BP, horizontal group plain shuffled BP for $G = 5$, horizontal replica shuffled BP with four sub-decoders and synchronous updating and its group version with $G = 10$	35
3.5	WER of a (4000, 2000) irregular LDPC code with plain shuffled BP and replica shuffled BP using vertical partitioning and horizontal partitioning.	36
4.1	EXIT charts of a 2-component turbo code with interleaver size 16384 and using parallel decoding at the SNR of 0.15 dB.	38
4.2	EXIT charts of a (3,6) regular LDPC code at the SNR of 1.5 dB.	39
4.3	EXIT charts of a (3,6) regular LDPC code at the SNR of 1.11 dB.	40
4.4	An example illustrating the ideal parity check matrix of a (2,3) regular LDPC code with length 9.	41
4.5	The mutual information updating process for the LDPC code with the ideal parity check matrix given in Figure 4.4.	43
4.6	Comparison between the EXIT curves obtained from the simulation method of [74] and the proposed closed forms for a (3, 6) regular LDPC code at the SNR 1.5 dB.	47

4.7	EXIT curves (in closed form) for plain shuffled BP and four types of replica shuffled BP decodings for a (3, 6) regular LDPC code at the SNR 1.5 dB (variable nodes (VND) and check nodes (CND)).	47
4.8	EXIT curves (in closed form) for standard BP, plain shuffled BP and replica shuffled BP with four sub-decoders with synchronous updating for a (3, 6) regular LDPC code at the SNR 1.5 dB, superimposed to constant-BER curves.	48
4.9	EXIT curves (in closed form) for standard BP, plain shuffled BP and four types of replica shuffled BP for a (3, 6) regular LDPC code at the SNR 1.11 dB (threshold).	49
4.10	WER of horizontal plain shuffled BP and horizontal group plain shuffled BP with 3 groups and 2 groups for decoding a (1800, 902) (3, 6) regular QC-LDPC code.	51
4.11	WER of vertical plain shuffled BP and vertical group plain shuffled BP with 6 groups and 2 groups for decoding a (1800, 902) (3, 6) regular QC-LDPC code.	52
4.12	WER of vertical plain shuffled BP and vertical group plain shuffled BP with 6 groups for decoding a (8000, 4000) (3, 6) regular LDPC code.	53
4.13	WER of horizontal plain shuffled BP and horizontal group plain shuffled BP with 2, 3, and 6 groups for decoding a (2016, 1512) irregular QC-LDPC code.	54
4.14	Comparison between the EXIT curves obtained from the simulation method of [74] and the proposed closed forms for vertical group plain shuffled BP and vertical group replica shuffled BP with four sub-decoders and synchronous updating, for decoding a (3, 6) regular LDPC code at the SNR 1.5 dB.	55
4.15	WER of vertical replica and vertical group replica shuffled BP with 24 groups with four sub-decoders and synchronous updating for decoding a (8000, 4000) (3, 6) regular LDPC code.	56
5.1	Base matrix of rate-2/3 LDPC codes in the IEEE 802.16e standard.	58
5.2	Performance comparison of horizontal replica shuffled normalized BP-based decoders without quantization and with 6-bit or 5-bit quantization for messages.	61
5.3	VLSI architecture of a horizontal group plain shuffled normalized BP-based decoder.	62
5.4	Structure of a super processor.	63
5.5	Structure of a check processor with 10 inputs/outputs.	64
5.6	Structure of a 10-input 2-output comparator.	65
5.7	Structure of a 11-input 2-output comparator.	66
5.8	Structure of comparator 3:2.	66
5.9	Structure of comparator 4:2 (1).	67
5.10	Structure of comparator 4:2 (2).	67

5.11	Structure of a link processor.	68
5.12	Structure of a bank of message registers.	69
5.13	Structure of a bank of belief registers.	70
5.14	VLSI architecture of a horizontal group replica shuffled normalized BP-based decoder with two sub-decoders.	72
5.15	Structure of a replica super processor with two sub-decoders.	73
6.1	Graph representation of a (2,7) regular GLDPC code.	77
6.2	An example that illustrates the process to obtain a (2,7) regular GLDPC code from its base matrix. (a) base matrix; (b) parity check matrix of the (7,4) Hamming code in the check nodes; (c) parity check matrix of the GLDPC code.	78
6.3	BER comparison of a (3,6) regular LDPC code of length 8000 and rate 1/2 and a (2,15) regular GLDPC code of length 7680 and rate 7/15.	79
6.4	Graph representation of a DGLDPC code.	80
6.5	An example that illustrates the process to obtain a (21, 3) DGLDPC code from its base matrix. (a) base matrix; (b) generator matrices of two sub-codes in the super variable nodes; (c) parity check matrix of the (7,4) Hamming code in the super check nodes; (d) new matrix \tilde{H} after row expansion; (e) parity check matrix of the DGLDPC code obtained from column expansion of \tilde{H}	81
6.6	Component code encoder of a super variable node.	83
6.7	Message passing at a super variable node and at a super check node.	84
6.8	A decoding model with two encoders.	87
6.9	Comparison of EXIT curves in closed form and those obtained from Monte Carlo simulation for the (6,5) systematic SPC code at the SNR value 1.9 dB.	89
6.10	EXIT curves in closed form for non-systematic SPC codes in cyclic form with different lengths over an AWGN channel at the SNR value 0.5 dB. A code rate of $R=1/2$ was used to calculate the SNR.	91
6.11	EXIT curve comparison of a non-systematic SPC code in cyclic form with length 1000 and an accumulator over an AWGN channel at several SNR values. A code rate of $R=1/2$ was used to calculate the SNR.	91
6.12	EXIT curves in closed form for non-systematic SPC codes in cyclic form with different lengths over a BEC with $q = 0.49$	93
6.13	EXIT curve comparison of a non-systematic SPC code in cyclic form with length 1000 and an accumulator over a BEC with $q = 0.49, 0.4, 0.3$ and 0.2	93
6.14	Comparison of EXIT curves in closed form and those obtained from Monte Carlo simulation for the (6,2) code at the SNR value 1.9 dB.	96
6.15	EXIT curve comparison of theoretical and simulation results for the (11,10), (12,11) SPC codes and the (7,4), (15,11), (31,26) Hamming codes as check component codes over an AWGN channel.	97

6.16	EXIT curve comparison of theoretical and simulation results for the (7,3), (15,4) and (31,5) simplex codes as variable component codes over an AWGN channel at $E_b/N_0 = 0.7$ dB and $R = 0.5$	99
6.17	EXIT curve comparison of theoretical and simulation results for a (31,10) random code as variable component codes over an AWGN channel at $E_b/N_0 = 0.5$ dB and $R = 0.5$	99
6.18	The process to obtain a larger graph from a protograph. (a) Protograph; (b) Copy 3 times; (c) Permute the edges.	102
6.19	A constituent code composed of p copies of super variable node v_i	103
6.20	A constituent code composed of p copies of super check node c_j	104
6.21	Protographs of the LDPC code and DGLDPC code for asymptotic weight enumerator calculation. (a) Protograph of a (3,12) LDPC code; (b) Protograph of a DGLDPC code that is equivalent to the (3,12) LDPC code; (c) Protograph of a DGLDPC code with a (6,2) code as super variable nodes.	107
6.22	WER/BER performances of a length-2016 rate-3/4 protograph-based LDPC code and DGLDPC 1.	108
6.23	WER/BER performances of DGLDPC 1.	109
6.24	BER performance comparison of randomly constructed rate-1/2 LDPC, GLDPC, DGLDPC codes with length 1000000.	111
6.25	BER performance comparison of randomly constructed rate-3/4 LDPC, DGLDPC 3 and DGLDPC 4 codes with length 100000.	113
6.26	BER performance comparison of DGLDPC 5 with a rate-7/15 length-7680 (2,15) GLDPC code.	114
6.27	WER performance comparison of DGLDPC 6 with a rate-1/2 length-1504 (2,4)-LDPC code over GF(16).	115

Chapter 1

Introduction

1.1 Introduction to Error Control Coding

In recent years, with the increasing demand for reliable data transmission and storage systems, it has become more and more crucial to control errors so that the data can be successfully recovered. Shannon's landmark paper [1] in 1948 proves that by proper encoding of the information, errors induced by a noisy channel can be reduced to any desired level as long as the information rate is less than channel capacity. This pioneering work gave birth to a brand-new field in digital communications – error control coding or channel coding. The goal of error control coding is to encode information in such a way that even if the channel introduces errors, the receiver can correct the errors and recover the original transmitted information.

Shannon's proof of his capacity theorem was based on random codes, i.e., the error probability averaged over all randomly selected codes can be made arbitrarily small for transmission rates below channel capacity. Although this result implies the existence of good codes, it has long been an open question about how to design such codes. Moreover, in practical applications, a good code should not only have performance approaching the limit predicted by Shannon's theory, but also have an implementable encoder and decoder.

Error control codes are basically divided into two classes, block codes and convolutional codes. The difference between block codes and convolutional codes is based on whether "memory" is used in the codeword generation. For a block code, the output of the encoder only depends on the current input sequence, while for a convolutional code, the

output of the encoder relates to not only the current input sequence, but also the previous v input sequences, where v is called “constraint length”. The input sequence of a block code has fixed size, while a convolutional code works on bit or symbol streams of arbitrary length. A convolutional code can be turned into a block code by truncation or termination.

Almost all the practical block codes are linear codes, i.e., the sum of any two codewords is also a codeword. A length- n linear block code having k information symbols and minimum distance d_{min} is referred to as an (n, k, d_{min}) code. The minimum distance determines the random-error-detecting and random-error-correcting capabilities of a code. It is defined as the minimum number of positions in which any two codewords differ. A large minimum distance is generally desirable in designing a good code. Many good linear block codes have been invented and explored during the past sixty years such as Hamming codes [2], Reed-Muller (RM) codes [3], Bose-Chaudhuri-Hocquenghem (BCH) codes [4] [5], and Reed-Solomon (RS) codes [6]. Hamming codes have been the first class of linear block codes proposed for error correction. They have a minimum distance of 3 and can correct any single error. The class of BCH codes is a generalization of the Hamming codes for multiple-error correction. The class of RS codes is a subclass of nonbinary BCH codes. Since RS codes have a powerful capability in correcting both random symbol errors and random burst errors, they have been widely used for error control in many areas, ranging from deep-space telecommunications to data storage systems.

Decoding of error-correcting codes can be categorized into hard-decision decoding (HDD) and soft-decision decoding (SDD), according to whether the decoder receives one-bit quantized values or multilevel quantized (or unquantized) values. The SDD provides better performance than the HDD, but generally requires more computational complexity. The Berlekamp-Massey decoding algorithm [7]-[10] for BCH and RS codes and the majority-logic decoding algorithm [11] for RM codes belong to the HDD class. Maximum *a posteriori* probability (MAP) decoding [12] and some probabilistic list decoding, such as generalized minimum distance (GMD) decoding [13], Chase type decoding [14] and ordered statistics decoding (OSD) [15] are SDD. The Viterbi algorithm for convolutional codes, which is a maximum likelihood decoding (MLD) algorithm, can be either HDD or SDD, with about the same complexity.

The invention of turbo codes [16] [17] created a revolution in the coding area. With reasonable encoding and decoding complexity, turbo coding succeeds in achieving near-Shannon limit error performance. Turbo codes are composed of two or more constituent codes, which are concatenated in parallel, along with one or more random interleavers. The interleaver plays a crucial role in turbo coding because it affects both distance properties and the complexity of the encoder and decoder. The basic random interleavers permute the information bits in a pseudorandom manner. On the decoding side, iterative soft-input soft-output (SISO) decoders are employed for each constituent code, in which the soft-output values of one decoder are passed as inputs to the others. This simple sub-optimum iterative decoding scheme can approach the maximum likelihood (ML) solution.

In the late 1990s, another class of Shannon limit-approaching codes, low-density parity-check (LDPC) codes, began to receive more and more attention. These codes were first discovered by Gallager [18] in 1960s. Unfortunately, due to the limited processing capabilities then, they have been mostly ignored until being rediscovered by MacKay and Neal [19] in 1996. It has been shown that long LDPC codes with iterative decoding based on belief propagation (BP) [20]–[21] can achieve an error performance only a fraction of a decibel away from the Shannon limit for several channels [22][23].

LDPC codes have many advantages over turbo codes. For example, they have better minimum distance properties, so that their error floor occurs much lower than for turbo codes; a parallel structure can be exploited in the hardware design of their decoder, so that they are suitable for high speed applications. Due to these advantages, LDPC codes have become one of the most promising error control codes and have been adopted or considered in many recent industry standards.

1.2 Motivation of this Work

For LDPC codes, the BP algorithm is the best-known decoding algorithm. However it often needs several tens or hundreds of iterations to converge, which causes large delay and may not be suitable for high speed communication systems. To speed up the convergence, the shuffled BP decoding algorithm [24]–[28] and its generalized version, replica shuffled BP decoding [29], have been proposed. Density evolution [30] can be used to

analyze the performance of these algorithms [29]; however, it is complex and not easy to visualize and to evaluate. Hence it becomes desirable to find an easy and efficient tool to analyze the above algorithms.

As LDPC codes have been adopted in more and more standards, efficient VLSI designs for implementations of LDPC codes has become crucial. Many architectures have been proposed for LDPC codes using the BP or shuffled BP algorithms [31]-[34]. Since replica shuffled decoding converges faster than other algorithms, it is useful to design a VLSI architecture to decode LDPC codes using this scheme.

Besides developing decoding algorithms and designing VLSI architectures, constructing good LDPC codes has also been studied intensively. In addition to the traditional LDPC structures, some generalized LDPC codes have been proposed [35]-[41]. These codes usually achieve better minimum distance properties compared with LDPC codes. However the resulting rate loss is not negligible. Therefore finding generalized LDPC codes with high rates remains an interesting challenge.

This dissertation investigates solutions to the above three issues. First, closed-form extrinsic mutual information transfer (EXIT) functions are proposed for shuffled BP and replica shuffled BP decoding. EXIT curves can be plotted accordingly, from which the speed of convergence of different algorithms can be visualized. Secondly, we designed a VLSI architecture for decoding LDPC codes using the replica shuffled BP algorithm. Finally, we proposed doubly generalized LDPC (DGLDPC) codes, which have better performances than their LDPC and GLDPC counterparts in both waterfall region and error floor region.

1.3 Outline of the Dissertation

The rest of this dissertation is organized as follows.

Chapter 2 presents the basic concepts and notation for LDPC codes. It introduces the definition of LDPC codes in terms of parity check matrix and graph representations. Regular and irregular LDPC codes are also defined. It finally discusses several approaches to construct LDPC codes.

Chapter 3 summarizes various iterative decoding algorithms for LDPC codes. Different algorithms are categorized into three types, the standard BP algorithm, the simplified BP algorithms, and the fast convergence BP algorithms. In standard BP decoding, all the variable nodes or check nodes are processed in parallel, with no simplifications in both variable node processing and check node processing. Simplified BP algorithms include the BP-based and normalized BP-based algorithms, both of which simplify check node processing. Fast convergence BP algorithms include plain shuffled BP decoding and replica shuffled BP decoding, both of which speed up the convergence compared with that of standard BP decoding or simplified BP decoding.

Closed-form EXIT functions for plain shuffled BP and replica shuffled BP decoding are proposed in Chapter 4. Based on EXIT charts, the convergence speeds of different decoding algorithms are compared. It is proved using EXIT functions that the threshold of a code decoded by plain shuffled BP or replica shuffled BP is the same as standard BP. In this chapter the estimates are also given for the least number of groups used by group shuffled BP and group replica shuffled BP to achieve at any given iteration the same performance (i.e., same convergence) as their corresponding non-group counterparts (nodes in a group are decoded in parallel as in standard BP decoding). These estimates are verified by performance simulation.

In Chapter 5, a VLSI architecture for replica shuffled normalized BP-based decoding is developed. FPGA testing results for a special case are presented.

DGLDPC codes are investigated in Chapter 6. Closed-form EXIT functions for specific variable component codes of DGLDPC codes are derived. EXIT functions of more general component codes are also discussed. Based on EXIT charts, differential evolution (DE) is used for threshold optimization. Ensemble weight enumerators for protograph-based DGLDPC codes are also presented. Several DGLDPC codes are constructed based on these results and compared with their LDPC and GLDPC counterparts.

Finally, some conclusions and further research directions are given in Chapter 7.

Chapter 2

LDPC Codes

2.1 Representations of LDPC Codes

An (N, K) LDPC code is defined as the null space of a parity check matrix $\mathbf{H} = [H_{mn}]$ with N columns and M rows, where N is the code length, K is the code dimension, and M is at least $N - K$. Each column of \mathbf{H} corresponds to a codeword bit position and each row corresponds to a check sum. The \mathbf{H} matrices of LDPC codes should be sparse, which means they have a small number of non-zero entries. This property enables us to use some efficient decoding algorithms.

The rate $R = K/N$ of an LDPC code is lower bounded by $1 - \frac{M}{N}$. If $M = N - K$, i.e., all the rows in \mathbf{H} are linearly independent, the lower bound is reached. For long randomly constructed LDPC codes, this lower bound is usually reached or nearly reached, while for some special families of LDPC codes, e.g., geometric LDPC codes, M can be much larger than $N - K$.

LDPC codes can be represented by bipartite graphs, among which the most popular one is the Tanner graph [35]. A Tanner graph is a bipartite graph, so that nodes are partitioned into two disjoint classes. The nodes of one class are denoted as bit nodes or variable nodes, which are associated with codeword bits; the nodes of the other class are denoted as check nodes, which specify the constraints imposed on codeword bits. Variable nodes and check nodes are connected by edges based on the parity check matrix \mathbf{H} of the code. If $H_{mn} = 1$, there exists an edge between bit node n and check node m . The degree of a node is defined as the number of edges incident to that node. An example of the \mathbf{H}

matrix of a (20,11) LDPC code is shown in (2.1.1) and its corresponding Tanner graph is depicted in Figure 2.1.

$$\mathbf{H} = \begin{bmatrix}
 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1
 \end{bmatrix}_{10 \times 20} \quad (2.1.1)$$

The girth of a graph is defined as the length of the shortest cycle in the graph. It is readily checked that the girth of the Tanner graph in Figure 2.1 is 8. A cycle of length 8 is shown in bold dash dot lines in Figure 2.1 and it consists of 4 variable nodes and 4 check nodes. Since the shortest cycle a Tanner graph can contain is of length 4, the possible smallest girth is also 4. For a girth-4 Tanner graph, the corresponding parity check matrix \mathbf{H} contains two rows having two bits in common.

Usually, short cycles, especially those of length 4, should be avoided in the code construction because they may degrade the performance of iterative decoding, which has been designed for a Tanner graph without cycles. However, if the connectivity of the short cycles is carefully designed, the performance of iterative decoding is not necessarily degraded by short cycles. This issue is further discussed in Chapter 6.

2.2 Regular Codes

An LDPC code is called regular if the row weight and the column weight of its parity check matrix are constants, saying d_c and d_v , respectively. Then this code is referred to as a (d_v, d_c) regular LDPC code. Since the parity check matrix \mathbf{H} is a “low density” matrix, d_c and d_v are generally much smaller than M or N . The rate $R \geq 1 - \frac{d_v}{d_c}$. In

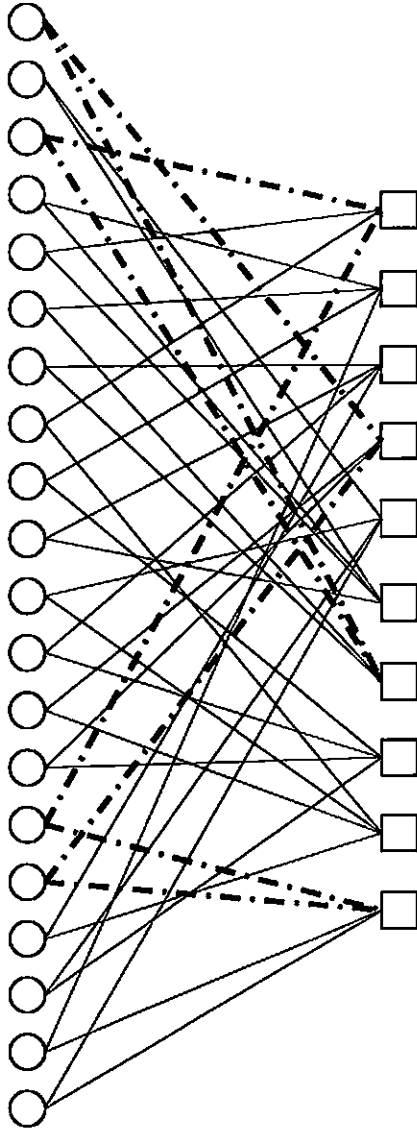


Figure 2.1: An LDPC code represented in Tanner graph.

the Tanner graph of a regular LDPC code, each variable node and each check node have constant degrees d_v and d_c , respectively. The matrix H in (2.1.1) exhibits an example of a (2,4) regular LDPC code.

2.3 Irregular Codes

If the row weight and the column weight of H are not constants, the LDPC code is irregular [42]. An irregular LDPC code is specified by degree distribution polynomials $\lambda(x)$ and $\rho(x)$

$$\lambda(x) = \sum_{j=2}^{d_{vmax}} \lambda_j x^{j-1} \quad (2.3.1)$$

and

$$\rho(x) = \sum_{j=2}^{d_{cmax}} \rho_j x^{j-1}, \quad (2.3.2)$$

where d_{vmax} and d_{cmax} represent the maximum degrees of variable and check nodes, respectively, and λ_j (ρ_j) is the fraction of edges incident to variable (check) nodes of degree j with $\sum_{j=2}^{d_{vmax}} \lambda_j = 1$ and $\sum_{j=2}^{d_{cmax}} \rho_j = 1$. A (d_v, d_c) regular LDPC code can be regarded as a special case of an irregular LDPC code with $\lambda_{d_v} = 1$ and $\rho_{d_c} = 1$. It has been shown in [22] that with carefully chosen degree distributions, irregular LDPC codes have better performance than regular ones and they can approach the Shannon limit for large code lengths. The rate R of an irregular code is lower bounded by

$$R \geq 1 - \frac{\int_0^1 \rho(x) dx}{\int_0^1 \lambda(x) dx} = 1 - \frac{\sum_{j=2}^{d_{cmax}} \frac{\rho_j}{j}}{\sum_{j=2}^{d_{vmax}} \frac{\lambda_j}{j}}. \quad (2.3.3)$$

Define node distribution polynomials $\chi(x)$ and $\vartheta(x)$ as

$$\chi(x) = \sum_{j=2}^{d_{vmax}} \chi_j x^{j-1} \quad (2.3.4)$$

and

$$\vartheta(x) = \sum_{j=2}^{d_{cmax}} \vartheta_j x^{j-1}, \quad (2.3.5)$$

where χ_j (ϑ_j) is the fraction of variable (check) nodes of degree j with $\sum_{j=2}^{d_{vmax}} \chi_j = 1$ and $\sum_{j=2}^{d_{cmax}} \vartheta_j = 1$. The node distributions can be expressed with respect to the edge distributions in (2.3.1) and (2.3.2) by

$$\chi_j = \frac{\lambda_j/j}{\int_0^1 \lambda(x) dx} \quad (2.3.6)$$

for $j = 2, 3, \dots, d_{vmax}$ and

$$\vartheta_j = \frac{\rho_j/j}{\int_0^1 \rho(x) dx} \quad (2.3.7)$$

for $j = 2, 3, \dots, d_{cmax}$.

2.4 Constructions of LDPC Codes

2.4.1 Random Constructions

Given the degree distributions and the code length, an LDPC code can be constructed by placing a random permutation between edges emanating from all the variable nodes and edges emanating from all the check nodes in the Tanner graph. The collection of LDPC codes resulting from all possible random permutations is defined as an ensemble. Figure 2.2 shows an example of the LDPC ensemble with code length 8 and degree distributions

$$\lambda(x) = 0.32x + 0.12x^2 + 0.32x^3 + 0.24x^5$$

and

$$\rho(x) = x^4.$$

In order to make the graph suitable for iterative decoding, a post processing routine may need to be carried out to remove some short cycles. Randomly constructed LDPC codes usually have high error floor, so many improved methods have been proposed, which are discussed next.

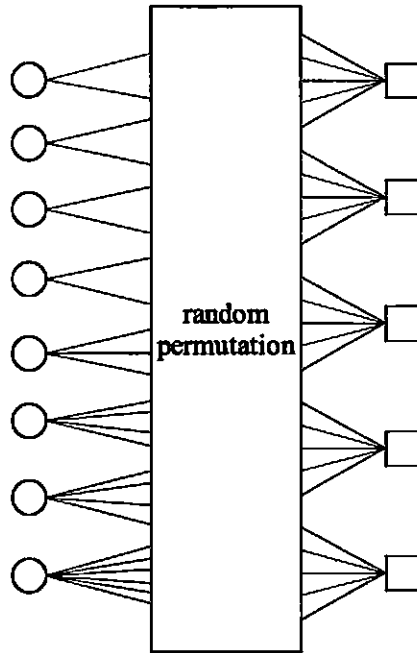


Figure 2.2: An example of an LDPC ensemble.

2.4.2 Pseudorandom Constructions

Progressive Edge-Growth (PEG) construction

The objective of the PEG construction [43] is to maximize the girth of the Tanner graph by progressively assigning edges between variable nodes and check nodes. The construction can be carried out as follows.

For bit n from 1 to N :

Step 1: Assign the first edge of bit n to a check node having the lowest degree under the current graph.

Step 2: Assign other edges to check nodes that are not among the neighbors of bit n within depth l in the current graph.

For bit n , its neighbors within depth l contain all the check nodes that are reached by a tree spreading from bit n within depth l . This tree is illustrated in Figure 2.3.

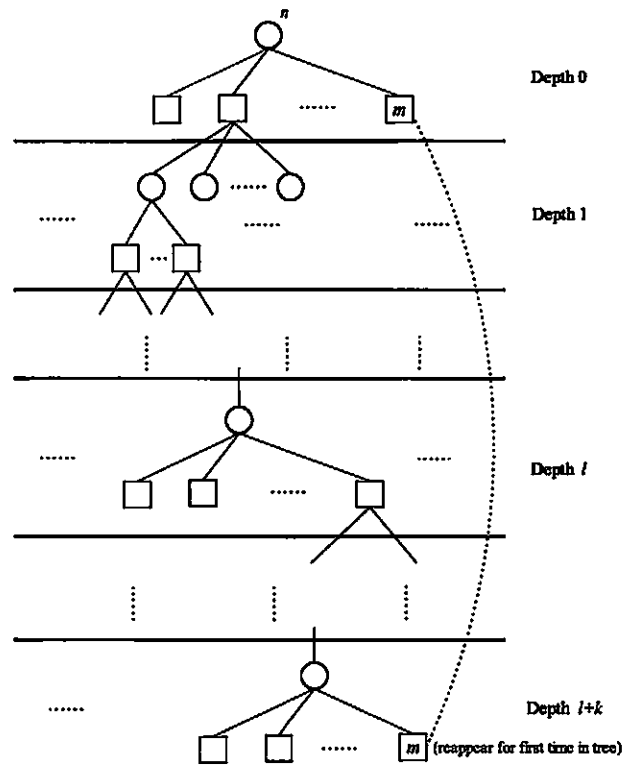


Figure 2.3: A tree spreading from bit n .

Based on the construction rule, two cases may occur for bit n . The first case is that all the check nodes it connects to do not reappear in any layer of its tree spreading graph. In this case, there exists no cycle in this graph. The second case is that the first reappearance of any check node m containing bit n occurs in depth $(l + k)$, where $k \geq 1$. This check is marked in shadow in Figure 2.3. In this case, the length of the smallest cycle is $2 \cdot (l + k + 1)$. Hence, if for any bit, all its adjacent check nodes are not among its neighbors within depth l , then the girth of the Tanner graph $g \geq 2 \cdot (l + 2)$.

It has been shown through simulations in [43] that the PEG algorithm results in 0.2 to 0.5 dB improvement at the bit error rate (BER) of 10^{-5} compared with randomly constructed LDPC codes.

ACE construction

Although a large girth is usually obtained with the PEG algorithm, a good girth does not necessarily guarantee a good weight distribution. To improve the weight distribution, cycle connectivity needs to be considered. In [44], the parameter ACE has been used as a measurement of the connectivity of a cycle, which is defined as $\sum_i (d_i - 2)$, where d_i is the degree of the i -th variable node in this cycle. If no variable nodes in a cycle share common check nodes outside the cycle, the ACE value can be also regarded as the total number of check nodes that are connected by this cycle only once. These check nodes can be helpful in improving minimum distance and correcting the errors trapped in the cycle. Therefore a large ACE value is preferred. An LDPC code has property (d_{ACE}, η_{ACE}) if all the cycles of length $2d_{ACE}$ or less have ACE values at least η_{ACE} . Methods to construct LDPC codes with good ACE properties have been proposed in [44]. It has been shown that the ACE algorithm exhibits a better minimum distance compared with the PEG algorithm and the obtained codes can achieve a much lower error floor compared with random LDPC codes without sacrificing much in the waterfall performance region.

However, an LDPC code with good ACE properties can still contain low-weight codewords. In [45], a recursive construction based on quasi-cyclic extension has been proposed, which can eliminate in the Tanner graph certain subgraphs that lead to low-weight codewords. An LDPC code constructed by this method exhibits a better minimum

distance as well as a thinner profile of low-weight codewords than that obtained by either the PEG or the ACE algorithm.

2.4.3 Finite Geometry LDPC Codes

Besides random and pseudorandom constructions, there is another important construction method based on finite geometry. Finite geometry codes were first investigated by Rudolph in 1967 for majority-logic decoding [46]. A finite geometry with N points and M lines can define the parity check matrix of an LDPC code with columns corresponding to the points and rows corresponding to the lines of the geometry. If a line passes through a point in the geometry, the intersection of the corresponding row and column in the parity check matrix is 1. Since two lines are either disjoint or intersect at one and only one point, the girth of the corresponding Tanner graph is at least 6. Compared with random and pseudorandom constructions, finite geometry LDPC codes have the following advantages. First they can be cyclic, so encoding can be easily implemented by shift registers. Second, finite geometry LDPC codes have relatively good minimum distances, so usually they have a low error floor in the high SNR region. Third many redundant rows in H can help iterative decoding. Two families of geometric LDPC codes, Euclidean geometry (EG) and projective geometry (PG) codes [47], are introduced in the following.

EG-LDPC codes

Let $\text{GF}(2^s)$ be the Galois field with 2^s elements. Let $\alpha = (a_0, a_1, \dots, a_{m-1})$ be an m -tuple with $a_i \in \text{GF}(2^s)$. There are 2^{ms} possible such m -tuples and all of them form an m -dimensional Euclidean geometry over $\text{GF}(2^s)$, which is denoted as $\text{EG}(m, 2^s)$. Each m -tuple α is regarded as a point. The all-zero m -tuple is called the origin of the geometry. Through two points α_1 and α_2 in $\text{EG}(m, 2^s)$, there is a line defined as the collection of the following 2^s points,

$$\{\alpha_1 + \beta\alpha_2\}, \quad (2.4.1)$$

with $\beta \in \text{GF}(2^s)$. Each point is intersected by $(2^{ms} - 1)/(2^s - 1)$ lines. The total number of non-origin points is $2^{ms} - 1$ and the total number of lines that do not pass through the origin is $(2^{ms} - 1)(2^{(m-1)s} - 1)/(2^s - 1)$.

Let H_{EG} be a matrix whose rows are the incidence vectors of all the lines in $EG(m, 2^s)$ that do not pass through the origin. The code defined by the null space of H_{EG} is a regular LDPC code of length $2^{ms} - 1$ and it is denoted as an m -dimensional EG-LDPC code.

A special subclass of EG-LDPC codes is the class of 2-dimensional EG-LDPC codes. Its parameters are:

- Length: $N = 2^{2s} - 1$
- Number of parity check bits: $M = 3^s - 1$
- Dimension: $K = 2^{2s} - 3^s$
- Minimum distance: $d_{min} = 2^s + 1$

The definition of a line or 1-flat given in (2.4.1) can be generalized as

$$\{\alpha_0 + \beta_1 \alpha_1 + \beta_2 \alpha_2 + \cdots + \beta_\mu \alpha_\mu\}, \quad (2.4.2)$$

with $1 \leq \mu < m$ and for $1 \leq j \leq \mu$, $\beta_j \in GF(2^s)$. Equation (2.4.2) defines a μ -flat of $EG(m, 2^s)$ that passes through the point α_0 . Then H_{EG} is obtained by associating its rows with the μ -flats of $EG(m, 2^s)$ (possibly not containing θ) and associating its columns with the points of $EG(m, 2^s)$ (possibly excluding θ). It follows that in general, an EG-LDPC code is totally defined by the three parameters m , s and μ .

PG-LDPC codes

Since $GF(2^{(m+1)s})$ contains $GF(2^s)$ as a subfield, it is possible to divide the $2^{(m+1)s} - 1$ non-zero elements of $GF(2^{(m+1)s})$ into $N(m, s) = (2^{(m+1)s} - 1)/(2^s - 1)$ disjoint subsets of $2^s - 1$ elements each. Then each subset is regarded as a point in $PG(m, 2^s)$, the m -dimensional projective geometry over the finite field $GF(2^s)$.

Two distinct points α_1 and α_2 of $PG(m, 2^s)$ define a unique line passing through them, which contains the following $2^s + 1$ points,

$$\{\beta_1 \alpha_1 + \beta_2 \alpha_2\}, \quad (2.4.3)$$

with β_1 and β_2 in $\text{GF}(2^s)$, and not both zero. Then we can obtain that there are $(2^{ms} - 1)/(2^s - 1)$ lines intersecting on each point of $\text{PG}(m, 2^s)$ and the total number of lines in $\text{PG}(m, 2^s)$ is $N(m, s) \cdot (2^{ms} - 1)/(2^s - 1)/(2^s + 1)$.

The construction of LDPC codes based on a projective geometry is similar to that based on Euclidean geometry. Let H_{PG} be a matrix whose rows are the incidence vectors of all the lines in $\text{PG}(m, 2^s)$ and whose columns correspond to the points of $\text{PG}(m, 2^s)$. The code defined by the null space of H_{PG} is a regular LDPC code of length $(2^{(m+1)s} - 1)/(2^s - 1)$ and it is denoted as an m -dimensional PG-LDPC code.

A special subclass of PG-LDPC codes is the class of 2-dimensional PG-LDPC codes with parameters:

- Length: $N = 2^{2s} + 2^s + 1$
- Number of parity check bits: $M = 3^s + 1$
- Dimension: $K = 2^{2s} + 2^s - 3^s$
- Minimum distance: $d_{min} = 2^s + 2$

For $1 \leq \mu < m$, a μ -flat of $\text{PG}(m, 2^s)$ is defined by the set of the points of the form

$$\{\beta_1 \alpha_1 + \beta_2 \alpha_2 + \dots + \beta_{\mu+1} \alpha_{\mu+1}\}, \quad (2.4.4)$$

with for $1 \leq j \leq \mu$, $\beta_j \in \text{GF}(2^s)$. Then H_{PG} is obtained by associating its rows and columns with the μ -flats and the points of $\text{PG}(m, 2^s)$, respectively.

2.4.4 Quasi-Cyclic LDPC Codes

Quasi-cyclic (QC) LDPC codes are another important class of LDPC codes. They were first proposed by Gallager in [18] and later investigated in [49] and [50].

QC-LDPC codes are constructed based on circulant permutation matrices, i.e., matrices obtained from the cyclic shift of an identity matrix. Finite geometry construction discussed in 2.4.3 is interrelated with this construction method because many geometry LDPC codes have an equivalent circulant permutation matrix representation [48]. For a

(d_v, d_c) regular LDPC code of length N , suppose $N = p \cdot d_c$. Let $I(s_{j,l})$ be a circulant permutation matrix which is obtained from cyclically right-shifting the $p \times p$ identity matrix $s_{j,l} \bmod p$ positions, where $1 \leq j \leq d_v$ and $1 \leq l \leq d_c$. The parity-check matrix \mathbf{H} can be represented by

$$\mathbf{H} = \begin{bmatrix} I(s_{1,1}) & I(s_{1,2}) & \cdots & I(s_{1,d_c}) \\ I(s_{2,1}) & I(s_{2,2}) & \cdots & I(s_{2,d_c}) \\ \vdots & \vdots & \ddots & \vdots \\ I(s_{d_v,1}) & I(s_{d_v,2}) & \cdots & I(s_{d_v,d_c}) \end{bmatrix}. \quad (2.4.5)$$

In order to achieve good performance, $s_{j,l}$ should be carefully chosen. One possible criterion is to choose $s_{j,l}$ so that the girth of the code is maximized. It has been shown in [51] that QC-LDPC codes cannot have a girth larger than 12. Consequently, given a code rate, the minimum distance cannot increase with the code length. Nevertheless, since the quasi-cyclic structure of this kind of codes enables them to be encoded easily with shift registers, they have been adopted in many standards, such as IEEE 802.16e [65].

The parity check matrix \mathbf{H} has an equivalent representation, the base matrix \mathbf{B} , which is defined by $\mathbf{B} = [s_{j,l}]$. For a given p , \mathbf{B} can be expanded to \mathbf{H} .

An irregular QC-LDPC code can be obtained from (2.4.5) by replacing a set of circulant permutation matrices by zero matrices and the corresponding shift values by -1. This technique is referred to as protograph [83] or masking [52] [54]. It results in a new Tanner graph with fewer edges and fewer short cycles. The matrix $\mathbf{D} = [d_{j,l}]$, which has the same dimensions as \mathbf{B} , is used to indicate the positions of circulant permutation and zero matrices. If the matrix $I(s_{j,l})$ is the all-zero matrix, $d_{j,l} = 0$; otherwise, $d_{j,l} = 1$. Hence \mathbf{D} is the all-one matrix for conventional QC-LDPC codes. The performance of an irregular QC-LDPC code constructed this way is highly related to the choice of \mathbf{D} . A good matrix \mathbf{D} can be constructed by computer search based on the variable and check node degree distributions.

Chapter 3

Iterative Decoding of LDPC Codes

In [18], two types of decoding algorithms were proposed for LDPC codes, bit-flipping decoding and probability-based decoding. Both of them are iterative decoding algorithms, which are also used in decoding turbo codes [17] to achieve near-Shannon-limit error performance. Iterative decoding is based on alternately decoding different component codes and passing the so-called extrinsic information to one another. The extrinsic information can be generated using a symbol-by-symbol soft-in/soft-out decoding algorithm like maximum *a posteriori* probability (MAP) decoding [12] or some other algorithms. Gallager's probabilistic decoding was later revealed to be an instance of Pearl's belief propagation (BP) algorithm [20]–[21]. In BP decoding, a Tanner graph is converted to a Bayesian network, based on which the marginal *a posteriori* probability of each codeword bit can be calculated [55]. If there is no cycle in the Tanner graph, the exact *a posteriori* probabilities can be derived with BP decoding. Even when the Tanner graph has cycles, BP decoding can still lead to a good estimate for the *a posteriori* probabilities.

Although the BP decoding algorithm is very powerful, it is complicated for hardware implementation due to the non-linear functions in the algorithm. Many algorithms have been proposed to simplify it, such as the BP-based algorithm [56], the min-sum algorithm [57] and the max-product algorithm [58]. These algorithms usually cause about 0.2 to 0.3 dB performance degradation for short LDPC codes and larger degradation for long LDPC codes. To compensate the performance loss and still preserve the simplicity of the decoder, normalized BP-based [59] and offset BP-based [60] algorithms have been proposed. Both of them apply some simple correction to check node messages and can

reduce the performance degradation to 0.1 dB or less. Other types of corrections also work well [53][61][62].

Another disadvantage of the BP decoding algorithm is that it needs several tens or hundreds of iterations to converge, which is not always realistic for high speed communication systems. To speed up the convergence, various schedulings of the BP decoding algorithm [24]–[28] have been proposed. Replica decoding [29] further generalizes this approach by combining different iterations, which can achieve further speed up of the convergence.

In this chapter, the BP algorithm and its modified versions are reviewed.

3.1 BP Algorithm

The BP algorithm for LDPC codes with Tanner graphs has two alternating parts: processing in variable nodes and processing in check nodes. When variable nodes receive messages from their neighbor check nodes, they process them and send back updated messages to the neighbor check nodes. Then check nodes carry out the similar procedure.

Messages can be represented in probabilities or log-likelihood ratios (LLRs). The corresponding BP algorithm is called probabilistic BP or LLR BP, respectively. The probabilistic BP algorithm is more general and it can be applied to both binary and non-binary LDPC codes. The LLR BP algorithm can lead to some low complexity decoding approaches, however it is only applicable to binary LDPC codes. The decoding procedure of these two BP algorithms is summarized in the following.

3.1.1 Probabilistic BP Algorithm

Suppose a binary (N, K) LDPC code C of length N and dimension K is used for channel coding. Assume binary phase-shift keying (BPSK) modulation with unit energy, which maps a codeword $\mathbf{c} = (c_1, c_2, \dots, c_N)$ into a sequence $\mathbf{x} = (x_1, x_2, \dots, x_N)$, according to $x_n = 1 - 2c_n$, for $n = 1, 2, \dots, N$. The vector \mathbf{x} is transmitted over an AWGN channel with zero mean and power spectral density $N_0/2$, then the received sequence is

$\mathbf{x} + \mathbf{n} = \mathbf{y} = [y_n]$, with $y_n = x_n + n_n$, where for $1 \leq n \leq N$, the n_n 's are statistically independent Gaussian random variables with zero mean and variance $N_0/2$.

Let $\mathbf{H} = [H_{mn}]$ be the parity check matrix which defines the LDPC code. We denote the set of bits that participate in check m by $\mathcal{N}(m) = \{n : H_{mn} = 1\}$ and the set of checks in which bit n participates as $\mathcal{M}(n) = \{m : H_{mn} = 1\}$. We also use $\mathcal{N}(m) \setminus n$ to denote the set $\mathcal{N}(m)$ with bit n excluded, and $\mathcal{M}(n) \setminus m$ to denote the set $\mathcal{M}(n)$ with check m excluded.

We define the following notations associated with a given iteration.

- f_n^x : The probability that c_n is x ($x = 0$ or 1). Let p_0 and p_1 be the conditional probability density functions (pdf's) of the AWGN channel, then

$$\begin{cases} p_0 = p(y_n | c_n = 0) = \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{(y_n - 1)^2}{N_0}} \\ p_1 = p(y_n | c_n = 1) = \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{(y_n + 1)^2}{N_0}} \end{cases} \quad (3.1.1)$$

f_n^x can be computed by

$$f_n^0 = \frac{p_0}{p_0 + p_1} \quad \text{and} \quad f_n^1 = \frac{p_1}{p_0 + p_1}. \quad (3.1.2)$$

- r_{mn}^x : The probability of check m being satisfied if bit n is considered fixed at x and the other bits have a separable distribution given by the probabilities $\{q_{mn'}^x : n' \in \mathcal{N}(m) \setminus n\}$.
- q_{mn}^x : The probability that bit n is x given the information obtained via checks other than check m .
- q_n^x : The ‘‘pseudoposteriori probabilities’’ of bit n .

The probabilistic BP decoding algorithm is carried out as follows.

Initialization: For each n , initialize f_n^0 and f_n^1 with (3.1.2); for each m and each $n \in \mathcal{N}(m)$, set $q_{mn}^0 = f_n^0$ and $q_{mn}^1 = f_n^1$. Set the maximum number of iterations to I_{Max} .

Step 1: Iterative processing:

- (i) Horizontal step, for each check node m and each $n \in \mathcal{N}(m)$, update r_{mn}^x by

$$\left\{ \begin{array}{l} r_{mn}^0 = \frac{1}{2} \left(1 + \prod_{n' \in \mathcal{N}(m) \setminus n} (q_{mn'}^0 - q_{mn'}^1) \right) \\ r_{mn}^1 = \frac{1}{2} \left(1 - \prod_{n' \in \mathcal{N}(m) \setminus n} (q_{mn'}^0 - q_{mn'}^1) \right) \end{array} \right. \quad (3.1.3)$$

(ii) Vertical step, for each bit n and each $m \in \mathcal{M}(n)$, update q_{mn}^x by

$$\left\{ \begin{array}{l} q_{mn}^0 = \alpha_{mn} f_n^0 \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m'n}^0 \\ q_{mn}^1 = \alpha_{mn} f_n^1 \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m'n}^1 \end{array} \right. \quad (3.1.4)$$

where α_{mn} is a normalization factor such that $q_{mn}^0 + q_{mn}^1 = 1$. For each n , update q_n^0 and q_n^1 by

$$\left\{ \begin{array}{l} q_n^0 = \alpha_n f_n^0 \prod_{m' \in \mathcal{M}(n)} r_{m'n}^0 \\ q_n^1 = \alpha_n f_n^1 \prod_{m' \in \mathcal{M}(n)} r_{m'n}^1 \end{array} \right. \quad (3.1.5)$$

where α_n is a normalization factor such that $q_n^0 + q_n^1 = 1$.

Step 2: Hard decision and stopping criterion test:

- (i) Create $\hat{c} = [\hat{c}_n]$ such that $\hat{c}_n = 0$ if $q_n^0 > q_n^1$, and $\hat{c}_n = 1$ otherwise.
- (ii) If $H\hat{c} = \mathbf{0}$ or I_{Max} is reached, stop the decoding iteration and go to Step 3. Otherwise go to Step 1.

Step 3: Output \hat{c} as the decoded codeword.

3.1.2 LLR BP Algorithm

Messages are represented by LLRs in LLR BP algorithm. We define the following notation associated with the i -th iteration.

- $U_{ch,n}$: The LLR of bit n which is derived from the channel output y_n , i.e., $U_{ch,n} \triangleq \ln \left(\frac{f_n^0}{f_n^1} \right)$. For the AWGN channel with conditional pdf's in (3.1.1), we have

$$U_{ch,n} = \frac{4}{N_0} y_n. \quad (3.1.6)$$

- $U_{mn}^{(i)}$: The LLR of bit n which is sent from the check node m to bit node n , i.e.,

$$U_{mn}^{(i)} \triangleq \ln \left(\frac{r_{mn}^0}{r_{mn}^1} \right). \quad (3.1.7)$$

- $V_{mn}^{(i)}$: The LLR of bit n which is sent from the bit node n to check node m , i.e.,

$$V_{mn}^{(i)} \triangleq \ln \left(\frac{q_{mn}^0}{q_{mn}^1} \right). \quad (3.1.8)$$

- $V_n^{(i)}$: The *a posteriori* LLR of bit n , i.e.,

$$V_n^{(i)} \triangleq \ln \left(\frac{q_n^0}{q_n^1} \right). \quad (3.1.9)$$

The LLR BP algorithm is carried out as follows [20]:

Initialization: Set $i = 1$, and the maximum number of iterations to I_{Max} . For each m and n , set

$$V_{mn}^{(0)} = U_{ch,n}. \quad (3.1.10)$$

Step 1: Iterative decoding

- (i) Horizontal step, for $1 \leq n \leq N$ and each $m \in \mathcal{M}(n)$, process:

$$U_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{V_{mn'}^{(i-1)}}{2} \right) \right). \quad (3.1.11)$$

- (ii) Vertical step, for $1 \leq n \leq N$ and each $m \in \mathcal{M}(n)$, process:

$$V_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} U_{m'n}^{(i)}. \quad (3.1.12)$$

Step 2: Hard decision and stopping criterion test:

- (i) Set $V_n^{(i)} = U_{ch,n} + \sum_{m \in \mathcal{M}(n)} U_{mn}^{(i)}$. Create $\hat{c}^{(i)} = [\hat{c}_n^{(i)}]$ such that $\hat{c}_n^{(i)} = 1$ if $V_n^{(i)} < 0$, and $\hat{c}_n^{(i)} = 0$ otherwise.
- (ii) If $H\hat{c}^{(i)} = \mathbf{0}$ or I_{Max} is reached, stop the decoding iteration and go to Step 3. Otherwise set $i := i + 1$ and go to Step 1.

Step 3: Output $\hat{c}^{(i)}$ as the decoded codeword.

To distinguish from the modified BP algorithms that will be discussed later, we call the BP algorithm presented in this section the standard BP algorithm.

3.2 Simplified BP Algorithms

3.2.1 BP-Based Algorithm

The complexity of standard BP decoding is dominated by the non-linear functions, $\tanh(x)$ and $\tanh^{-1}(x)$, in check node processing. Since both $\tanh(x)$ and $\tanh^{-1}(x)$ are odd functions, with the properties

$$\begin{cases} \tanh(x) = \text{sgn}(x) \cdot \tanh(|x|) \\ \tanh^{-1}(x) = \text{sgn}(x) \cdot \tanh^{-1}(|x|), \end{cases} \quad (3.2.1)$$

the check node processing following (3.1.11) can be expressed by

$$\begin{aligned} U_{mn}^{(i)} &= 2 \tanh^{-1} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \text{sgn}(V_{mn'}^{(i-1)}) \cdot \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{|V_{mn'}^{(i-1)}|}{2} \right) \right) \\ &= \prod_{n' \in \mathcal{N}(m) \setminus n} \text{sgn}(V_{mn'}^{(i-1)}) \cdot 2 \tanh^{-1} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{|V_{mn'}^{(i-1)}|}{2} \right) \right). \end{aligned} \quad (3.2.2)$$

Then the following approximation can be applied

$$\begin{aligned} \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{|V_{mn'}^{(i-1)}|}{2} \right) &\approx \min_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{|V_{mn'}^{(i-1)}|}{2} \right) \\ &= \tanh \left(\frac{\min_{n' \in \mathcal{N}(m) \setminus n} |V_{mn'}^{(i-1)}|}{2} \right). \end{aligned} \quad (3.2.3)$$

Therefore, (3.1.11) in standard BP can be simplified by

$$U_{mn}^{(i)} = \prod_{n' \in \mathcal{N}(m) \setminus n} \text{sgn}(V_{mn'}^{(i-1)}) \cdot \min_{n' \in \mathcal{N}(m) \setminus n} |V_{mn'}^{(i-1)}|. \quad (3.2.4)$$

This algorithm is referred to as BP-based algorithm or min-sum algorithm [56][57]. The BP-based algorithm is much simpler than the BP algorithm since only additions and comparisons are used in check node processing and bit node processing. Moreover, for the AWGN channel, another advantage of the BP-based algorithm is that it does not need the channel information since the constant term $4/N_0$ in (3.1.6) has no effect on check node comparison. Hence, (3.1.10) in standard BP can be modified to $V_{mn}^{(0)} = y_n$ for each m and n .

3.2.2 Normalized BP-Based Algorithm

Denote $U_{mn}^{(i)}$ in standard BP by $U_1^{(i)}$ and $U_{mn}^{(i)}$ in the BP-based algorithm by $U_2^{(i)}$.

The normalized BP-based algorithm was derived based on two properties [59]:

1. The signs of $U_1^{(i)}$ and $U_2^{(i)}$ are the same.
2. The magnitude of $U_2^{(i)}$ is always greater than that of $U_1^{(i)}$.

To approximate $U_1^{(i)}$ from $U_2^{(i)}$, $U_{mn}^{(i)}$ in the normalized BP-based algorithm is updated by

$$U_{mn}^{(i)} \leftarrow U_2^{(i)} / \alpha \quad (3.2.5)$$

where α is a normalization factor greater than 1.

In [64], a two-dimensional correction algorithm has been proposed for irregular LDPC codes, which assigns a normalization factor to each variable degree and each check degree. This new algorithm achieves better performance for irregular codes compared with that of the normalized BP-based algorithm. However, for irregular codes with few degrees only, the normalized BP-based algorithm can still have satisfactory performance. An alternative correction to scaling is offsetting, with similar performances [60][64].

3.2.3 Simulation Results

Figure 3.1 depicts the word error rate (WER) performance of the standard BP, BP-based and normalized BP-based algorithms of the (1056, 704) irregular LDPC code from [65]. The degree distributions of variable nodes and check nodes are $\lambda(x) = 0.172840x + 0.037037x^2 + 0.790123x^3$ and $\rho(x) = 0.864198x^9 + 0.135802x^{10}$, respectively. The maximum number of iterations equals 100. The normalization factor of the normalized BP-based algorithm is set to 1.4. We observe that the performance gap between the BP-based algorithm and standard BP is about 0.25 dB. The normalized BP-based algorithm reduces this gap to less than 0.05 dB for the waterfall region. In the high SNR region, the performance of the normalized BP-based algorithm is even slightly better than that of standard BP.

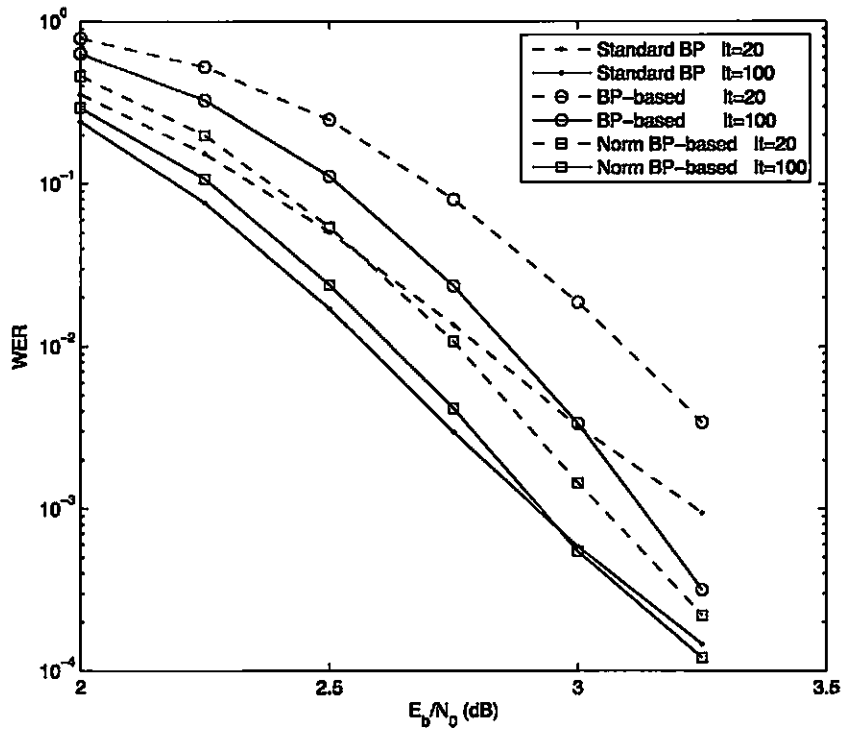


Figure 3.1: The WER performance of the standard BP, BP-based and normalized BP-based algorithms for a (1056, 704) irregular LDPC code.

3.3 Fast Convergence BP Algorithms

3.3.1 Plain Shuffled BP Decoding of LDPC Codes

The basic idea of plain shuffled BP decoding is to process the horizontal step and vertical step in standard BP **jointly** so that they can exchange and benefit from new information within the same iteration. Following the definitions in [63], deterministic schedulings can be implemented either based on horizontal [26, 27] or vertical partitioning [24, 25] of the parity check matrix.

Vertical shuffled BP decoding

Iteration- i of the two-step implementation of the standard BP algorithm uses all values $V_{mn'}^{(i-1)}$ computed at the previous iteration in (3.1.12). However certain values $V_{mn'}^{(i)}$ could already be computed based on a partial computation of the values $U_{mn}^{(i)}$ obtained from (3.1.11), and then be used instead of $V_{mn'}^{(i-1)}$ in (3.1.11) to compute the remaining values $U_{mn}^{(i)}$.

In the vertical shuffled BP algorithm, the initialization, stopping criterion test and output steps remain the same as in the standard BP algorithm. The only difference between the two algorithms lies in the updating procedure. Step 1 of the shuffled BP algorithm is modified as: for $1 \leq n \leq N$ and each $m \in \mathcal{M}(n)$, process the horizontal step and vertical step jointly, with (3.1.11) modified as:

$$U_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' < n}} \tanh \left(\frac{V_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' > n}} \tanh \left(\frac{V_{mn'}^{(i-1)}}{2} \right) \right). \quad (3.3.1)$$

Horizontal shuffled BP decoding

In the horizontal shuffled BP algorithm, stopping criterion test and output steps remain the same as in the standard BP algorithm. The initialization and the updating procedure are modified as

Initialization: Set $i = 1$, and the maximum number of iterations to I_{Max} . For each m and n , set $U_{mn}^{(0)} = 0$.

Step 1: For $1 \leq m \leq M$ and each $n \in \mathcal{N}(m)$, process:

(i) Vertical step:

$$V_{mn}^{(i)} = U_{ch,n} + \sum_{\substack{m' \in \mathcal{M}(n) \setminus m \\ m' < m}} U_{m'n}^{(i)} + \sum_{\substack{m' \in \mathcal{M}(n) \setminus m \\ m' > m}} U_{m'n}^{(i-1)} \quad (3.3.2)$$

(ii) Horizontal step:

$$U_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{V_{mn'}^{(i)}}{2} \right) \right). \quad (3.3.3)$$

Vertical shuffled BP and horizontal shuffled BP have similar error performances, which are shown in Section 3.3.4. In hardware implementation, both of them have their own advantages. This issue is discussed in Chapter 5.

3.3.2 Replica Shuffled BP Decoding of LDPC Codes

Shuffled BP decoding described in Section 3.3.1 is based on a natural increasing updating order, i.e., in vertical shuffled BP, the messages at bit nodes are updated according to the order $n = 1, 2, \dots, N$ and in horizontal shuffled BP, the messages at check nodes are updated according to the order $m = 1, 2, \dots, M$. The larger the value of n or m , the more reliable these messages become. If more decoders are used, they can exchange their most reliable messages with one another and achieve faster convergence. This algorithm is referred to as replica shuffled BP decoding. In the following, we describe replica shuffled BP decoding based on vertical partitioning. It can be readily applied to horizontal partitioning.

In replica shuffled BP decoding, several shuffled sub-decoders based on different updating orders operate simultaneously and cooperatively. After each iteration, each sub-decoder receives more reliable messages from and sends more reliable messages to other sub-decoders. Based on these more reliable messages, all replica sub-decoders begin the next iteration. Hence replica decoding can be viewed as a way to parallelize iterations. For two replicas, let \vec{D} and \overleftarrow{D} denote the sub-decoders with natural increasing and decreasing updating orders, respectively. Let $\vec{U}_{mn}^{(i)}$ and $\overleftarrow{V}_{mn}^{(i)}$ be the variables associated with \vec{D} at iteration i . The variables associated with \overleftarrow{D} are defined in a similar way. The replica shuffled BP decoding with two replica sub-decoders is carried out as follows:

Initialization: Set $i = 1$, and the maximum number of iterations to I_{Max} . For each m and n , set $\vec{V}_{mn}^{(0)} = \overleftarrow{V}_{mn}^{(0)} = U_{ch,n}$.

Step 1: Each replica sub-decoder processes the following two steps simultaneously. For $1 \leq n \leq N$ and each $m \in \mathcal{M}(n)$, process

(i) Horizontal step

$$\vec{U}_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' < n}} \tanh \left(\frac{\vec{V}_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' > n}} \tanh \left(\frac{\vec{V}_{mn'}^{(i-1)}}{2} \right) \right)$$

$$\overleftarrow{U}_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' > n}} \tanh \left(\frac{\overleftarrow{V}_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' < n}} \tanh \left(\frac{\overleftarrow{V}_{mn'}^{(i-1)}}{2} \right) \right)$$

(ii) Vertical step

$$\vec{V}_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} \vec{U}_{m'n}^{(i)}$$

$$\overleftarrow{V}_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} \overleftarrow{U}_{m'n}^{(i)}$$

Step 2: Set $\vec{V}_{mn}^{(i)} = \overleftarrow{V}_{mn}^{(i)}$ for $1 \leq n \leq N/2$ and $\overleftarrow{V}_{mn}^{(i)} = \vec{V}_{mn}^{(i)}$ for $N/2 < n \leq N$.

Step 3: Hard decision and stopping criterion test:

(i) Create $\hat{c}^{(i)} = [\hat{c}_n^{(i)}]$ such that for $1 \leq n \leq N/2$, $\hat{c}_n^{(i)} = 1$ if $U_{ch,n} + \sum_{m \in \mathcal{M}(n)} \vec{U}_{mn}^{(i)} < 0$, and $\hat{c}_n^{(i)} = 0$ otherwise; for $N/2 < n \leq N$, $\hat{c}_n^{(i)} = 1$ if $U_{ch,n} + \sum_{m \in \mathcal{M}(n)} \overleftarrow{U}_{mn}^{(i)} < 0$, and $\hat{c}_n^{(i)} = 0$ otherwise.

(ii) If $H\hat{c}^{(i)} = \mathbf{0}$ or I_{Max} is reached, stop the decoding iteration and go to Step 4. Otherwise set $i := i + 1$ and go to Step 1.

Step 4: Output $\hat{c}^{(i)}$ as the decoded codeword.

Another possible implementation is to let these two sub-decoders exchange more reliable messages synchronously with each other during the decoding process. Define $R(n) = \{n' | n \leq n' \leq N - n\}$, and $\bar{R}(n) = \{n' | 1 \leq n' \leq N, n' \notin R(n)\}$, for $1 \leq n \leq N$. In synchronous scheme, the updating and exchanging procedures operate simultaneously as follows:

Step 1: For $1 \leq n \leq N$ and each $m \in \mathcal{M}(n)$, for $p = N - n$ and each $q \in \mathcal{M}(p)$, two replica sub-decoders process the following two steps simultaneously

(i) Horizontal step

$$U_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \in \bar{R}(n)}}} \tanh \left(\frac{V_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \in \bar{R}(n)}}} \tanh \left(\frac{V_{mn'}^{(i-1)}}{2} \right) \right)$$

$$U_{qp}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{q' \in \mathcal{N}(q) \setminus p \\ q' \in \bar{R}(N-p)}}} \tanh \left(\frac{V_{qp'}^{(i)}}{2} \right) \prod_{\substack{q' \in \mathcal{N}(q) \setminus p \\ q' \in \bar{R}(N-p)}}} \tanh \left(\frac{V_{qp'}^{(i-1)}}{2} \right) \right).$$

(ii) Vertical step

$$V_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} U_{m'n}^{(i)}$$

$$V_{qp}^{(i)} = U_{ch,p} + \sum_{q' \in \mathcal{M}(p) \setminus q} U_{q'p}^{(i)}$$

Notice that in this case the two replica sub-decoders use the same set of bit-to-check LLR values. It is also straightforward to extend the replica shuffled BP decoding to the cases in which more than two replica sub-decoders are used.

For most Gallager type LDPC codes, synchronous replica shuffled BP achieves a similar ultimate performance as that of non-synchronous replica shuffled BP. However for some LDPC codes with relatively higher density parity check matrices (such as EG-LDPC codes or PG-LDPC codes), non-synchronous replica shuffled BP may provide a better ultimate performance than that of synchronous decoding. In Table 3.1, synchronous

and non-synchronous replica shuffled BP algorithms with 2 sub-decoders and 200 iterations are compared for the (273,191) PG-LDPC code (the large iteration number was chosen to ensure convergence in both cases). For this code, the non-synchronous schedule provides a better performance. The synchronous approach reduces storage by half but increases memory conflicts.

SNR (dB)	Non-synchronous	Synchronous
2.0	1.5e-2	3.0e-2
2.5	3.0e-3	5.0e-3
3.0	3.5e-4	6.3e-4
3.5	2.0e-5	5.7e-5

Table 3.1: Performance comparison of non-synchronous and synchronous replica shuffled BP decoding for the (273,191) PG-LDPC code.

3.3.3 Group Plain Shuffled BP and Group Replica Shuffled BP Decoding

To take advantage of as many newly delivered messages as possible and therefore to achieve the best performance, a fully serial replica shuffled BP is necessary. However, this scheme is not attractive for hardware implementation due to its serial nature. A totally parallel implementation is not realistic either for large code lengths, or codes with highly connected graph.

In [24], a method called “group shuffled” BP was presented. This “group” concept can also be used in replica shuffled BP decoding. Our description of group plain and group replica shuffled BP is still based on vertical partitioning.

In group plain shuffled BP, the bits of a codeword are processed in groups in a semi-parallel manner. The groups are processed serially while the bits within a group are processed in parallel. Assume the N bits of a codeword are divided into G groups and each group contains $\frac{N}{G} = B$ bits (assuming $N \bmod G = 0$ for simplicity).

Step 1 of the group shuffled BP algorithm is carried out as follows:

Step 1: For $1 \leq g \leq G$,

(i) Horizontal step: for $(g-1) \cdot B + 1 \leq n \leq g \cdot B$ and each $m \in \mathcal{M}(n)$, process:

$$U_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \leq (g-1) \cdot B}} \tanh \left(\frac{V_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \geq (g-1) \cdot B + 1}} \tanh \left(\frac{V_{mn'}^{(i-1)}}{2} \right) \right)$$

(ii) Vertical step: for $(g-1) \cdot B + 1 \leq n \leq g \cdot B$ and each $m \in \mathcal{M}(n)$, process:

$$V_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} U_{m'n}^{(i)}$$

Step 1 of the non-synchronous group replica shuffled BP algorithm is carried out as follows:

Step 1: For $1 \leq g \leq G$, each replica sub-decoder processes jointly the following two steps

(i) Horizontal step: for $(g-1) \cdot B + 1 \leq n \leq g \cdot B$ and each $m \in \mathcal{M}(n)$, process:

$$\bar{U}_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \leq (g-1) \cdot B}} \tanh \left(\frac{\bar{V}_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \geq (g-1) \cdot B + 1}} \tanh \left(\frac{\bar{V}_{mn'}^{(i-1)}}{2} \right) \right)$$

$$\bar{U}_{mn}^{(i)} = 2 \tanh^{-1} \left(\prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \geq (G-g+1) \cdot B + 1}} \tanh \left(\frac{\bar{V}_{mn'}^{(i)}}{2} \right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \leq (G-g+1) \cdot B}} \tanh \left(\frac{\bar{V}_{mn'}^{(i-1)}}{2} \right) \right)$$

(ii) Vertical step: for $(g-1) \cdot B + 1 \leq n \leq g \cdot B$ and each $m \in \mathcal{M}(n)$, process:

$$\bar{V}_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} \bar{U}_{m'n}^{(i)}$$

$$\bar{V}_{mn}^{(i)} = U_{ch,n} + \sum_{m' \in \mathcal{M}(n) \setminus m} \bar{U}_{m'n}^{(i)}$$

Synchronous group replica shuffled decoding is defined in a similar way.

Replica shuffled BP can also update messages in groups based on nonnatural increasing or decreasing orders. Suppose the updating order of one replica is $\mathcal{O} = \{O_1, O_2,$

$\dots, O_G\}$, where $O_g \in \{1, 2, \dots, G\}$. Assume the updating orders of \vec{D} and \overleftarrow{D} are \vec{O} and \overleftarrow{O} , respectively. Then replica shuffled BP with nonnatural updating ordering can be described with the above updating rules by replacing $(g - 1) \cdot B$ and $(G - g + 1) \cdot B$ with $\vec{O}_g \cdot B$ and $\overleftarrow{O}_g \cdot B$, respectively.

Replica shuffled BP can be further generalized to various forms. One example is that in the nonnatural updating scheme, some groups of bit nodes may be updated more than once at one iteration while other groups of bit nodes are updated only once. The updating of LLR values at the i -th iteration is now based on the LLR values delivered at the $(i - 1)$ -th or $(i - 2)$ -th iteration.

Relationship between group plain and group replica shuffled BP

Group plain shuffled BP can be viewed as a special case of synchronous group replica shuffled BP. Assume in group plain shuffled BP decoding, the N bits of a codeword are divided into $G = \frac{N}{B}$ groups and each group contains B bits. Consider a group replica shuffled BP decoder with two sub-decoders D_1 and D_2 . For both D_1 and D_2 , the N bits of a codeword are divided into $2G = \frac{N}{(B/2)}$ groups and each group contains $B/2$ bits. For $1 \leq g \leq G$, let bits in group- $(2g - 1)$ in D_1 and bits in group- $2g$ in D_2 compose group- g in group plain shuffled BP decoding. In synchronous group replica shuffled BP decoding, if sub-decoder D_1 updates group- $(2g - 1)$ and sub-decoder D_2 updates group- $2g$ simultaneously, group replica shuffled BP decoding with two sub-decoders becomes group plain shuffled BP decoding.

Since each sub-decoder in group replica shuffled BP decoder can take any updating order, group replica shuffled BP decoding provides more flexibility than group plain shuffled BP decoding. Hence we can expect to find some scheduling for group replica shuffled BP decoder that has better performance than group plain shuffled BP using the same decoding time and the same hardware resources, i.e., the same number of sub-decoders. For example, consider a $(16200, 7200)$ irregular LDPC code which was constructed in a semi-random manner [67]. The variable node and check node degree distributions are $\lambda(x) = 0.00006x + 0.57772x^2 + 0.3111x^3 + 0.11111x^8$ and $\rho(x) = 0.00006x^2 + 0.14917x^3 + 0.29851x^4 + 0.44777x^5 + 0.10449x^6$, respectively. We compare

group plain shuffled BP decoding with 2 groups and group replica shuffled BP decoding with 2 sub-decoders and 4 groups.

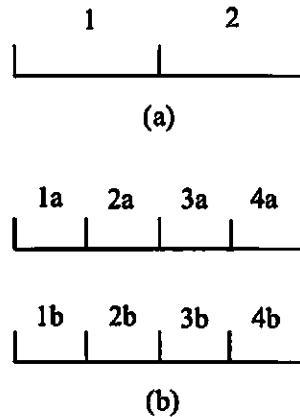


Figure 3.2: Illustration of the scheduling of group plain shuffled BP decoding with 2 groups and group replica shuffled BP decoding with 2 sub-decoders and 4 groups.

Figure 3.2 (a) illustrates the scheduling of group plain shuffled BP decoding. The variable nodes are divided into 2 parts, 1 and 2. The processing follows the order: $1 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow \dots$. Figure 3.2 (b) illustrates the scheduling of group replica shuffled BP decoding. With respect to the first sub-decoder, the variable nodes are divided into 4 parts, $1a, 2a, 3a$ and $4a$. The processing follows the order: $1a \rightarrow 2a \rightarrow 1a \rightarrow 2a \rightarrow 3a \rightarrow 4a \rightarrow 1a \rightarrow 2a \rightarrow 1a \rightarrow 2a \rightarrow 3a \rightarrow 4a \dots$. With respect to the second sub-decoder, the variable nodes are divided into $1b, 2b, 3b$ and $4b$. The processing follows the order: $2b \rightarrow 1b \rightarrow 4b \rightarrow 3b \rightarrow 2b \rightarrow 1b \rightarrow 2b \rightarrow 1b \rightarrow 4b \rightarrow 3b \rightarrow 2b \rightarrow 1b \dots$. Since the decoding time for one iteration in group replica shuffled BP triples that in group plain shuffled BP decoding, we compare their performance after 6 and 18 iterations, respectively, in Table 3.2. We observe that with this particular scheduling and the same number of sub-decoders, group replica shuffled BP outperforms group plain shuffled BP.

3.3.4 Simulation Results

Figure 3.3 depicts the WER of iterative decoding of a $(8000, 4000)(3, 6)$ LDPC code, with standard BP, vertical plain shuffled BP, vertical group plain shuffled BP for

SNR (dB)	group plain shuffled BP (It=18)	group replica shuffled BP (It=6)
1.1	9.8e-4	8.6e-4
1.2	2.6e-4	2.0e-4
1.3	6.2e-5	4.0e-5
1.4	1.5e-5	1.0e-5
1.5	4.3e-6	1.8e-6

Table 3.2: Performance comparison of group plain and group replica shuffled BP decoding.

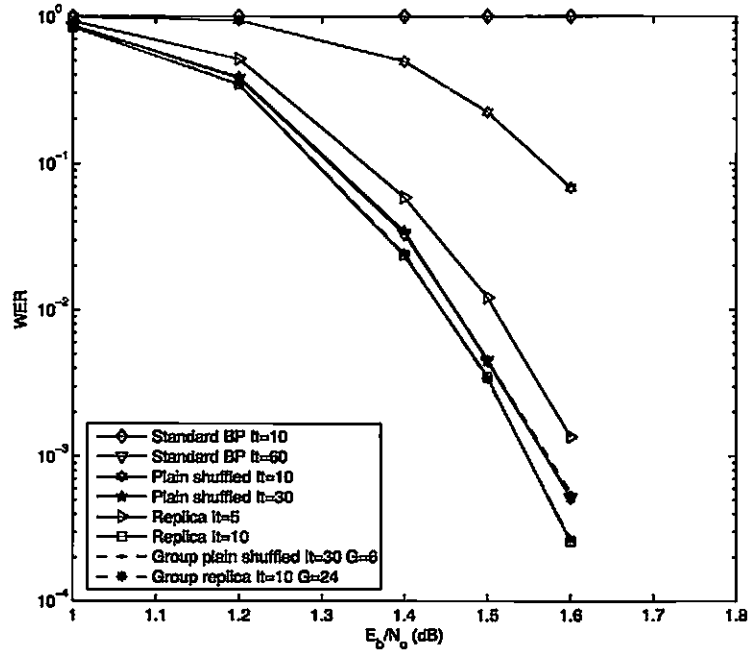


Figure 3.3: WER of a $(8000, 4000)(3, 6)$ LDPC code with standard BP, vertical plain shuffled BP, vertical group plain shuffled BP for $G = 6$, vertical replica shuffled BP with four sub-decoders and synchronous updating and its group version with $G = 24$.

$G = 6$, vertical replica shuffled BP with four sub-decoders and synchronous updating and its group version with $G = 24$. We observe that the WER performance of replica shuffled BP with 10 iterations is approximately the same as that of plain shuffled BP with 30 iterations and standard BP with 60 iterations. Vertical group plain shuffled BP with 6 groups and 30 iterations has almost the same performance as that of vertical plain shuffled BP with 30 iterations. Vertical group replica shuffled BP with 24 groups and 10 iterations has almost the same performance as that of vertical replica shuffled BP with 10 iterations.

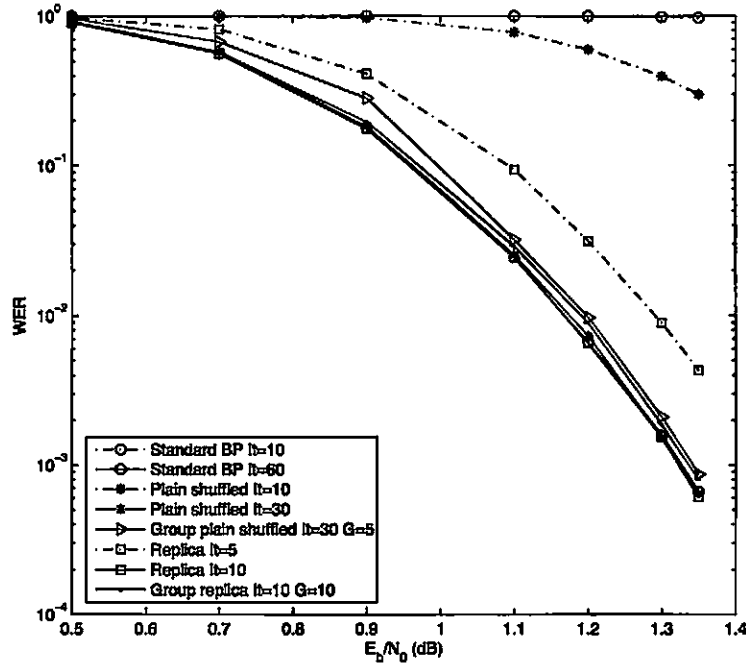


Figure 3.4: WER of a $(4000, 2000)$ irregular LDPC code with standard BP, horizontal plain shuffled BP, horizontal group plain shuffled BP for $G = 5$, horizontal replica shuffled BP with four sub-decoders and synchronous updating and its group version with $G = 10$.

Figure 3.4 depicts the WER of a $(4000, 2000)$ irregular LDPC code with standard BP, horizontal plain shuffled BP, horizontal group plain shuffled BP for $G = 5$, horizontal replica shuffled BP with four sub-decoders and synchronous updating and its group version with $G = 10$. The degree distribution is from [22] with $d_{v_{max}} = 15$. The variable node and check node degree distributions are $\lambda(x) = 0.23802x + 0.20997x^2 + 0.03492x^3 +$

$0.12015x^4 + 0.01587x^6 + 0.00480x^{13} + 0.37627x^{14}$ and $\rho(x) = 0.98013x^7 + 0.01987x^8$, respectively. For this irregular LDPC code decoded in horizontal partitioning, we obtain similar conclusions about convergence speed as those of the regular LDPC code in vertical partitioning in Figure 3.3. We also observe that for this code, horizontal group plain shuffled BP with 5 groups and 30 iterations has similar performance as that of horizontal plain shuffled BP with 30 iterations. Horizontal group replica shuffled BP with 10 groups and 10 iterations has similar performance as that of horizontal replica shuffled BP with 10 iterations.

Figure 3.5 depicts the WER comparison of the same irregular LDPC code with plain shuffled BP and replica shuffled BP using vertical partitioning and horizontal partitioning. The number of replica sub-decoders was four and updating was synchronous. With 30 iterations we observe that the performance of vertical partitioning and horizontal partitioning is almost the same.

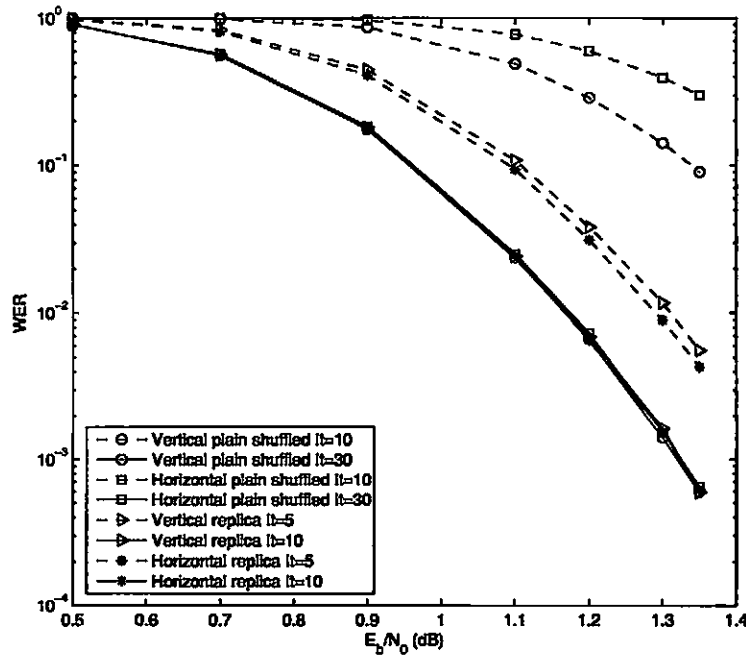


Figure 3.5: WER of a (4000, 2000) irregular LDPC code with plain shuffled BP and replica shuffled BP using vertical partitioning and horizontal partitioning.

Chapter 4

Analysis of Fast Convergence BP Algorithms by EXIT Charts

EXIT charts [68]-[74] are an effective tool to study the convergence behavior of iterative decoding. They are easy to visualize and to program and are a good complement to density evolution [30].

For LDPC codes using standard BP decoding, EXIT charts are especially useful because the EXIT functions can be calculated in closed forms. This enables us to easily estimate the threshold, the speed of convergence and even the bit error rate (BER) in the waterfall region of LDPC codes.

In this chapter, we propose closed-form EXIT functions for LDPC codes using plain shuffled BP, replica shuffled BP and their group versions. These EXIT functions allow us to analyze the above decoding algorithms and compare their speed of convergence from a theoretical view point. We also obtain some interesting results for group plain shuffled BP and group replica shuffled BP, which indicate the minimum number of groups these two decoding algorithms need to achieve almost the same performance as their corresponding non-group counterparts.

4.1 Introduction to EXIT Charts

EXIT charts were first introduced for serial concatenated codes [68, 69] by ten Brink. They were then extended to parallel concatenated codes [70], LDPC codes [71] and repeat accumulate (RA) codes [72]. In the EXIT charts, each EXIT curve depicts the

relationship between the input *a priori* messages and the output extrinsic messages of a soft-in/soft-out decoder based on mutual information. The exchange of extrinsic information is visualized as a decoding trajectory in the EXIT charts.

EXIT-chart techniques are based on the empirical evidence that extrinsic messages in LLR forms can be approximated by a Gaussian random variable that satisfies the consistency condition [75]. Suppose the message associated with the binary value x is t , with $x \in \{\pm 1\}$. Then t can be represented as $t = x + n$ with $n \sim \mathcal{N}(0, \sigma_n^2)$. Let the LLR of the message be Λ . Then $\Lambda|x \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu = \frac{2}{\sigma_n^2}x$ and $\sigma^2 = \frac{4}{\sigma_n^2}$. The mutual information between x and Λ can be written as

$$I(x; \Lambda) = J(\sigma) = 1 - \int_{-\infty}^{\infty} \frac{e^{-(\xi - \sigma^2/2)^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}} \cdot \log_2[1 + e^{-\xi}] d\xi. \quad (4.1.1)$$

Suppose the mutual information between the *a priori* messages and x is I_A and the mutual information between the extrinsic messages and x is I_E . Then I_E can be written as a function of I_A , which is referred to as an EXIT function.

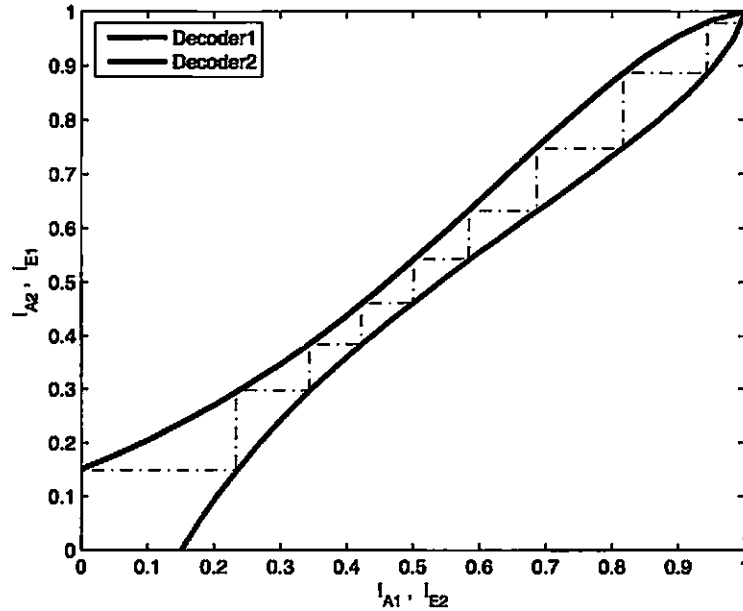


Figure 4.1: EXIT charts of a 2-component turbo code with interleaver size 16384 and using parallel decoding at the SNR of 0.15 dB.

Figure 4.1 depicts the EXIT charts of a 2-component turbo code with interleaver size 16384 and using parallel decoding. The top and bottom EXIT curves are for decoder-1 and decoder-2, respectively. Since the extrinsic messages of decoder-1 serve as the *a priori* messages for decoder-2, the axes are swapped for the EXIT curve of decoder-2. From the trajectory between the two EXIT curves, we can easily trace the exchange of extrinsic information between the two decoders.

4.2 EXIT Charts of LDPC Codes Using Standard BP Decoding

EXIT charts of LDPC codes contain two curves with one representing the EXIT function of the variable nodes and the other representing that of the check nodes.

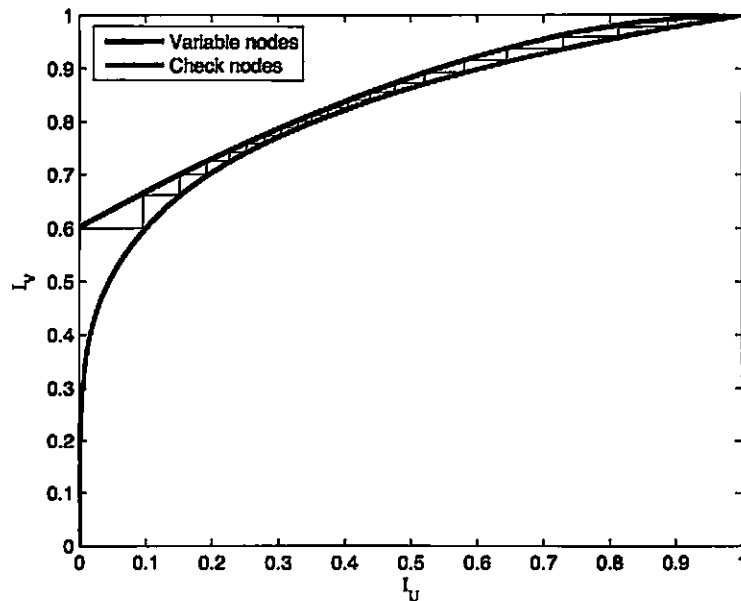


Figure 4.2: EXIT charts of a (3,6) regular LDPC code at the SNR of 1.5 dB.

Figure 4.2 depicts the EXIT charts of a (3,6) regular LDPC code at the SNR of 1.5 dB. The x axis I_U is the average mutual information between the bits on the edges of

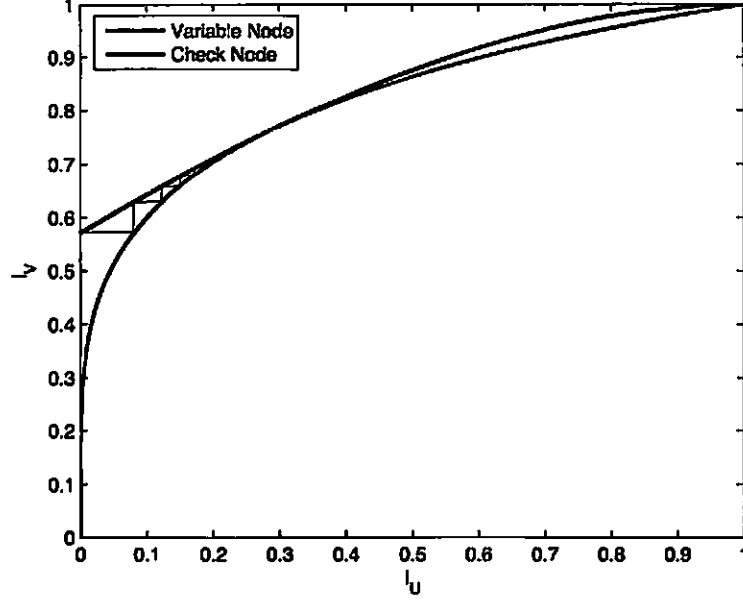


Figure 4.3: EXIT charts of a (3,6) regular LDPC code at the SNR of 1.11 dB.

the Tanner graph and the *a priori* (extrinsic) LLRs of the variable (check) nodes. The y axis I_V is the average mutual information between the bits on the edges of the Tanner graph and the extrinsic (*a priori*) LLRs of the variable (check) nodes. In Figure 4.2, the top curve is for variable nodes and the bottom curve is for check nodes. If the two curves do not cross each other, then the decoding trajectory finally converges to the top right corner, i.e., point (1,1), and decoding is successful. If the two curves intersect as shown in Figure 4.3, then the trajectory is blocked at the crossing point and decoding fails.

Both the variable node and check node EXIT curves can be computed in closed form [71] for standard BP decoding. The EXIT functions of a degree- d_v variable node and a degree- d_c check node are respectively

$$I_{V,STD} \left(I_U, d_v, \frac{E_b}{N_0}, R \right) = J \left(\sqrt{(d_v - 1)[J^{-1}(I_U)]^2 + \sigma_{ch}^2} \right) \quad (4.2.1)$$

$$I_{U,STD} (I_V, d_c) \approx 1 - J \left(\sqrt{d_c - 1} \cdot J^{-1}(1 - I_V) \right) \quad (4.2.2)$$

where $\sigma_{ch}^2 = 8R \cdot \frac{E_b}{N_0}$ and $J(\cdot)$ is defined as in (4.1.1); $J^{-1}(\cdot)$ is the inverse function of $J(\cdot)$. The approximation functions of $J(\cdot)$ and $J^{-1}(\cdot)$ are given in the Appendix of [71].

4.3 EXIT Charts of Plain Shuffled BP

In order to find a closed form for shuffled BP decoding, the following ideal model is constructed for a regular LDPC code. First, vertical partitioning is considered. Suppose the variable nodes can be divided into d_c sets and those in the i -th set only connect to the i -th edge of the check nodes. For example, regular QC-LDPC codes have this feature. This ideal model is also suitable for codes constructed using the PEG method. The parity check matrix corresponding to this ideal model is referred to as the “ideal” parity check matrix. Figure 4.4 illustrates an example of the ideal parity check matrix of a (2,3) regular LDPC code with length 9. In the following, we derive the EXIT functions only for vertical shuffled BP decoding. The principle can be readily generalized to horizontal shuffled BP decoding.

Set 1	Set 2	Set 3
1 0 0	0 0 1	0 1 0
0 1 0	1 0 0	0 0 1
0 0 1	0 1 0	1 0 0
0 1 0	1 0 0	0 0 1
0 0 1	0 1 0	1 0 0
1 0 0	0 0 1	0 1 0

Figure 4.4: An example illustrating the ideal parity check matrix of a (2,3) regular LDPC code with length 9.

Based on the above ideal model, since all the edges of the variable nodes in the same set connect to different check nodes, they can not benefit from one another. However they can equally make use of the updated information of the previous edges. The processing of each check node also becomes identical.

Let the mutual information between the bits on any edge connected to a check node and their corresponding *a priori* LLRs be equal to the average input mutual information I_V . Let I_{V_i} be the updated mutual information between the bit on the i -th edge of the same check node and its *a priori* LLRs. Denote by I_{U_i} the mutual information between the bit on the i -th edge of this check node and its extrinsic LLRs. Then the EXIT function for

a check node of a (d_v, d_c) regular LDPC code decoded with shuffled BP decoding is

$$I_{U,SHF}(I_V, d_c) = \frac{1}{d_c} \sum_{i=1}^{d_c} I_{U_i} \quad (4.3.1)$$

It is worth mentioning that for standard BP, I_{U_i} 's are the same for all edges of a check node since all of them are processed simultaneously. However, that is not the case for plain shuffled BP. In vertical plain shuffled BP, variable nodes are processed in a fully serial manner and in our ideal model, since the edges of a check node are processed serially, I_{U_i} is improved as i increases. For example, consider the ideal parity check matrix in Figure 4.4. Figure 4.5 illustrates its updating process using vertical plain shuffled BP. Since the processing of each check node is identical, Figure 4.5 depicts only one check node. The dark dots in Figure 4.5 represent the variable nodes that are being processed. Based on the ideal model assumption, we know that the i -th edge of all the check nodes only connects to the variable nodes in the i -th set. Assuming variable nodes are processed from set-1 to set-3, all the i -th edges of the check nodes are processed before any j -th edge of any check node for $j > i$. When the variable nodes in set-1 are processed, they take the output extrinsic information I_{U_1} from the first edges of the check nodes as their input *a priori* information as shown in Figure 4.5 (b). Since the *a priori* information of a check node is I_V initially, following (4.2.2), we have $I_{U_1} \approx I_{U,STD}(I_V, d_c)$. Based on (4.2.1), the output extrinsic information from the variable nodes in set-1 is $I'_{V_1} = I_{V,STD}\left(I_{U_1}, d_v, \frac{E_b}{N_0}, R\right)$ as shown in Figure 4.5 (c). Then the updating of set-1 is completed. Next we process the variable nodes in set-2 as shown in Figure 4.5 (d) and (e). The variable nodes in set-2 take the output extrinsic information I_{U_2} from the second edges of the check nodes as their input *a priori* information. To calculate I_{U_2} , we follow (4.2.2) and take the average of I'_{V_1} and I_V as the input *a priori* information, i.e., $I_{U_2} \approx I_{U,STD}\left(\frac{I'_{V_1} + I_V}{2}, d_c\right)$. Then based on (4.2.1) the output extrinsic information I'_{V_2} of the variable nodes in set-2 equals $I_{V,STD}\left(I_{U_2}, d_v, \frac{E_b}{N_0}, R\right)$. Finally we process the variable nodes in set-3. They take the output extrinsic information I_{U_3} from the third edges of the check nodes as their input *a priori* information. Similarly, I_{U_3} is obtained from (4.2.2) with the average of I'_{V_1} and I'_{V_2} as the input *a priori* information, i.e., $I_{U_3} \approx I_{U,STD}\left(\frac{I'_{V_1} + I'_{V_2}}{2}, d_c\right)$. Then the variable nodes in

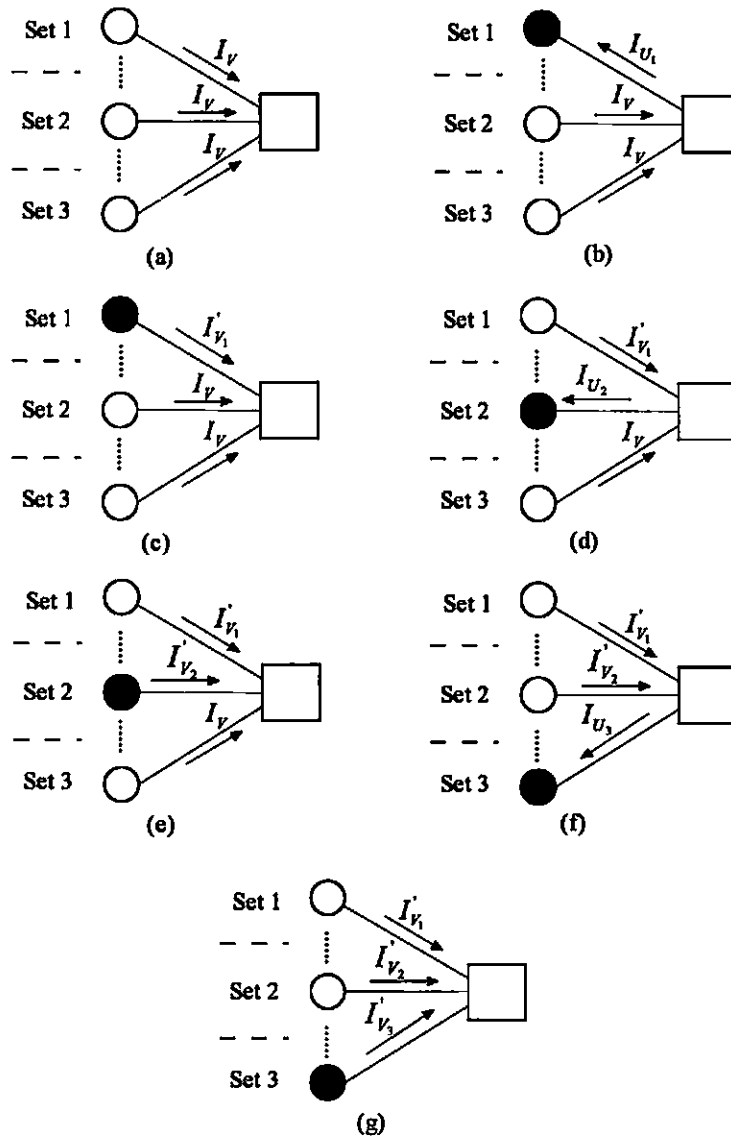


Figure 4.5: The mutual information updating process for the LDPC code with the ideal parity check matrix given in Figure 4.4.

the third set have output extrinsic information I_{V_3} , which equals $I_{V,STD} \left(I_{U_3}, d_v, \frac{E_b}{N_0}, R \right)$ as shown in Figure 4.5 (g); one iteration is completed.

The above updating process can be generalized to any (d_v, d_c) regular LDPC code with the ideal parity check matrix, i.e.,

$$I_{U_i} = I_{U,STD} \left(\frac{(d_c - i)I_V + \sum_{k=1}^{i-1} I_{V_k}}{d_c - 1}, d_c \right) \quad (4.3.2)$$

$$I_{V_i} = I_{V,STD} \left(I_{U_i}, d_v, \frac{E_b}{N_0}, R \right) \quad (4.3.3)$$

for $i = 1, 2, \dots, d_c$.

The average input mutual information of all the variable nodes is

$$I_{U_{av}} = \sum_{i=1}^{d_c} I_{U_i} / d_c$$

and the average output mutual information is

$$I_{V_{av}} = \sum_{i=1}^{d_c} I_{V,STD}(I_{U_i}, d_v, \frac{E_b}{N_0}, R) / d_c.$$

The EXIT function for a variable node in shuffled BP decoding is given by

$$I_{V,SHF} \left(I_{U_{av}}, d_v, \frac{E_b}{N_0}, R \right) = I_{V_{av}}. \quad (4.3.4)$$

Next, we compare $I_{V,STD}$ and $I_{V,SHF}$. Let $J_1(\sigma^2) = J(\sigma)$ and $I_{U_i} = J_1(\sigma_i^2)$. Since $J_1(\sigma^2)$ is approximately linear with σ^2 when σ^2 is within a small range, we obtain in that case $I_{U_{av}} = \sum_{i=1}^{d_c} I_{U_i} / d_c = \sum_{i=1}^{d_c} J_1(\sigma_i^2) / d_c \approx J_1(\frac{1}{d_c} \sum_{i=1}^{d_c} \sigma_i^2)$. Therefore, it follows

$$\begin{aligned} I_{V,STD} \left(I_{U_{av}}, d_v, \frac{E_b}{N_0}, R \right) &= J_1 \left((d_v - 1) J_1^{-1}(I_{U_{av}}) + \sigma_{ch}^2 \right) \\ &\approx J_1 \left((d_v - 1) \left(\frac{1}{d_c} \sum_{i=1}^{d_c} \sigma_i^2 \right) + \sigma_{ch}^2 \right) \\ &= J_1 \left(\frac{1}{d_c} \sum_{i=1}^{d_c} ((d_v - 1)\sigma_i^2 + \sigma_{ch}^2) \right) \end{aligned}$$

$$\begin{aligned}
&\approx \frac{1}{d_c} \sum_{i=1}^{d_c} J_1 \left((d_v - 1) \sigma_i^2 + \sigma_{ch}^2 \right) \\
&= \frac{1}{d_c} \sum_{i=1}^{d_c} I_{V,STD} \left(I_{U_i}, d_v, \frac{E_b}{N_0}, R \right) \\
&= I_{V,SHF} \left(I_{U_{av}}, d_v, \frac{E_b}{N_0}, R \right).
\end{aligned}$$

From simulations, we observe that the variances σ_i^2 of the *a priori* inputs to different variable nodes at one iteration vary within a small range. Hence the EXIT function for a variable node in shuffled BP decoding is almost the same as that in standard BP decoding.

4.4 EXIT Charts of Replica Shuffled BP

It is straightforward to extend this method to replica shuffled BP. Using a similar approach, we can show that the EXIT function for a variable node in replica shuffled BP decoding is also almost the same as that in standard BP decoding. Since in the non-synchronous scheme, sub-decoders only exchange information at the end of each iteration, the EXIT function for a check node in replica shuffled BP with two sub-decoders and the non-synchronous updating can be written as

$$I_{U,REP_2,NS}(I_V, d_c) = \frac{1}{d_c} \sum_{i=d_c/2}^{d_c} 2I_{U_i} \quad (\text{even } d_c) \quad (4.4.1)$$

$$I_{U,REP_2,NS}(I_V, d_c) = \frac{1}{d_c} \left(\sum_{i=\lceil d_c/2 \rceil + 1}^{d_c} 2I_{U_i} + I_{U_{\lceil d_c/2 \rceil}} \right) \quad (\text{odd } d_c). \quad (4.4.2)$$

The EXIT function for a check node in replica shuffled BP with more than two sub-decoders can be obtained in a similar way.

In the synchronous scheme, sub-decoders exchange information immediately. Suppose D sub-decoders are used. Then we can divide each of the d_c sets of the ideal

model into D subsets. Each sub-decoder processes the variable nodes in a distinct subset of the same set at the same time. After all the variable nodes have been processed once, the sub-decoders go back to the first set and process a subset different from those they have already processed. Thus, in this case the replica shuffled BP can be regarded as applying the shuffled BP D times. Therefore the EXIT function for a check node in the synchronous scheme with D sub-decoders is given by

$$I_{U,REP_{D,S}}(I_V, d_c) = I_{U,SHF}(I_{V_D}, d_c) \quad (4.4.3)$$

$$I_{V_i} = I_{V,SHF}\left(I_{U,SHF}(I_{V_{i-1}}, d_c), d_v, \frac{E_b}{N_0}, R\right) \quad i = 2, 3, \dots, D \quad (4.4.4)$$

with $I_{V_1} = I_V$.

While these derivations allow us to model the convergence of each method, it is well known that the threshold derived on a tree can not be changed by modifying the scheduling of the algorithm only. Therefore the threshold value remains the same for all methods as expressed in the next theorem.

Theorem 1. *Based on EXIT chart analysis, the threshold of a code decoded by plain shuffled BP or replica shuffled BP is the same as for BP.*

Proof. Let γ be the threshold in standard BP decoding. When $E_b/N_0 \leq \gamma$, the EXIT curves of variable and check nodes cross each other at some point, say A . If $I_E = I_{V,STD}(I_A, d_v, \frac{E_b}{N_0}, R)$, then $I_A = I_{U,STD}(I_E, d_c)$. In plain shuffled BP decoding, if we use I_E as the input *a priori* information to check nodes, then the extrinsic information of the first edge in a check node is I_A because $I_A = I_{U,STD}(I_E, d_c)$. Variable nodes take I_A as input and send back I_E to check nodes because $I_E = I_{V,STD}(I_A, d_v, \frac{E_b}{N_0}, R)$. It follows that the input mutual information to check nodes is not improved during the process of updating variable nodes serially, i.e., $I_{U_i} \equiv I_A$ and $I_{V_i} \equiv I_E$. So $I_{U,SHF}(I_E, d_c) = I_A$ and $I_E = I_{V,SHF}(I_A, d_v, \frac{E_b}{N_0}, R)$, which means the EXIT curves of variable and check nodes in plain shuffled BP also cross each other at the point A . The same result can be proved for replica shuffled BP. \square

In general, the Tanner graph of an LDPC code does not satisfy all the constraints of our ideal model, but in many cases the convergence behavior can still be well approximated by the ideal model as shown next. Figure 4.6 compares the EXIT functions obtained

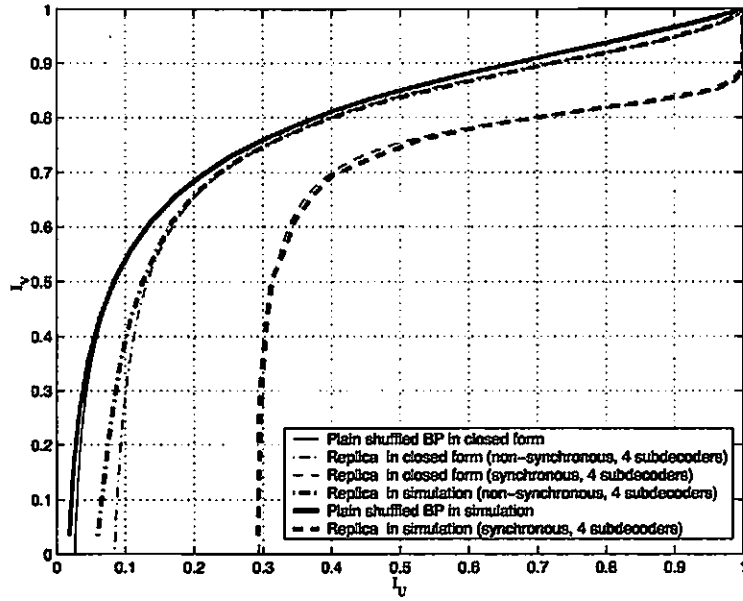


Figure 4.6: Comparison between the EXIT curves obtained from the simulation method of [74] and the proposed closed forms for a (3, 6) regular LDPC code at the SNR 1.5 dB.

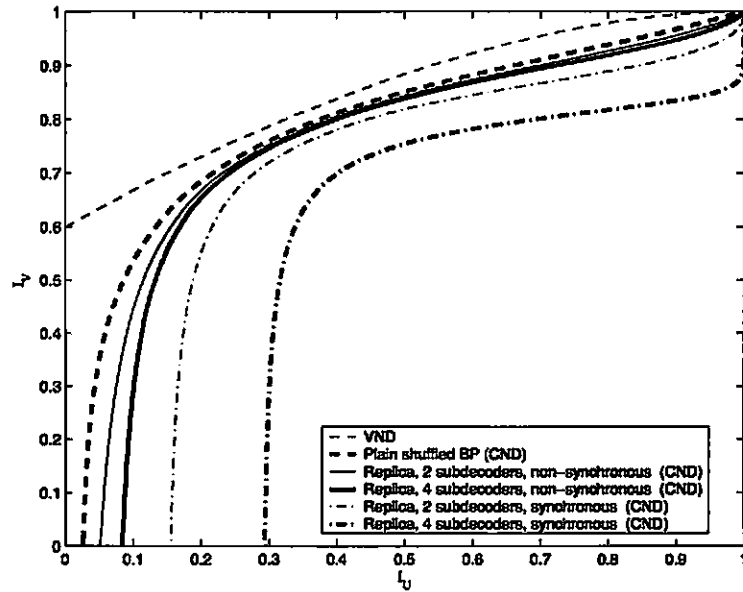


Figure 4.7: EXIT curves (in closed form) for plain shuffled BP and four types of replica shuffled BP decodings for a (3, 6) regular LDPC code at the SNR 1.5 dB (variable nodes (VND) and check nodes (CND)).

from the simulation method of [74] and the proposed closed forms for a $(3, 6)$ regular LDPC code at the SNR value of 1.5 dB. Both methods assume the input LLRs have a Gaussian distribution. We observe that the EXIT functions of these two methods are almost the same, which validates the derived EXIT functions.

We also verified by EXIT charts that the non-synchronous scheduling converges slower than the synchronous one. Figure 4.7 depicts the EXIT charts of five decoding methods. We observe that replica shuffled BP with four sub-decoders using the synchronous scheme converges much faster than the other methods.

Figure 4.8 depicts EXIT curves superimposed to constant-BER curves [76, Chapter 9]. For the same BER, we observe that the iteration number of standard BP is twice that of plain shuffled BP and 8 times that of replica shuffled BP with four sub-decoders and synchronous updating.

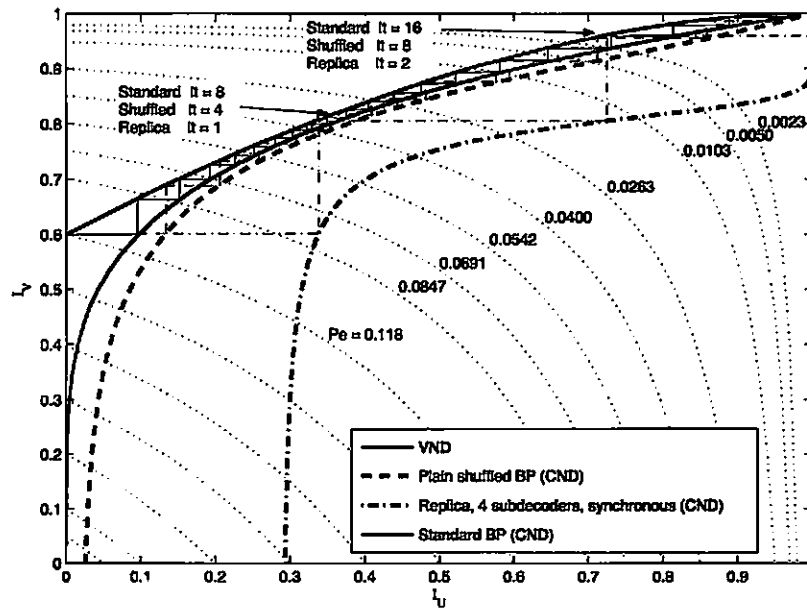


Figure 4.8: EXIT curves (in closed form) for standard BP, plain shuffled BP and replica shuffled BP with four sub-decoders with synchronous updating for a $(3, 6)$ regular LDPC code at the SNR 1.5 dB, superimposed to constant-BER curves.

Figure 4.9 depicts the EXIT curves of different decoding methods at the SNR 1.11 dB, which is the threshold of the (3, 6) regular LDPC code. We observe that the EXIT curves of variable and check nodes cross each other at the same point for all the methods. Hence they have the same threshold as expected from Theorem 1.

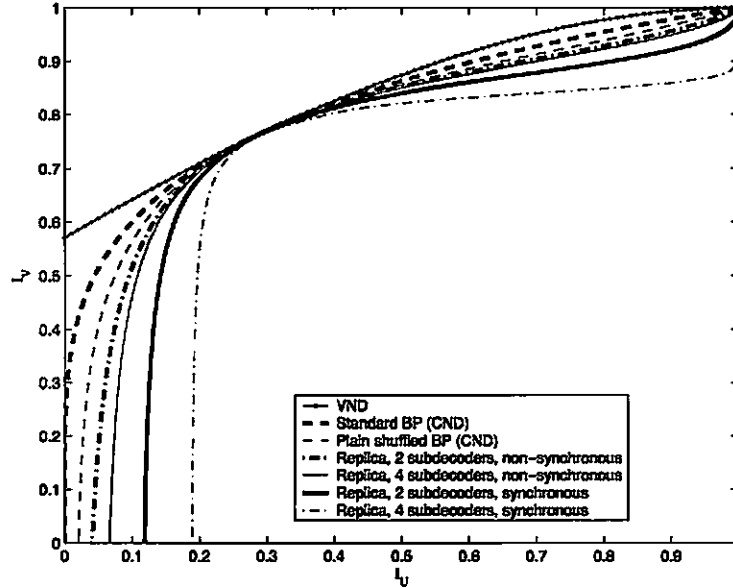


Figure 4.9: EXIT curves (in closed form) for standard BP, plain shuffled BP and four types of replica shuffled BP for a (3, 6) regular LDPC code at the SNR 1.11 dB (threshold).

The extension of these results to irregular LDPC codes follows in a straightforward way.

4.5 EXIT Charts of Group Plain Shuffled BP

In this section, we investigate the minimum number of groups necessary for a group decoding to achieve the same performance as the fully serial corresponding decoding. Based on the analysis of plain shuffled BP, we deduce the following theorem.

Theorem 2. *When decoding a regular LDPC code, vertical group plain shuffled BP should have at least d_c groups in order to have at any given iteration the same performance as vertical plain shuffled BP based on the ideal model.*

This result directly follows from the ideal parity check matrix as the variable nodes in each set do not benefit from each other and they can be processed in parallel without changing the performance. Simulation results confirm that this value is a good estimate of the least number of groups necessary to achieve the same performance as plain shuffled BP. Consequently Theorem 2 indicates that the speed-up obtained by shuffled BP over standard BP can still be achieved with a high level of parallelism since in general d_c is quite small. For completeness, we develop the case $G < d_c$ next.

When the group number is less than d_c , the EXIT function of vertical group plain shuffled BP can be easily obtained if the check node degree is divisible by the group number, but it becomes cumbersome otherwise. Let G be the number of groups. Suppose the check node degree d_c is divisible by G with $S_G = d_c/G$. Then the EXIT function of vertical group plain shuffled BP can be described as

$$I_{U,SHF,GRG}(I_V, d_c) = \frac{1}{d_c} \sum_{i=1}^{d_c} I_{U_i}. \quad (4.5.1)$$

If $i \bmod S_G = 1$, then

$$I_{U_i} = I_{U,STD} \left(\frac{(d_c - i)I_V + \sum_{k=1}^{i-1} I_{V_k}}{d_c - 1}, d_c \right) \quad (4.5.2)$$

$$I_{V_i} = I_{V,STD} \left(I_{U_i}, d_v, \frac{E_b}{N_0}, R \right). \quad (4.5.3)$$

Otherwise,

$$I_{U_i} = I_{U_m} \quad (4.5.4)$$

$$I_{V_i} = I_{V_m} \quad (4.5.5)$$

where $m = \lfloor (i-1)/S_G \rfloor \cdot S_G + 1$. It is readily checked that I_{U_i} and I_{V_i} obtained from (4.5.4) and (4.5.5) can not be larger than those obtained from (4.3.2) and (4.3.3). Therefore in the case $G < d_c$, vertical group plain shuffled BP performs worse than vertical plain shuffled BP at any given iteration.

Similar results can be obtained for horizontal group plain shuffled BP.

Theorem 3. *When decoding a regular LDPC code, horizontal group plain shuffled BP should have at least d_v groups in order to have at any given iteration the same performance as horizontal plain shuffled BP based on the ideal model.*

In horizontal shuffled BP, instead of dividing variable nodes into d_c sets, we divide check nodes into d_v sets. The check nodes in each set do not benefit from each other and they can be processed in parallel without changing the performance.

Similar results can also be obtained for irregular codes. Suppose we divide the variable nodes of an irregular LDPC code into P groups. Denote by P_{min} the smallest P so that no variable nodes in each group have common check nodes. Then vertical group plain shuffled BP should have at least P_{min} groups in order to have at any given iteration the same performance as vertical plain shuffled BP. Similarly, suppose we divide the check nodes of an irregular LDPC code into Q groups. Denote by Q_{min} the smallest Q so that no check nodes in each group have common variable nodes. Then horizontal group plain shuffled BP should have at least Q_{min} groups in order to have at any given iteration the same performance as horizontal plain shuffled BP.

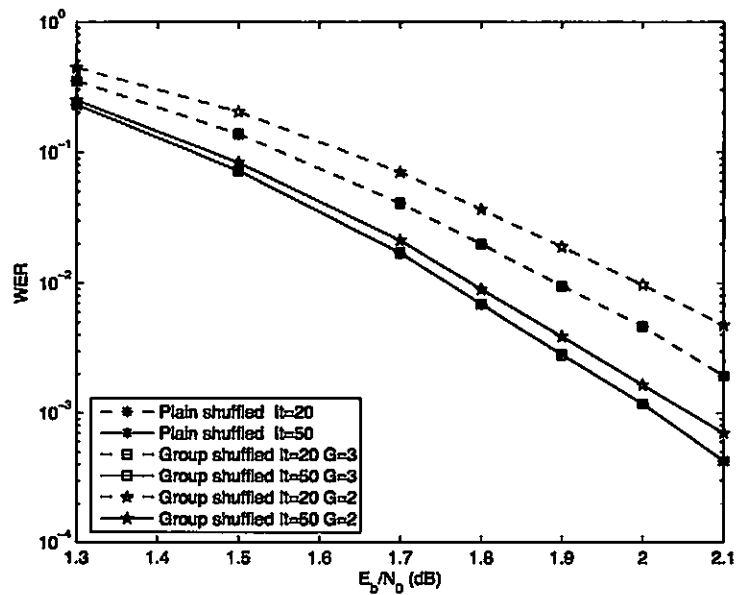


Figure 4.10: WER of horizontal plain shuffled BP and horizontal group plain shuffled BP with 3 groups and 2 groups for decoding a (1800, 902) (3, 6) regular QC-LDPC code.

Figure 4.10 depicts the WER performance of horizontal plain shuffled BP and horizontal group plain shuffled BP with 3 groups and 2 groups for decoding a (1800, 902)

(3, 6) regular QC-LDPC code with girth 8. The base matrix of this code is

$$\begin{bmatrix} 22 & 40 & 5 & 25 & 57 & 77 \\ 2 & 76 & 0 & 33 & 7 & 47 \\ 59 & 50 & 3 & 43 & 56 & 71 \end{bmatrix}_{3 \times 6} \quad (4.5.6)$$

Since this code has the QC structure that belongs to our ideal model, based on Theorem 3 horizontal group plain shuffled BP with 3 groups should have at any given iteration the same performance as horizontal plain shuffled BP. We verify this property in Figure 4.10 as the two algorithms do have the same performance at any given iteration. When the number of groups is reduced to 2, the performance of horizontal group plain shuffled BP gets worse.

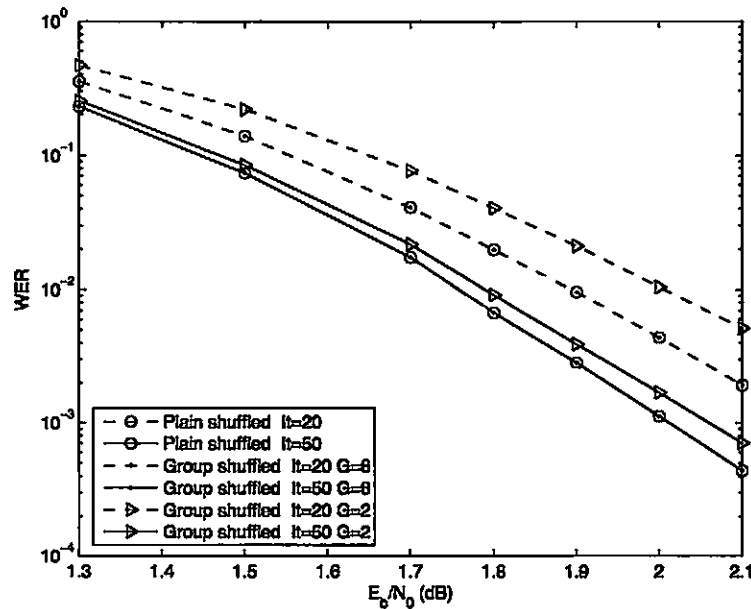


Figure 4.11: WER of vertical plain shuffled BP and vertical group plain shuffled BP with 6 groups and 2 groups for decoding a (1800, 902) (3, 6) regular QC-LDPC code.

Figure 4.11 depicts the WER performance of vertical plain shuffled BP and vertical group plain shuffled BP with 6 groups and 2 groups for decoding the same code as in Figure 4.10. Based on Theorem 2, vertical group plain shuffled BP with 6 groups should

have at any given iteration the same performance as vertical plain shuffled BP, which is verified in Figure 4.11. When the number of groups is reduced to 2, the performance of vertical group plain shuffled BP becomes worse. We have also tried other codes than QC-LDPC

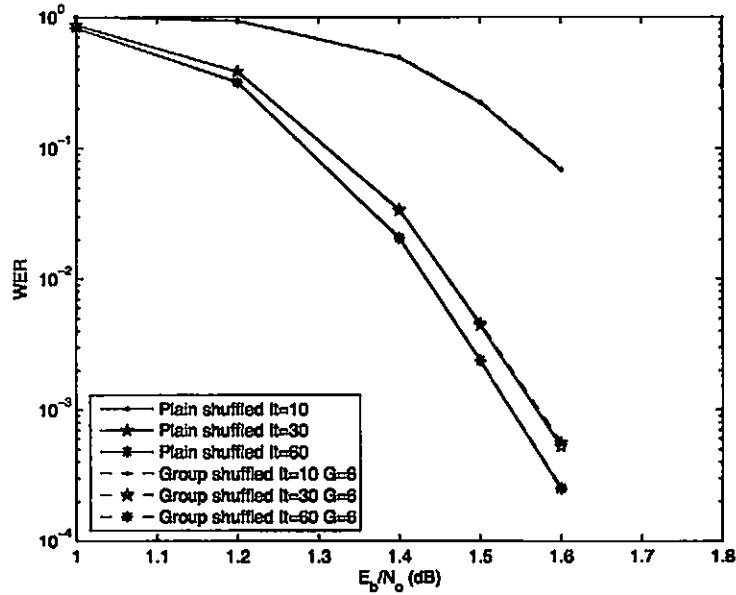


Figure 4.12: WER of vertical plain shuffled BP and vertical group plain shuffled BP with 6 groups for decoding a $(8000, 4000)$ $(3, 6)$ regular LDPC code.

codes, for which the ideal model no longer holds. Figure 4.12 depicts the WER performance of vertical plain shuffled BP and vertical group plain shuffled BP with 6 groups for decoding a $(8000, 4000)$ $(3, 6)$ regular LDPC code, which is constructed by the PEG method [43]. Since the number of the bit nodes, 8000, cannot be divided by 6, the remaining bit nodes are assigned to the corresponding last group. From Figure 4.12, we observe that vertical group plain shuffled BP with 6 groups has almost the same performance as its corresponding non-group counterpart. Figure 4.13 depicts the WER performance of horizontal plain shuffled BP and horizontal group plain shuffled BP with 2, 3, and 6 groups for decoding a $(2016, 1512)$ irregular QC-LDPC code from [65]. The value Q_{min} for this code is 6. We observe that horizontal group plain shuffled BP with Q_{min} groups has the same performance as horizontal plain shuffled BP. Again when the number of groups decreases from this minimum value, the performance becomes worse.

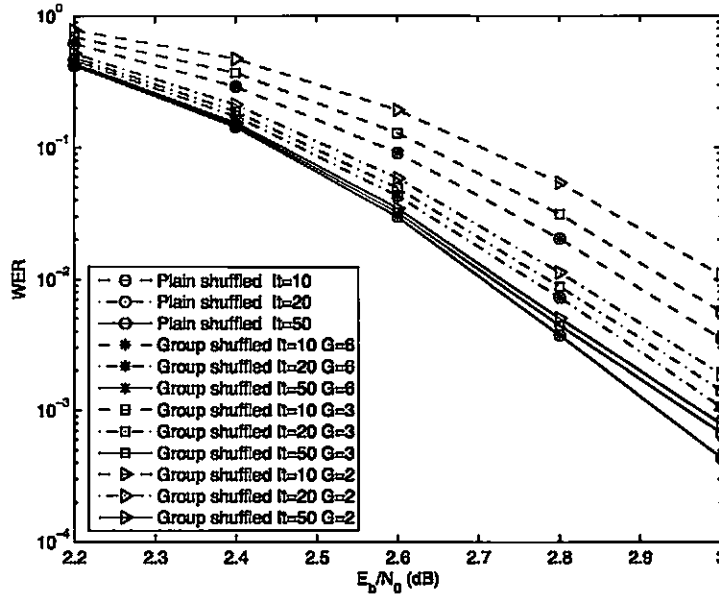


Figure 4.13: WER of horizontal plain shuffled BP and horizontal group plain shuffled BP with 2, 3, and 6 groups for decoding a (2016, 1512) irregular QC-LDPC code.

4.6 EXIT Charts of Group Replica Shuffled BP

In this section we analyze group replica shuffled BP based on vertical partitioning in this section. The results can be readily generalized to horizontal partitioning.

The EXIT function of group replica shuffled BP with non-synchronous updating is almost the same as that of replica shuffled BP (i.e., $G = N$) except that I_{U_i} 's in (4.4.1) and (4.4.2) are obtained from (4.5.2) and (4.5.4).

For the synchronous scheme, when $G \leq D$, group replica shuffled BP can be regarded as applying standard BP G times. Therefore the corresponding EXIT function is

$$I_{U,REP,D,S,GR_G}(I_V, d_c) = I_{U,STD}(I_{V_G}, d_c) \quad (4.6.1)$$

$$I_{V_i} = I_{V,STD}\left(I_{U,STD}(I_{V_{i-1}}, d_c), d_v, \frac{E_b}{N_0}, R\right) \quad i = 2, 3, \dots, G \quad (4.6.2)$$

where $I_{V_1} = I_V$. When $D \cdot d_c > G > D$, if G is divisible by D and d_c is divisible by $\frac{G}{D}$, group replica shuffled BP is equivalent to applying group shuffled BP with $\frac{G}{D}$ groups D

times. Let $T = \frac{G}{D}$. Then the EXIT function becomes

$$I_{U,REP_D,S,GR_G}(I_V, d_c) = I_{U,SHF,GR_T}(I_{V_D}, d_c) \quad (4.6.3)$$

$$I_{V_i} = I_{V,SHF,GR_T} \left(I_{U,SHF,GR_T}(I_{V_{i-1}}, d_c), d_v, \frac{E_b}{N_0}, R \right) \quad i = 2, 3, \dots, D \quad (4.6.4)$$

where $I_{V_1} = I_V$. When $G \geq D \cdot d_c$, the EXIT function of group replica shuffled BP with synchronous updating is the same as for $G = N$. Hence we have the following theorem.

Theorem 4. *When decoding a regular LDPC code, vertical group replica shuffled BP should have at least $D \cdot d_c$ groups in order to have at any given iteration the same performance as vertical replica shuffled BP based on the ideal model.*

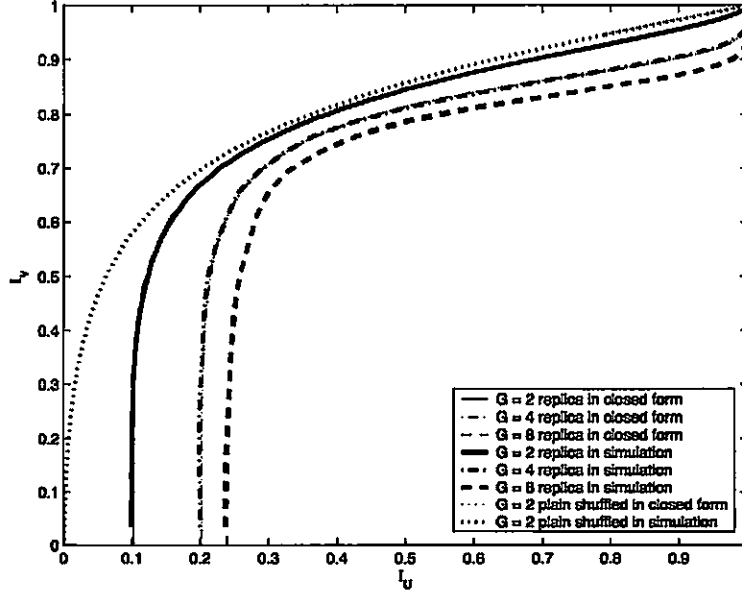


Figure 4.14: Comparison between the EXIT curves obtained from the simulation method of [74] and the proposed closed forms for vertical group plain shuffled BP and vertical group replica shuffled BP with four sub-decoders and synchronous updating, for decoding a (3, 6) regular LDPC code at the SNR 1.5 dB.

Figure 4.14 depicts the EXIT curves obtained from the simulation method of [74] and the proposed closed forms for vertical group plain shuffled BP and vertical group

replica shuffled BP with four sub-decoders and synchronous updating. We observe that the curves obtained with these two methods match each other well, which again validates our derived EXIT functions.

Figure 4.15 depicts the WER performance of vertical replica and vertical group replica shuffled BP with 24 groups with four sub-decoders and synchronous updating for decoding a $(8000, 4000)$ $(3, 6)$ regular LDPC code, whose Tanner graph was constructed by the PEG method. As in Section 4.5, since 8000 cannot be divided by 24, the remaining bit nodes are assigned to the corresponding last group. From this figure, we observe that vertical group replica shuffled BP with the smallest group number G derived theoretically in Theorem 4 has almost the same performance as its corresponding non-group counterpart.

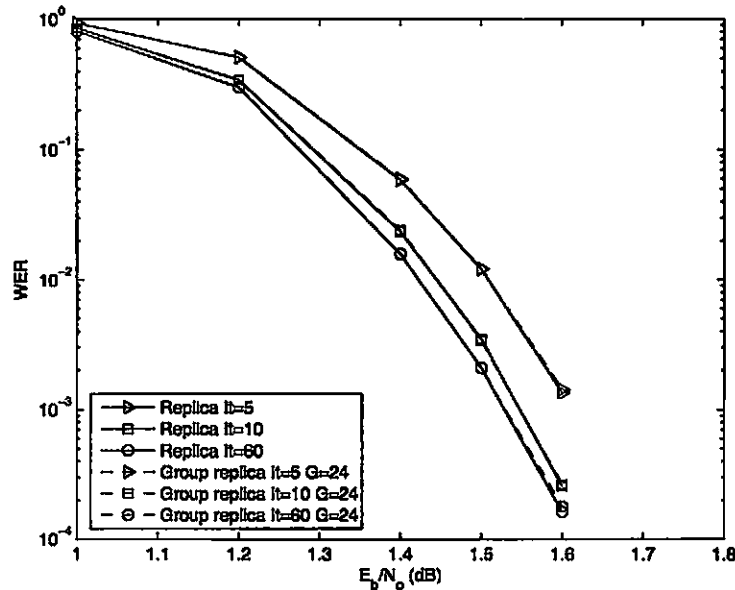


Figure 4.15: WER of vertical replica and vertical group replica shuffled BP with 24 groups with four sub-decoders and synchronous updating for decoding a $(8000, 4000)$ $(3, 6)$ regular LDPC code.

Chapter 5

Hardware Implementation of Replica Shuffled Normalized BP-Based Decoder

As discussed in Chapter 1, compared to turbo codes [17], LDPC codes can achieve lower error floor, better threshold, higher degree of parallelism and simpler decoding processing. These advantages allow LDPC codes to become strong competitors to turbo codes. In fact, they have been adopted in next generation digital video broadcasting (DVB-S2) via satellite [67] and IEEE 802.16e [65] and considered for adoption in many other standards, such as wireless local area network (WLAN) air interface (802.11) [96]. With more and more applications of LDPC codes, the development of LDPC codec VLSI architectures has become an important issue. A fully parallel decoder [31] can have large throughput, but it is not practical for relatively long codes because of high complexity. A serial decoder [32] has reasonable complexity, but it sacrifices throughput and is not suitable for high speed applications. Thus partly parallel decoders that achieve appropriate trade-off between complexity and throughput are highly desirable. Many partly parallel decoder architectures have been proposed [33] [97]. In [33], $(3, k)$ regular LDPC codes were mainly discussed and a joint design approach was introduced. In [97], turbo decoding message passing (TDMP) decoding was proposed, then it was referred to as layered decoding in [98]. This algorithm is equivalent to horizontal group plain shuffled BP decoding discussed in Chapter 3 and it converges faster than standard BP decoding. In Chapter 3, we also introduced the replica shuffled BP decoding algorithm, which can further speed up the convergence compared with plain shuffled BP decoding. In this chapter, we combine

replica shuffled decoding with a simplified BP algorithm and design a VLSI architecture for codes in the 802.16e standard using replica shuffled normalized BP-based decoding.

5.1 Code Structure in the 802.16e Standard

The LDPC codes in the 802.16e standard have quasi-cyclic structures. The parity check matrix H can be obtained from expanding its corresponding base matrix $B = [B_{ij}]$. All base matrices have a constant number of columns, which equals 24, while the number of rows varies according to the code rate. Let p be the dimension of a circulant permutation matrix and N be the length of the code, then $p = N/24$. To obtain H , each -1 in the base matrix is replaced with a $p \times p$ zero matrix and any other number is replaced with a $p \times p$ circulant permutation matrix. In the base matrix the part that corresponds to parity check bits has a dual-diagonal structure and the first column of this part has adopted an “ a -0- a ” structure, which enables efficient encoding. Figure 5.1 describes the base matrix of rate-2/3 LDPC codes in the IEEE 802.16e standard.

$$\begin{bmatrix} 2 & -1 & 19 & -1 & 47 & -1 & 48 & -1 & 36 & -1 & 82 & -1 & 47 & -1 & 15 & -1 & 95 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 69 & -1 & 88 & -1 & 33 & -1 & 3 & -1 & 16 & -1 & 37 & -1 & 40 & -1 & 48 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ 10 & -1 & 86 & -1 & 62 & -1 & 28 & -1 & 85 & -1 & 16 & -1 & 34 & -1 & 73 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\ -1 & 28 & -1 & 32 & -1 & 81 & -1 & 27 & -1 & 88 & -1 & 5 & -1 & 56 & -1 & 37 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\ 23 & -1 & 29 & -1 & 15 & -1 & 30 & -1 & 66 & -1 & 24 & -1 & 50 & -1 & 62 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\ -1 & 30 & -1 & 65 & -1 & 54 & -1 & 14 & -1 & 0 & -1 & 30 & -1 & 74 & -1 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\ 32 & -1 & 0 & -1 & 15 & -1 & 56 & -1 & 85 & -1 & 5 & -1 & 6 & -1 & 52 & -1 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\ -1 & 0 & -1 & 47 & -1 & 13 & -1 & 61 & -1 & 84 & -1 & 55 & -1 & 78 & -1 & 41 & 95 & -1 & -1 & -1 & -1 & -1 & -1 & 0 \end{bmatrix}$$

Figure 5.1: Base matrix of rate-2/3 LDPC codes in the IEEE 802.16e standard.

5.2 Decoding Algorithm

The check node processing in BP decoding involves the calculation of $\tanh(\cdot)$ and $\tanh^{-1}(\cdot)$ functions, which are usually realized by Loop-Up Table (LUT) in hardware implementation. Since the number of LUTs increases with the number of check node processors and check node degree, we adopt the normalized BP-based algorithm to avoid using

LUTs. As discussed in Chapter 3, plain and replica shuffled BP can be implemented based on either horizontal or vertical partitioning of the parity check matrix. From a hardware implementation point of view, each partitioning has its own advantages and drawbacks. Vertical partitioning processes along variable nodes, therefore syndrome computation can be carried out during decoding. For horizontal partitioning, usually syndrome can only be calculated at the end of each iteration; however, in practice we can fix the number of iterations and remove the syndrome computation part. As shown in Section 3.3.1, the initialization step of horizontal partitioning is simpler than that of vertical partitioning because it just needs to clear all the memories and registers, while vertical partitioning needs to distribute intrinsic information to specified places according to H .

In our design, we combine the normalized BP-based algorithm with group replica shuffled decoding based on horizontal partitioning. Two replica sub-decoders are used to describe the decoding process. It is straightforward to extend the algorithm to the cases in which more replica sub-decoders are used. Denote the number of groups by G and the inverse of the normalization factor by γ . Let $\mathcal{O}_{r,g}$ be the set of check nodes in the g -th group of the r -th replica sub-decoder, where $r = 1, 2$ and $g = 1, 2, \dots, G$. Assume T_n is the *a posteriori* LLR (or the belief) of the n -th codeword bit. Other notations are the same as those defined in Section 3.1.2 except that we omit the superscript that denotes the number of iterations. To facilitate hardware implementation, the algorithm is carried out as follows.

Initialization: Set $i = 1$, and the maximum number of iterations to I_{Max} . For each m and n , set $U_{mn} = 0$ and $T_n = U_{ch,n}$.

Step 1: For $1 \leq g \leq G$, process

(i) Vertical step: For each $m \in \mathcal{O}_{1,g}$ and $n \in \mathcal{N}(m)$

$$V_{mn} = T_n - U_{mn}. \quad (5.2.1)$$

For each $l \in \mathcal{O}_{2,g}$ and $q \in \mathcal{N}(l)$

$$V_{lq} = T_q - U_{lq}. \quad (5.2.2)$$

(ii) Horizontal step: For each $m \in \mathcal{O}_{1,g}$ and $n \in \mathcal{N}(m)$

$$U_{mn} = \gamma \cdot \prod_{n' \in \mathcal{N}(m) \setminus n} \text{sgn}(V_{mn'}) \cdot \min_{n' \in \mathcal{N}(m) \setminus n} |V_{mn'}|. \quad (5.2.3)$$

For each $l \in \mathcal{O}_{2,g}$ and $q \in \mathcal{N}(l)$

$$U_{lq} = \gamma \cdot \prod_{q' \in \mathcal{N}(l) \setminus q} \text{sgn}(V_{lq'}) \cdot \min_{q' \in \mathcal{N}(l) \setminus q} |V_{lq'}|. \quad (5.2.4)$$

(ii) Update the *a posteriori* LLR: For each $m \in \mathcal{O}_{1,g}$ and $n \in \mathcal{N}(m)$

$$T_n = V_{mn} + U_{mn}. \quad (5.2.5)$$

For each $l \in \mathcal{O}_{2,g}$ and $q \in \mathcal{N}(l)$

$$T_q = V_{lq} + U_{lq}. \quad (5.2.6)$$

Step 2: Hard decision and stopping criterion test:

- (i) Create $\hat{c} = [\hat{c}_n]$ such that $\hat{c}_n = 1$ if $T_n < 0$, and $\hat{c}_n = 0$ otherwise.
- (ii) If I_{Max} is reached, stop the decoding iteration and go to Step 3. Otherwise set $i := i + 1$ and go to Step 1.

Step 3: Output \hat{c} as the decoded codeword.

In the vertical step, we have used the total sum first implementation [32][34][63][97] instead of calculating $\sum_{m' \in \mathcal{M}(n) \setminus m} U_{m'n}$ and $\sum_{l' \in \mathcal{M}(q) \setminus l} U_{l'q}$.

5.3 Quantization and Parameter Selection

We use 6-bit quantization for bit-to-check or check-to-bit messages and 9-bit quantization for beliefs with one bit representing the sign and the remaining bits representing the magnitude. Uniform quantization is adopted in our design. Figure 5.2 depicts the performance comparison of horizontal replica shuffled normalized BP-based decoders without quantization and with 6-bit or 5-bit quantization for messages. The number of sub-decoders is 2. The coefficient γ in (5.2.3) and (5.2.4) is set to 0.75. We observe that the performance of 6-bit quantization is almost the same as that of no quantization, while when 5-bit quantization is used, the performance gap between the quantization case and the non-quantization case becomes larger.

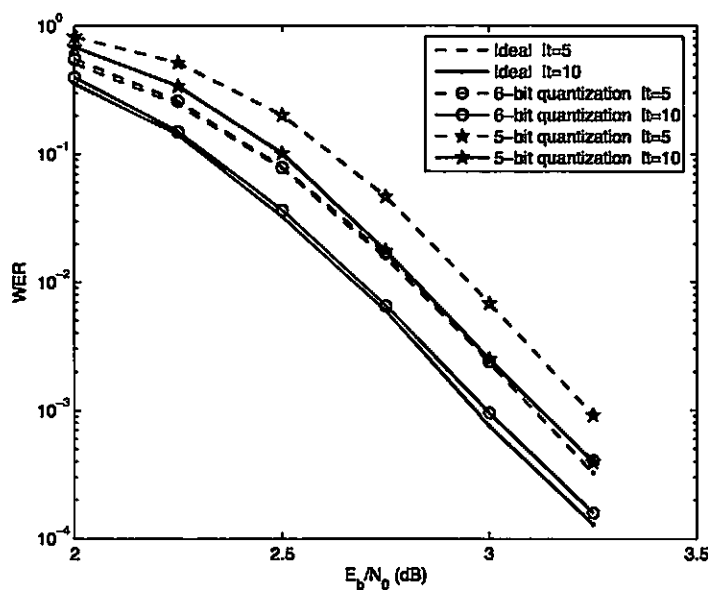


Figure 5.2: Performance comparison of horizontal replica shuffled normalized BP-based decoders without quantization and with 6-bit or 5-bit quantization for messages.

The value of γ is selected with both decoding performance and the facility of hardware implementation taken into account. When $\gamma = 0.75$, the multiplication of a value x and γ can be calculated by summing up the right-shifted x by one bit and the right-shifted x by two bits. By doing so, we can not only reduce latency but also save logic gates.

5.4 VLSI Architecture of Replica Shuffled Normalized BP-Based Decoder

Since group plain shuffled decoding is a special case of group replica shuffled decoding, we first describe the architecture of horizontal group plain shuffled normalized BP-based decoder. We form p groups of M/p checks, where M is the number of check nodes. We devote one super processor to each of the checks in one group. Therefore there are M/p super processors, which work in parallel, stepping through the p groups. The belief of each bit is stored in a single register, which is denoted as a belief register.

Every p belief registers are grouped to form a bank of belief registers and each bank is associated with one column in the base matrix. Then there are 24 banks of belief registers. We select the length-1056 rate-2/3 LDPC code from the IEEE 802.16e standard with base matrix shown in Figure 5.1 as our design example. Since $p = 44$ and $M/p = 8$, we have 44 groups and 8 super processors. Super processors and banks of belief registers are connected according to the base matrix. If $B_{ij} \neq -1$, the i -th super processor is connected to the j -th bank of belief registers. Figure 5.3 depicts the VLSI architecture of a horizontal group plain shuffled normalized BP-based decoder for this code.

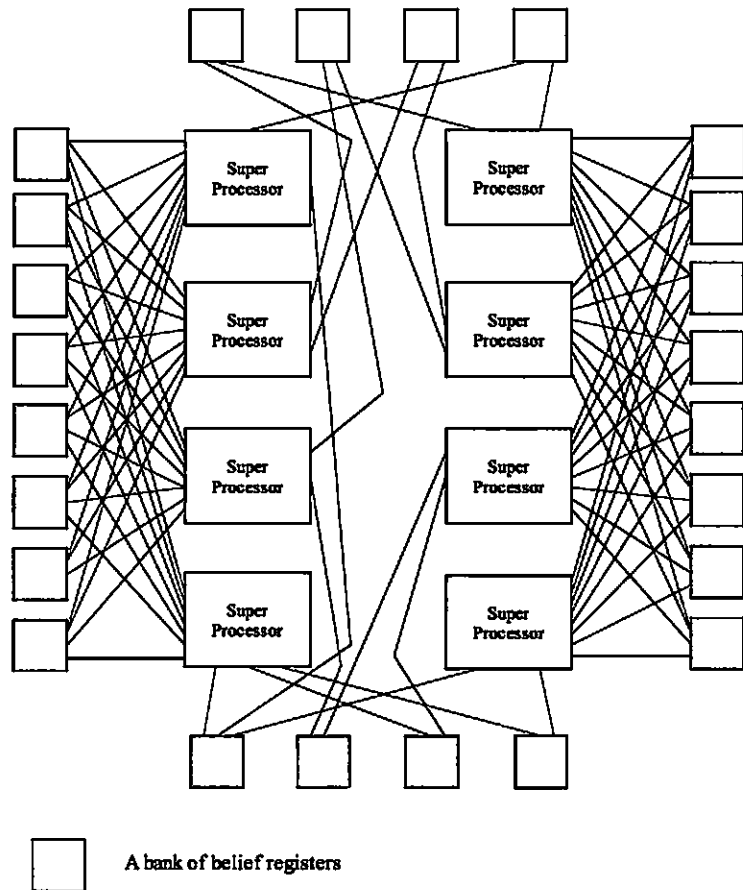


Figure 5.3: VLSI architecture of a horizontal group plain shuffled normalized BP-based decoder.

5.4.1 Structure of Super Processor

Figure 5.4 depicts the structure of a super processor. Each super processor contains a check processor, a set of link processors and a set of banks of message registers. Message registers are used to store the messages from checks to bits and those associated with the same circulant permutation matrix in H are grouped to form one bank. There is a one-to-one correspondence between link processors and banks of message registers. Each link processor operates in two directions. In one direction, it receives the belief and the corresponding check-to-bit message, then generates the bit-to-check message and sends it to the check processor. In the other direction, it receives the new check-to-bit message from the check processor and sends it to the corresponding bank of message registers. At the same time, it generates the new belief and sends it to the corresponding bank of belief registers. The function of the check processor is to process the bit-to-check messages and sends the new check-to-bit messages to the corresponding link processors.

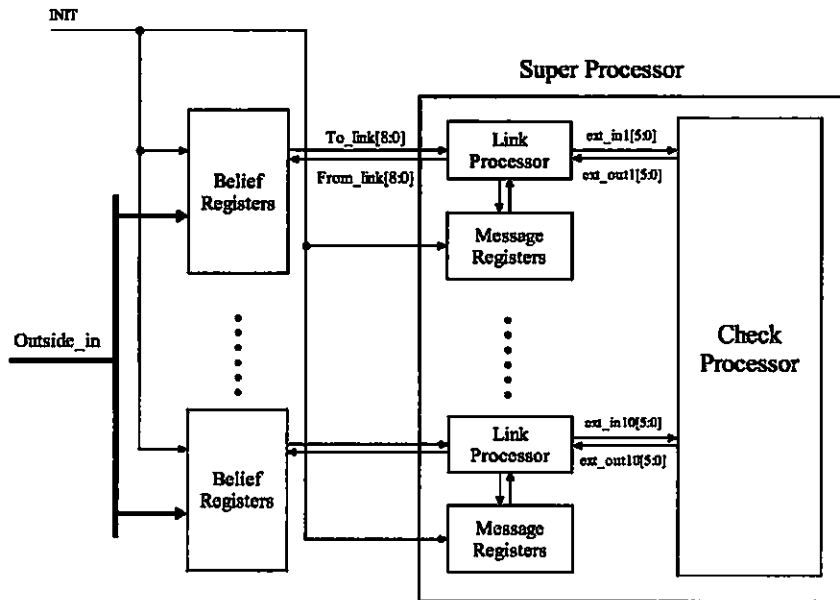


Figure 5.4: Structure of a super processor.

5.4.2 Structure of Check Processor

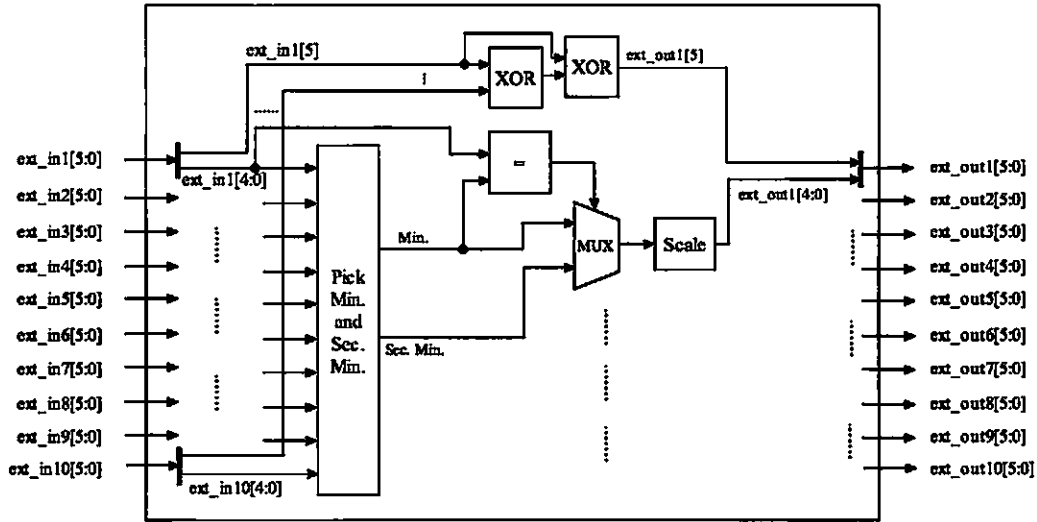


Figure 5.5: Structure of a check processor with 10 inputs/outputs.

A check processor carries out the calculation in (5.2.3). Figure 5.5 depicts the structure of a check processor with 10 inputs/outputs. The number of inputs or outputs is determined by the degree of checks it processes. A check processor receives bit-to-check messages with each of them having a sign and a magnitude. After these messages enter the check processor, their signs and magnitudes are separated. The magnitudes go through a block that calculates the first minimum value and the second minimum value. Then the magnitude of each input message is compared with the first minimum value. If it is equal to the first minimum value, the second minimum value is selected; otherwise, the first minimum value is selected. The selection is realized using a multiplexer, which is denoted as MUX in Figure 5.5. Then we scale the selected value according to γ and the scaled value is the magnitude of the corresponding output message. For the sign, we use bit 0 to represent the sign of a positive LLR and bit 1 to represent the sign of a negative LLR. Then the product of the signs in (5.2.3) corresponds to the XOR of the bit values. Since the sign of the output is the product of the signs of all the inputs excluding that of its corresponding input, we use two XOR blocks to fulfill this function as shown in Figure 5.5. Then, the

magnitude of each output is combined with its corresponding sign, which generates the complete output message.

Comparators are built to calculate the first minimum value and the second minimum value. Figure 5.6 and Figure 5.7 depict the structures of a 10-input 2-output com-

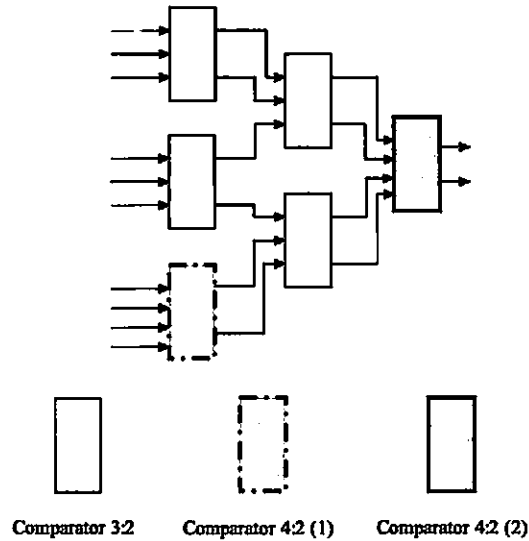


Figure 5.6: Structure of a 10-input 2-output comparator.

parator and a 11-input 2-output comparator, respectively. They are constructed as a cascade of three kinds of comparators, Comparator 3:2, Comparator 4:2 (1) and Comparator 4:2 (2), whose structures are shown in Figure 5.8, Figure 5.9 and Figure 5.10, respectively. For a 10-input comparison, the input messages are divided into three groups, with 3, 3, and 4 messages, respectively. For a 11-input comparison, the input messages are also divided into three groups, with 3, 4, and 4 messages, respectively. Comparator 3:2 receives three inputs and compares each pair among them. Thus, there are three parallel comparisons and according to the comparison results, it outputs the minimum value and the second minimum value. Comparator 4:2 (1) receives four inputs and compares each pair. Hence there are six parallel comparisons and according to the comparison results, it outputs the minimum value and the second minimum value. Since we know the ordering of the outputs of Comparator 3:2 in the second stage in Figure 5.6 and Figure 5.7, we do not need to com-

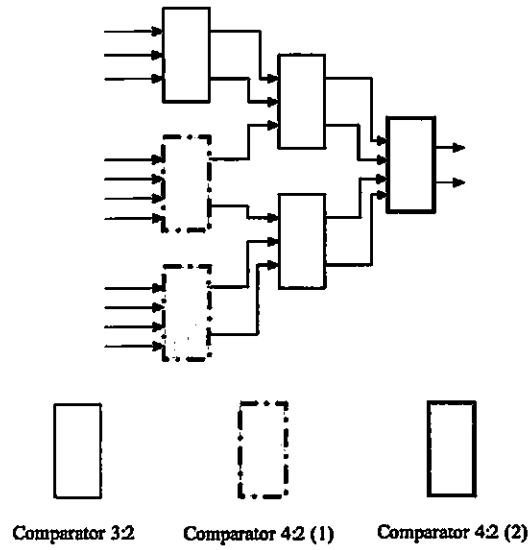


Figure 5.7: Structure of a 11-input 2-output comparator.

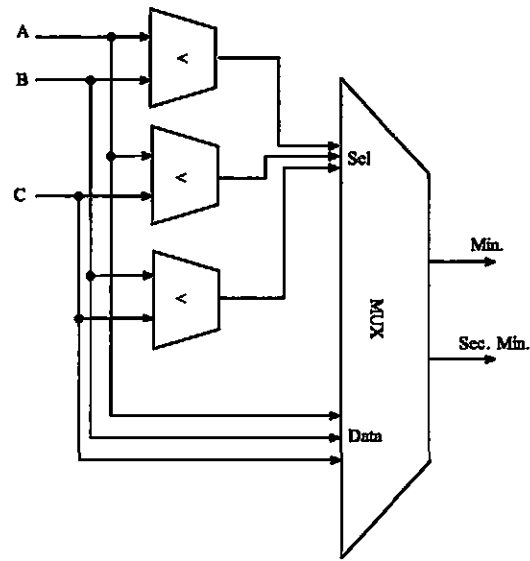


Figure 5.8: Structure of comparator 3:2.

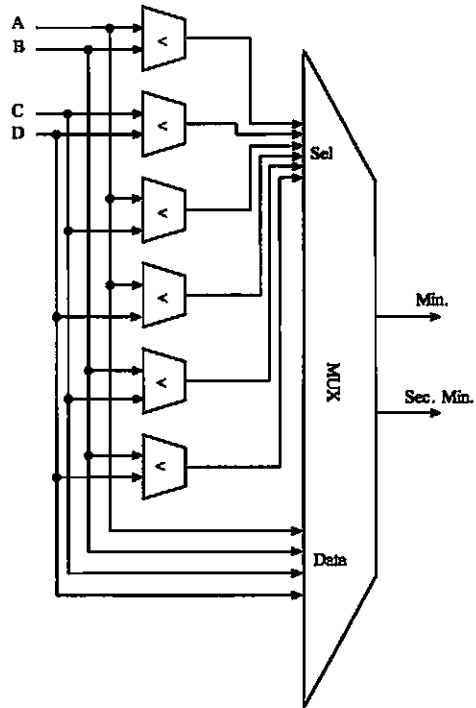


Figure 5.9: Structure of comparator 4:2 (1).

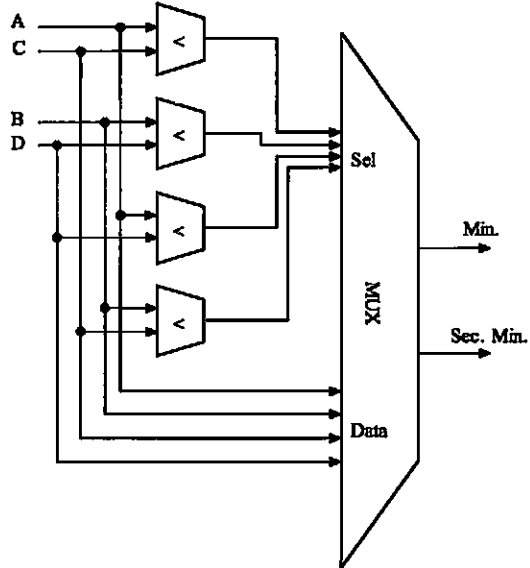


Figure 5.10: Structure of comparator 4:2 (2).

pare them again in the third stage. Therefore Comparator 4:2 (2) saves two comparisons compared with Comparator 4:2 (1).

5.4.3 Structure of Link Processor

Link processors carry out the calculation in (5.2.1) and (5.2.5). Figure 5.11 de-

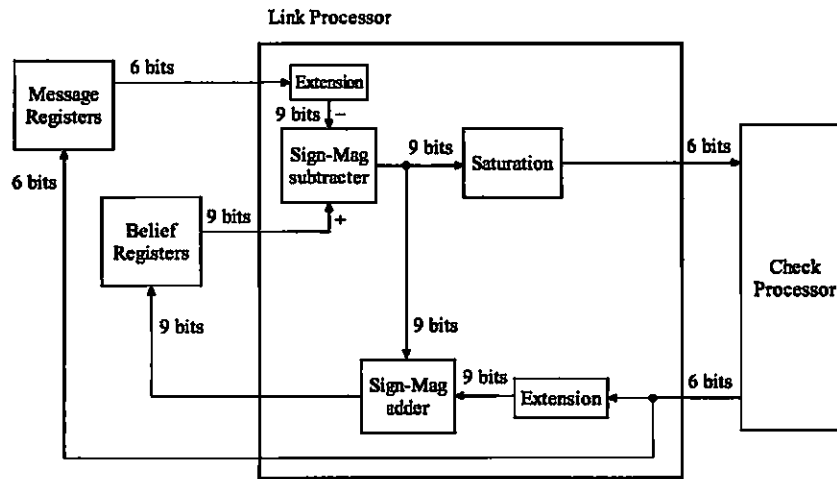


Figure 5.11: Structure of a link processor.

It takes inputs from a belief register and a message register. After subtracting the message from the belief, and limiting the maximum value of the magnitude of the remainder to a 5-bit value using the saturation block, we send the resulting value to the corresponding check processor. To recover the beliefs from the check-to-bit messages sent by the check processor, we perform an addition operation.

5.4.4 Structure of Message Registers

Figure 5.12 depicts the structure of a bank of message registers. Since the code is quasi-cyclic, we design each bank of message registers as shift registers so that the messages automatically cycle from one stage to another, until they are sent to the appropriate link processor. A bank of message registers contains p stages. Each stage either passes its message to the next stage or outputs its message to a connected link processor. The input is

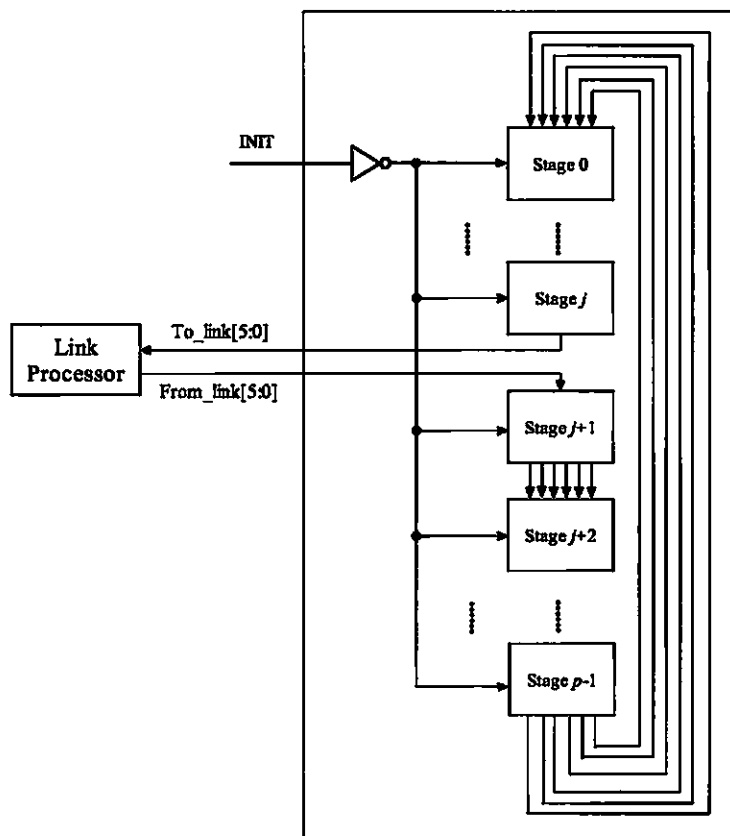


Figure 5.12: Structure of a bank of message registers.

either the message coming from the previous stage or the updated message from the connected link processor. The signal INIT is a synchronous reset that forces all the stages to output all zeroes at a rising edge when the signal is '1'. The INIT signal is set to '1' at the beginning of decoding each block, and set to '0' after one clock cycle.

5.4.5 Structure of Belief Registers

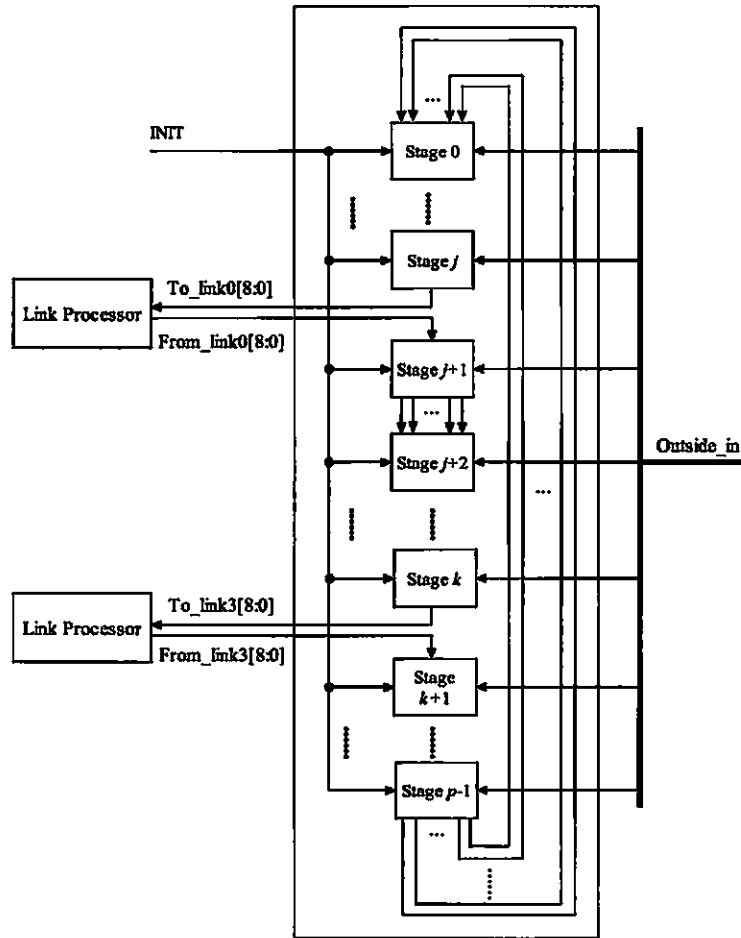


Figure 5.13: Structure of a bank of belief registers.

Figure 5.13 depicts the structure of a bank of belief registers. Similar to message registers, we design each bank of belief registers as shift registers so that the beliefs

automatically cycle from one stage to another, until they are sent to the appropriate super processor.

A bank of belief registers contains p stages. The beliefs are shifted in a circular manner so that each stage either passes its belief to the next stage or outputs its belief to the connected link processor. The input for a stage is either the belief coming from the previous stage or the updated belief from the connected link processor. The INIT signal is the same as that in Figure 5.12 and it forces all the stages to load the channel information of a new block to be decoded.

It should be noted that only selected stages are connected to the link processors. To avoid access conflict, the placement of the connections should ensure that two super processors do not simultaneously access the same belief register. There are many possible placements. For example, if a certain super processor is connected to a given bank of belief registers, and the base matrix has entry t for that connection, then we can connect the t -th stage to the super processor. However, we can also choose, for a particular super processor, to always connect to stage $t + k$ instead of stage t . As long as one does that consistently for every connection coming out of a super processor, the decoder still operates correctly.

5.4.6 VLSI Architecture of Replica Shuffled Normalized BP-Based Decoder with Two Sub-Decoders

It is straightforward to modify the above architecture for a replica shuffled normalized BP-based decoder with multiple sub-decoders. Figure 5.14 depicts the VLSI architecture of a horizontal group replica shuffled normalized BP-based decoder with two sub-decoders. Figure 5.14 has almost the same architecture as that in Figure 5.3 except that we replace super processors with replica super processors and replace every single connection with a double connection. Figure 5.15 depicts the structure of a replica super processor with two sub-decoders. We double the number of check processors and link processors. The number of belief register banks and message register banks remains unchanged, while the number of connections of each bank to link processors is doubled. Hence we need to choose the placement of these connections carefully to avoid access conflict as discussed in Section 5.4.5.

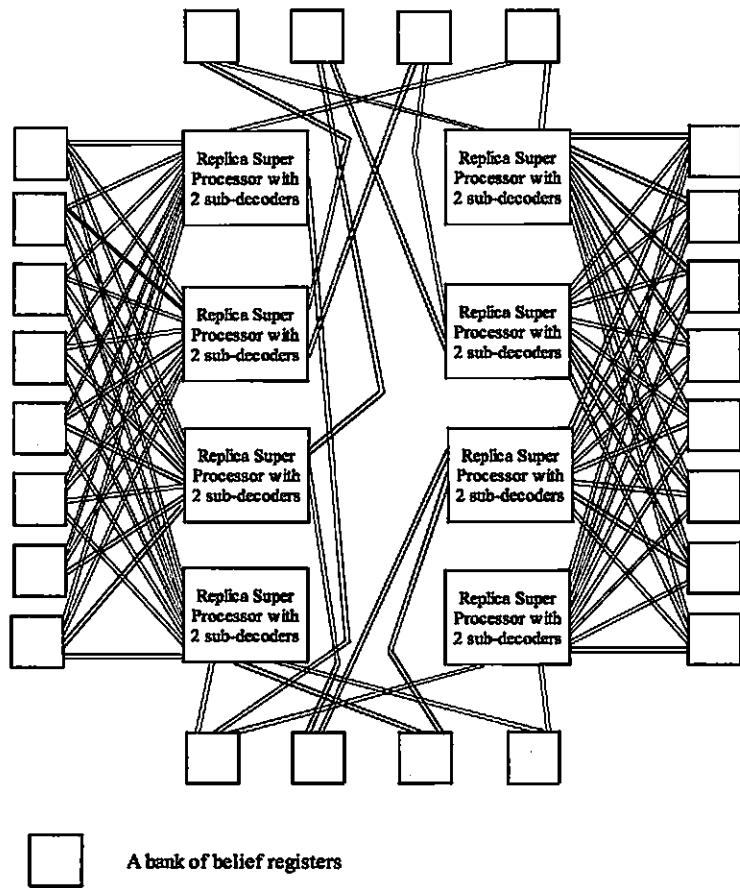


Figure 5.14: VLSI architecture of a horizontal group replica shuffled normalized BP-based decoder with two sub-decoders.

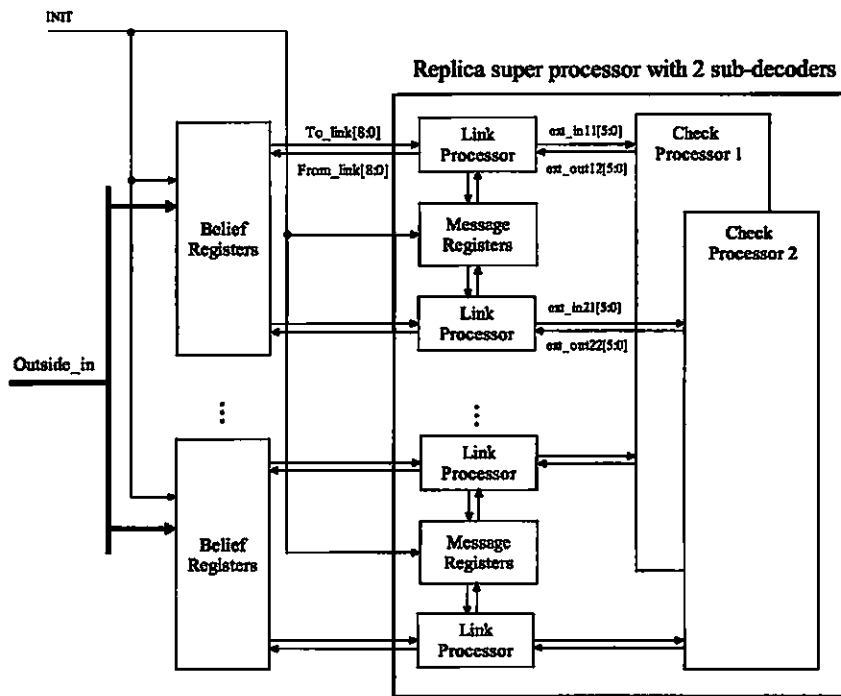


Figure 5.15: Structure of a replica super processor with two sub-decoders.

5.5 FPGA Testing Results

We implemented the decoder architecture in Figure 5.3 using Altera Stratix II Device EP2S180F1020C5. The hardware resource utilization statistics are listed in Table 5.1. The maximum clock frequency f can be 40 MHz based on the results reported by the Al-

Resource	Utilization
Total ALUTs	36,808 / 143,520 (26 %)
Total registers	30889
Total memory bits	0

Table 5.1: FPGA resource utilization statistics.

tera timing analysis tool. Since each iteration needs p clock cycles to finish, the throughput can be calculated by

$$\text{Throughput} = N \cdot R \cdot f / (p \cdot \text{Iter}) \quad (5.5.1)$$

where R is the rate of the code and Iter is the number of iterations used. If we choose $\text{Iter} = 10$, the throughput of the decoder with the architecture in Figure 5.3 is 64 Mbps. When the decoder with the architecture in Figure 5.14 is used, the throughput increases since the number of iterations is reduced. At the same time, the total ALUTs also increase because we double the number of link and check processors. However, the number of registers remains unchanged. It follows that given certain requirements, we can find a good compromise between throughput and hardware complexity based on our architecture.

Although we only developed the hardware architecture for length-1056 rate-2/3 LDPC codes, it is straightforward to generalize the design to other codes in the 802.16e standard.

Chapter 6

Doubly Generalized LDPC Codes

As stated in Chapter 2, based on its Tanner graph, a standard LDPC code can be characterized by random connections between variable nodes and check nodes, which can be viewed as repetition codes and SPC codes, respectively. Tanner proposed the use of short component codes other than SPC codes as check nodes and these new codes are referred to as generalized LDPC (GLDPC) codes [35]. Within this framework, many codes have been considered as check node component codes, such as Hamming codes [36][37], BCH codes [37][38], RS codes [38] and Hadamard codes [39]. Some hybrid constructions have also been studied. In [40], a hybrid structure mixing SPC codes and Hamming codes at the check nodes has been presented for the Gilbert-Elliott channel. In [41], another hybrid construction referred to as doped LDPC code has been studied for the AWGN channel. All these constructions are based on strengthening the check node component codes at the expense of a rate loss. In [77], GLDPC codes referred to as molecular codes have been constructed by allowing both variable and check nodes to represent codes more general than repetition and SPC codes, and regular molecular codes are mainly discussed. In this chapter, GLDPC codes more closely related to conventional LDPC codes and with more diversities in variable and check nodes than the codes of [77] are studied. We refer to them as doubly GLDPC (DGLDPC) codes.

For LDPC codes, both density evolution [30] and EXIT charts [68]-[74] are effective ways to analyze the performance of a code. For the binary erasure channel (BEC), these two methods are equivalent. For the AWGN channel, density evolution is more accurate, while EXIT charts are easier to visualize and to evaluate. For (D)GLDPC codes, density

evolution becomes often intractable, and only EXIT charts are left to analyze their performance and estimate their threshold. In [78]-[80], a concept termed information combining has been introduced. In particular a lower bound and an upper bound on the combined information are presented. Using the concept of information combining, we derive closed forms for some variable component codes of DGLDPC codes. For more general codes, we apply the method of [82], which has been derived for extrinsic mutual information at the output of a pseudo-MAP decoder over any binary input memoryless symmetric channel. For the AWGN channel, the general expression of [82] is accurate for SPC codes and it is also a good approximation for Hamming codes and simplex codes. While the results of [82] are directly applicable to check component codes, we generalize them to variable component codes in the process.

In conjunction with EXIT charts, differential evolution (DE) is used in threshold optimization. DE is a parallel direct search technique featured by evolution of variable vectors [88]. This technique has been successfully applied to design irregular LDPC codes with excellent thresholds for the BEC [89], the AWGN channel [90], and the Rayleigh fading channel [91]. In [92], it has been used for threshold optimization of DGLDPC codes over the BEC. We use this technique for threshold optimization of DGLDPC codes over the AWGN channel. From DE optimization of EXIT charts and simulation results, we observe that carefully chosen variable and check component codes can significantly improve the threshold of DGLDPC codes compared with that of LDPC and GLDPC codes with the same maximum variable degree.

6.1 A Brief Review of GLDPC Codes

6.1.1 Structure of GLDPC Codes

In GLDPC codes, all the variable nodes connected to the same check node must form a valid codeword in the corresponding check node component code, rather than just satisfying a single parity check. If the degrees of all the variable nodes and all the check nodes are constant, say d_v and d_c , respectively, and all the check nodes represent the same

component code, then the GLDPC code is referred to as a (d_v, d_c) regular GLDPC code; otherwise, it is an irregular (or hybrid) GLDPC code.

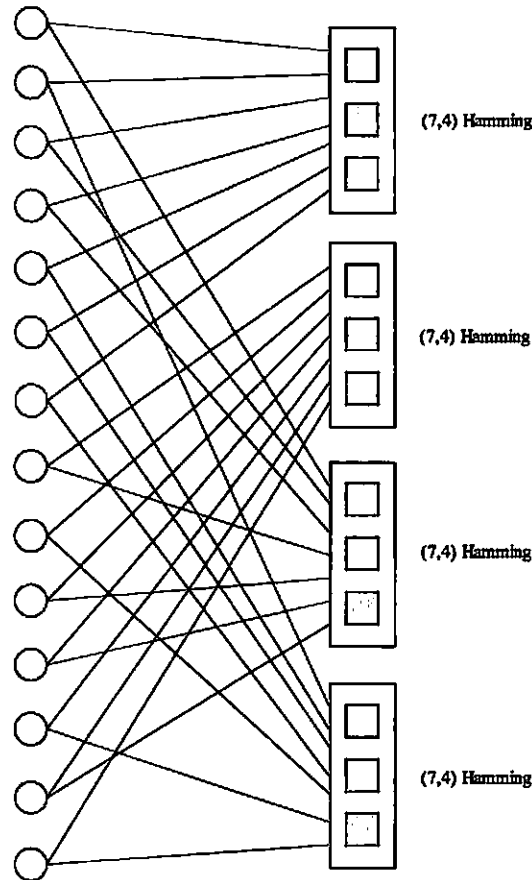


Figure 6.1: Graph representation of a $(2,7)$ regular GLDPC code.

Figure 6.1 depicts the graph representation of a $(2,7)$ regular GLDPC code. Each check node represents a $(7,4)$ Hamming code. The corresponding graph adjacency matrix, $\mathbf{H} = [H_{mn}]$, referred to as the base matrix, is shown in Figure 6.2 (a). Figure 6.2 illustrates the process to obtain the parity check matrix of a $(2,7)$ regular GLDPC code from its base matrix. In every row of the base matrix \mathbf{H} , each “1” is replaced with a column vector from the parity check matrix of the $(7,4)$ Hamming code shown in Figure 6.2 (b) based on a one-to-one correspondence, and each “0” is replaced with a zero column vector. The resulting parity check matrix of the GLDPC code is shown in Figure 6.2 (c). The assignment of

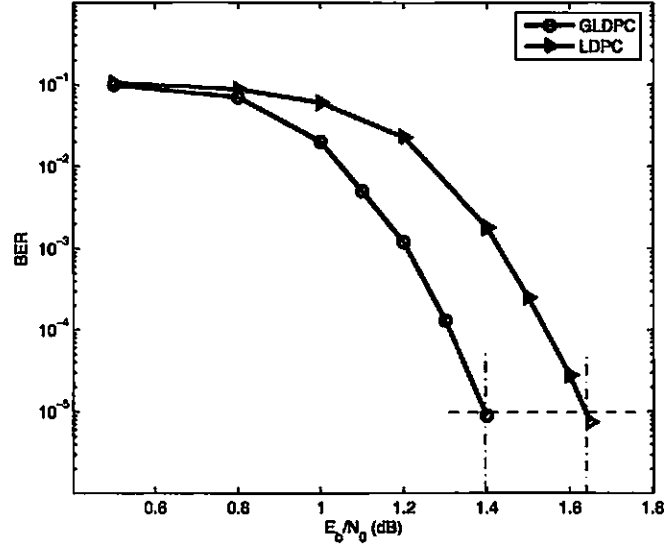


Figure 6.3: BER comparison of a (3,6) regular LDPC code of length 8000 and rate 1/2 and a (2,15) regular GLDPC code of length 7680 and rate 7/15.

codes with lengths equal to the degrees of their corresponding super nodes, as shown in Figure 6.4.

Let the number of super variable and super check nodes be N and M , respectively, let the corresponding node degrees be $d_{v1}, d_{v2}, \dots, d_{vN}$ and $d_{c1}, d_{c2}, \dots, d_{cM}$, respectively, and let the corresponding component codes be $(d_{v1}, k_{v1}), (d_{v2}, k_{v2}), \dots, (d_{vN}, k_{vN})$ and $(d_{c1}, k_{c1}), (d_{c2}, k_{c2}), \dots, (d_{cM}, k_{cM})$, respectively. The information bits of each super variable node compose the codeword bits of the DGLDPC code and are transmitted through the channel. Similarly to GLDPC codes, the graph adjacency matrix, $\mathbf{H} = [H_{mn}]$, is referred to as the base matrix. We denote the set of super variable nodes that participate in super check node m by $\mathcal{N}(m) = \{n : H_{mn} = 1\}$ and the set of super check nodes in which super variable node n participates as $\mathcal{M}(n) = \{m : H_{mn} = 1\}$. The parity check matrix of a DGLDPC code can be obtained from its base matrix with the following two steps.

Step 1 : row expansion

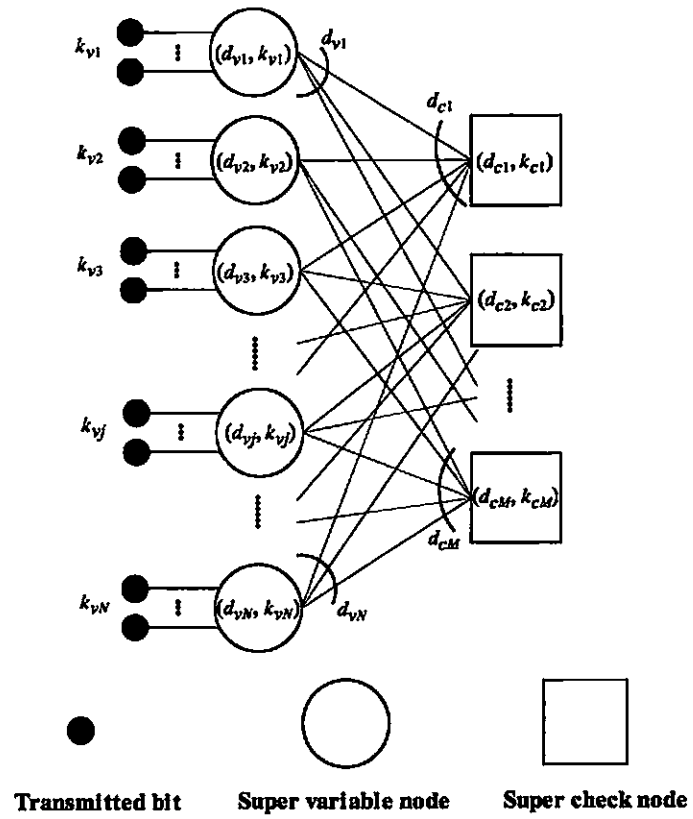


Figure 6.4: Graph representation of a DGLDPC code.

As an example, Figure 6.5 (e) depicts the parity check matrix of a (21, 3) DGLDPC code. It is expanded from the base matrix given in Figure 6.5 (a). All the super check nodes are (7, 4) Hamming codes whose parity check matrix is given in Figure 6.5 (c). The first half of the super variable nodes are (3, 2) SPC codes and the other half are (3, 1) repetition codes. Their generator matrices, G_1 and G_2 , are shown in Figure 6.5 (b). To illustrate column expansion, take the first and 8-th columns in \tilde{H} in Figure 6.5 (d) for example. Since the first super variable node is a (3, 2) SPC code, the 3 non-zero column vectors, $(1, 1, 1)^T$, $(0, 1, 1)^T$, and $(1, 0, 0)^T$ in the first column of \tilde{H} are replaced with $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}^T$, $\begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}^T$, and $\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}^T$, respectively, where $(A)^T$ is the transpose of matrix A . All the zero vectors are replaced with $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}^T$. For the 8-th column in \tilde{H} , since its corresponding super variable node is a (3, 1) repetition code, it remains the same in H_{DGLDPC} .

Denote the super check nodes that participate in super variable node n by $m_{n,1}, \dots, m_{n,d_{vn}}$. Similarly, denote the super variable nodes that participate in super check node m by $n_{m,1}, \dots, n_{m,d_{cm}}$. After row expansion there are d_{vi} non-zero column vectors in column i with dimensions $(d_{cm_{i,1}} - k_{cm_{i,1}}), \dots, (d_{cm_{i,d_{vi}}} - k_{cm_{i,d_{vi}}})$, respectively. Similarly to GLDPC codes, the assignment of columns from the component code parity check matrix should be random in order to produce good codes [37]. After column expansion, a DGLDPC parity check matrix is obtained with $M_G = \sum_{i=1}^M (d_{ci} - k_{ci})$ rows and $N_G = \sum_{i=1}^N k_{vi}$ columns. As a result, the rate of this DGLDPC code is lower bounded by $1 - M_G/N_G$.

6.3 Iterative Decoding of DGLDPC Codes

Iterative decoding based on BP [20] is used to decode a DGLDPC code. Let $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_N)$ be a codeword of a DGLDPC code, where $\mathbf{b}_n = (b_{n,1}, b_{n,2}, \dots, b_{n,k_{vn}})$ for $n = 1, 2, \dots, N$. Assume BPSK signaling with unit energy, which maps a codeword \mathbf{b} into a transmitted sequence \mathbf{c} , where $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_N)$ and $\mathbf{c}_n = (c_{n,1}, c_{n,2}, \dots, c_{n,k_{vn}})$, according to $c_{n,i} = 1 - 2b_{n,i}$, for $n = 1, 2, \dots, N$ and $i = 1, 2, \dots, k_{vn}$. Let $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N)$ be the received sequence, where $\mathbf{y}_n = (y_{n,1}, y_{n,2}, \dots, y_{n,k_{vn}})$ and $y_{n,i} =$

$c_{n,i} + g_{n,i}, g_{n,i}$'s being statistically independent Gaussian random variables with zero mean and variance $N_0/2$.

Assume super variable node n is connected by the $t_{n,i}$ -th edge of super check node $m_{n,i}$ and similarly super check node m is connected by the $s_{m,i}$ -th edge of super variable node $n_{m,i}$. Since each super variable node is associated with a component code, the bits on the outgoing edges of a super variable node should form a codeword of its corresponding component code. We denote the codeword associated with super variable node n and information sequence \mathbf{b}_n by $\mathbf{x}_n = (x_{n,1}, x_{n,2}, \dots, x_{n,d_{vn}})$, as shown in Figure 6.6. We also denote the codeword associated with super check node m by $\mathbf{z}_m = (z_{m,1}, z_{m,2}, \dots, z_{m,d_{cm}})$, where $z_{m,j} = x_{n_{m,j}, s_{m,j}}$ for $j = 1, \dots, d_{cm}$.

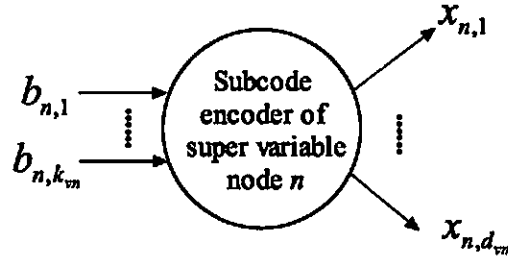


Figure 6.6: Component code encoder of a super variable node.

Let $\mathbf{u}_n^{(i)} = (u_{n,1}^{(i)}, u_{n,2}^{(i)}, \dots, u_{n,d_{vn}}^{(i)})$ and $\mathbf{v}_m^{(i)} = (v_{m,1}^{(i)}, v_{m,2}^{(i)}, \dots, v_{m,d_{cm}}^{(i)})$ be the *a priori* information incoming to super variable node n and super check node m at the i -th iteration, respectively. The corresponding LLR values are $(U_{n,1}^{(i)}, U_{n,2}^{(i)}, \dots, U_{n,d_{vn}}^{(i)})$ and $(V_{m,1}^{(i)}, V_{m,2}^{(i)}, \dots, V_{m,d_{cm}}^{(i)})$, with

$$U_{n,p}^{(i)} = \log \frac{P(u_{n,p}^{(i)} | x_{n,p} = 0)}{P(u_{n,p}^{(i)} | x_{n,p} = 1)} \quad (6.3.1)$$

$$V_{m,q}^{(i)} = \log \frac{P(v_{m,q}^{(i)} | z_{m,q} = 0)}{P(v_{m,q}^{(i)} | z_{m,q} = 1)} \quad (6.3.2)$$

for $p = 1, \dots, d_{vn}$ and $q = 1, \dots, d_{cm}$. Denote the *a posteriori* LLR of the information bits of super variable node n at the i -th iteration by $\mathbf{W}_n^{(i)} = (W_{n,1}^{(i)}, W_{n,2}^{(i)}, \dots, W_{n,k_{vn}}^{(i)})$. Let

$\mathbf{u}_{n[j]}^{(i)}$ and $\mathbf{v}_{m[j]}^{(i)}$ be the vectors $\mathbf{u}_n^{(i)}$ and $\mathbf{v}_m^{(i)}$ with the j -th entry removed, respectively, i.e., $\mathbf{u}_{n[j]}^{(i)} = (u_{n,1}^{(i)}, \dots, u_{n,j-1}^{(i)}, u_{n,j+1}^{(i)}, \dots, u_{n,d_{vn}}^{(i)})$ and $\mathbf{v}_{m[j]}^{(i)} = (v_{m,1}^{(i)}, \dots, v_{m,j-1}^{(i)}, v_{m,j+1}^{(i)}, \dots, v_{m,d_{cm}}^{(i)})$.

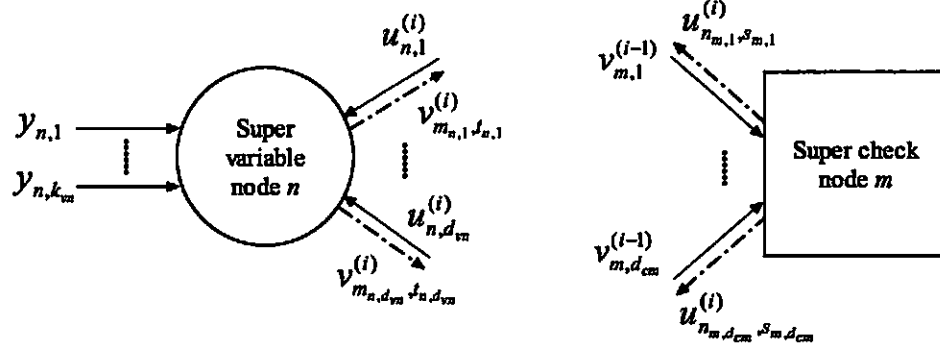


Figure 6.7: Message passing at a super variable node and at a super check node.

The super variable node n takes \mathbf{y}_n and $\mathbf{u}_n^{(i)}$ as input and outputs $v_{m_{n,1}, t_{n,1}}^{(i)}, \dots, v_{m_{n,d_{vn}}, t_{n,d_{vn}}}^{(i)}$ and the super check node m takes $\mathbf{v}_m^{(i-1)}$ as input and outputs $u_{n_{m,1}, s_{m,1}}^{(i)}, \dots, u_{n_{m,d_{cm}}, s_{m,d_{cm}}}^{(i)}$ as shown in Figure 6.7. We perform maximum *a posteriori* probability (MAP) decoding [93][94] of the component codes, so that

$$U_{n_{m,q}, s_{m,q}}^{(i)} = \log \frac{P(z_{m,q} = 0 | \mathbf{v}_{m[q]}^{(i-1)})}{P(z_{m,q} = 1 | \mathbf{v}_{m[q]}^{(i-1)})}$$

$$= \log \frac{\sum_{\mathbf{z}_m: z_{m,q}=0} P(\mathbf{v}_{m[q]}^{(i-1)} | \mathbf{z}_m[q])}{\sum_{\mathbf{z}_m: z_{m,q}=1} P(\mathbf{v}_{m[q]}^{(i-1)} | \mathbf{z}_m[q])}$$

$$V_{m_{n,p}, t_{n,p}}^{(i)} = \log \frac{P(x_{n,p} = 0 | \mathbf{u}_{n[p]}^{(i)}, \mathbf{y}_n)}{P(x_{n,p} = 1 | \mathbf{u}_{n[p]}^{(i)}, \mathbf{y}_n)}$$

$$= \log \frac{\sum_{\mathbf{b}_n: x_{n,p}=0} P(\mathbf{u}_{n[p]}^{(i)} | \mathbf{x}_{n[p]}) P(\mathbf{y}_n | \mathbf{c}_n)}{\sum_{\mathbf{b}_n: x_{n,p}=1} P(\mathbf{u}_{n[p]}^{(i)} | \mathbf{x}_{n[p]}) P(\mathbf{y}_n | \mathbf{c}_n)}$$

Therefore the BP algorithm is carried out as follows.

Initialization: Set $i = 1$, and the maximum number of iterations to I_{Max} . For each n , set

$$V_{m_n, p, t_n, p}^{(0)} = \log \frac{\sum_{\mathbf{b}_n: x_{n,p}=0} \prod_{j=1}^{k_{vn}} e^{\frac{2v_{n,j}c_{n,j}}{N_0}}}{\sum_{\mathbf{b}_n: x_{n,p}=1} \prod_{j=1}^{k_{vn}} e^{\frac{2v_{n,j}c_{n,j}}{N_0}}}$$

Step 1: (i) Horizontal Step, for $1 \leq m \leq M$ and each $n \in \mathcal{N}(m)$, suppose $n = n_{m,q}$ and process:

$$U_{n_{m,q}, s_{m,q}}^{(i)} = \log \frac{\sum_{\mathbf{z}_m: z_{m,q}=0} \prod_{j=1, j \neq q}^{d_{cm}} e^{-V_{m,j}^{(i-1)} z_{m,j}}}{\sum_{\mathbf{z}_m: z_{m,q}=1} \prod_{j=1, j \neq q}^{d_{cm}} e^{-V_{m,j}^{(i-1)} z_{m,j}}} \quad (6.3.3)$$

(ii) Vertical Step, for $1 \leq n \leq N$ and each $m \in \mathcal{M}(n)$, suppose $m = m_{n,p}$ and process:

$$V_{m_{n,p}, t_{n,p}}^{(i)} = \log \frac{\sum_{\mathbf{b}_n: x_{n,p}=0} \prod_{j=1, j \neq p}^{d_{vn}} e^{-U_{n,j}^{(i)} x_{n,j}} \prod_{j=1}^{k_{vn}} e^{\frac{2v_{n,j}c_{n,j}}{N_0}}}{\sum_{\mathbf{b}_n: x_{n,p}=1} \prod_{j=1, j \neq p}^{d_{vn}} e^{-U_{n,j}^{(i)} x_{n,j}} \prod_{j=1}^{k_{vn}} e^{\frac{2v_{n,j}c_{n,j}}{N_0}}} \quad (6.3.4)$$

Step 2: Hard decision and stopping criterion test, for $n = 1, 2, \dots, N$ and $k = 1, 2, \dots, k_{vn}$,

$$W_{n,k}^{(i)} = \log \frac{\sum_{\mathbf{b}_n: b_{n,k}=0} \prod_{j=1}^{d_{vn}} e^{-U_{n,j}^{(i)} x_{n,j}} \prod_{j=1}^{k_{vn}} e^{\frac{2v_{n,j}c_{n,j}}{N_0}}}{\sum_{\mathbf{b}_n: b_{n,k}=1} \prod_{j=1}^{d_{vn}} e^{-U_{n,j}^{(i)} x_{n,j}} \prod_{j=1}^{k_{vn}} e^{\frac{2v_{n,j}c_{n,j}}{N_0}}} \quad (6.3.5)$$

(i) Create $\hat{\mathbf{b}}^{(i)} = [\hat{b}_n^{(i)}]$ and $\hat{\mathbf{b}}_n^{(i)} = [\hat{b}_{n,k}^{(i)}]$ such that $\hat{b}_{n,k}^{(i)} = 1$ if $W_{n,k}^{(i)} < 0$, and $\hat{b}_{n,k}^{(i)} = 0$ if $W_{n,k}^{(i)} \geq 0$.

- (ii) If $\hat{\mathbf{b}}^{(i)} \mathbf{H}_{\text{DGLDPC}}^T = \mathbf{0}$ or I_{Max} is reached, stop the decoding iteration and go to Step 3. Otherwise set $i := i + 1$ and go to Step 1.

Step 3: Output $\hat{\mathbf{b}}^{(i)}$ as the decoded codeword.

Other APP decoding methods, such as Max-Log-MAP, can also be used to decode the component codes. Also the schedulings discussed in Chapter 3 can be extended to the decoding of DGLDPC codes in a straightforward way.

6.4 Analysis by EXIT Charts

For a DGLDPC code, let D be the number of different component codes in super variable nodes and let λ_i be the fraction of edges incident to the i -th super variable node type, for $i = 1, 2, \dots, D$. Let E be the number of different component codes in super check nodes and let ρ_j be the fraction of edges incident to the j -th super check node type, for $j = 1, 2, \dots, E$. Denote by $I_{V_i}(\cdot)$ and $I_{U_j}(\cdot)$ the EXIT functions of the i -th super variable node type and the j -th super check node type, respectively. Denote by $I_V(\cdot)$ and $I_U(\cdot)$ the average EXIT functions of all super variable nodes and all super check nodes, respectively. Hence, transfer curves for super variable and super check nodes are respectively

$$I_V \left(I_U, \frac{E_b}{N_0}, R \right) = \sum_{i=1}^D \lambda_i \cdot I_{V_i} \left(I_U, \frac{E_b}{N_0}, R \right) \quad (6.4.1)$$

$$I_U(I_V) = \sum_{j=1}^E \rho_j \cdot I_{U_j}(I_V) \quad (6.4.2)$$

where $\sum_i \lambda_i = 1$ and $\sum_j \rho_j = 1$.

6.4.1 Closed Form EXIT Functions Obtained from Information Combining

In [93], closed-form EXIT functions for the BEC have been derived based on a general decoding model, which is depicted in Figure 6.8. The vector \mathbf{u} is the information

sequence with k independent bits. It is encoded into \mathbf{x} of length n and \mathbf{v} of length m through linear Encoder 1 and Encoder 2, respectively. The code formed by all pairs (\mathbf{v}, \mathbf{x}) is a linear $(m + n, k)$ code. Then \mathbf{x} and \mathbf{v} are transmitted through a communication channel and an extrinsic channel, respectively. The corresponding outputs \mathbf{y} and \mathbf{a} are sent to the decoder. In [93], both communication and extrinsic channels are BECs.

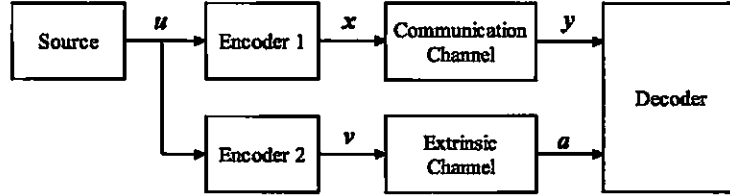


Figure 6.8: A decoding model with two encoders.

These results can be used as an estimate for the AWGN channel, but they are not very accurate. In the following, we present more accurate closed forms for some variable node component codes in DGLDPC codes.

In order to obtain the closed form of $I_{V_i} \left(I_U, \frac{E_b}{N_0}, R \right)$, first consider the following model for a length- L code. Assume its L code bits, X_1, \dots, X_L , are passed through L independent AWGN channels. The outputs of the L AWGN channels are Y_1, \dots, Y_L . The mutual information of each channel is defined as $I_i := I(X_i, Y_i)$, for $i = 1, 2, \dots, L$. Let $\mathbf{Y} = (Y_1, \dots, Y_L)$ and $\mathbf{Y}_{[i]}$ be \mathbf{Y} with the i -th entry removed. Then the extrinsic mutual information between code bit X_i and $\mathbf{Y}_{[i]}$ is $I_{ext,i} = I(X_i; \mathbf{Y}_{[i]})$. If the code is a repetition code,

$$I_{ext,i} = J \left(\sqrt{\sum_{j=1, j \neq i}^L [J^{-1}(I_j)]^2} \right). \quad (6.4.3)$$

If the code is a SPC code,

$$I_{ext,i} \approx 1 - J \left(\sqrt{\sum_{j=1, j \neq i}^L [J^{-1}(1 - I_j)]^2} \right). \quad (6.4.4)$$

Closed Form EXIT Functions for Systematic SPC Codes

For a systematic $(L, L - 1)$ SPC code, suppose the L code bits are X_1, \dots, X_L . Then the $L - 1$ information bits are X_1, \dots, X_{L-1} . Following the decoding model in [93], for a systematic SPC code used as a super variable node, the $L - 1$ information bits go through a communication channel while the L code bits go through an extrinsic channel whose mutual information is I_A . An extrinsic channel can also be called an *a priori* channel because its outputs are used as *a priori* information. Both channels are AWGN channels and their outputs are $Y = (Y_1, \dots, Y_{L-1})$ and $A = (A_1, \dots, A_L)$, respectively. The vectors Y and A are sent to an APP decoder which outputs L extrinsic information values. Denote the extrinsic mutual information for L code bits by $I_{ext,1}, \dots, I_{ext,L}$. Then according to [93], $I_{ext,i} = I(X_i; Y A_{[i]})$, which for $i = 1, \dots, L - 1$, should be the same because information bits are interchangeable. Therefore we can just focus on calculating $I_{ext,1}$ and $I_{ext,L}$. Based on (6.4.3),

$$I(X_i; Y_i A_i) = J \left(\sqrt{[J^{-1}(I_A)]^2 + 8R \cdot \frac{E_b}{N_0}} \right) \quad (6.4.5)$$

for $i = 1, \dots, L - 1$. We denote it by I_{temp1} . Since $X_L = X_1 + X_2 + \dots + X_{L-1}$, based on (6.4.4),

$$I_{ext,L} \approx 1 - J \left(\sqrt{(L-1) \cdot [J^{-1}(1 - I_{temp1})]^2} \right). \quad (6.4.6)$$

Since $X_1 = X_2 + X_3 + \dots + X_L$, based on (6.4.4),

$$\begin{aligned} & I(X_1; Y_2 A_2 Y_3 A_3 \dots Y_{L-1} A_{L-1} A_L) \\ & \approx 1 - J \left(\sqrt{(L-2) \cdot [J^{-1}(1 - I_{temp1})]^2 + [J^{-1}(1 - I_A)]^2} \right), \end{aligned}$$

which we denote by I_{temp2} . Based on (6.4.3), it follows

$$\begin{aligned} I_{ext,1} &= I(X_1; Y_1 Y_2 A_2 Y_3 A_3 \dots Y_{L-1} A_{L-1} A_L) \\ &\approx J \left(\sqrt{[J^{-1}(I_{temp2})]^2 + 8R \cdot \frac{E_b}{N_0}} \right). \end{aligned} \quad (6.4.7)$$

The average extrinsic mutual information becomes

$$I_{ext,av} \approx \frac{L-1}{L} I_{ext,1} + \frac{1}{L} I_{ext,L}. \quad (6.4.8)$$

Figure 6.9 depicts the comparison of EXIT curves in closed form and those obtained from Monte Carlo simulations for the (6,5) SPC code in systematic form. We observe that the EXIT curves of these two methods are almost the same, which validates the derived EXIT functions.

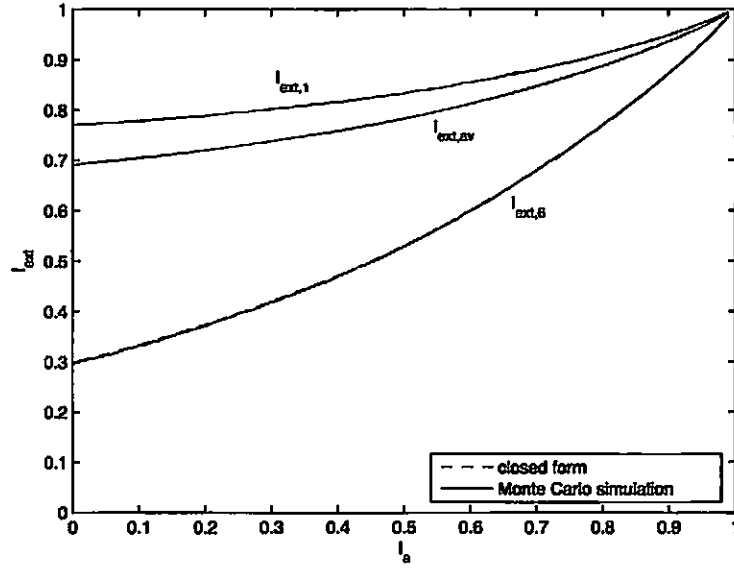


Figure 6.9: Comparison of EXIT curves in closed form and those obtained from Monte Carlo simulation for the (6,5) systematic SPC code at the SNR value 1.9 dB.

Closed Form EXIT Functions for Non-systematic SPC Codes

Similarly, we can also obtain the closed-form EXIT function for a non-systematic $(L, L-1)$ SPC code with generator matrix in cyclic form

$$\begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 0 & 1 & 1 \end{pmatrix}. \quad (6.4.9)$$

Let

$$I_{L-1}^{(1)} = J \left(\sqrt{[J^{-1}(I_A)]^2 + 8R \cdot \frac{E_b}{N_0}} \right),$$

and for $k = 1, 2, \dots, L-2$, let

$$I_k^{(2)} = 1 - J \left(\sqrt{[J^{-1}(1 - I_{k+1}^{(1)})]^2 + [J^{-1}(1 - I_A)]^2} \right)$$

$$I_k^{(1)} = J \left(\sqrt{[J^{-1}(I_k^{(2)})]^2 + 8R \cdot \frac{E_b}{N_0}} \right).$$

When L is even, let $I_1 = I_1^{(1)}$ and for $k = 2, 3, \dots, L/2$, let

$$I_k = 1 - J \left(\sqrt{[J^{-1}(1 - I_{L-k+1}^{(1)})]^2 + [J^{-1}(1 - I_k^{(1)})]^2} \right).$$

Then we obtain

$$I_{ext,av} \approx \frac{2}{L} \cdot \sum_{k=1}^{L/2} I_k. \quad (6.4.10)$$

When L is odd, let $I_1 = I_1^{(1)}$ and for $k = 2, 3, \dots, (L+1)/2$, let

$$I_k = 1 - J \left(\sqrt{[J^{-1}(1 - I_{L-k+1}^{(1)})]^2 + [J^{-1}(1 - I_k^{(1)})]^2} \right).$$

Then we obtain

$$I_{ext,av} \approx \frac{2}{L} \cdot \sum_{k=1}^{(L-1)/2} I_k + \frac{1}{L} \cdot I_{(L+1)/2}. \quad (6.4.11)$$

Figure 6.10 depicts the EXIT curves in closed form for non-systematic SPC codes in cyclic form with different lengths over an AWGN channel at the SNR value 0.5 dB. We observe with the increase of the length, the EXIT curves converge to a certain curve, which is actually the EXIT curve of an accumulator. Figure 6.11 depicts the EXIT curve comparison of a non-systematic SPC code in cyclic form with length 1000 and an accumulator over an AWGN channel at the SNR values of 0.5 dB, 2 dB, 3 dB and 4 dB. The EXIT curves of the accumulator are obtained from [72] through simulation of the full decoding procedure. We observe that the EXIT curves of these two codes are the same. Therefore,

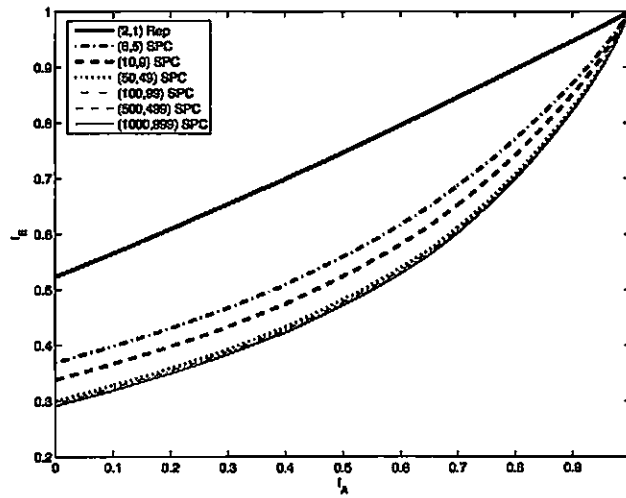


Figure 6.10: EXIT curves in closed form for non-systematic SPC codes in cyclic form with different lengths over an AWGN channel at the SNR value 0.5 dB. A code rate of $R=1/2$ was used to calculate the SNR.

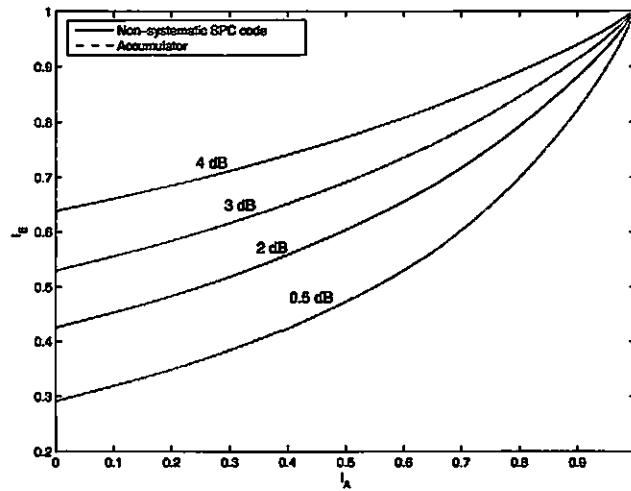


Figure 6.11: EXIT curve comparison of a non-systematic SPC code in cyclic form with length 1000 and an accumulator over an AWGN channel at several SNR values. A code rate of $R=1/2$ was used to calculate the SNR.

we can use the closed-form EXIT function of a long non-systematic SPC code to replace that of an accumulator. Then we can avoid the computer-intensive simulation and facilitate the threshold optimization of some codes, such as irregular repeat-accumulate (IRA) codes.

The above analysis for the AWGN channel can be extended to the BEC. Denote the erasure probability by q . Let

$$I_{L-1}^{(1)} = 1 - (1 - I_A)q,$$

and for $k = 1, 2, \dots, L - 2$, let

$$I_k^{(2)} = I_{k+1}^{(1)} I_A$$

$$I_k^{(1)} = 1 - (1 - I_k^{(2)})q.$$

When L is even, let $I_1 = I_1^{(1)}$ and for $k = 2, 3, \dots, L/2$, let

$$I_k = I_{L-k+1}^{(1)} I_k^{(1)}.$$

Then we obtain

$$I_{ext,av} = \frac{2}{L} \cdot \sum_{k=1}^{L/2} I_k. \quad (6.4.12)$$

When L is odd, let $I_1 = I_1^{(1)}$ and for $k = 2, 3, \dots, (L + 1)/2$, let

$$I_k = I_{L-k+1}^{(1)} I_k^{(1)}.$$

Then we obtain

$$I_{ext,av} = \frac{2}{L} \cdot \sum_{k=1}^{(L-1)/2} I_k + \frac{1}{L} \cdot I_{(L+1)/2}. \quad (6.4.13)$$

Figure 6.12 depicts the EXIT curves in closed form for non-systematic SPC codes in cyclic form with different lengths over a BEC with $q = 0.49$. Similarly to what we observed for the AWGN channel, with the increase of the length, the EXIT curves converge to a certain curve, which is actually the EXIT curve of an accumulator.

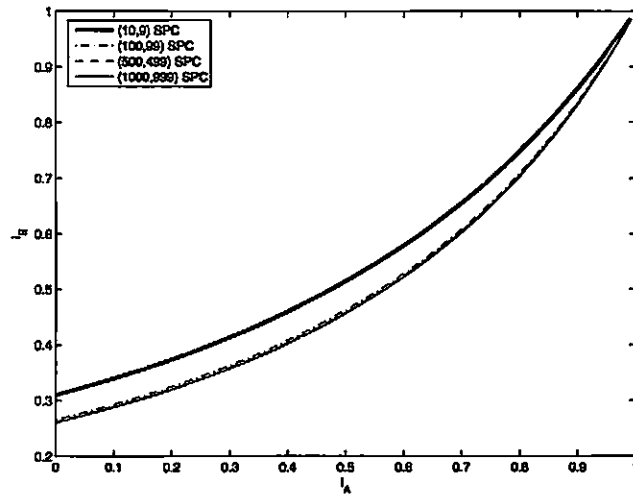


Figure 6.12: EXIT curves in closed form for non-systematic SPC codes in cyclic form with different lengths over a BEC with $q = 0.49$.

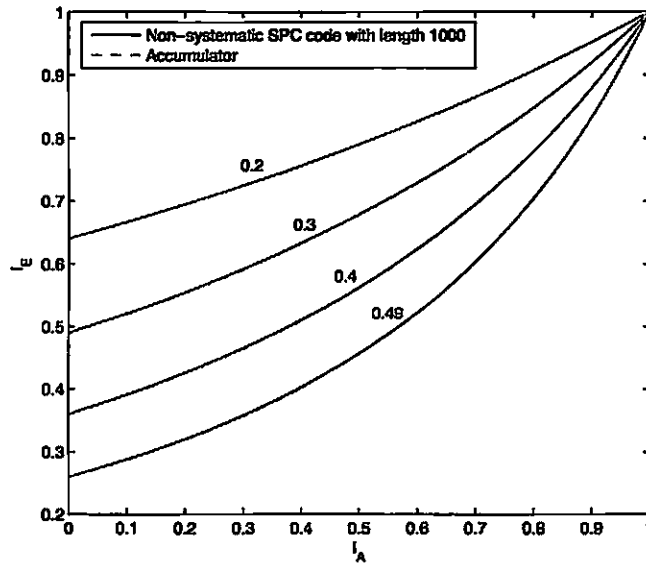


Figure 6.13: EXIT curve comparison of a non-systematic SPC code in cyclic form with length 1000 and an accumulator over a BEC with $q = 0.49, 0.4, 0.3$ and 0.2 .

The EXIT function of an accumulator has been given by (see [81, eq. (14)])

$$I_{ext,acc} = \left[\frac{1-q}{1-qI_A} \right]^2. \quad (6.4.14)$$

Figure 6.13 depicts the EXIT curve comparison of a non-systematic SPC code in cyclic form with length 1000 and an accumulator over a BEC with $q = 0.49, 0.4, 0.3$ and 0.2 . We observe that the EXIT curves of these two codes are the same. This verifies our results for the AWGN channel that when the length of a non-systematic SPC code in cyclic form is large enough, its EXIT function is the same as that of an accumulator.

Closed Form EXIT Functions for $(L, 2)$ Codes

For a $(L, 2)$ code with generator matrix

$$\begin{pmatrix} 1 & \cdots & 1 & 1 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1 & 1 & \cdots & 1 \end{pmatrix}, \quad (6.4.15)$$

suppose the number of ones in each row is $\lfloor 2L/3 \rfloor$. Then the two rows have $2 \cdot \lfloor 2L/3 \rfloor - L$ ones in common. Let the L code bits be X_1, \dots, X_L . Based on the generator matrix (6.4.15) the two information bits can be X_1 and X_L . It is readily checked that the extrinsic information for all code bits that have two ones in the corresponding columns of (6.4.15) should be the same; so should be the extrinsic information for all code bits that have a single one in their corresponding columns. Therefore we can just focus on calculating $I_{ext,1}$ and $I_{ext,\lfloor 2L/3 \rfloor}$. We use the same decoding model as previously and denote the outputs of the communication channel by $Y = (Y_1, \dots, Y_L)$ and the outputs of the extrinsic channel by $A = (A_1, \dots, A_L)$, respectively. Let $I_{temp1} = I(X_1; Y_1 A_1 A_2 \cdots A_{L-\lfloor 2L/3 \rfloor})$. Then

$$I_{temp1} = J \left(\sqrt{(L - \lfloor 2L/3 \rfloor) \cdot [J^{-1}(I_A)]^2 + 8R \cdot \frac{E_b}{N_0}} \right).$$

Since $X_{\lfloor 2L/3 \rfloor} = X_1 + X_L$,

$$\begin{aligned} I(X_{\lfloor 2L/3 \rfloor}; Y_1 A_1 A_2 \cdots A_{L-\lfloor 2L/3 \rfloor} Y_L A_{\lfloor 2L/3 \rfloor+1} \cdots A_L) \\ \approx 1 - J \left(\sqrt{2 \cdot [J^{-1}(1 - I_{temp1})]^2} \right), \end{aligned}$$

which we denote by I_{temp2} . It follows

$$I_{ext,[2L/3]} = J \left(\sqrt{(2 \cdot \lfloor 2L/3 \rfloor - L - 1)[J^{-1}(I_A)]^2 + [J^{-1}(I_{temp2})]^2} \right).$$

Let $I_{temp3} = I(X_{\lfloor 2L/3 \rfloor}; A_{L-\lfloor 2L/3 \rfloor+1} \cdots A_{\lfloor 2L/3 \rfloor})$. Then

$$I_{temp3} = J \left(\sqrt{(2 \cdot \lfloor 2L/3 \rfloor - L) \cdot [J^{-1}(I_A)]^2} \right).$$

Similarly $I_{temp4} \approx I(X_1; Y_L A_{L-\lfloor 2L/3 \rfloor+1} \cdots A_L)$ with

$$I_{temp4} = 1 - J \left(\sqrt{[J^{-1}(1 - I_{temp1})]^2 + [J^{-1}(1 - I_{temp3})]^2} \right).$$

It follows

$$I_{ext,1} \approx J \left(\sqrt{(L - \lfloor \frac{2L}{3} \rfloor - 1) \cdot [J^{-1}(I_A)]^2 + 8R \cdot \frac{E_b}{N_0} + [J^{-1}(I_{temp4})]^2} \right).$$

Combining $I_{ext,1}$ and $I_{ext,[2L/3]}$, we obtain

$$I_{ext,av} \approx \frac{2L - 2 \cdot \lfloor 2L/3 \rfloor}{L} I_{ext,1} + \frac{2 \cdot \lfloor 2L/3 \rfloor - L}{L} I_{ext,[2L/3]}. \quad (6.4.16)$$

Figure 6.14 depicts the comparison of EXIT curves in closed form and those obtained from Monte Carlo simulations for the (6,2) code with generator matrix (6.4.15) for $L = 6$. We observe that in both cases, the EXIT curves of these two methods are almost the same, which validates the derived EXIT functions.

6.4.2 EXIT Functions over the AWGN Channel Obtained from the BEC EXIT Functions

For more general variable or check component codes, the method in [82] has been applied. It also uses the general decoding model in Figure 6.8.

If the erasure probabilities of the communication and extrinsic channels are q and p , respectively, and if Encoders 1 and 2 have no idle components, then as shown in [93], the EXIT function of the $(m + n, k)$ code in Figure 6.8 is

$$I_E^{BEC}(p, q) = 1 - \frac{1}{m} \sum_{h=0}^n (1 - q)^h q^{n-h} \sum_{g=1}^m (1 - p)^{g-1} p^{m-g}$$

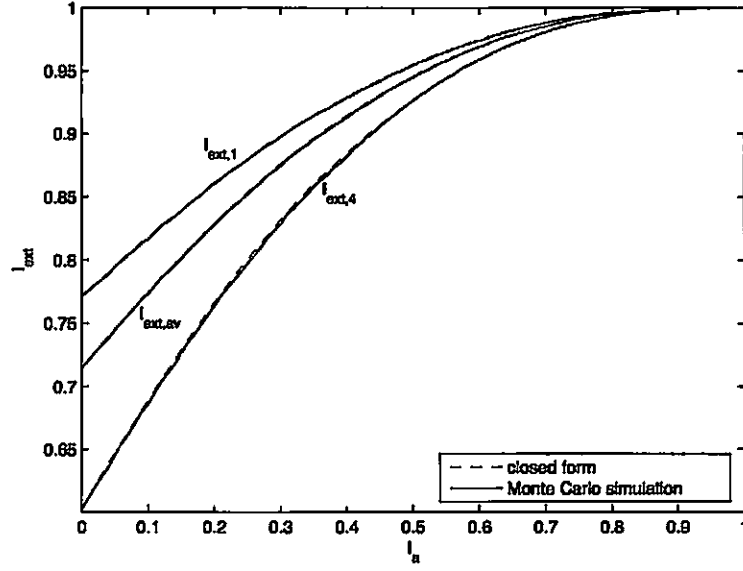


Figure 6.14: Comparison of EXIT curves in closed form and those obtained from Monte Carlo simulation for the (6,2) code at the SNR value 1.9 dB.

$$\cdot [g \cdot \tilde{e}_{g,h} - (m - g + 1) \cdot \tilde{e}_{g-1,h}] \quad (6.4.17)$$

where $\tilde{e}_{g,h}$ is the (g, h) -th unnormalized split information function, which is defined as the summation of the dimensions of all possible codes composed of g positions among v and h positions among x . If there is no communication channel, i.e., $n = 0$, we have

$$I_E^{BEC}(p) = 1 - \frac{1}{m} \sum_{g=1}^m (1-p)^{g-1} p^{m-g} \cdot [g \cdot \tilde{e}_g - (m - g + 1) \cdot \tilde{e}_{g-1}] \quad (6.4.18)$$

where \tilde{e}_g is the g -th unnormalized information function, which is defined as the summation of the dimensions of all possible codes composed of g positions among v .

For an AWGN channel, EXIT functions have been approximated in [82] as

$$I_E^{AWGN} \left(\frac{E_b}{N_0} \right) \cong \frac{1}{\ln 2} \sum_{i=1}^{\infty} \frac{1}{(2i-1)(2i)} I_E^{BEC}(\epsilon_i) \quad (6.4.19)$$

where $I_E^{BEC}(\cdot)$ is given in (6.4.18) with $p = \epsilon_i$, and

$$\epsilon_i = 1 - \Phi_i \left(4R \cdot \frac{E_b}{N_0} \right)$$

with

$$\Phi_i(m) = \int_{-1}^{+1} \frac{2t^{2i}}{(1-t^2)\sqrt{4\pi m}} e^{-\frac{(\ln \frac{1+t}{1-t} - m)^2}{4m}} dt. \quad (6.4.20)$$

We can use (6.4.19) directly to estimate the EXIT functions of check component codes of a DGLDPC code, by just replacing $4R \cdot \frac{E_b}{N_0}$ with $\frac{[J^{-1}(I_A)]^2}{2}$, where $J(\cdot)$ is defined in (4.1.1).

As shown in [82], (6.4.19) is a good estimate of the EXIT functions of high-rate codes, such as SPC codes and Hamming codes. Figure 6.15 compares the EXIT curves obtained by theoretical estimates and Monte-Carlo simulations for several SPC and Hamming codes. We observe that for high-rate codes, (6.4.19) is indeed a good estimate.

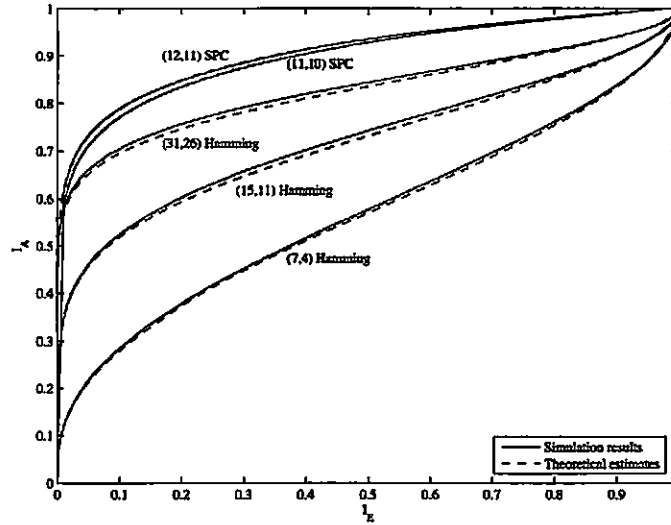


Figure 6.15: EXIT curve comparison of theoretical and simulation results for the (11,10), (12,11) SPC codes and the (7,4), (15,11), (31,26) Hamming codes as check component codes over an AWGN channel.

The EXIT functions for low-rate codes can be obtained from the duality property. They can be approximated by [82]

$$I_E^{AWGN} \left(\frac{E_b}{N_0} \right) \cong 1 - \frac{1}{\ln 2} \sum_{i=1}^{\infty} \frac{1}{(2i-1)(2i)} I_E^{L,BEC}(\epsilon_i) \quad (6.4.21)$$

where $I_E^{L,BEC}(\epsilon_i)$ is the average extrinsic information at the output of a MAP decoder for the dual code of the corresponding low-rate code over a BEC with erasure probability ϵ_i , and

$$\epsilon_i = 1 - \Phi_i \left(\frac{\left[J^{-1} \left(1 - J \left(\sqrt{8R \cdot \frac{E_b}{N_0}} \right) \right) \right]^2}{2} \right).$$

The results in [82] can be readily extended to variable component codes. A variable node is modelled as a $(m+k, k)$ code (i.e., set $n = k$ in Figure 6.8) with generator matrix in the form $[G|I_k]$. Since its rate is $\frac{k}{m+k}$, it is usually a low-rate code and we need to use (6.4.21) to derive its EXIT function. According to Figure 6.8, the outputs from both communication and extrinsic channels contribute to MAP decoding. Let the erasure probabilities of these two channels be η_i and ϵ_i , respectively. We can rewrite (6.4.21) as

$$I_E^{AWGN} \left(I_A, \frac{E_b}{N_0}, R \right) \cong 1 - \frac{1}{\ln 2} \sum_{i=1}^{\infty} \frac{1}{(2i-1)(2i)} \cdot I_E^{L,BEC}(\epsilon_i, \eta_i) \quad (6.4.22)$$

where $\epsilon_i = 1 - \Phi_i \left(\frac{1}{2} [J^{-1}(1 - I_A)]^2 \right)$ and $\eta_i = 1 - \Phi_i \left(\frac{1}{2} \left[J^{-1} \left(1 - J \left(\sqrt{8E_b R / N_0} \right) \right) \right]^2 \right)$. Given the split information function of the dual code, $I_E^{L,BEC}(\epsilon_i, \eta_i)$ is evaluated according to (6.4.17).

Figure 6.16 compares the EXIT curves of theoretical and simulation results for several simplex codes. We observe that for low-rate codes, (6.4.22) is again a good estimate.

However for other moderate-rate codes, the estimates from (6.4.19) and (6.4.22) are not as good as those for high-rate and low-rate codes. Figure 6.17 depicts an EXIT curve comparison of theoretical and simulation results for a (31,10) random code. Compared with the results of Figure 6.16, the EXIT function estimate of (31,10) random codes is not as good as that of (31,5) simplex codes.

It is worth mentioning that we can also use (6.4.19) and (6.4.22) to obtain the EXIT functions of systematic or non-systematic SPC codes or $(L, 2)$ codes, which have been derived in Section 6.4.1 using information combining. However, (6.4.19) and (6.4.22) are based on EXIT functions over the BEC, for which we need to calculate information

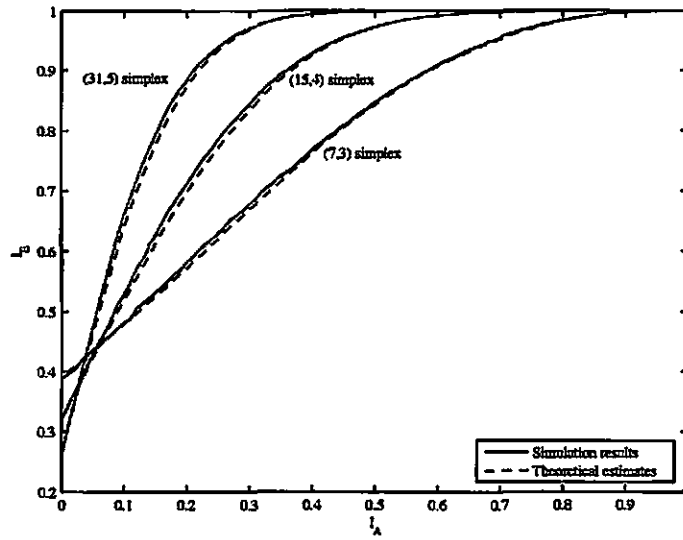


Figure 6.16: EXIT curve comparison of theoretical and simulation results for the (7,3), (15,4) and (31,5) simplex codes as variable component codes over an AWGN channel at $E_b/N_0 = 0.7$ dB and $R = 0.5$.

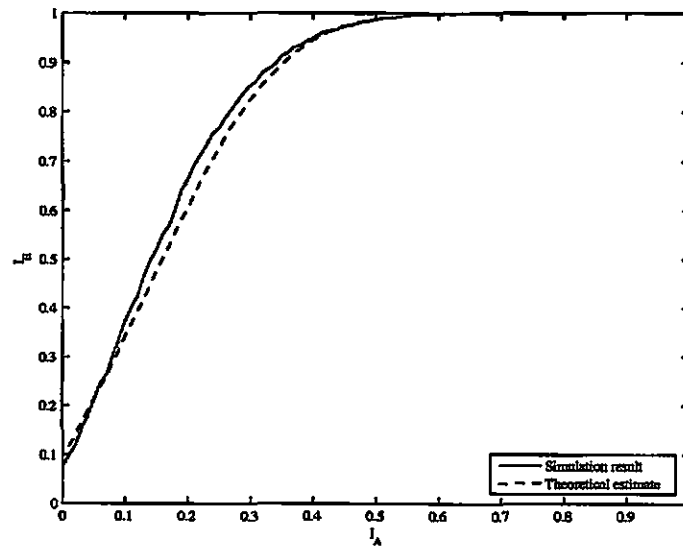


Figure 6.17: EXIT curve comparison of theoretical and simulation results for a (31,10) random code as variable component codes over an AWGN channel at $E_b/N_0 = 0.5$ dB and $R = 0.5$.

functions or split information functions. For long codes, this computation becomes cumbersome. To overcome this problem, the expected information function of random codes has been introduced in [92]. The EXIT functions obtained from information combining do not have such a problem. Their computation is as fast for long codes as it is for short codes.

6.5 Threshold Optimization over the AWGN Channel

In [93], it is shown that the EXIT curve of the variable nodes must match with that of the check nodes in order to approach capacity for the BEC. This curve-fitting technique also works well for BPSK signaling over an AWGN channel.

Suppose the maximum degrees of variable repetition codes and check SPC codes are d_{vmax} and d_{cmax} , respectively and the corresponding distributions are $(\lambda_2, \lambda_3, \dots, \lambda_{d_{vmax}})$ and $(\rho_2, \rho_3, \dots, \rho_{d_{cmax}})$, respectively. Assume there are T variable component codes other than repetition codes and S check component codes other than SPC codes and the corresponding distributions are $(\lambda_{V_1}, \lambda_{V_2}, \dots, \lambda_{V_T})$ and $(\rho_{C_1}, \rho_{C_2}, \dots, \rho_{C_S})$, respectively. The distributions should satisfy the following constraints:

- $1 \geq \lambda_i \geq 0$ for $i = 2, 3, \dots, d_{vmax}$
- $1 \geq \lambda_{V_i} \geq 0$ for $i = 1, \dots, T$
- $1 \geq \rho_i \geq 0$ for $i = 2, 3, \dots, d_{cmax}$
- $1 \geq \rho_{C_i} \geq 0$ for $i = 1, \dots, S$
- $\sum_{i=2}^{d_{vmax}} \lambda_i + \sum_{i=1}^T \lambda_{V_i} = 1$
- $\sum_{i=2}^{d_{cmax}} \rho_i + \sum_{i=1}^S \rho_{C_i} = 1$
-

$$R = 1 - \frac{\sum_{i=2}^{d_{cmax}} \rho_i \left(\frac{1}{i}\right) + \sum_{i=1}^S \rho_{C_i} (1 - R_{C_i})}{\sum_{i=2}^{d_{vmax}} \lambda_i \left(\frac{1}{i}\right) + \sum_{i=1}^T \lambda_{V_i} R_{V_i}}$$

where R_{V_i} and R_{C_i} are the rates of the i -th variable and check component codes other than repetition and SPC codes, respectively.

Form an L -dimensional vector $\mathbf{p} = (\lambda_3, \dots, \lambda_{d_{vmax}-1}, \lambda_{v_1}, \dots, \lambda_{v_T}, \rho_3, \dots, \rho_{d_{cmax}}, \rho_{C_1}, \dots, \rho_{C_S})$ with $L = d_{vmax} + d_{cmax} + T + S - 5$. Note that λ_2 , $\lambda_{d_{vmax}}$ and ρ_2 are not included in \mathbf{p} as implicitly given by the last three constraints. Then the target is to find \mathbf{p} that yields the smallest threshold, i.e., the smallest $\frac{E_b}{N_0}$ that guarantees the average EXIT curve of the variable nodes lies above that of the check nodes.

The optimization can be realized by differential evolution (DE) [88]. DE is carried out as follows.

Step 1: Set the maximum number of generations to G_{max} and generation index $g = 0$. Initialize K vectors \mathbf{p} and denote them as $\mathbf{p}_i^{(0)}$ with $i = 1, \dots, K$. For each $\mathbf{p}_i^{(0)}$, evaluate the EXIT curves and obtain the corresponding threshold $\left(\frac{E_b}{N_0}\right)_i^{(0)}$. The best $\mathbf{p}_i^{(0)}$ is the one with the smallest $\left(\frac{E_b}{N_0}\right)_i^{(0)}$. Denote it as $\mathbf{p}_{l_{best}}^{(0)}$, where $l_{best} = \arg \min_{i=1}^K \left\{ \left(\frac{E_b}{N_0}\right)_i^{(0)} \right\}$. The best threshold is $\left(\frac{E_b}{N_0}\right)_{min}^{(0)} = \min_{i=1}^K \left\{ \left(\frac{E_b}{N_0}\right)_i^{(0)} \right\}$.

Step 2: For each $i = 1, \dots, K$, randomly generate four distinct integers $\{r_j | r_j \in [1, K], r_j \neq i\}$ with $j = 1, 2, 3, 4$ and define the test vector $\mathbf{q}_i^{(g+1)} = \mathbf{p}_{l_{best}}^{(g)} + \gamma(\mathbf{p}_{r_1}^{(g)} - \mathbf{p}_{r_2}^{(g)} + \mathbf{p}_{r_3}^{(g)} - \mathbf{p}_{r_4}^{(g)})$, where γ is a real constant that controls the amplification of the differential variation. We set $\gamma = 0.5$ in our DE. Calculate the threshold of each test vector and denote them as $\left(\frac{E_b}{N_0}\right)_{i,t}^{(g+1)}$.

Step 3: For each $i = 1, \dots, K$, compare the threshold of the test vector $\mathbf{q}_i^{(g+1)}$ with the original vector $\mathbf{p}_i^{(g)}$. Updating follows the rule:

$$\mathbf{p}_i^{(g+1)} = \begin{cases} \mathbf{q}_i^{(g+1)} & \text{if } \left(\frac{E_b}{N_0}\right)_{i,t}^{(g+1)} \leq \left(\frac{E_b}{N_0}\right)_i^{(g)} \\ \mathbf{p}_i^{(g)} & \text{otherwise} \end{cases}$$

and $\left(\frac{E_b}{N_0}\right)_i^{(g+1)} = \min \left\{ \left(\frac{E_b}{N_0}\right)_{i,t}^{(g+1)}, \left(\frac{E_b}{N_0}\right)_i^{(g)} \right\}$. Denote the new best \mathbf{p} as $\mathbf{p}_{l_{best}}^{(g+1)}$, where $l_{best} = \arg \min_{i=1}^K \left\{ \left(\frac{E_b}{N_0}\right)_i^{(g+1)} \right\}$ and the new best threshold $\left(\frac{E_b}{N_0}\right)_{min}^{(g+1)} = \min_{i=1}^K \left\{ \left(\frac{E_b}{N_0}\right)_i^{(g+1)} \right\}$.

Step 4: Set $g := g + 1$. If G_{max} is reached or $\left(\frac{E_b}{N_0}\right)_{min}^{(g)}$ stops decreasing, output $\mathbf{p}_{l_{best}}^{(g)}$ and $\left(\frac{E_b}{N_0}\right)_{min}^{(g)}$.

6.6 Ensemble Weight Enumerators for Protograph-Based DGLDPC Codes

A protograph [83] is defined as a Tanner graph with a relatively small number of nodes. Each node and each edge in a protograph represent a node type and an edge type, respectively. A larger graph can be obtained from a protograph by a “copy-and-permute” process as shown in Figure 6.18. In this process, a protograph is first copied p times and then the p edges of the same type are permuted under the constraint that they still connect to the same type of nodes. A protograph can contain parallel edges. However, these parallel edges should be eliminated in the copy-and-permute process in order to obtain an appropriate Tanner graph for the parity check matrix.

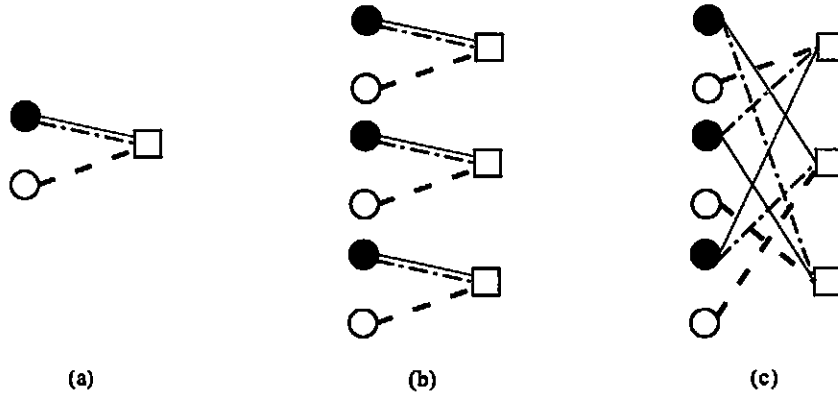


Figure 6.18: The process to obtain a larger graph from a protograph. (a) Protograph; (b) Copy 3 times; (c) Permute the edges.

Asymptotic weight enumerators for ensembles of protograph-based LDPC codes and GLDPC codes have been discussed in [84] and [85], respectively. In this section, we derive the asymptotic weight enumerators for protograph-based DGLDPC ensembles.

A DGLDPC protograph with n_v super variable nodes and n_c super check nodes is denoted as $G = (V, C, E)$, where $V = \{v_1, v_2, \dots, v_{n_v}\}$ is the set of super variable nodes, $C = \{c_1, c_2, \dots, c_{n_c}\}$ is the set of super check nodes, and E is the set of edges. The codes corresponding to the super nodes in V and C are $(l_{v_1}, k_{v_1}), (l_{v_2}, k_{v_2}), \dots, (l_{v_{n_v}}, k_{v_{n_v}})$ and $(l_{c_1}, k_{c_1}), (l_{c_2}, k_{c_2}), \dots, (l_{c_{n_c}}, k_{c_{n_c}})$, respectively. A DGLDPC code is obtained by copying

the protograph G p times and permuting edges among the p copies. In order to apply the

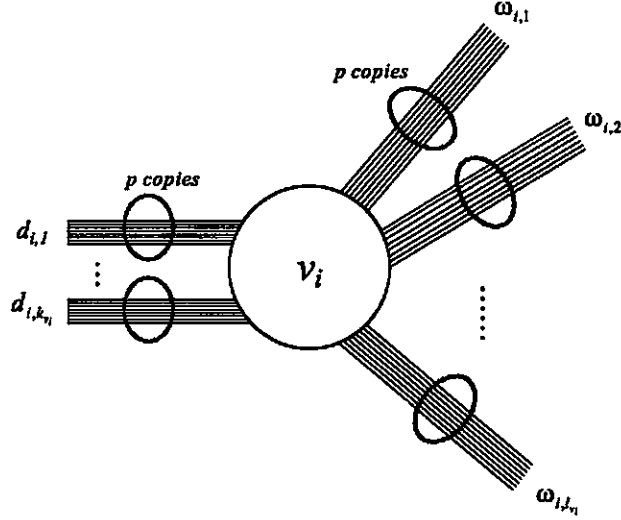


Figure 6.19: A constituent code composed of p copies of super variable node v_i .

results in [87], we regard the group of p copies of super variable node v_i as a constituent code with k_{v_i} length- p inputs and l_{v_i} length- p outputs. As shown in Figure 6.19, the weight vectors of k_{v_i} inputs and l_{v_i} outputs are denoted as $\mathbf{d}_i = [d_{i,1}, d_{i,2}, \dots, d_{i,k_{v_i}}]$ and $\mathbf{w}_i = [\omega_{i,1}, \omega_{i,2}, \dots, \omega_{i,l_{v_i}}]$, respectively, where $d_{i,j}$ and $\omega_{i,j}$ are the weights of the j -th input and output of node v_i , respectively. Similarly, we regard the group of p copies of super check node c_j as a constituent code with l_{c_j} length- p inputs and no output. The weight vector of l_{c_j} inputs is denoted as $\mathbf{z}_j = [z_{j,1}, z_{j,2}, \dots, z_{j,l_{c_j}}]$ as shown in Figure 6.20, where $z_{j,i}$ is the weight of the i -th input of node c_j .

Let $\mathbf{d} = [\mathbf{d}_1 \mathbf{d}_2 \dots \mathbf{d}_{n_v}]$, $\mathbf{w} = [\mathbf{w}_1 \mathbf{w}_2 \dots \mathbf{w}_{n_v}]$, and $\mathbf{z} = [\mathbf{z}_1 \mathbf{z}_2 \dots \mathbf{z}_{n_c}]$. We also define the following vector weight enumerators.

- $A_{\mathbf{w}_i}^{v_i}$: the vector weight enumerator for constituent variable node v_i with \mathbf{w}_i as the output weight vector.
- $A_{\mathbf{z}_j}^{c_j}$: the vector weight enumerator for constituent check node c_j with \mathbf{z}_j as the output weight vector. We have $z_{j,t} = \omega_{i,h}$ if the t -th edge of super check node c_j is the h -th

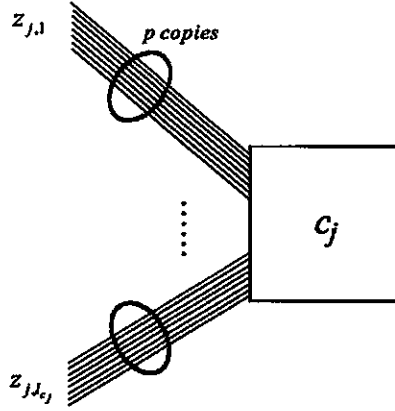


Figure 6.20: A constituent code composed of p copies of super check node c_j .

edge of super variable node v_i . Denote the mapping between w and z by π , then $z = \pi(w)$.

- A_d : the ensemble vector weight enumerator for a protograph-based DGLDPC code with d as the input weight vector.
- A_d : the average number of weight- d codewords in the ensemble.

Let K_{c_j} be the number of codewords in super check node c_j and \mathbf{Q}^{c_j} be the $K_{c_j} \times l_{c_j}$ matrix with all the codewords of c_j as its rows. Let $\mathbf{n}_j = [n_{j,1}, n_{j,2}, \dots, n_{j,K_{c_j}}]$ be the frequency at which each codeword appears in the p copies of node c_j . Then $n_{j,1}, n_{j,2}, \dots, n_{j,K_{c_j}} > 0$ and $\sum_{i=1}^{K_{c_j}} n_{j,i} = p$. From [85], $A_{z_j}^{c_j}$ can be calculated by

$$A_{z_j}^{c_j} = \sum_{\mathbf{n}_j \in S_{c_j}} C(p; n_{j,1}, n_{j,2}, \dots, n_{j,K_{c_j}}), \quad (6.6.1)$$

where $S_{c_j} = \{\mathbf{n}_j : z_j = \mathbf{n}_j \cdot \mathbf{Q}^{c_j}\}$ and $C(y; y_1, y_2, \dots, y_q) = \frac{y!}{y_1! y_2! \dots y_q!}$.

Similarly, we can derive $A_w^{v_i}$. Let K_{v_i} be the number of codewords in super variable node v_i , \mathbf{B}^{v_i} be the $K_{v_i} \times k_{v_i}$ matrix with all the possible binary k_{v_i} -tuples as its rows and \mathbf{T}^{v_i} be the $K_{v_i} \times l_{v_i}$ matrix with all the codewords of v_i as its rows. If the generator matrix of v_i is \mathbf{G}^{v_i} , then $\mathbf{T}^{v_i} = (\mathbf{B}^{v_i} \cdot \mathbf{G}^{v_i}) \bmod 2$. Let $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \dots, x_{i,K_{v_i}}]$ be the frequency at which each codeword appears in the p copies of node v_i . Then $x_{i,1}, x_{i,2}, \dots, x_{i,K_{v_i}}$

> 0 and $\sum_{j=1}^{K_{v_i}} x_{i,j} = p$. Denote $S_{v_i} = \{\mathbf{x}_i : \mathbf{x}_i \cdot \mathbf{T}^{v_i} = \mathbf{w}_i\}$. Then $A_{\mathbf{w}_i}^{v_i}$ can be written as

$$A_{\mathbf{w}_i}^{v_i} = \sum_{\mathbf{x}_i \in S_{v_i}} C(p; x_{i,1}, x_{i,2}, \dots, x_{i,K_{v_i}}). \quad (6.6.2)$$

For example, suppose v_i is a (6,2) code with generator matrix $\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$ and $p = 4$. Then $k_{v_i} = 2$, $l_{v_i} = 6$, $K_{v_i} = 4$,

$$\mathbf{B}^{v_i} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad (6.6.3)$$

and

$$\mathbf{T}^{v_i} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}. \quad (6.6.4)$$

(a) When $\mathbf{w}_i = [3, 3, 2, 2, 1, 1]$, we obtain $S_{v_i} = \{[1, 0, 2, 1]\}$ by solving $\mathbf{x}_i \cdot \mathbf{T}^{v_i} = \mathbf{w}_i$ and $\sum_{j=1}^4 x_{i,j} = 4$. Therefore, $A_{\mathbf{w}_i}^{v_i} = C(4; 1, 0, 2, 1) = 12$.

(b) When $\mathbf{w}_i = [3, 3, 1, 1, 3, 3]$, $S_{v_i} = \emptyset$. Therefore, $A_{\mathbf{w}_i}^{v_i} = 0$.

As shown in [84][85], $A_{\mathbf{d}}$ can be computed as

$$A_{\mathbf{d}} = \sum_{\mathbf{w}} \frac{\prod_{i=1}^{n_v} \Delta_i(\mathbf{w}_i; \mathbf{d}_i) \cdot \prod_{i=1}^{n_v} A_{\mathbf{w}_i}^{v_i} \cdot \prod_{j=1}^{n_c} A_{\mathbf{z}_j}^{c_j}}{\prod_{s=1}^{n_v} \prod_{r=1}^{l_{v_s}} C(p; \omega_{s,r}, p - \omega_{s,r})} \quad (6.6.5)$$

where $\Delta_i(\mathbf{a}; \mathbf{b})$ is defined as

$$\Delta_i(\mathbf{a}; \mathbf{b}) = \begin{cases} 1 & \text{if there exists an } \mathbf{x}_i \text{ such that } \mathbf{x}_i \cdot \mathbf{B}^{v_i} = \mathbf{b} \text{ and } \mathbf{x}_i \cdot \mathbf{T}^{v_i} = \mathbf{a} \\ 0 & \text{otherwise} \end{cases}$$

Suppose all the codeword bits are transmitted and there is no punctured bit, then

$$A_{\mathbf{d}} = \sum_{\mathbf{d} \in S_{\mathbf{d}}} A_{\mathbf{d}} \quad (6.6.6)$$

where $S_{\mathbf{d}} = \{\mathbf{d} : \sum_{i=1}^{n_v} \sum_{j=1}^{k_{v_i}} d_{i,j} = \mathbf{d}\}$.

Define the asymptotic weight enumerator

$$\gamma(\delta) = \limsup_{N \rightarrow \infty} \frac{\ln A_d}{N}, \quad (6.6.7)$$

where N is the length of the DGLDPC code and $\delta = d/N$. Since $N = p \cdot \sum_{i=1}^{n_v} k_{v_i}$, we have $\delta = d/(p \cdot \sum_{i=1}^{n_v} k_{v_i})$. Let $\tilde{\delta} = d/p$, then $\delta = \tilde{\delta}/(\sum_{i=1}^{n_v} k_{v_i})$. Define

$$\tilde{\gamma}(\tilde{\delta}) = \limsup_{p \rightarrow \infty} \frac{\ln A_d}{p}, \quad (6.6.8)$$

then

$$\gamma(\delta) = \frac{1}{\sum_{i=1}^{n_v} k_{v_i}} \cdot \tilde{\gamma} \left(\delta \cdot \sum_{i=1}^{n_v} k_{v_i} \right). \quad (6.6.9)$$

Define the following normalized vectors:

- $\tilde{\delta} = [\tilde{\delta}_1, \tilde{\delta}_2, \dots, \tilde{\delta}_{n_v}]$ with $\tilde{\delta}_i = d_i/p$ and $\tilde{\delta}_{i,j} = d_{i,j}/p$ for $i = 1, 2, \dots, n_v$ and $j = 1, 2, \dots, k_{v_i}$.
- $\beta = [\beta_1, \beta_2, \dots, \beta_{n_v}]$ with $\beta_i = w_i/p$ and $\beta_{i,j} = w_{i,j}/p$ for $i = 1, 2, \dots, n_v$ and $j = 1, 2, \dots, l_{v_i}$.
- $\xi = [\xi_1, \xi_2, \dots, \xi_{n_c}]$ with $\xi_j = z_j/p$ and $\xi_{j,i} = z_{j,i}/p$ for $j = 1, 2, \dots, n_c$ and $i = 1, 2, \dots, l_{c_j}$. Since $z = \pi(w)$, then $\xi = \pi(\beta)$.
- $\eta = [\eta_1, \eta_2, \dots, \eta_{n_v}]$ with $\eta_i = x_i/p$ and $\eta_{i,j} = x_{i,j}/p$ for $i = 1, 2, \dots, n_v$ and $j = 1, 2, \dots, K_{v_i}$. Since for $i = 1, 2, \dots, n_v$, $\sum_{j=1}^{K_{v_i}} \eta_{i,j} = 1$, η_i is the empirical probability distribution of each codeword when given p codewords in v_i .
- $\phi = [\phi_1, \phi_2, \dots, \phi_{n_c}]$ with $\phi_j = n_j/p$ and $\phi_{j,i} = n_{j,i}/p$ for $j = 1, 2, \dots, n_c$ and $i = 1, 2, \dots, K_{c_j}$. Since for $j = 1, 2, \dots, n_c$, $\sum_{i=1}^{K_{c_j}} \phi_{j,i} = 1$, ϕ_j is the empirical probability distribution of each codeword when given p codewords in c_j .

Let $S_{\tilde{\delta}} = \{\tilde{\delta} : \sum_{i=1}^{n_v} \sum_{j=1}^{k_{v_i}} \tilde{\delta}_{i,j} = \tilde{\delta}\}$ and $S_{\beta} = \{\beta : \prod_{i=1}^{n_v} \Delta_i(\beta_i; \tilde{\delta}_i) = 1\}$. Define

$$a^{v_i}(\beta_i) \triangleq \limsup_{p \rightarrow \infty} \frac{\ln A_{w_i}^{v_i}}{p}$$

$$a^{c_j}(\xi_j) \triangleq \limsup_{p \rightarrow \infty} \frac{\ln A_{z_j}^{c_j}}{p}$$

From Stirling's formula, $\lim_{p \rightarrow \infty} \sup \ln[C(p; w_{s,r}, p-w_{s,r})]/p = H(\beta_{s,r}) = -(1-\beta_{s,r}) \ln(1-\beta_{s,r}) - \beta_{s,r} \ln(\beta_{s,r})$. Since for large and distinct x and y , $\ln(e^x + e^y) \simeq \max(x, y)$ and similarly for more than two variables, we have

$$\tilde{\gamma}(\tilde{\delta}) \simeq \max_{\tilde{\delta} \in S_{\tilde{\delta}}} \left\{ \max_{\beta \in S_{\beta}} \left\{ \sum_{i=1}^{n_v} a^{v_i}(\beta_i) + \sum_{j=1}^{n_c} a^{c_j}(\xi_j) - \sum_{s=1}^{n_v} \sum_{r=1}^{l_{v_s}} H(\beta_{s,r}) \right\} \right\} \quad (6.6.10)$$

Let $S_{\tilde{v}_i} = \{\eta_i : \beta_i = \eta_i \cdot T^{v_i}\}$ and $S_{\tilde{c}_j} = \{\phi_j : \xi_j = \phi_j \cdot Q^{c_j}\}$. From [85], we have

$$a^{v_i}(\beta_i) = \max_{\eta_i \in S_{\tilde{v}_i}} H(\eta_i)$$

$$a^{c_j}(\xi_j) = \max_{\phi_j \in S_{\tilde{c}_j}} H(\phi_j).$$

Therefore

$$\tilde{\gamma}(\tilde{\delta}) \simeq \max_{\tilde{\delta} \in S_{\tilde{\delta}}} \left\{ \max_{\beta \in S_{\beta}} \left\{ \sum_{i=1}^{n_v} \max_{\eta_i \in S_{\tilde{v}_i}} H(\eta_i) + \sum_{j=1}^{n_c} \max_{\phi_j \in S_{\tilde{c}_j}} H(\phi_j) - \sum_{s=1}^{n_v} \sum_{r=1}^{l_{v_s}} H(\beta_{s,r}) \right\} \right\} \quad (6.6.11)$$

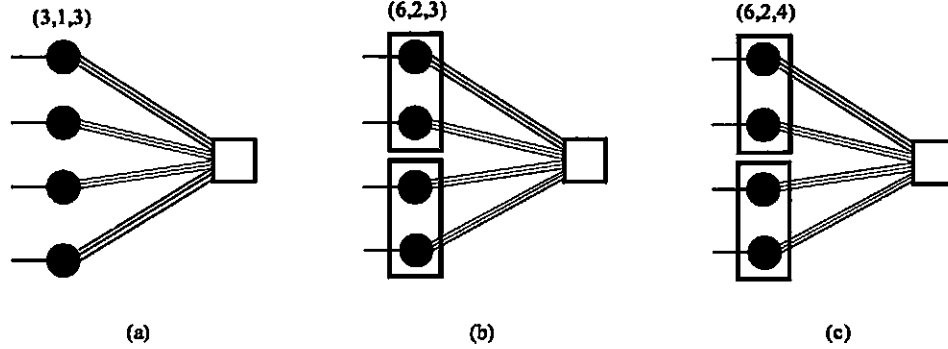


Figure 6.21: Protographs of the LDPC code and DGLDPC code for asymptotic weight enumerator calculation. (a) Protograph of a (3,12) LDPC code; (b) Protograph of a DGLDPC code that is equivalent to the (3,12) LDPC code; (c) Protograph of a DGLDPC code with a (6,2) code as super variable nodes.

We compare next the asymptotic weight enumerators of an LDPC code and its DGLDPC code counterpart. Figure 6.21 depicts the protographs of these codes. Figure 6.21 (a) is the protograph of a (3,12) LDPC code. If we combine every two variable nodes in Figure 6.21 (a) and regard it as a super variable node that represents the (6,2,3) code

with generator matrix $\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$, then we obtain the DGLDPC protograph shown in Figure 6.21 (b), which is equivalent to Figure 6.21 (a). The DGLDPC protograph in Figure (c) is obtained by keeping everything the same as in Figure 6.21 (b) and just changing the code that super variable nodes represent from the (6,2,3) code to the (6,2,4) code with generator matrix $\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$.

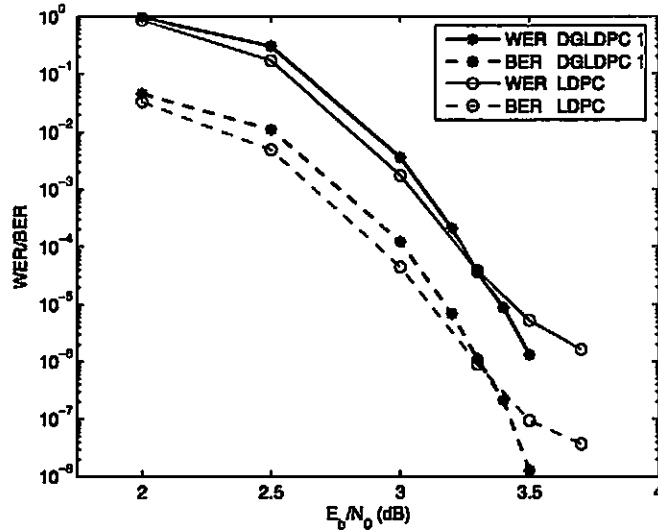


Figure 6.22: WER/BER performances of a length-2016 rate-3/4 protograph-based LDPC code and DGLDPC 1.

The asymptotic growth rate of the minimum distance of the (3,12) LDPC code has been calculated in [86], as 0.00206. The asymptotic growth rate of rate-3/4 random codes is 0.04169. Following (6.6.11), we calculate the asymptotic growth rate of the DGLDPC code in Figure 6.21 (c), as 0.01. Therefore, without changing the number of nodes and the graph connection, an LDPC code can be modified to achieve an improved asymptotic growth rate by regarding a group of variable nodes as a different code. We construct a length-2016 protograph-based DGLDPC code with protograph depicted in Figure 6.21 (c) and denote it as DGLDPC 1. We compare it with a length-2016 protograph-based LDPC code with protograph depicted in Figure 6.21 (a). The weight enumerators of the LDPC code and DGLDPC 1 are estimated as $4z^{12} + 13z^{14} + 171z^{16} + 859z^{18} + 2527z^{20} + \dots$

and $4z^{51} + z^{52} + z^{53} + z^{55} + 5z^{56} + 4z^{57} + 5z^{58} + \dots$, respectively, using the approach in [99][100]. Therefore the estimated minimum distances of these two codes are 12 and 51, respectively, which have a similar proportion to their asymptotic growth rate. The WER and BER performances of the (3,12) LDPC code and DGLDPC 1 are depicted in Figure 6.22. We observe that DGLDPC 1 has a lower error floor than the LDPC code with a slight loss in the waterfall region.

6.7 Simulation Results

Figure 6.23 depicts the WER and BER performances of DGLDPC 1. We have used two decoding algorithms to decode this code. The first one has been described in Section 6.3; we denote it as “MAP” in Figure 6.23 because the MAP algorithm is used to decode the super variable nodes. The second algorithm is denoted as “BP” as it applies standard BP to the parity check matrix of DGLDPC 1. We observe that “MAP” is a little

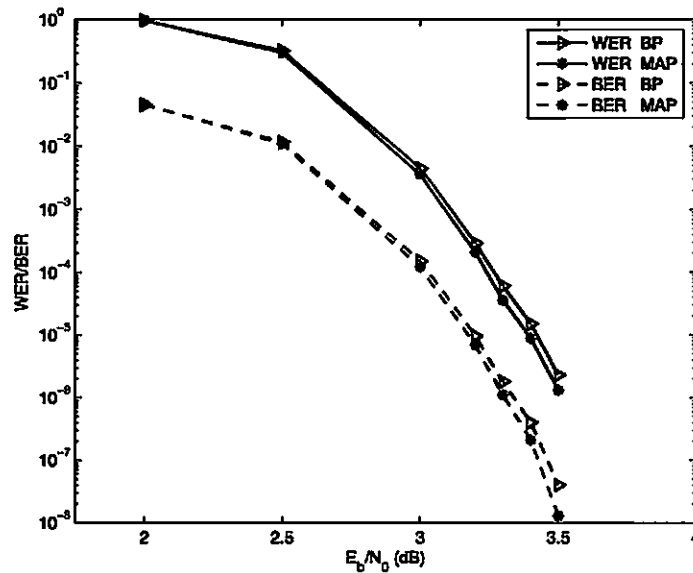


Figure 6.23: WER/BER performances of DGLDPC 1.

better than “BP” and both of them have low error floor. Since the parity check matrix

of DGLDPC 1 contains many four cycles, it is surprising that “BP” performs so close to “MAP”. This shows that four cycles do not necessarily degrade much the performance of an LDPC code as long as they are arranged in appropriate places.

Next, we attempt to optimize the thresholds of rate-1/2 LDPC codes, GLDPC codes and DGLDPC codes with the same maximum variable degree $d_{vmax} = 6$ and the same maximum check degree $d_{cmax} = 15$. The optimized distributions are given in Table 6.1. For the GLDPC code, we mix the (15,11) Hamming code with SPC codes at check

Code type	LDPCopt	LDPC 1	LDPC 2	GLDPC	DGLDPC 2
Variable nodes					
(2,1) Rep.	0.332433	0.339952	0.340206	0.383584	0.018198
(3,1) Rep.	0.239904	0.224844	0.225658	0.277719	0.098646
(4,1) Rep.					0.000048
(5,1) Rep.					0.001625
(6,1) Rep.	0.427663	0.435204	0.434136	0.338697	0.438911
(6,5) SPC					0.442572
Check nodes					
(6,5) SPC	0.666700	0.666616	0.671245		0.245267
(7,6) SPC	0.333300	0.333384	0.328755	0.777479	
(15,11) Ham.				0.222521	0.754733
Thresholds (dB)					
	0.59 (D)	0.56 (E)	0.56 (E)	0.51 (E)	0.32 (E)

Table 6.1: Threshold comparison of LDPC codes, a GLDPC code, and a DGLDPC code. (D): from density evolution (E): from EXIT charts.

nodes. For the DGLDPC code (denoted DGLDPC 2), in addition to mixing the (15,11) Hamming code with SPC codes at check nodes, we also mix the (6,5) SPC code in cyclic form with repetition codes at variable nodes. From DE, we can usually obtain several distributions with the same threshold. In Table 6.1, we recorded two distributions from EXIT charts analysis for LDPC codes with the same threshold (denoted LDPC 1 and LDPC 2). We also compared the degree distribution of our LDPC codes with that from [66], which is the best one returned when the maximum left degree is set to 6 (denoted LDPCopt). We observe from Table 6.1 that these distributions are very similar. As already known, the EXIT chart threshold (obtained using the approximations in the Appendix of [71] for the calculation of $J(\cdot)$ and $J^{-1}(\cdot)$) is slightly better than that obtained from [66]. However,

since we use this technique to design good codes only, the exact threshold value becomes secondary. A better threshold is obtained for the DGLDPC code compared with that of the LDPC and GLDPC codes with the same maximum variable and check degrees.

In order to verify these asymptotic results, we simulated three long codes. For the LDPC code, we have used LDPCopt obtained from [66] and for the GLDPC and DGLDPC codes, we have followed the distributions in Table 6.1. Their graphs were randomly constructed except that double edges and four-cycles were avoided. The length of these three codes is 1000000. Their error performances are depicted in Figure 6.24. At the BER around 10^{-5} , the GLDPC code is about 0.05 dB better than the LDPC code and the DGLDPC code is about 0.2 dB better than the LDPC code, which matches the threshold analysis. The corresponding capacity is 0.19 dB. The error floor of the DGLDPC code is lower than that of the LDPC and GLDPC codes.

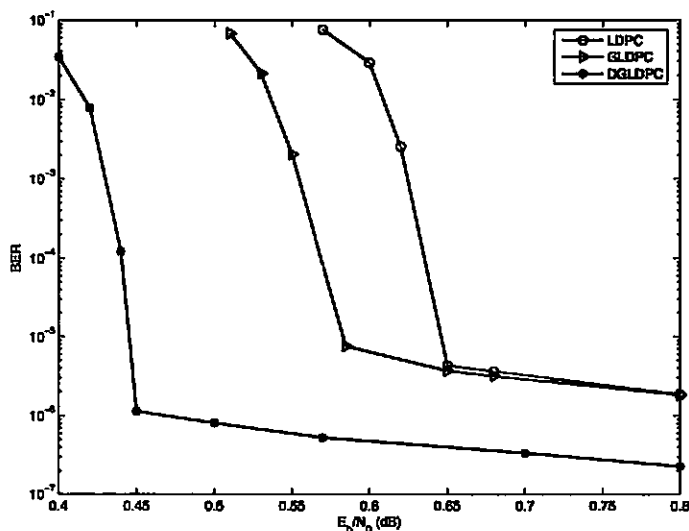


Figure 6.24: BER performance comparison of randomly constructed rate-1/2 LDPC, GLDPC, DGLDPC codes with length 1000000.

We also tried to optimize the thresholds of rate-3/4 LDPC codes and DGLDPC codes with the same maximum variable degree $d_{vmax} = 6$ and the same maximum check degree $d_{cmax} = 31$. The optimized distributions are given in Table 6.2. Again, the LDPC code is obtained from [66]. The two DGLDPC codes are denoted as DGLDPC 3 and

Code type	LDPCopt	DGLDPC 3	DGLDPC 4
Variable nodes			
(2,1) Rep.	0.247607		0.000143
(3,1) Rep.	0.219072	0.000110	0.000016
(4,1) Rep.	0.000001		0.000047
(5,1) Rep.			0.000170
(6,1) Rep.	0.533320	0.374649	0.335842
(6,5) SPC		0.625241	0.663782
Check nodes			
(14,13) SPC	1.0	0.171510	0.099913
(31,26) Ham.		0.828490	0.900087
Thresholds (dB)			
	1.93 (D)	1.82 (E)	1.85 (E)

Table 6.2: Threshold comparison of an LDPC code and two DGLDPC codes with rate 3/4. (D): from density evolution (E): from EXIT charts.

DGLDPC 4, respectively. For DGLDPC 4, we impose the additional constraint that the degree distribution of the (31,26) Hamming code for super check nodes must be greater than 0.9. Figure 6.25 depicts the BER performance of randomly constructed rate-3/4 LDPC, DGLDPC 3 and DGLDPC 4 codes with length 100000. At the BER around 10^{-5} , DGLDPC 3 is about 0.1 dB better than the LDPC code, which matches the threshold analysis. The capacity is 1.63 dB. The error floor of DGLDPC 3 is slightly better than that of the LDPC code. DGLDPC 4 achieves much better error floor performance by slightly sacrificing the waterfall region performance compared with DGLDPC 3.

Figure 6.26 depicts the BER performance of a rate-7/15 length-7650 DGLDPC code and we denote it as DGLDPC 5. DGLDPC 5 uses the (6,1) repetition code, the (6,2) code with generator matrix $\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$, the (6,4) code with generator

matrix $\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$, and the (6,5) SPC code as super variable nodes with $\lambda_1 =$

0.425, $\lambda_2 = 0.075$, $\lambda_3 = 0.075$, $\lambda_4 = 0.425$, respectively, and (15,11) Hamming codes for all super check nodes. The threshold of DGLDPC 5 is 0.3 dB. Compared with the threshold 0.88 dB of a (2,15) GLDPC code with the same rate, it improves by 0.58 dB. The

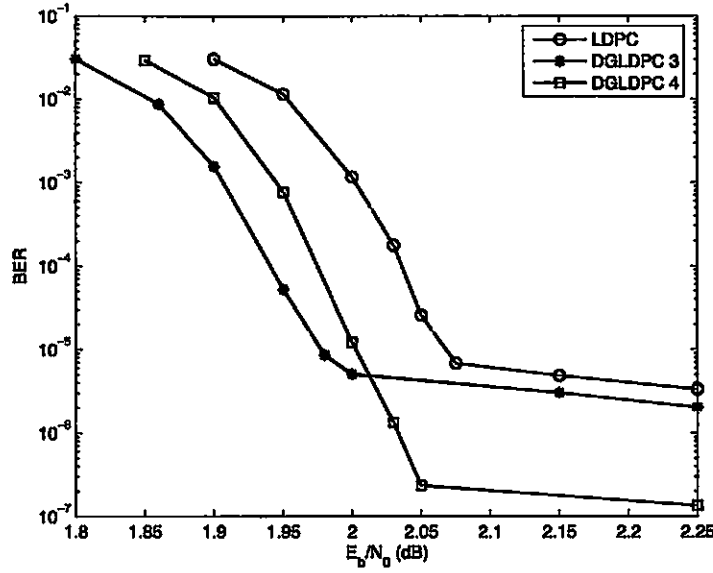


Figure 6.25: BER performance comparison of randomly constructed rate-3/4 LDPC, DGLDPC 3 and DGLDPC 4 codes with length 100000.

improvement of threshold results from introducing the (6,5) SPC code in the super variable nodes, whose EXIT curve is convex. Compared with the concave EXIT curves of repetition codes, it better fits the Hamming code. Since the capacity limit for rate-7/15 is 0.04dB, the threshold of DGLDPC 5 is 0.26 dB away from the capacity. Figure 6.26 depicts the BER performance comparison of DGLDPC 5 with a rate-7/15 length-7680 (2,15) GLDPC code, both with $I_{Max} = 50$: DGLDPC 5 outperforms its counterpart by about 0.35dB.

We finally constructed a short DGLDPC code, denoted as DGLDPC 6. It has length 1536 and rate 1/2. Its super variable nodes contain only the (4,1) repetition code and the (4,3) SPC code in order to reduce decoding complexity. All super check nodes use the (15,11) Hamming code as component code. The threshold predicted by EXIT charts in closed form is 0.77 dB. Figure 6.27 depicts the performance comparison of DGLDPC 6 with an optimized rate-1/2 length-1504 (2,4)-LDPC code over GF(16) [95]. DGLDPC 6 outperforms its counterpart by about 1 dB at the WER of 10^{-6} due to a lower error floor. The computational complexity per check node of DGLDPC 6 is $O(d_c \times 2^{d_c - k_c}) = O(15 \times 2^{15-11}) = O(15 \times 16)$, while that of the (2,4)-LDPC code over GF(q) is $O(d_c \times q \log_2 q) =$

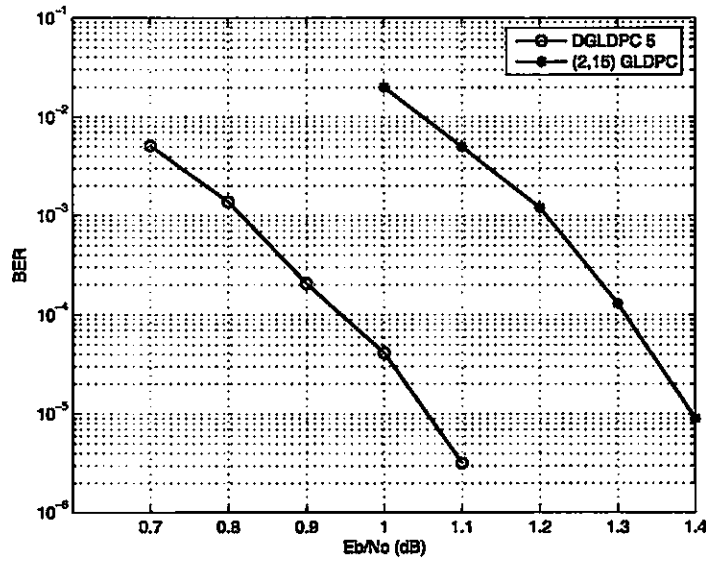


Figure 6.26: BER performance comparison of DGLDPC 5 with a rate-7/15 length-7680 (2,15) GLDPC code.

$O(4 \times 16 \log_2 16) = O(16 \times 16)$. So the computational complexity in check nodes of these two codes is comparable. The variable node processing of DGLDPC codes remains lower. The low error floor of DGLDPC 6 is mainly due to the improved minimum distance obtained by introducing Hamming codes in the check nodes. This leads to a loss of rate, but since for a DGLDPC code component codes are also used in the variable nodes, we can make up for this rate loss.

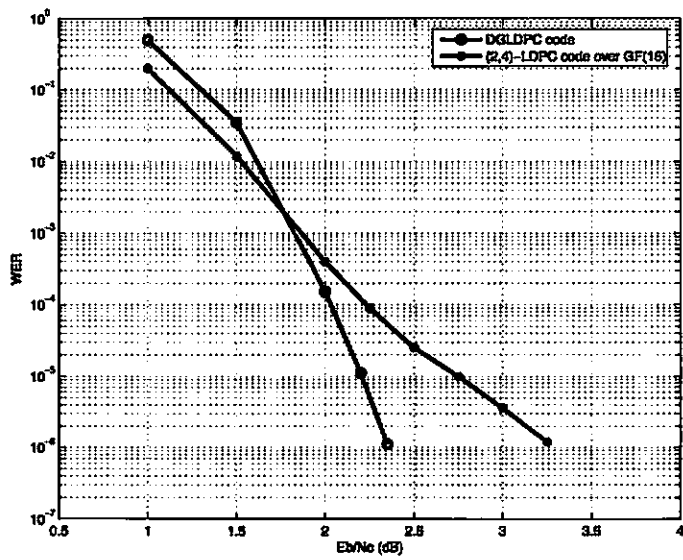


Figure 6.27: WER performance comparison of DGLDPC 6 with a rate-1/2 length-1504 (2,4)-LDPC code over GF(16).

Chapter 7

Conclusion

In this dissertation, we have investigated generalized constructions, decoding and implementation of LDPC codes.

For fast convergence algorithms (plain shuffled BP and replica shuffled BP), we proposed closed-form EXIT functions which avoid the use of computer-intensive simulations. Based on EXIT charts, we can easily compare the speed of convergence of different decoding algorithms. We also proved that the threshold of a code decoded by plain shuffled BP or replica shuffled BP is the same as that of standard BP. Then we investigated group plain shuffled BP and group replica shuffled BP and determined the smallest number of groups (i.e., the highest level of parallelism) to achieve at any given iteration the same performance as that of their corresponding non-group counterparts. All these results have been verified by simulation.

We have designed a VLSI architecture for replica shuffled normalized BP-based decoding of LDPC codes. Replica shuffled decoding can converge very fast, while normalized BP-based decoding can greatly reduce complexity. Combining them leads to a fast and simple decoder operating at 64 Mbps. Our designed architecture can be applied to any quasi-cyclic LDPC code.

We have also proposed DGLDPC codes and used EXIT charts to analyze them. We derived closed-form EXIT functions for several variable component codes of DGLDPC codes, such as systematic SPC codes, non-systematic SPC codes in cyclic form, $(L, 2)$ codes and accumulators. EXIT functions of more general component codes have also been discussed. Based on EXIT charts, differential evolution has been used for threshold op-

timization. It allowed us to match variable and check EXIT curves quickly. We investigated ensemble weight enumerators for protograph-based DGLDPC codes and analyzed their distance properties. We found that without changing the code parameters (N and R), the number of nodes and the graph connections, an LDPC protograph code can be modified to achieve an improved asymptotic weight enumerator by replacing a group of LDPC variable nodes by a stronger component code of the same length and dimension. Several DGLDPC codes have been constructed based on these results and simulation results show that DGLDPC codes have improved performance in both waterfall region and error floor region compared with that of their LDPC and GLDPC counterparts.

In summary, the main contributions of this research include:

- Closed-form EXIT functions for plain shuffled BP and replica shuffled BP decoding algorithms.
- VLSI architecture for replica shuffled normalized BP-based decoding of quasi-cyclic LDPC codes.
- Introduction of DGLDPC codes.
- Closed-form EXIT functions for specific variable component codes of DGLDPC codes.
- Differential evolution for threshold optimization of DGLDPC codes.
- Ensemble weight enumerators for protograph-based DGLDPC codes.

We propose the following possible future works:

- Doubly generalized irregular repeat-accumulate (IRA) codes.
- Density evolution for DGLDPC codes.
- Non-binary DGLDPC codes.

Bibliography

- [1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379-423 (Part 1); pp. 623-656 (Part 2), Jul. 1948.
- [2] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147-160, Apr. 1950.
- [3] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection," *IRE Trans. Electron. Comput.*, vol. 3, pp. 6-12, Sept. 1954.
- [4] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147-156, Sept. 1959.
- [5] R. C. Bose and D. X. Ray-Chaudhuri, "On a class of error-correcting binary group codes," *Inform. and Control*, vol. 3, pp. 68-79, Mar. 1960.
- [6] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Indust. Appl. Math.*, vol. 8, pp. 300-304, 1960.
- [7] E. R. Berlekamp, "On decoding binary Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. 11, pp. 577-580, Oct. 1965.
- [8] E. R. Berlekamp, *Algebraic Coding Theory*. McGraw-Hill, New York, 1968.
- [9] J. L. Massey, "Step-by-step decoding of the Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. 11, pp. 580-585, Oct. 1965.
- [10] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. 15, pp. 122-127, Jan. 1969.

- [11] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IRE Trans. Inform. Theory*, vol. 4, pp. 38-49, Sept. 1954.
- [12] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284-287, Mar. 1974.
- [13] G. D. Forney Jr., "Generalized minimum distance decoding," *IEEE Trans. Inform. Theory*, vol. 12, pp. 125-131, Apr. 1966.
- [14] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theory*, vol. 18, pp. 170-182, Jan. 1972.
- [15] M. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1379-1396, Sept. 1995.
- [16] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo codes," *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064-1070.
- [17] C. Berrou and A. Glavieux, "Near-optimum error-correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261-1271, Oct. 1996.
- [18] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: M.I.T. Press, 1963.
- [19] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.* vol. 32, pp. 1645-1646, Aug. 1996.
- [20] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.
- [21] R. J. McEliece, D. J. C. MacKay and J. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE J. on Selected Areas in Commun.*, vol. 16, pp. 140-151, Feb. 1998.

- [22] T. Richardson, A. Shokrollahi and R. Urbanke, "Design of capacity-approaching irregular low-density parity check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619-637, Feb. 2001.
- [23] S. Chung, G. D. Forney Jr., T. Richardson and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon Limit," *IEEE Commun. Lett.*, vol. 5, pp. 58-60, Feb. 2001.
- [24] J. Zhang and M. Fossorier, "Shuffled belief propagation decoding," *IEEE Trans. Commun.*, vol. 53, pp. 209-213, Feb. 2005.
- [25] H. Kfir and I. Kanter, "Parallel versus sequential updating for belief propagation decoding," *Physica A*, vol. 330, pp. 259-270, 2003.
- [26] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharam, "High throughput low-density parity-check decoder architectures," *Proc. IEEE Global Telecommun. Conf.*, San Antonio, TX, Nov. 2001, pp. 3019-3024.
- [27] M. M. Mansour and N. R. Shanbhag, "Turbo decoder architecture for low-density parity-check codes" *Proc. IEEE Global Telecommun. Conf.*, Taipei, Taiwan, R. O. C., Nov. 2002, pp. 1383-1388.
- [28] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976-996, Dec. 2003.
- [29] J. Zhang, Y. Wang, M. Fossorier, and J. S. Jonathan, "Iterative decoding with replicas," *IEEE Trans. Inform. Theory*, vol. 53, pp. 1644-1663, May 2007.
- [30] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599-618, Feb. 2001.
- [31] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes," *Proc. 2001 IEEE Int. Symp. Circuits Syst.*, Sydney, Australia, May 2001, pp. 742-745.

- [32] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magn.*, vol. 37, no. 2, pp. 748-755, Mar. 2001.
- [33] T. Zhang, *Efficient VLSI architectures for error-correcting coding*, Ph.D. dissertation, 2002.
- [34] E. Boutillon, J. Castura, and F. R. Kschischang, "Decoder-first code design," *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, Brest, France, Sept. 2000, pp. 459-462.
- [35] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533-547, Sept. 1981.
- [36] M. Lentmaier and K. S. Zigangirov, "On generalized low-density parity-check codes based on Hamming component codes," *IEEE Commun. Lett.*, vol. 3, pp. 248-250, Aug. 1999.
- [37] J. Boutros, O. Pothier, and G. Zemor, "Generalized low-density (Tanner) codes," *Proc. IEEE Int. Conf. Commun.*, Vancouver, Canada, vol. 1, Jun. 1999, pp. 441-445.
- [38] N. Miladinovic and M. Fossorier, "Generalized LDPC codes with Reed-Solomon and BCH codes as component codes for binary channels," *Proc. IEEE Global Telecommun. Conf.*, St. Louis, MO, Nov. 2005, pp. 1239-1244.
- [39] G. Yue, L. Ping, and X. Wang, "Low-rate generalized low-density parity-check codes with Hadamard constraints," *Proc. IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, Sept. 2005, pp. 1377-1381.
- [40] J. Chen and R. M. Tanner, "A hybrid coding scheme for the Gilbert-Elliott channel," *Proc. 42th Allerton Conf. on Commun., Control and Computing*, Monticello, IL, Sept. 2004.
- [41] S. Abu-Surra, G. Liva, and W. Ryan, "Low-floor Tanner codes via Hamming-node or RSCC-node doping," *Proc. Applied Algebra, Algebraic Algorithms and Error Correcting Codes-16 Conf.*, Las Vegas, NV, Feb. 2006.

- [42] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585-598, Feb. 2001.
- [43] X. Hu, E. Eleftheriou, and D. Arnold, "Progressive edge-growth Tanner graphs," *Proc. IEEE Global Telecommun. Conf.*, San Antonio, TX, Nov. 2001, pp. 995-1001.
- [44] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242-1247, Aug. 2004.
- [45] J. Chen, R. M. Tanner, J. Zhang, and M. Fossorier, "Construction of irregular LDPC codes by quasi-cyclic extension", *IEEE Trans. Inform. Theory*, vol. 53, no. 4, pp. 1479-1483, Apr. 2007.
- [46] L. D. Rudolph, "A class of majority logic decodable codes," *IEEE Trans. Inform. Theory*, vol. 13, pp. 305-307, Apr. 1967.
- [47] Y. Kou, S. Lin and M. Fossorier, "Low density parity check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, Vol. 47, pp. 2711-2736, Nov. 2001.
- [48] J. Denes and A. D. Keedwell, *Latin Squares and their Applications*. New York: Academic Press, 1974.
- [49] J. L. Fan, "Array codes as low-density parity-check codes," *Proc. 2nd Int. Symp. Turbo Codes*, Brest, France, Sept. 2000, pp. 545-546.
- [50] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," *Proc. ISTA*, Ambleside, England, 2001.
- [51] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices", *IEEE Trans. Inform. Theory*, vol. 50, no. 8, pp. 1788-1793, Aug. 2004.
- [52] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. 2nd ed., Upper Saddle River, NJ: Prentice-Hall, 2004.

- [53] L. Sakai, W. Matsumoto, and H. Yoshida, "Reduced complexity decoding based on approximation of update function for low-density parity-check codes," *IEICE Trans. Fund. Electron. Commun. Comput. Sci. (Japanese Edition)*, vol. J90-A no. 2 pp. 83-91, Feb. 2007.
- [54] J. Xu, L. Chen, I. Djurdjevic, S. Lin, and K. Abdel-Ghaffar, "Construction of regular and irregular LDPC codes: geometry decomposition and masking," *IEEE Trans. Inform. Theory*, vol. 53, no. 1, pp. 121-134, Jan. 2007.
- [55] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
- [56] M. Fossorier, M. Mihaljević and H. Imai, "Reduced complexity iterative decoding of low density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673-680, May 1999.
- [57] N. Wiberg, "Codes and decoding on general graphs," Linköping Studies in Sci. and Technol., dissertations no. 440, Linköping, Sweden, 1996.
- [58] S. Chung, *On the construction of some capacity-approaching coding schemes*. Ph.D. dissertation, M.I.T., Sept. 2000.
- [59] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 50, No. 3, pp. 406-414, Mar. 2002.
- [60] J. Chen and M. Fossorier, "Density evolution for two improved BP-based decoding algorithms of LDPC codes," *IEEE Commun. Lett.*, vol. 6, no. 5, May 2002.
- [61] S. L. Howard, C. Schlegel, and V. C. Gaudet, "A degree-matched check node approximation for LDPC decoding," *Proc. IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, Sept. 2005, pp. 1131-1135.
- [62] J. Chen; A. Dholakia, E. Eleftheriou, M. Fossorier, and X-Y Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, pp. 1288-1299, Aug. 2005.

- [63] F. Guilloud, *Generic architecture for LDPC codes decoding*, Ph.D. thesis, ENST Paris, France, 2004.
- [64] J. Zhang, M. Fossorier, D. Gu, and J. Zhang, "Two-dimensional correction for min-sum decoding of irregular LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 3, Mar. 2006.
- [65] IEEE Std 802.16e, *Air interface for fixed and mobile broadband wireless access systems*, [Online]. Available: <http://standards.ieee.org/getieee802/download/802.16e-2005.pdf>.
- [66] LDPCopt, *An efficient optimization program for LDPC codes*, [Online]. Available: <http://lthcwww.epfl.ch/research/ldpcopt/>.
- [67] Draft DVB-S2 Standard, [Online]. Available: <http://www.dvb.org>.
- [68] S. ten Brink, "Convergence of iterative decoding," *Electron Lett.*, vol. 35, no. 10, pp. 806-808, May 1999.
- [69] S. ten Brink, "Iterative decoding for multicode CDMA," *Proc. IEEE VTC*, May 1999, pp. 1876-1880.
- [70] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727-1737, Oct. 2001.
- [71] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, pp. 670-678, Apr. 2004.
- [72] S. ten Brink and G. Kramer, "Design of repeat-accumulate codes for iterative detection and decoding", *IEEE Trans. Signal Processing*, vol. 51, no. 11, Nov. 2003.
- [73] M. Tüchler, S. ten Brink, and J. Hagenauer, "Measures for tracing convergence of iterative decoding algorithms," *Proc. 4th IEEE/ITG Conf. on Source and Channel Coding*, Berlin, Germany, Jan. 2002, pp. 53-60.

- [74] M. Tüchler and J. Hagenauer, "EXIT charts of irregular codes," *Proc. Inform. Sci. and Syst.*, Princeton, NJ, Mar. 2002, pp. 748-753.
- [75] T. Richardson, A. Shokrollahi and R. Urbanke, "Design of provably good low-density parity-check codes," *Proc. IEEE Int. Symp. Inform. Theory*, Sorrento, Italy, Jun. 2000, p. 199.
- [76] E. Biglieri, *Coding for Wireless Channels*. New York: Springer-Verlag, 2005.
- [77] S. Dolinar, "Design and iterative decoding of networks of many small codes," *Proc. IEEE Int. Symp. Inform. Theory*, Yokohama, Japan, Jun. 2003, p. 381.
- [78] I. Land, P. A. Hoeher, and J. B. Huber, "Bounds on information combining for parity-check equations," *Int. Zurich Seminar*, Zurich, Switzerland, Feb. 2004, pp. 68-71.
- [79] I. Land, P. A. Hoeher, and J. B. Huber, "Analytical derivation of EXIT charts for simple block codes and for LDPC codes using information combining," *Proc. of the 12th European Signal Processing Conf. (EUSIPCO04)*, Vienna, Austria, Sept. 2004, pp. 1561-1564.
- [80] I. Land, S. Huettinger, P. A. Hoeher, and J. B. Huber, "Bounds on information combining," *IEEE Trans. Inform. Theory*, vol. 51, pp. 612-619, Feb. 2005.
- [81] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," *Proc. 2nd Int. Symp. Turbo Codes Related Topics*, Brest, France, Sept. 2000, pp. 1-8.
- [82] E. Sharon, A. Ashikhmin, and S. Litsyn, "EXIT functions for binary input memory-less symmetric channels," *IEEE Trans. Commun.*, vol. 54, pp. 1207-1214, Jul. 2006.
- [83] J. Thorpe, "Low density parity check (LDPC) codes constructed from protographs," *JPL IPN Progress Report 42-154*, Aug. 2003.
- [84] D. Divsalar, "Ensemble weight enumerators for protograph LDPC codes," *Proc. IEEE Int. Symp. Inform. Theory*, Seattle, WA, Jul. 2006, pp. 1554-1558.

- [85] S. Abu-Surra, W. E. Ryan, and D. Divsalar, "Ensemble weight enumerators for protograph-based generalized LDPC codes," *Proc. Appl. Algebra, UCSD ITA Workshop*, San Diego, CA, Jan. 2007.
- [86] D. Divsalar, C. Jones, S. Dolinar, and J. Thorpe, "Protograph based LDPC codes with minimum distance linearly growing with block size," *Proc. IEEE Global Telecommun. Conf.*, St. Louis, MO, Nov. 2005, pp. 1152-1156.
- [87] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 909-926, May 1998.
- [88] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, vol. 11, pp. 341-359, Dec. 1997.
- [89] A. Shokrollahi and R. Storn, "Design of efficient erasure codes with differential evolution," *Proc. 2000 IEEE Int. Symp. Inform. Theory*, Sorrento, Italy, Jun. 2000, p. 5.
- [90] T. J. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619-637, Feb. 2001.
- [91] J. Hou, P. H. Siegel, and L. B. Milstein, "Performance analysis and code optimization of low-density parity-check codes on Rayleigh fading channels," *IEEE J. on Selected Areas in Commun.*, vol. 19, no. 5, pp. 924-934, May 2001.
- [92] E. Paolini, M. Fossorier, and M. Chiani, "Doubly generalized LDPC codes for the binary erasure channel," *Proc. 44th Allerton Conf. on Commun., Control, and Comput.*, Urbana, IL, Sept. 2006.
- [93] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: model and erasure channel properties," *IEEE Trans. Inform. Theory*, vol. 50, pp. 2657-2673, Nov. 2004.

- [94] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429-445, Mar. 1996.
- [95] C. Poulliat, M. Fossorier, and D. Declercq, "Using binary images of non binary LDPC codes to improve overall performance," *Proc. of the 4th Int. Symp. on Turbo Codes and Related Topics*, Munich, Germany, Apr. 2006.
- [96] B. Bangerter *et al.*, "High-throughput wireless LAN air interface," *J. Intel Technol.*, vol. 7, pp. 47-57, Aug. 2003.
- [97] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE Trans. VLSI Syst.*, vol. 41, no. 3, pp. 684-698, Mar. 2006.
- [98] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," *Proc. IEEE Workshop Signal Processing Syst.*, Austin, USA, Oct. 2004, pp. 107-112.
- [99] X.-Y. Hu, M. Fossorier, and E. Eleftheriou, "Approximate algorithms for computing the minimum distance of low-density parity-check codes," *Proc. 2004 IEEE Int. Symp. Inform. Theory*, Chicago, IL, Jun. 2004, p. 475.
- [100] X.-Y. Hu, M. Fossorier, and E. Eleftheriou, "On the computation of the minimum distance of low-density parity-check codes," *Proc. 2004 IEEE Int. Conf. Commun.*, Paris, France, Jun. 2004, pp. 767-771.