Bootstrap Algorithms and Reliability Based Schedule for Decoding Low-Density Parity-Check Codes

By

Ahmed Galal Nouh, B.Eng.

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

**Master of Engineering
in Electrical Engineering**

Ottawa-Carleton Institute For Electrical and Computer Engineering
Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
Ottawa, Ontario
August, 2002

© Copyright
Ahmed Nouh, 2002

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

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

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

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

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

0-612-79772-4

Canada

The undersigned hereby recommend to the faculty of Graduate Studies and Research acceptance of the thesis, "Bootstrap Algorithms and Reliability Based Schedule for Decoding Low-Density Parity-Check Codes", submitted by Ahmed Galal Nouh, B.Eng. in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering.

Professor Amir H. Banihashemi,
Thesis Supervisor

Professor Rafik Goubran,
Chair, Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
Ottawa, Ontario
August, 2002

Abstract

Error control coding is one of the challenging tasks in communications. Designing a decoding algorithm, that has good error performance and yet easy to implement, is challenging. In this work, we modify the existing decoding algorithms and design new ones to enhance the error performance along with alleviating the decoding complexity, for hard and/or hard-soft algorithms for low-density parity-check (LDPC) codes.

Three generic methods are proposed for modifying current algorithms: *i*) Reliability based schedule; *ii*) Hybrid decoding; and *iii*) Bootstrap decoding. A reliability based schedule prevents the unreliable information from propagating through the graph. In hybrid decoding, switching among different soft/hard message-passing algorithms is performed throughout the iteration process with the aim of improving the performance and/or the complexity of decoding. Bootstrap decoding employs both ideas to achieve better performance at lower complexity. Decoding is initiated by first erasing a number of less reliable bits. New values and reliabilities are then assigned to erasure bits by passing messages from non-erasure bits through the reliable check equations. The error performance is dramatically improved. The added complexity, for the initial bootstrap step, is negligible and the overall complexity is reduced considerably.

Acknowledgements

After praising God, I would like to appreciate the valuable efforts of Dr. Amir H. Banihashemi in supervising this research. I would also like to sincerely thank Dr. Banihashemi for his ongoing support and encouragement. In addition, I thankfully acknowledge the financial support provided by Zarlink Semiconductor Corp. (formerly, Mitel Semiconductor Corp.) and Carleton University.

Moreover, I am especially grateful to my professors at Ain Shams University, Cairo, Egypt, Carleton University, Ottawa, Canada and University of Ottawa, Ottawa, Canada, for the strong technical background that I learned from my undergraduate and graduate courses. I would like to thank Dr. Marc Fossorier, Dr. Yu Kou, Dr. David MacKay and Dr. Thomas Richardson for providing me with the parity-check matrices of the LDPC codes.

Finally, I would like to thank my family, especially my father, for their support and understanding and their sacrifices throughout the years of this research. I would like to truthfully thank my wife for her great sacrifices and I would like to dedicate this thesis to my little *Maryam*.

Contents

ABSTRACT	III
ACKNOWLEDGEMENTS	IV
CONTENTS	V
LIST OF TABLES	VIII
LIST OF FIGURES	X
LIST OF SYMBOLS	XIII
INTRODUCTION	1
1.1. Thesis Motivation.....	1
1.2. Problem Definition.....	4
1.3. Thesis Contributions	6
1.4. Thesis Outline	9
LDPC CODES.....	10
2.1 Principles and Constructions	10
2.1.1 Tanner Graph (TG).....	11
Constructions of	13
2.1.2 LDPC Codes	13
2.1.3 Finite Geometry LDPC Codes	13
2.2 Iterative Decoding Algorithms.....	14

2.2.1	Belief-Propagation (BP) Algorithm.....	17
2.2.2	Min-Sum (MS) Algorithm.....	18
2.2.3	Weighted Bit-Flipping (WBF) Algorithm	19
2.2.4	Gallager's Algorithm A (GA).....	21
2.2.5	Sipser-Spielman (SS) Algorithm	22
2.3	Decoding Schedules	23
2.3.1	Graph Independent Schedules	23
2.3.2	Graph-Based Schedules	24
RELIABILITY BASED SCHEDULE.....		26
3.1	Reliability Based Schedule (RBS)	26
3.2	RBS Belief-Propagation (RBS-BP) Algorithm.....	29
3.3	RBS Min-Sum (RBS-MS) Algorithm.....	30
3.4	RBS Gallager's Algorithm A (RBS-GA).....	30
3.5	RBS Sipser-Spielman (RBS-SS) Algorithm	31
3.6	Simulation Results	32
3.7	Discussion	38
HYBRID (MULTISTAGE) DECODING.....		40
4.1	Description	40
4.2	Simulation Results	42
4.3	Discussion	47
BOOTSTRAP DECODING ALGORITHMS		48
5.1	Bootstrap (BS) Decoding	48
5.2	BS Weighted Bit-Flipping Algorithm.....	51
5.3	BS Gallager's Algorithm A.....	51
5.4	BS Sipser-Spielman Algorithm.....	51
5.5	Simulation Results	52
5.6	Comparison among RBS, HB and BS Schemes.....	57
5.7	Discussion	62
CONCLUDING REMARKS.....		64
APPENDIX A: ACRONYMS.....		66
APPENDIX B: LDPC CODES		68
B.1	(273, 191) Regular Code; Column Weight = 3	68
B.2	(273, 191) Regular Code; Column Weight = 4	69
B.3	(1008, 504) Regular LDPC Code.....	69
B.4	(2640, 1320) Regular LDPC Code.....	69

B.5	(8000, 4000) Regular LDPC Code.....	69
B.6	(273, 191) PG LDPC Code	70
B.7	(1057, 813) PG LDPC Code	70
B.8	(1000, 500) Irregular LDPC Code.....	70
APPENDIX C: MORE RESULTS.....		71
C.1	Reliability Based Schedule.....	72
C.2	Hybrid (Multistage) Decoding	84
C.3	Bootstrap Decoding Algorithms.....	104
REFERENCES		130

List of Tables

Table 3.6.1 Statistics of the Number of Decoding Iterations for B.8 Code for Converged Cases for BP, MS, GA, RBS-BP, RBS-MS and RBS-GA Algorithms	34
Table 3.6.2 The Percentage of Wrongly-Converged Code-Words for B.8 Code for BP, MS, RBS-BP and RBS-MS Algorithms.....	35
Table 4.2.1 Statistics on the Complexity of BP, MS, GA, (iBP, GA) and (iMS, GA) Decoding Algorithms for B.8 Code.	44
Table 4.2.2 The Percentage of Wrongly-Converged Code-Words for B.8 Code for BP and MS Algorithms.....	45
Table 5.5.1 Statistics of the Number of Decoding Iterations for B.8 Code for Converged Cases for WBF, GA, BWBF and BGA-MS Algorithms	54
Table 5.5.2 Statistics of the Number of Decoding Iterations for B.8 Code for Converged Cases for GA and BGA-MR Algorithms	54
Table C.1.1 Statistics of the Number of Decoding Iterations for B.1 Code for Converged Cases for MS, GA, RBS-MS, RBS-GA and RBS-SS Algorithms.....	76
Table C.1.2 The Percentage of Wrongly-Converged Code-Words for B.1 Code for MS, SS, RBS-MS and RBS-SS Algorithms	76
Table C.1.3 Statistics of the Number of Decoding Iterations for B.6 Code for Converged Cases for BP, MS, GA, SS, RBS-BP, RBS-MS RBS-GA and RBS-SS Algorithms	82
Table C.1.4 The Percentage of Wrongly-Converged Code-Words for B.6 Code for BP, MS, SS, RBS-BP, RBS-MS and RBS-SS Algorithms.....	83
Table C.2.1 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.1 Code.	86
Table C.2.2 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.2 Code.	90
Table C.2.3 Statistics on the Complexity of BP, MS, GA, (iBP, GA) and (iMS, GA) Decoding Algorithms for B.5 Code.	94
Table C.2.4 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.6 Code.	98
Table C.2.5 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.7 Code.	102

Table C.3.1 Statistics of the Number of Decoding Iterations for B.1 Code for Converged Cases for WBF, GA, SS, BWBF, BGA-MS and BSS Algorithms	109
Table C.3.2 Statistics of the Number of Decoding Iterations for B.1 Code for Converged Cases for GA and BGA-MR Algorithms	109
Table C.3.3 Statistics of the Number of Decoding Iterations for B.2 Code for Converged Cases for WBF, GA, BWBF and BGA-MS Algorithms	114
Table C.3.4 Statistics of the Number of Decoding Iterations for B.2 Code for Converged Cases for GA and BGA-MR Algorithms	114
Table C.3.5 Statistics of the Number of Decoding Iterations for B.3 Code for Converged Cases for GA and BGA-MS Algorithms.....	118
Table C.3.6 Statistics of the Number of Decoding Iterations for B.3 Code for Converged Cases for GA and BGA-MR Algorithms	118
Table C.3.7 Statistics of the Number of Decoding Iterations for B.4 Code for Converged Cases for GA and BGA-MS Algorithms.....	122
Table C.3.8 Statistics of the Number of Decoding Iterations for B.4 Code for Converged Cases for GA and BGA-MR Algorithms	122
Table C.3.9 Statistics of the Number of Decoding Iterations for B.5 Code for Converged Cases for WBF and BWBF Algorithms.....	124
Table C.3.10 Statistics of the Number of Decoding Iterations for B.6 Code for Converged Cases for WBF, SS, BWBF and BSS Algorithms	127
Table C.3.11 Statistics of the Number of Decoding Iterations for B.7 Code for Converged Cases for WBF and BWBF Algorithms.....	129

List of Figures

Figure 2.1.1 A (7, 3) Hamming Code, Where the Blank Nodes and the Crossed Nodes Represent the Columns and Rows of the Parity-Check Matrix H, Respectively	12
Figure 3.6.1 BER (—) and MER (---) Curves for B.8 Code Decoded by BP, MS, GA, RBS-BP, RBS-MS and RBS-GA Algorithms.....	33
Figure 3.6.2 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by BP (—) and RBS-BP (---) Algorithms.	35
Figure 3.6.3 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by MS (—) and RBS-MS (---) Algorithms.	36
Figure 3.6.4 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by GA (—) and RBS-GA (---) Algorithms.....	37
Figure 4.2.1 BER Curves for B.8 Code Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.....	43
Figure 4.2.2 MER Curves for B.8 Code Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.....	46
Figure 5.5.1 BER Curves for B.8 Code Decoded by BP, MS, WBF, GA, BWBF, BGA-MS and BGA-MR Algorithms.	53
Figure 5.5.2 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by WBF (—) and BWBF (---) Algorithms.	55
Figure 5.5.3 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by GA (—) and BGA-MS (---) Algorithms.....	56
Figure 5.5.4 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by GA (—) and BGA-MR (---) Algorithms.	57
Figure 5.6.1 BER Curves for the B.8 Code Decoded by BP, MS, GA, RBS-GA, (iBP, GA), (iMS, GA) , BGA-MS and BGA-MR Algorithms.....	58
Figure 5.6.2 MER Curves for B.8 Code Decoded by BP, MS, GA, RBS-GA, (iBP, GA), (iMS, GA) , BGA-MS and BGA-MR Algorithms.	59
Figure 5.6.3 BER Curves for B.6 Code Decoded by BP, MS, WBF, (iBP, WBF), (iMS, WBF) and BWBF Algorithms.....	61
Figure 5.6.4 MER Curves for B.6 Code Decoded by BP, MS, WBF, (iBP, WBF), (iMS, WBF) and BWBF Algorithms.....	62

Figure C.1.1 BER and MER Curves for B.1 Code, Decoded by MS, GA, SS, RBS-MS, RBS-GA and RBS-SS Algorithms.....	72
Figure C.1.2 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by MS (—) and RBS-MS (---) Algorithms.	73
Figure C.1.3 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by GA (—) and RBS-GA (---) Algorithms.....	74
Figure C.1.4 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by SS (—) and RBS-SS (---) Algorithms.....	75
Figure C.1.5 BER and MER Curves for B.6 Code, Decoded by BP, MS, GA, SS, RBS-BP, RBS-MS, RBS-GA and RBS-SS Algorithms.	77
Figure C.1.6 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by BP (—) and RBS-BP (---) Algorithms.	78
Figure C.1.7 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by MS (—) and RBS-MS (---) Algorithms.	79
Figure C.1.8 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by GA (—) and RBS-GA (---) Algorithms.....	80
Figure C.1.9 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by SS (—) and RBS-SS (---) Algorithms.....	81
Figure C.2.1 BER Curves for B.1 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	84
Figure C.2.2 MER Curves for B.1 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	85
Figure C.2.3 BER Curves for B.2 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	88
Figure C.2.4 MER Curves for B.2 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	89
Figure C.2.5 BER Curves for B.5 Code, Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.....	92
Figure C.2.6 MER Curves for B.5 code, Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.....	93
Figure C.2.7 BER Curves for B.6 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	96
Figure C.2.8 MER Curves for B.6 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	97
Figure C.2.9 BER Curves for B.7 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	100
Figure C.2.10 MER Curves for B.7 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.	101
Figure C.3.1 BER and MER Curves for B.1 Code Decoded by BP, MS, WBF, GA, SS, BWBF, BGA-MS, BGA-MR and BSS Algorithms.....	104
Figure C.3.2 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by WBF (—) and BWBF (---) Algorithms.	105

Figure C.3.3 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by GA (—) and BGA-MS (---) Algorithms.....	106
Figure C.3.4 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by GA (—) and BGA-MR (---) Algorithms.	107
Figure C.3.5 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by SS (—) and BSS (---) Algorithms.	108
Figure C.3.6 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by BP, MS, WBF, GA, BWBF, BGA-MS and BGA-MR Algorithms.....	110
Figure C.3.7 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by WBF (—) and BWBF (---) Algorithms.	111
Figure C.3.8 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by GA (—) and BGA-MS (---) Algorithms.....	112
Figure C.3.9 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by GA (—) and BGA-MR (---) Algorithms.	113
Figure C.3.10 BER and MER Curves for B.3 Code Decoded by BP, GA, BGA-MS and BGA-MR Algorithms.....	115
Figure C.3.11 Effect of α on BER Given for Selected SNRs for B.3 Code Decoded by GA (—) and BGA-MS (---) Algorithms.....	116
Figure C.3.12 Effect of α on BER Given for Selected SNRs for B.3 Code Decoded by GA (—) and BGA-MR (---) Algorithms.	117
Figure C.3.13 BER and MER Curves for B.4 Code Decoded by GA, BGA-MS and BGA-MR Algorithms.	119
Figure C.3.14 Effect of α on BER Given for Selected SNRs for B.4 Code Decoded by GA (—) and BGA-MS (---) Algorithms.....	120
Figure C.3.15 Effect of α on BER Given for Selected SNRs for B.4 Code Decoded by GA (—) and BGA-MR (---) Algorithms.	121
Figure C.3.16 BER and MER Curves for B.5 Code Decoded by BP, MS, WBF and BWBF Algorithms.	123
Figure C.3.17 Effect of α on BER Given for Selected SNRs for B.5 Code Decoded by WBF (—) and BWBF (---) Algorithms.	124
Figure C.3.18 BER and MER Curves for B.6 Code Decoded by BP, MS, WBF, SS, BWBF and BSS Algorithms.	125
Figure C.3.19 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by WBF (—) and BWBF (---) Algorithms.	126
Figure C.3.20 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by SS (—) and BSS (---) Algorithms.	127
Figure C.3.21 BER and MER Curves for B.7 Code Decoded by BP, MS, WBF and BWBF Algorithms.	128
Figure C.3.22 Effect of α on BER Given for Selected SNRs for B.7 Code Decoded by WBF (—) and BWBF (---) Algorithms.	129

List of Symbols

$\mathcal{C}$	codebook of a code
c_i	a check-node represented by the i^{th} row in a parity-check matrix $\mathbf{H}$
D_s	flipping decision parameter for WBF algorithm
$\mathbf{E}_b$	average energy per information bit
$\mathbf{H}$	parity-check matrix
h_i	the i^{th} row in a parity-check matrix
i	index used for a row (check-node) in a parity-check matrix of a code
j	index used for a column (bit-node) in a parity-check matrix of a code
k	code dimension
$m_j^{(0)}$	initial messages generated at bit-nodes v_j
$m_{p \rightarrow s}$	message sent from check-node p to bit-node s
$m_{s \rightarrow p}$	message sent from bit-node s to check-node p
n	block length of a code
$\underline{\mathbf{n}}$	Gaussian random vector (noise) with independent and identically distributed elements
$N(p)$	the set of all neighboring bit-nodes of p

$N(s)$	the set of all neighboring check-nodes of s
$N(p)\backslash s$	the set of all neighboring bit-nodes of p other than s
$N(s)\backslash p$	the set of all neighboring check-nodes of s other than p
$N_0/2$	power spectral density of AWGN
P_1 (P_{-1})	is the a posterior probability of having a ‘0’ (‘1’)
$\mathbf{R}$	code rate
$\mathbf{r}$	received vector from a channel
$\hat{s}$	hard-decision of the estimated value at bit-node s
(s, p)	represents a neighboring pair of bit and check-nodes (v_j, c_i), respectively
U_s	the number of the unsatisfied check-nodes in the neighboring of bit-node s
$\mathbf{v}$	a code-word in a codebook $\mathbf{C}$
v_j	a bit-node in a code or the j^{th} bit in code-word’s vector $\mathbf{v}$
$\mathbf{x}$	transmitted vector over a binary input channel
y_p	the absolute minimum received bit of all the neighboring bit-nodes of p
$\mathbf{z}$	the vector consisting of the hard-decision estimates for a received vector $\mathbf{r}$
z_j	the j^{th} term in $\mathbf{z}$
x_j	bipolar signal $\{\pm 1\}$
α	bit-nodes reliability threshold for RBS and BS schemes
β	a threshold parameter defined for GB algorithm
δ	a value proportional to the girth passing through a bit or a check-node
γ	check-nodes reliability threshold for RBS and BS schemes
σ^2	noise variance

$\psi(s)$	the set of all the neighboring reliable check-nodes of s
$\psi(s)\backslash p$	the set of all the neighboring reliable check-nodes of s other than p

Chapter One

Introduction

1.1. Thesis Motivation

Gallager introduced low-density parity-check (LDPC) codes in the early sixties [15]. They were however, soon forgotten due to limited computational resources at the time which made the corresponding iterative decoding algorithms look impractical. It was not until the invention of turbo codes [8] that researchers realized the phenomenal strength of iterative coding schemes, and the attractive performance/complexity tradeoff that they offer.

In particular, for the past few years, there has been a great amount of research devoted to LDPC codes and the performance and complexity of associated iterative decoding algorithms [2]. Furthermore, LDPC codes [15], decoded by iterative message-

passing algorithms have proved to be a viable alternative for high-performance low-complexity coding schemes (for a list of references, see the related papers in [1], [2], [3], [4], and the references therein).

There have been a wide variety of iterative algorithms, ranging from full hard-decision to a spectrum of soft-decision algorithms, proposed for the decoding of LDPC codes. Each algorithm provides a different performance/complexity tradeoff. The best performing algorithm, known to be “belief-propagation” (BP) or “sum-product” [14], is considered to be the most complex to implement. Another well-known soft iterative decoding algorithm is “min-sum” (MS). Although, in general, the error performance of MS is a few tenths of a dB inferior to that of BP, it is much simpler to implement [5], [37].

There are also a variety of hard message-passing (bit-flipping) algorithms including the well-known Gallager’s algorithms A and B (GA and GB, respectively) [15], [31]. The performances, of these algorithms, are inferior to that of soft algorithms, but they are much simpler to implement and in some cases converge much faster [38].

Additionally, among the different iterative message-passing algorithms for the decoding of LDPC codes, weighted bit-flipping (WBF) [16] appears to offer a good tradeoff between error performance and decoding complexity. It combines the simplicity

of bit-flipping algorithms [15] with the improved performance due to including some measure of reliability (soft information) in the messages.

Interestingly, from the study of these different algorithms, we can see that the performance/complexity tradeoff is normally far away from linearity, i.e., adding high decoding complexity might result in a gain of only few tenths of a dB and on the contrary sacrificing some of the decoding complexity could lead to a huge degradation in error performance.

Also, from [26] and [36] it appears that incorporating schedules, to the decoding algorithm, plays a very important role in the performance and complexity of iterative decoding algorithms, particularly at short block lengths. As the complexity of the employed schedule increases, the performance in general improves [36], and there still seems to be a room for improvement.

As a result, in this thesis, we propose new schedules combined with hybrid steps, to efficiently use the current algorithms at lower complexity. The expected benefits of these new methodologies provided the motivation to study the feasibility of enhancing the current iterative algorithms and to design models that can be tailored to any of the suitable algorithms.

1.2. Problem Definition

The objective of this research is to introduce decoding schemes that achieve both better error performance and lower decoding complexity. In addition, it is favorable that the new schemes be generic, in the sense that they can be tailored to any suitable algorithm, to accomplish the expected objective. In order to achieve this goal, our research studied the dynamics of different decoding algorithms and ways of enhancing them.

Iterative decoding algorithms are naturally described using a graph of the code constructed based on the parity-check matrix $\mathbf{H}$. Such a graph is called a Tanner graph (TG) [35]. To describe TGs and the associated decoding algorithms, we assume that the codes are binary. A TG is a bipartite graph that contains two types of nodes, bit (symbol) nodes and check-nodes, corresponding to the columns and the rows of the parity-check matrix $\mathbf{H}$, respectively.

Fundamentally, every iterative algorithm passes messages back and forth, between bit-nodes and check-nodes, according to a schedule, referred to as “message-passing schedule”. The most popular schedule is the *flooding schedule*. In the flooding schedule, all the check-node operations are evaluated at the same time [14], and accordingly, the bit-nodes are updated simultaneously.

Nevertheless, it has been shown in [17], [26] and [36] that preventing to update some of the bit and/or check-nodes in each iteration can improve the performance noticeably. The principle idea here is to control the exchange of messages, between a node and its neighbors, based on its position within the TG of the code. However, employing different schedules to improve the performance of iterative decoding algorithms is a very recent development.

Besides, although the existing schedules might have the same average decoding complexity, the overall complexity is increased due to the added hardware [36]. In this dissertation, we propose the idea of adding a schedule that has both, lowered overall complexity and graph independence [29]. The purpose of this schedule is to prevent the unreliable information from propagating within the graph. This methodology is called ***Reliability Based Schedule***.

Moreover, we introduce the idea of ***Hybrid (Multistage) Decoding*** [6], [7], where the decoder switches among different soft/hard message-passing algorithms throughout the iteration process with the aim of improving the performance and/or the complexity of decoding. As an example of a two-stage version, the decoding starts with a few iterations of a soft message-passing algorithm, such as BP or MS, and then switches to a less complex hard message-passing algorithm, such as GA.

Incorporating the idea of reliability based schedule along with hybrid decoding, we propose the idea of ***Bootstrap Decoding*** [27], [28]. In the first iteration, the algorithm performs a reliability based scheduled iteration using a soft algorithm, such as BP or MS, then switches to a less complex algorithm, such as a hard/soft or hard algorithm, till the end of the decoding process.

All the new schemes have been tested with different algorithms. For reliability based schedule and bootstrap decoding, large improvements in both error performance and decoding complexity are observed, while the hybrid decoding shows the expected performance/complexity tradeoff. Comparing bootstrap decoding to hybrid decoding, there were some cases where the bootstrap scheme, yet based on two hard-decision algorithms, has outperformed the hybrid decoding, that is more complex.

1.3. Thesis Contributions

The major contribution of the research presented in this thesis is to introduce a set of decoding algorithms¹ that generally achieve both, better error performance and lower decoding complexity. Furthermore, the architecture of these bootstrapped decoding schemes incorporates in its design two major contributions; *i*) reliability based schedule; and *ii*) hybrid decoding, each has a set of supporting algorithms.

¹ The contributions of this thesis have been presented partly in [6], [7], [27], [28], [29].

The following is a summary of the major and minor contributions of the research reported in this dissertation:

- ***Reliability Based Schedule***

The reliability based schedule, proposed and studied in this thesis, introduces and incorporates in its methodology several new ideas [29]. Not only it prevents the unreliable bits from propagating their information through the graph, but also it passes reliable extrinsic information to them. Decoding is initiated by first identifying a number of less reliable bits to be unreliable. More reliable extrinsic information is delivered to unreliable bits by passing messages from reliable bits through the reliable check equations. Different from existing schedules [26], [36], that are graph dependent, two types of reliability schedule are proposed, one of which is graph independent. Our results show that the schedule performs well only with hard and/or hard-soft iterative algorithms and only decrease the average number of iterations required for convergence for soft algorithms. However, in principle, it can be applied to any iterative message-passing algorithm, and for other families of codes.

- ***Hybrid Decoding***

The main proposal is to switch among different soft/hard message-passing algorithms throughout the iteration process [6], [7]. As one possible manifestation of a two-stage

version, the decoding starts with a few iterations of a soft message-passing algorithm and then switches to a less complex hard message-passing algorithm. Depending on the number of iterations for the first decoding stage, and the convergence properties of the two algorithms, different performance/complexity tradeoffs, which in many cases are more attractive than those of the individual algorithms, are achievable. To study the performance and convergence properties, two sets of hybrid iterative algorithms were implemented.

- ***Bootstrap Decoding***

Bootstrap decoding [27], [28] combines the idea of reliability based scheduling and the hybrid decoding, along with the bootstrapped algorithm. Such an alteration would impact the first iteration in the original decoding process.

In bootstrap decoding, a two-stage hybrid algorithm is reliability based scheduled at the first iteration. Decoding is initiated by first erasing a number of less reliable bits. New values and reliabilities are then assigned to erasure bits by passing messages from non-erasure bits through the reliable check equations.

The results show great improvement in both bit error rate and message error rate. The added complexity of the bootstrap step is negligible and, in some cases, the overall complexity is lowered by more than a factor of two.

1.4. Thesis Outline

The contents of this thesis are organized as follows:

In Chapter 2, a general review of the LDPC codes is provided. More details are given on the TG of the code and the different types of LDPC codes. Iterative decoding algorithms are explained next. Then existing decoding schedules are reviewed.

Chapters 3 and 4 contain the thesis contributions to the reliability based scheduling and hybrid decoding, respectively. The implementation of these algorithms along with simulation results are also presented in these chapters, followed by discussions.

In chapter 5, bootstrap decoding is explained in details. Simulation results on different bootstrap decoding algorithms are given and the results are compared to those of reliability based schedule and hybrid decoding. The discussion is then followed.

Finally, chapter 6 is devoted to concluding remarks and suggestions for future research.

Chapter Two

LDPC Codes

In this chapter, some principles and properties, of LDPC codes and their graphical representation, are given. Iterative decoding algorithms for LDPC codes are explained next, followed by a discussion on decoding schedules.

2.1 Principles and Constructions

An LDPC code C [15], like any other linear block code, can be fully described by its parity-check matrix $\mathbf{H}$, through the parity-check equations $\mathbf{y}\mathbf{H}^T = \mathbf{0}$, $\mathbf{y} \in C$, where $\mathbf{y}$ is the code-word.

For a code C with parameters $(\mathbf{n}, \mathbf{k})$, $\mathbf{H}$ has typically $\mathbf{n}$ columns and $\mathbf{n} - \mathbf{k}$ rows, where $\mathbf{n}$ and $\mathbf{k}$ being the block length and the dimension of the code, respectively. In the

parity-check matrix $\mathbf{H}$, the number of nonzero elements per column (row) is referred as the column (row) weight. For LDPC codes however, the parity-check matrix is sparse (only a small fraction of elements are nonzero). For a fixed column weight and fixed rate $\mathbf{R}$, the minimum distance of these codes increases linearly with block length $\mathbf{n}$ [15].

A good way to represent codes is to draw an equivalent graphical picture for the code corresponding to the set of constraints that specify the code. In the following subsections, the Tanner Graph (TG) [35] is being explained as one possible solution for this, followed by different codes' constructions and a special class of LDPC codes, "finite geometry LDPC codes [16]," that offers some good properties.

2.1.1 Tanner Graph (TG)

Iterative decoding algorithms are naturally described using a graph of the code constructed based on $\mathbf{H}$. Such a graph is called a Tanner graph (TG) [35]. To best describe the TGs [35], the codes are assumed to be binary.

Definition 2.1.1.a A TG [35] is a bipartite graph that contains two types of nodes, bit (symbol) nodes and check-nodes, corresponding to the columns and the rows of the parity-check matrix $\mathbf{H}$, respectively. A '1' located at position (i, j) of $\mathbf{H}$ corresponds to an edge between bit-node j and check-node i .

Example 2.1.1 A (7, 3) linear block code, given by the following parity-check matrix $\mathbf{H}$, can be represented by a TG as shown in Figure 2.1.1.

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

Figure 2.1.1 A (7, 3) Hamming Code, Where the Blank Nodes and the Crossed Nodes Represent the Columns and Rows of the Parity-Check Matrix $\mathbf{H}$, Respectively

From the TG, the number of edges connected to any bit or check-node is referred as the node degree. One important property that relates the relative position of bit-nodes, as well as edges connecting different nodes is the girth.

Definition 2.1.1.b A cycle on a TG is defined as the sequence of connected edges which starts and ends at the same vertex (bit or check-node) and satisfies the condition that no vertex appears more than once, except for the initial vertex

Definition 2.1.1.c A girth of any node on a TG is defined as the length of the shortest cycle that passes through that node, while the girth for a directed edge connecting nodes s

and p , and starting from s , is the shortest cycle that passes through s and does not contain that edge.

From definition 2.1.1.c, bit-node 4 has a girth 6; while for bit-node 1, the girth is infinity.

2.1.2 Constructions of LDPC Codes

Generally, LDPC codes have random constructions [9], [12], [15], [23], [32], [33], however, based on their parity-check matrix, LDPC codes can be differentiated into *Regular* and *Irregular* LDPC codes.

Definition 2.1.2 An LDPC code C is said to be regular if, in the corresponding TG, all nodes of the same type have the same degrees, and is said to be irregular otherwise.

Irregular LDPC codes, introduced in [18] and [20], have been shown [19], [21], [25] to outperform regular ones. Furthermore, using irregular structures, capacity approaching LDPC codes have been designed that perform at only 0.0045 dB away from Shannon limit [12], [31], [33].

2.1.3 Finite Geometry LDPC Codes

Within the different classes of LDPC codes, finite geometry LDPC codes have shown to possess interesting properties [16], [22].

Definition 2.1.3 A finite geometry is a finite set of points, a finite set of lines, and a relation incidence, between them.

Under certain conditions according to those incidence relations [16], a matrix whose columns and rows corresponds to the points and lines of a finite geometry, respectively, is a low-density matrix and its null space is then an LDPC code.

In [16], the authors presented four different classes of finite geometry LDPC codes. LDPC codes constructed based on these geometries are cyclic or quasi-cyclic, and thus easy to encode (linear time). The graphs for these codes are regular and have a girth 6. They also have relatively large minimum distances. In fact, the exact minimum distances for some classes of these codes are known.

Two important families of finite geometry are *Euclidean* and *Projective* geometries over finite fields [16] (EG and PG, respectively). Their parity-check matrices are square and the TG of the codes has n bit-nodes and n check-nodes (k redundant check equations). Because of the redundant check equations, these codes perform well with bit-flipping schemes.

2.2 Iterative Decoding Algorithms

In its conventional form, an iterative decoding algorithm, also called “message-passing algorithm,” starts by creating initial messages at bit-nodes from the observations

at the output of the channel and the channel characteristics. It then passes these messages to check-nodes through the edges of the graph.

Iterations then start by each iteration consisting of two steps: *i*) check-nodes processing the received information and passing new messages to the bit-nodes. These messages usually reflect the estimate of check-nodes from the value and reliability of bit-nodes; *ii*) bit-nodes processing the messages sent by check-nodes, and sending back updated information about their value and the associated reliability to check-nodes.

In each iteration, hard-decision on the value of each bit is made at bit-nodes, and the algorithm stops if the hard-decision assignment for bits satisfies all the check equations, or a maximum number of iterations is reached. The messages can be “hard” and/or “soft” information, distinguished by binary vs. non-binary message alphabets, respectively. The operations performed in bit and check-nodes to generate messages depend on the nature of the decoding algorithm and the message type(s).

Consider a binary LDPC code $\mathbf{C}$ of length $\mathbf{n}$, defined as the null space of the row span of the parity-check matrix $\mathbf{H}$ with $\mathbf{I}$ rows, $h_1, h_2, \dots, h_I$, and $\mathbf{n}$ columns. Each row h_i in $\mathbf{H}$ represents a check-node c_i . Suppose that a code-word $\mathbf{v}$ in $\mathbf{C}$ is transmitted over a binary input additive white Gaussian noise (AWGN) channel using a BPSK modulation, where each bit v_j of the code-word $\mathbf{v} = (v_1, \dots, v_n)$ is mapped to the bipolar signal $x_j = (-1)^{v_j}$

(note that the bit values $\{0, 1\}$ and the corresponding signal values $\{+1, -1\}$ are used interchangeably throughout the thesis).

At the output of the channel, we have $\mathbf{r} = \mathbf{x} + \mathbf{n}$, where $\mathbf{n}$ is a Gaussian random vector independent of $\mathbf{x}$ and with independent and identically distributed elements. Each element n_j of $\mathbf{n}$ is a zero-mean Gaussian random variable with variance $\sigma^2 = N_0 / 2\mathbf{R}\mathbf{E}_b$, in which $\mathbf{R}$ is the rate of the code, and $\mathbf{E}_b$ and $N_0/2$ are the average energy per information bit and the power spectral density of AWGN, respectively.

For such a communication scenario, let (s, p) represent a neighboring pair of bit and check-nodes (v_j, c_i) , respectively. $N(p)$ and $N(s)$ are the sets of all the neighboring bit and check-nodes of p and s , respectively. $N(p) \setminus s$ is the set of all the neighboring bit-nodes of p , other than s . $N(s) \setminus p$ is the set of all the neighboring check-nodes of s , other than p . Starting from $r_j, j = 1, \dots, \mathbf{n}$, iterative decoding algorithms generate initial messages $m_j^{(0)}$ at bit-nodes v_j , and estimate $\mathbf{y}$ by performing one of the suitable algorithms iteratively.

For the sake of brevity, in the following subsections, we explain some of these algorithms that have been used throughout this thesis.

2.2.1 Belief-Propagation (BP) Algorithm

Belief-propagation (BP) [14], [15], also known as “sum-product,” is a soft algorithm, i.e., it passes soft information (real messages) between check and bit-nodes, and converges to a posteriori probabilities (APP) for bits if the TG is cycle-free [14]. We assume that all the messages for BP are log-likelihood ratio; $\text{LLR} = \log\left(\frac{P_1}{P_{-1}}\right)$, where P_1 (P_{-1}) is the a posteriori probability of having a ‘0’ (‘1’) being transmitted. The BP algorithm is then carried out, iteratively, as follows:

Initialization (at bit-node s):

$$m_{s \rightarrow p} = m_j^{(0)} = \frac{2}{\sigma^2} r_j. \quad (2.2.1.1)$$

Check-node step (at node p):

$$m_{p \rightarrow s} = \log \frac{1 + \prod_{k \in N(p) \setminus s} \tanh(0.5 m_{k \rightarrow p})}{1 - \prod_{k \in N(p) \setminus s} \tanh(0.5 m_{k \rightarrow p})}. \quad (2.2.1.2)$$

Bit-node step (at node s):

$$m_{s \rightarrow p} = m_j^{(0)} + \sum_{k \in N(s) \setminus p} m_{k \rightarrow s}. \quad (2.2.1.3)$$

In (2.2.1.2) and (2.2.1.3), incorporating $N(p) \setminus s$ and $N(s) \setminus p$, guarantees that only extrinsic information is passed along to s and p , respectively.

Hard decision and stopping criteria (at bit-node s):

$$w_s = m_j^{(0)} + \sum_{k \in N(s)} m_{k \rightarrow s} . \quad (2.2.1.4)$$

$$\hat{s} = \frac{1 - \text{sgn}(w_s)}{2} . \quad (2.2.1.5)$$

The algorithm stops if the estimated hard-decision assignment $\underline{\mathbf{z}}$ (consisting of $\hat{s}$'s, i.e., $z_j = \hat{s}$) is a code-word, or a maximum number of iterations is reached

2.2.2 Min-Sum (MS) Algorithm

Min-sum (MS) is another soft iterative decoding algorithm. Other common names for min-sum are “max-product” and “max-sum,” depending on the type of messages and the corresponding operations performed in bit and check-nodes [14]. MS can be considered as an approximation to BP in LLR domain [14] (i.e., all MS messages are LLR, as well). On cycle-free graphs, MS converges to the maximum-likelihood (ML) solution for code-words [14].

All the steps for MS are the same as those of BP, except for the following initialization and check-node operation.

Initialization (at bit-node s):

$$m_{s \rightarrow p} = m_j^{(0)} = r_j. \quad (2.2.2.1)$$

Check-node step (at node p):

$$m_{p \rightarrow s} = \prod_{k \in N(p) \setminus s} \text{sgn}(m_{k \rightarrow p}) \text{Min}_{k \in N(p) \setminus s} |m_{k \rightarrow p}|, \quad (2.2.2.2)$$

where $\text{sgn}(x) = 1$ or -1 , depending on $x \geq 0$ or < 0 , respectively. As described above, the algorithm only requires addition and minimum operations, hence the name “min-sum.” (Note that the product in (2.2.2.2) is equivalent to a binary exclusive OR if 1 and -1 are mapped to 0 and 1, respectively.)

Although good codes do not have cycle-free graphs [12], [33], BP appears to perform very well even on graphs with cycles [16], [19], [22], and MS is just a few tenths of a dB inferior to BP [5], [37].

2.2.3 Weighted Bit-Flipping (WBF) Algorithm

Mixing ideas from MS and bit-flipping (see section 2.2.4), weighted bit-flipping (WBF) [16] is categorized as a soft/hard algorithm. In WBF, check-nodes operations are close to those in MS, while the bit-nodes perform a bit-flipping operation, where a bit is

flipped if it happens to be in more unsatisfied than satisfied check equations [15]. It appears to perform particularly well when applied to finite geometry LDPC codes.

Initialization (at bit-node s and check-node p):

$$m_{s \rightarrow p} = m_j^{(0)} = \text{sgn}(r_j) . \quad (2.2.3.1)$$

$$y_p = \min\{|r_j| : 1 \leq j \leq n, h_{pj} = 1\} , \quad (2.2.3.2)$$

where y_p is the minimum absolute received value of all the neighboring bit-nodes of p .

Check-node step (at node p):

$$m_{p \rightarrow s} = y_p \cdot \prod_{k \in N(p)} m_{k \rightarrow p} . \quad (2.2.3.3)$$

Bit-node step (at node s): D_s (flipping decision parameter) is calculated for every bit-node;

$$D_s = \sum_{k \in N(s)} -m_{k \rightarrow s} . \quad (2.2.3.4)$$

Among all the bit-nodes, flip bit s (i.e., $m_{s \rightarrow p}^{(l)} = -m_j^{(l-1)} = -m_{s \rightarrow p}^{(l-1)}$) for which D_s is the largest (note that l is the iteration number).

Hard decision and stopping criteria (at bit-node s): The algorithm stops if the estimated hard-decision assignment $\underline{\mathbf{z}}$ (consisting of $m_s^{(l)}$'s, i.e., $z_j = m_s^{(l)}$) is a code-word, or a maximum number of iterations is reached.

2.2.4 Gallager's Algorithm A (GA)

The first bit-flipping algorithms were proposed by Gallager and were called algorithms A and B (GA and GB, respectively) [15]. In GA algorithm, a bit-node s sends the hard message it receives from the channel to check-node p unless all the neighboring check-nodes, other than p , vote in favor of sending its complement. Algorithm A is a special case of algorithm B, in which the required number of check-node votes for flipping is equal to or greater than β , where β is an integer that is a function of the bit and check-node degrees and the iteration number [15], [31].

In this work, we only use GA algorithm. The GA algorithm performs the following operations:

Initialization:

$$m_{s \rightarrow p} = m_j^{(0)} = \text{sgn}(r_j). \quad (2.2.4.1)$$

Check-node step:

$$m_{p \rightarrow s} = \prod_{k \in N(p) \setminus s} m_{k \rightarrow p} . \quad (2.2.4.2)$$

Bit-node step (at node s): $m_{s \rightarrow p} = -m_j^{(0)}$ if all the extrinsic incoming messages are equal to $-m_j^{(0)}$, otherwise $m_{s \rightarrow p} = m_j^{(0)}$.

Hard decision and stopping criteria (at bit-node s): Choose w_s as the majority of outgoing messages and the initial weight $m_j^{(0)}$. In the case of a tie, $w_s = m_j^{(0)}$. Then, using equation (2.2.1.5), the algorithm stops if the estimated hard-decision assignment, consisting of $\hat{s}$'s, is a code-word, or a maximum number of iterations is reached.

2.2.5 Sipser-Spielman (SS) Algorithm

In [34], the authors used a sequential algorithm to decode regular codes. As in WBF the Sipser-Spielman (SS) algorithm employs the methodology of bit-flipping [15]. In fact, SS can be interpreted as a hard version of WBF. The SS algorithm uses the same initialization and check-node operation as GA, and executes the following routine for the rest:

Bit-node step (at node s): Count the number U_s of the unsatisfied check-nodes in the neighborhood of bit-node s , where check-node p is satisfied if $m_{p \rightarrow s}^{(l)} = m_s^{(l-1)}$ and unsatisfied otherwise. Then, randomly select a bit-node from the set of nodes that has the

greatest U_s and complement (flip) its value, i.e., $m_{s \rightarrow p}^{(l)} = -m_j^{(l-1)} = -m_{s \rightarrow p}^{(l-1)}$. Throughout this work, we modify the SS algorithm by flipping the whole set of bits with the greatest U_s , in order to accelerate the decoding process, while the performance is unchanged.

Hard decision and stopping criteria: The algorithm stops if the estimated hard-decision assignment $\mathbf{z}$ (consisting of $m_j^{(l)}$'s, i.e., $z_j = m_j^{(l)}$) is a code-word, or a maximum number of iterations is reached.

2.3 Decoding Schedules

The order of message-passing, between check-nodes and bit-nodes in a TG of a code, referred to as an updating rule or a schedule, plays a vital rule in the performance of the decoding algorithm. Different schedules have been proposed [14], [26], [34], [36], that have different performance/complexity tradeoffs. In principle, schedules can be categorized into: *i*) Graph independent schedules; and *ii*) Graph-based schedules. In the following we define the different types of existing schedules.

2.3.1 Graph Independent Schedules

Conventionally, iterative algorithms employ a schedule that is independent of the code's TG [14]. Currently, there exist two types of schedules that are graph independent.

Flooding schedule [14], which is the first type and is also known as parallel schedule, is the most widely used schedule. The name “flooding” reflects the fact that in each iteration, all check-nodes, and consequently all bit-nodes in the TG, pass new messages to their neighbors, simultaneously. Among the algorithms introduced in the previous section, BP, MS and GA algorithm employ the flooding schedule.

The second type of graph independent schedules is the sequential schedule [10], [16], [34]. Different from the former, in each iteration, a sequential schedule will update only a bit-node or a set of bit-nodes, and then messages are sent simultaneously from all updated bit-nodes to their neighboring check-nodes. Consequently, neighboring check-nodes update their values and send out messages to all their neighboring bit-nodes. As examples, WBF and SS algorithm employ the sequential scheduling.

2.3.2 Graph-Based Schedules

As the name reflects, “Graph-based schedules” are all tailored to the TG of the code. These schedules provide different tradeoffs between error performance and complexity and, in general, they outperform the conventional flooding schedule in error performance [26], [36]. Two types of graph-based schedules have been introduced in [26] and [36]: *i*) Probabilistic schedules; and *ii*) Deterministic schedules.

Probabilistic schedule, first introduced in [26] and later generalized in [36], updates the outgoing-messages of a bit-node s (check-node p) with an average frequency

proportional to the girth of $s(p)$. In its general form [36], the probabilistic schedule can be configured to operate on edges instead of nodes (i.e., edge-based schedule). In such a design [36], an edge connecting two nodes in the TG, will update its outgoing message from bit-node s (check-node p) to check-node p (bit-node s) with an average frequency proportional to its girth.

Another graph-based schedule is the deterministic schedule [36]. For the updating rule, consider a bit-node s (check-node p), a down counter starting with a value δ proportional to the girth passing through $s(p)$ will permit $s(p)$ to update its outgoing-message for δ times. Considering edges instead of nodes, the author proposed in [36] the edge-based deterministic schedule. In such a version, the updating rule is governed by a down counter which is proportional to the girth of the directed edge.

For either probabilistic or deterministic scheduling, the schedules can be applied to one type or both types of nodes, referred as unidirectional vs. bidirectional schedule, respectively. For edge-based scheduling, the schedule can be unidirectional or bidirectional by applying it in either one direction (e.g., from s to p) or both directions, respectively.

Chapter Three

Reliability Based Schedule

In this chapter, an introduction to the idea of reliability based schedule is presented [28], [29]. A proposed set of supporting algorithms is given later in the chapter. Next, simulation results are presented followed by a discussion on error performance and complexity issues.

3.1 Reliability Based Schedule (RBS)

The main ideas in reliability based schedule (RBS) are: *i*) preventing unreliable information from propagating in the TG of the code; and *ii*) updating the values and/or outgoing messages of unreliable bits using more reliable extrinsic information. Incorporating these ideas, decoding is initiated by first identifying a number of less

reliable bits to be unreliable. Outgoing messages are then updated for unreliable bits by passing messages from more reliable bits through the reliable check equations.

In principle, the schedule can be tailored to any of the existing algorithms. As one proposed manifestation for implementing this scheme, the following algorithm is carried out in each scheduled iteration:

Initialization (at bit-node s and check-node p): Let α and γ represent the bit and check-nodes reliability thresholds, respectively. For a fixed α , where α is a real value, if $|r_s| = |r_{v_j}| \leq \alpha$, then mark s as unreliable. After examining all bit-nodes, for a fixed γ , if the number of unreliable bit-nodes in the neighborhood of $p > \gamma$, then mark p as an unreliable check-node. Otherwise, if a bit or check-node is not unreliable, then, relatively, it is declared as reliable. Since, in principle, a check-node would send out an unreliable message if any of its inputs is unreliable, the γ parameter is chosen to be 1. Thus, the extrinsic information, sent to an unreliable bit-node s through check-node p , is reliable if all other bit-nodes connected to p are reliable.

Bit and check-node step: The algorithm, then, proceeds as the original algorithm, in using the same check and bit operations defined for the original algorithm. The only differences are: *i*) Unreliable bit and check-nodes remain idle, i.e., they do not send any messages; *ii*) For reliable check-nodes, only reliable extrinsic messages, (i.e., only

received from the reliable bit-nodes), are employed in transmitting messages to unreliable bit-nodes; *iii*) For unreliable bit-nodes, reliable extrinsic messages are used in updating their outgoing messages; *iv*) Reliable bit-nodes remain unchanged.

Updating values and reliabilities: Consider s an unreliable bit-node with at least one of the neighboring check-nodes being reliable. Then s can update its outgoing messages with new estimates that depend on the received messages and its initial reliability, based on the iterative algorithm used.

In general, for a given code, α depends on the TG of the code, E_b/N_0 , iteration number, as well as the used algorithm. For the sake of simplicity, for any given algorithm, α is fixed for a given code and at a given E_b/N_0 and is assumed to be the same for every bit-node. Thus the schedule is a special case from its general form. Such a simple schedule would require neither a counter nor a random generator, as may be required by other schedules, and hence, has lower complexity. In addition, it does not involve graph calculations (e.g., calculating the girth of a node), and so is graph independent.

Applying this simple RBS only to the first iteration, our results show that, for a given code and a given algorithm, α can be fixed for a wide band of E_b/N_0 , and still perform well, i.e., optimal normalized $\alpha(\alpha/\sigma)$ is highly insensitive to E_b/N_0 variations.

For simplicity, the research is focused on studying the behavior of the simplified RBS over different algorithms. Because of time limitation, in all our implementations, RBS is employed only in the first iteration, and then the original algorithm is left to run till the end of the decoding process.

3.2 RBS Belief-Propagation (RBS-BP) Algorithm

In RBS belief-propagation (RBS-BP) algorithm, BP algorithm is used as the base algorithm. Consider an unreliable bit-node s ; equations (2.2.1.3) and (2.2.1.4) are replaced respectively by the following two equations:

$$m_{s \rightarrow p} = m_j^{(0)} + \sum_{k \in \psi(s) \setminus p} m_{k \rightarrow s} . \quad (3.2.1)$$

$$w_s = m_j^{(0)} + \sum_{k \in \psi(s)} m_{k \rightarrow s} . \quad (3.2.2)$$

where $\psi(s)$ is the set of all the neighboring reliable check-nodes of s , and $\psi(s) \setminus p$ is the set of all the neighboring reliable check-nodes of s other than p .

Equations (3.2.1) and (3.2.2) are the same as equations (2.2.1.3) and (2.2.1.4) defined for BP algorithm, but carried out on just reliable inputs. Starting the second iteration, reliable bit-nodes send the same messages sent in the first iteration, while unreliable bit-

nodes send the messages calculated by equation (3.2.1). The algorithm then executes the same routine as BP till the end of the decoding process.

3.3 RBS Min-Sum (RBS-MS) Algorithm

In RBS min-sum (RBS-MS) algorithm, the algorithm is exactly the same as RBS-BP except for using MS as the base algorithm, and consequently, replacing the BP operations with MS operations, in the schedule and the decoding process.

It is worth mentioning that in our simulations for BP and MS algorithms, one would expect the RBS step, defined in the first iteration, not to considerably improve the performance. The reason may be attributed to the facts that we use a fixed value of α for all the bit-nodes as well as RBS is only applied to the first iteration. And since, in general, BP and MS have very good error performance, the improvement due to the RBS step might be negligible.

3.4 RBS Gallager's Algorithm A (RBS-GA)

Based on the original GA algorithm, its scheduled version (RBS-GA), is carried out by performing the same steps as the original algorithm except for the bit-node step in the first iteration which is replaced by the following.

Bit-node step (at node s): $m_{s \rightarrow p} = -m_j^{(0)}$ if all the extrinsic incoming messages defined in $\psi(s) \setminus p$ are equal to $-m_j^{(0)}$, otherwise $m_{s \rightarrow p} = m_j^{(0)}$.

Starting the next iteration, reliable bit-nodes send the same messages sent in the first iteration, while unreliable bit-nodes send the messages generated from the bit-node step. The algorithm then executes the same routine as GA till the end of the decoding process. It is worth mentioning that the proposed modifications for GA algorithm do not add any soft operation, except for erasing unreliable bit-nodes. Hence, for such a case, RBS is offering a hard-decision scheme.

3.5 RBS Sipser-Spielman (RBS-SS) Algorithm

The scheduled version of SS algorithm (RBS-SS) is exactly the same as the mentioned SS algorithm. The only difference is that in the first iteration, the values of the set of unreliable bit-nodes are updated using majority votes from reliable check-nodes only. Thus, an unreliable bit-node is flipped if it happens in more unsatisfied reliable check-nodes than satisfied ones, and only unreliable bit-nodes are updated in the first iteration. Again we have a modified algorithm with no additional soft steps.

3.6 Simulation Results

Simulations are performed on a number of LDPC codes and on different decoding algorithms. For all the simulation results in this dissertation, unless otherwise is mentioned, the maximum number of iterations is chosen to be 200, and enough code-words are simulated at each signal to noise ratio (SNR), to generate 100 code-word errors. The number of iterations is counted from receiving the channel output till decoding is done. Also, for every code, the received vectors are decoded simultaneously using all algorithms reported for that code. Moreover, the parameters for all reported codes are given in Appendix B.

Results are presented here for B.8 code, while more results for other codes are given in Appendix C. Curves of the bit and block error rates (BER and MER, respectively) for B.8 are given in Figure 3.6.1. The figure contains the results of BP, MS, GA, RBS-BP, RBS-MS and RBS-GA. Results for SS and RBS-SS algorithms are given for B.1 and B.6 codes in Appendix C. The statistics of the number of decoding iterations for converged cases are shown in Table 3.6.1 for the RBS algorithms and their conventional counterparts. Percentages of wrongly-converged code-words are given in Table 3.6.2 only for BP, MS and their RBS algorithms, while no code-words are reported to converge wrongly, for either GA or RBS-GA schemes, for this code.

Figure 3.6.1 BER (—) and MER (---) Curves for B.8 Code Decoded by BP, MS, GA, RBS-BP, RBS-MS and RBS-GA Algorithms.

For each SNR, for the RBS schemes, the results are obtained based on the optimal value for α . The optimal value of α is not only a function of E_b/N_0 and the decoding algorithm, but also depends on the code and its Tanner graph. The effect of α on the BER for the RBS-BP, RBS-MS and RBS-GA decoders is shown in Figure 3.6.2, Figure 3.6.3 and Figure 3.6.4, respectively, for selected SNRs.

For RBS-BP and RBS-MS schemes, the results show no noticeable improvement for both cases over conventional algorithms. In fact, for the B.8 code which is optimized for

BP, the RBS scheme is in general slightly inferior to conventional BP and MS. However, from Table 3.6.1, the average number of iterations required for convergence is lowered by a factor of two or more for both RBS-BP and RBS-MS. Thus the overall complexity is lowered considerably.

Table 3.6.1 Statistics of the Number of Decoding Iterations for B.8 Code for Converged Cases for BP, MS, GA, RBS-BP, RBS-MS and RBS-GA Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		RBS Version	
		Mean	Standard Deviation	Mean	Standard Deviation
BP	1.6	14.65	17.68	5.88	4.64
	1.8	11.96	9.90	4.41	2.01
	2	10.22	6.15	3.44	1.47
MS	2	17.05	30.26	10.27	5.72
	2.2	12.18	17.29	7.73	5.53
	2.4	9.65	7.19	5.88	4.64
GA	5	14.07	7.10	10.27	5.72
	5.5	10.09	10.97	7.73	5.53
	6	7.64	4.41	5.88	1.56

In addition, it can be seen that the RBS scheme is outperforming the conventional GA with a gain of 0.3 – 0.35 dB, while the average number of iterations is lowered considerably. Moreover, for other codes, the RBS-GA scheme provides a gain of up to 0.8 dB over its counterpart. This is due to using soft information in the first step.

Table 3.6.2 The Percentage of Wrongly-Converged Code-Words for B.8 Code for BP, MS, RBS-BP and RBS-MS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional	RBS Version
BP	1.6	6.0E-3%	1.7E-2%
	1.8	1.1E-2%	1.03E-2%
	2	9.0E-3%	1.18E-3%
MS	2	0.00%	0.00%
	2.2	0.00%	0.00%
	2.4	0.00%	6.61E-4%

Figure 3.6.2 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by BP (—) and RBS-BP (---) Algorithms.

Figure 3.6.3 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by MS (—) and RBS-MS (---) Algorithms.

In general, for a given SNR, the RBS BER curve crosses the conventional curve at $\alpha = 0$ and also at least at another elevated α . It is also observed that, the error rate is highly insensitive to the value of α , especially in the range nearby the optimal normalized α (α/σ). Besides, for the B.8 code, for all SNRs, the optimal normalized value of α (α/σ) is around 0.6 for the RBS-GA algorithm, while for the soft algorithms; the BER is almost independent of α .

The same trends also apply to other codes but with different values of optimal normalized value of α (see Appendix C). However, for regular codes, the RBS scheme for soft algorithms shows slightly improvement over the conventional scheme in error rate performance, while the complexity is almost the same for both.

Figure 3.6.4 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by GA (—) and RBS-GA (---) Algorithms.

Furthermore, it was noticed that although undetected errors happen as a function of the code as well as the decoding algorithm, their occurrence can not be predicted. Our results show that for most of the cases, the RBS scheme does not introduce undetected

errors more than what conventional algorithms do. In fact, in some cases, the RBS scheme eliminated all the undetected errors happened in the conventional algorithm.

The complexity cost due to the reliability schedule step is negligible compared to the advantages. In fact, in the first iteration, only a small percentage of bit-nodes is declared unreliable. For example, for the B.1 and B.6 codes given in appendix C, this number, on average, is only about 3.8 – 11.6% for the SNRs between 2.5 – 5dB (larger percentages correspond to smaller SNRs). Furthermore, for hardware implementation, a reliability schedule would be represented by enabling or disabling the hardware representing the original base algorithm, and the schedule does not pass any extra soft messages or require graph calculations. Hence the schedule is graph independent and therefore is more flexible.

3.7 Discussion

Scheduling appears to play a very important role in the performance and complexity of iterative decoding algorithms [36]. Our results indicate that large improvements in error performance and/or decoding complexity can be achieved by employing the reliability schedule with the existing algorithms. The significance of this new schedule is greater when the base algorithm is a hard-decision, e.g. RBS-SS. For simulated codes, the overall complexity is lowered considerably. In addition, the reliability based schedule

scheme neither sends extra soft messages nor depends on the TG of the code. Hence it has low complexity, is graph independent and is more flexible to code changes.

On top of that, more improvement might be achieved when utilizing the RBS for more than one iteration. A simple manifestation is to start with a large α and decrease its value gradually, per iteration, till reaching zero then switching to the base algorithm. In such a case, α is optimized as a function of iteration number, which makes it difficult using Monte Carlo simulations.

Chapter Four

Hybrid (Multistage) Decoding

This chapter presents the hybrid (multistage) iterative decoding of LDPC codes [6], [7]. Two sets of hybrid decoders are developed based on WBF and GA, respectively. Next, simulation results are presented. A discussion and some concluding remarks are given at the end.

4.1 Description

Hybrid (HB) decoding switches among different soft/hard message-passing algorithms throughout the iteration process with the aim of improving the performance and/or the complexity of decoding. As one possible manifestation of a two-stage version, the decoding starts with a few iterations of a soft message-passing algorithm, such as BP

or MS, and then switches to a less complex hard message-passing algorithm, such as WBF or GA.

Depending on the number of iterations for the first decoding stage, and the convergence properties of the two algorithms, different performance/complexity tradeoffs, which in many cases are more attractive than those of the individual algorithms, are achievable.

Although the concept of HB or multistage decoding is quite general and can be applied to different algorithms and in different number of stages depending on the application, our focus here is on a two-stage decoder with the first stage consisting of few iterations of a soft algorithm, such as BP or MS, and the second stage continuing on by performing iterations of a hard algorithm, such as GA.

Consider an instance of HB decoding in which an iterative hard algorithm follows a finite number of iterations of a soft algorithm. The switch happens at the output of bit-nodes, where all the outgoing messages m_i are replaced by $m'_i = \text{sgn}(m_i)$. In fact the initial message $m^{(0)}$ is also replaced by $\text{sgn}(m^{(0)})$. Here, we consider BP and MS followed by GA. The HB algorithms are denoted by (iBP, GA), and (iMS, GA), where i denotes the number of iterations for the soft algorithm.

Another example for HB decoding is where BP and MS are followed by WBF. For the case of WBF, the initial reliability r_j is replaced by w_s (calculated from equation (2.2.1.4)) and the outgoing messages m_i are replaced by $m'_i = \text{sgn}(w_s)$. Consequently, the algorithms are denoted by (iBP, WBF), and (iMS, WBF) and the simulation results are given in Appendix C.

4.2 Simulation Results

To study the performance and complexity of HB algorithms, simulation results on the B.8 code are presented here, while results for other codes are reported in appendix C. The BER and MER curves are given in Figure 4.2.1 and Figure 4.2.2. The figures contain the results of HB decoding for 1, 2 and 3 iterations of BP and MS followed by GA. The figures also contain the BER curves for pure BP, MS and GA as references.

It can be seen that for HB algorithms, the largest improvement per soft iteration compared to pure GA is obtained by the first iteration. Also, for a small number of iterations of the soft algorithm, the difference between BP and MS is negligible.

To investigate the complexity of HB algorithms, we have listed the average number of soft/hard iterations required for convergence of (iBP, GA) and (iMS, GA), $i = 1, 2, 3$, at the waterfall region in Table 4.2.1. The statistics for BP, MS and GA are also listed as references. Percentages of wrongly-converged code-words are only given for BP and MS

algorithms in Table 4.2.2, while for other algorithms, no undetected cases are reported. Thus HB decoding eliminates all undetected errors.

It can be seen that as performance improves, the complexity increases, and that HB algorithms sweep a wide range of performance/complexity tradeoffs between those of pure BP and/or MS and GA. In addition, (iBP, GA) is not offering any considerable gain over (iMS, GA) which is less complex, and the results given in Table 4.2.1 are another testimony for this.

Figure 4.2.1 BER Curves for B.8 Code Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.

Table 4.2.1 Statistics on the Complexity of BP, MS, GA, (iBP, GA) and (iMS, GA) Decoding Algorithms for B.8 Code.

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
GA	6	0	7.64
	6.5	0	5.97
	7	0	4.59
1BP-GA	5	1	5.80
	5.5	1	3.52
	6	1	2.21
2BP-GA	4	2	5.53
	4.5	2	3.30
	5	2	1.58
3BP-GA	3.5	3	2.81
	4	3	1.77
	4.5	3	0.48
1MS-GA	5	1	5.46
	5.5	1	3.47
	6	1	2.22
2MS-GA	4	2	5.65
	4.5	2	2.97
	5	2	1.68
3MS-GA	3.5	3	2.88
	4	3	1.84
	4.5	3	0.56

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
BP	1.6	14.65	0
	1.8	11.96	0
	2	10.22	0
MS	2	17.05	0
	2.2	12.18	0
	2.4	9.65	0

Table 4.2.2 The Percentage of Wrongly-Converged Code-Words for B.8 Code for BP and MS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional	HB Version
BP	1.6	6.0E-3%	0.00%
	1.8	1.1E-2%	0.00%
	2	9.0E-3%	0.00%
MS	2	0.00%	0.00%
	2.2	0.00%	0.00%
	2.4	0.00%	0.00%

Unlike the case for regular codes, for PG LDPC codes (see Appendix C), the change of the performance of BP and MS with the number of iterations follows an unusual pattern. By increasing the number of iterations, the performance of these algorithms (BP and MS) does not always improve [7]. As it can be seen in Figure C.2.7, Figure C.2.8, Figure C.2.9 and Figure C.2.10, similar behavior is observed for (iBP, WBF) and (iMS,

WBF). For (iMS, WBF) algorithms, (1MS, WBF) is giving the best performance when applied to any of the PG LDPC codes B.6 and B.7.

Figure 4.2.2 MER Curves for B.8 Code Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.

In [11], the authors reported similar phenomenon for PG codes decoded with MS algorithms. They observed that, for a certain block, if in the second iteration, the block is not converged to a valid code-word, then the number of bits in error is exceeding the number in the first iteration. They related this to the inaccuracy of the soft information delivered from the first iteration and that MS does not perform well for codes with high degree check-nodes, (e.g., when the check-nodes have degree greater than 6).

In fact, our results show that this phenomenon is not restricted to the first iteration, as well as it is happening with BP algorithm. However, at this point, we do not have a good explanation for such phenomenon.

4.3 Discussion

Hybrid or multistage algorithms are introduced as a method of achieving a wide range of performance/complexity tradeoffs for the decoding of iterative coding schemes. Although the results are presented for a two-stage soft-hard decoder of LDPC codes over an AWGN channel, the same ideas can be applied to decoders with a larger number of stages performing over the more general family of binary-input memoryless output-symmetric channels, and also to other iterative coding schemes, such as turbo codes.

Chapter Five

Bootstrap Decoding Algorithms

In the following subsections, bootstrap algorithms [27], [28] are developed, and simulation results are presented. Next, a comparison is given among reliability based schedule, hybrid decoding and bootstrap decoding. A discussion and concluding remarks are given at the end.

5.1 Bootstrap (BS) Decoding

Resembling RBS, bootstrap (BS) decoding is based on identifying the unreliable (or less reliable) received bits and estimating their values and corresponding reliabilities by passing information to them from more reliable bits and through “reliable” check-nodes. This is done as the first step of the decoding, preventing the less reliable information to propagate through the graph.

However, different from RBS schemes, BS algorithms employ the idea of hybrid decoding in estimating the new reliabilities and values for unreliable bits. For instance, in the first iteration, the algorithm performs a “reliability based scheduled iteration” using a soft algorithm, such as BP or MS (bootstrap step). Continuing on the decoding process, the decoder then switches to a less complex algorithm, such as a hard/soft or hard algorithm.

In principle, a BS decoding algorithm would perform the same steps mentioned in RBS, in the first iteration, except that, new values and reliabilities are assigned to erasure bits. For instance, consider s an unreliable bit-node with at least one of the neighboring check-nodes, to s , being reliable. Then s can replace its existing reliability r_s with a new estimated one r'_s , that depends on the received messages and its initial value r_s as well as the used algorithm in the first iteration.

In this work, hard/soft and/or hard algorithms, such as WBF, GA and SS, respectively, are used to illustrate the idea of “bootstrap decoding” that can be applied, in principle, to many iterative message-passing algorithms. Based on our results for HB decoding, for the bootstrap step, MS is chosen as the soft algorithm, because of its lower complexity compared to BP. Hence, the estimation step at unreliable bit-node s can be represented by:

$$r'_s = r_s + \sum_{p \in \psi(s)} \prod_{k \in N(p) \setminus s} \text{sgn}(m_{k \rightarrow p}). \min_{k \in N(p) \setminus s} |m_{k \rightarrow p}|, \quad (5.1.1)$$

If there is no reliable check-node connected to an erased bit-node, the original received value is reassigned to that node.

In addition, we also show that switching from a complex algorithm to a less complex one is not an essential condition for improving the performance. As an example, to reduce complexity, a hard algorithm, such as the majority rule (MR), might be chosen to replace MS in the bootstrap step for bootstrapped hard message-passing algorithms and, yet, still perform well. In such a case, the estimation step at unreliable bit-node s is carried out by the following equation:

$$m'_s = \begin{cases} \text{sgn}(m_s^{(0)} + \sum_{k \in \psi(s)} m_{k \rightarrow s}) & ; \quad m_s^{(0)} \neq - \sum_{k \in \psi(s)} m_{k \rightarrow s} \\ m_s^{(0)} & ; \quad m_s^{(0)} = - \sum_{k \in \psi(s)} m_{k \rightarrow s} \end{cases}, \quad (5.1.2)$$

where m'_s replaces $m_s^{(0)} = m_j^{(0)}$, for unreliable bit-nodes, through the next operations and $m_{k \rightarrow s}$ is the operation defined in (2.2.4.2).

For simplicity, bootstrapped algorithm based on MS or MR bootstrap step are abbreviated with a suffix '-MS' and '-MR', respectively. It is worth mentioning that a bootstrapped SS algorithm, with MR bootstrap step is the same as RBS-SS.

5.2 BS Weighted Bit-Flipping Algorithm

An MS bootstrap step is added to WBF algorithm, and the combination is called bootstrapped weighted bit-flipping (BWBF-MS). Decoding is initiated by identifying and erasing all the unreliable bit-nodes. Improved values and reliabilities are, then, assigned to the erased bits using equation (5.1.1). Next, WBF algorithm is initialized using the new reliabilities and will continue till the rest of the decoding process.

5.3 BS Gallager's Algorithm A

Gallager's algorithm A is used with BS decoding instead of WBF as in section 5.2. Using an MS or MR bootstrap step, the new combination is known as bootstrapped Gallager's algorithm A (BGA-MS and BGA-MR, respectively). Decoding is initiated by carrying out the appropriate bootstrap step and unreliable bit-nodes are updated using equation (5.1.1) or (5.1.2), respectively. Hence, the new estimates are used to replace $m^{(0)}$. Starting the second iteration, the GA algorithm is initialized using equation (2.2.4.1) for all bit-nodes and the algorithm then follows as BWBF with GA steps replacing WBF steps.

5.4 BS Sipser-Spielman Algorithm

An MS bootstrap step is added to SS algorithm to form the bootstrapped Sipser-Spielman (BSS) algorithm. The algorithm is initialized by performing the bootstrap step,

using equation (5.1.1), in the first iteration. Next, SS algorithm is initialized using the estimated reliabilities, and decoding follows using SS algorithm.

5.5 Simulation Results

Simulations are performed on a number of LDPC codes. Results of B.8 code are presented, here, and more results are given in Appendix C. Decoding is carried out using BWBF, BGA-MS, BGA-MR and BSS decoding algorithms.

To illustrate the performance of different BS schemes, the received vectors are decoded using WBF, GA, SS, BWBF, BGA-MS, BGA-MR and BSS decoders, and also BP and MS decoders as references. BER and MER curves of BP, MS, WBF, GA, BWBF, BGA-MS and BGA-MR decoders are plotted in Figure 5.5.1. The statistics of the number of decoding iterations for converged cases for the BS algorithms and their conventional ones are shown in Table 5.5.1 and Table 5.5.2 for the bootstrapped algorithms based on an MS and MR bootstrap step, respectively. For B.8 code, no code-words are reported to converge wrongly, for any of the BS algorithm.

For each SNR, the results are obtained based on the optimal value for α . The optimal value of α is not only a function of E_b/N_0 and the decoding algorithm, but also depends on the code and its Tanner graph. The effect of α on the BER of B.8 is shown in Figure

5.5.2, Figure 5.5.3 and Figure 5.5.4 for SNR = 5, 5.5, 6, 6.5 and 7 dB, for BWBF, BGA-MS and BGA-MR algorithms, respectively.

Figure 5.5.1 BER Curves for B.8 Code Decoded by BP, MS, WBF, GA, BWBF, BGA-MS and BGA-MR Algorithms.

The results, for this code, show significant improvement in both BER and MER. At a given probability of error, the BS schemes are out performing the conventional ones with up to 1.7–1.75 dB and 1.75–1.8 dB for BER and MER, respectively. In fact, the BS algorithms outperform (1BP, GA) and (1MS, GA) almost over the whole range of SNRs

and (2BP, GA) and (2MS, GA) at low SNR (see section 5.6 for a more detailed comparison).

Table 5.5.1 Statistics of the Number of Decoding Iterations for B.8 Code for Converged Cases for WBF, GA, BWBF and BGA-MS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
WBF	5	35.86	5.49	16.46	4.74
	5.5	29.37	4.98	11.22	3.91
	6	23.29	4.55	7.88	2.96
GA	5	14.07	7.10	6.63	2.45
	5.5	10.09	10.97	3.35	1.67
	6	7.64	4.41	4.44	1.29

Table 5.5.2 Statistics of the Number of Decoding Iterations for B.8 Code for Converged Cases for GA and BGA-MR Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MR-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	14.07	7.10	7.75	3.36
	5.5	10.09	10.97	6.05	2.21
	6	7.64	4.41	4.97	1.56

Besides, the BGA-MR algorithm, which is less complex, is only 0.5 dB inferior to BGA-MS, while BWBF, the most complex, is inferior to all other BS schemes. This is because, in general, WBF performs well only on finite geometry codes [16]. Also, it is worth mentioning that although BGA-MR is a hard message-passing algorithm, the bit-node operation, for the unreliable bits in the bootstrap step, is a soft one.

Furthermore, for B.6 and B.7 codes, the results show that, BWBF is outperforming not only WBF in both BER and MER but also MS (see Appendix C). As expected, PG codes (B.6 and B.7) perform, in general, better than regular codes (B.1 and B.2), with both BWBF and WBF algorithms [16].

For a given SNR, our statistics show that the average number of iterations required for convergence for the BS algorithms are lower than the conventional algorithms by up to a factor of two or more, especially at high SNRs.

Figure 5.5.2 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by WBF (—) and BWBF (---) Algorithms.

Figure 5.5.3 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by GA (—) and BGA-MS (---) Algorithms.

The complexity cost due to the initial bootstrap step is negligible compared to the advantages. In fact, in RBS and BS schemes, the same percentage of bit-nodes is declared unreliable in the initial step. This percentage is a function of the SNR, i.e., different codes with the same block length result in the same percentage. Moreover, all other codes follow the same trends.

In addition, it is also observed that, in general, the error rate is highly insensitive to the value of α in the range nearby the optimal normalized α (α/σ), especially at lower

SNR. Although, for all SNRs, the optimal normalized value of α (α/σ) is seen to lie in a small range for every BS algorithm, a general optimum α can be chosen for every code and the performance degradation is negligible. For example, for the B.8 code, $\alpha/\sigma = 0.8$ is a good choice for all the BS schemes. The same trends also apply to other codes.

Figure 5.5.4 Effect of α on BER Given for Selected SNRs for B.8 Code Decoded by GA (—) and BGA-MR (---) Algorithms.

5.6 Comparison among RBS, HB and BS Schemes

To explore the relative performance between RBS, HB and BS schemes, two codes are selected and their error performances are investigated. The first code is the B.8 code

which is an optimized code for BP decoding, while the second is the B.6, which is a PG code that demonstrates the behavior of the PG codes with HB when WBF is the second algorithm.

The BER and MER curves for the B.8 code, for schemes with GA as the second algorithm, as well as conventional algorithms, are given in Figure 5.6.1 and Figure 5.6.2, respectively.

Figure 5.6.1 BER Curves for the B.8 Code Decoded by BP, MS, GA, RBS-GA, (iBP, GA), (iMS, GA), BGA-MS and BGA-MR Algorithms.

In general, consider a given hard/soft or hard algorithm, it can be seen that, its RBS version offers a considerable improvement in both BER and MER by using the soft information received from the channel in the RBS step. However, the added complexity due to the RBS step is negligible compared to the overall reduction in complexity (see Table 3.6.1).

Figure 5.6.2 MER Curves for B.8 Code Decoded by BP, MS, GA, RBS-GA, (iBP, GA), (iMS, GA), BGA-MS and BGA-MR Algorithms.

On the other hand, for the algorithm under consideration, its HB decoding schemes, (i.e., iBP- or iMS-, where i denotes maximum number of iterations assigned to the BP or

MS algorithms, respectively, in the first decoding stage, while the algorithm under consideration is the second algorithm), offer performance/complexity tradeoff that fills the gap between the given algorithm and the superior BP or MS algorithm. The added complexity for the HB decoding also reflects the reduction in the average number of iterations required for convergence (see Table 4.2.2).

However, in general, the BS scheme for that algorithm presents superior performance to its RBS as well as its 1BP- and 1MS- hybrid algorithms. In fact, the BS is also surpassing the 2BP-, 2MS-, 3BP- and 3MS- versions at low SNR, as well. At a given SNR, the average number of iterations required for convergence is reduced for the BS scheme compared to the RBS and HB schemes (see Table 5.5.1). Hence, the added complexity due to the BS step is minimal compared to other schemes. In addition, using MR, which is simpler, in the bootstrap step if suitable, reduces the complexity dramatically, compared to MS bootstrap step, while the performance is still outperforming the 1MS- hybrid decoding over a wide range of SNR, for the same algorithm.

For the PG codes, an example is given by the B.6 code, where Figure 5.6.3 and Figure 5.6.4 present its BER and MER curves respectively, for (iBP, WBF), (iMS, WBF), BWBF, as well as BP, MS and WBF as references. For these results, the maximum number of iterations is chosen to be 100.

As mentioned before in section 4.2, MS and consequently (iMS, WBF) are not performing well [11]. In general, as i increases, (iBP, WBF) as well as (iMS, WBF) perform worse. In addition, (1MS, WBF) algorithm is outperforming MS as well as (iMS, WBF) for $i > 1$, while BWBF is surpassing MS and (iMS, WBF) over a wide range of SNR. In general, the same trends apply to other PG codes.

Figure 5.6.3 BER Curves for B.6 Code Decoded by BP, MS, WBF, (iBP, WBF), (iMS, WBF) and BWBF Algorithms.

Figure 5.6.4 MER Curves for B.6 Code Decoded by BP, MS, WBF, (iBP, WBF), (iMS, WBF) and BWBF Algorithms.

5.7 Discussion

The bootstrap scheme can be considered as an example of hybrid decoding, in which different decoding algorithms are employed at different iterations throughout the decoding process. On the other hand, the bootstrap step can be viewed as a reliability-based message-passing schedule. In fact for the WBF algorithm, applying RBS scheme to WBF algorithm would result in the same BWBF algorithm, if the local reliability of the unreliable bit is not added in equation (5.1.1).

Interestingly, our results show that mixing hard algorithms can also improve the performance. The BGA-MR which is very simple and passes only hard messages along the edges of the graph is out performing the more complex hybrid algorithms. In addition, different codes may require appropriate BS algorithms. For PG codes, BWBF is the best choice and performs better than MS.

The results indicate that the BS performance is highly insensitive to α variations, especially in the range nearby the optimal α . In fact, for a given code and a given BS algorithm, the optimum α lies in a wide band. Moreover, it is found that for a given code, a general optimum α can be selected for all the BS schemes, and yet, the performance degradation is negligible. In addition, large gain in performance is achieved and the complexity of the bootstrap step is negligible to its advantages. The overall complexity is reduced by a factor of two or more.

Chapter Six

Concluding Remarks

This thesis has made contributions in developing a family of iterative algorithms for the decoding of LDPC codes that achieve both better error performance and lower decoding complexity. It has also contributed to introducing a new schedule as well as exploring the idea of multistage decoding.

The reliability based schedule, which can be tailored to any decoding algorithm, offers considerable improvements, especially with hard/soft and hard message-passing algorithms. The decoding complexity, measured in terms of the average number of iterations, is lowered considerably, and the added complexity, due to the reliability based schedule, is negligible. Moreover, this special case of the schedule is graph independent.

Tuning the schedule parameter is highly insensitive to noise power, for a particular code and a particular algorithm.

On the other hand, hybrid decoding explores the normal idea of mixing several algorithms, with the aim of achieving a wide range of performance/complexity tradeoffs. Combining both proposals, bootstrap decoding is introduced which possesses the performance enhancement of hybrid decoding and reliability based schedule, to offer significant improvement at lower decoding complexity. Also, the new decoding scheme can be customized to any algorithm.

There are a few directions in which the contributions of this work can be extended. Studying the dynamics of reliability based schedule in its general form is of definite interest.

The fading channels are of great interest due to the high demand for wireless applications. Studying the performance/complexity tradeoff for bootstrap schemes over fading channels is an important issue.

Evaluating the threshold for iterative algorithms is an important fundamental problem. In particular, a continuation of this work could be to apply these algorithms to other combinations of iterative decoding algorithms and try to analytically explain the performance/complexity tradeoffs.

Appendix A: Acronyms

AWGN	additive white Gaussian noise
BER	bit error rate
BGA-MS	bootstrapped Gallager's algorithm A with the MS bootstrapped step
BGA-MR	bootstrapped Gallager's algorithm A with the SS bootstrapped step
BP	belief-propagation
BS	bootstrap
BSS	bootstrapped Sipser-Spielman
BWBF	bootstrapped weighted bit-flipping
EG	Euclidean geometry(ies)
GA	Gallager's algorithm A
GB	Gallager's algorithm B
HB	hybrid or multistage
LDPC	low-density parity-check

LLR	log-likelihood ratio
MER	message error rate
MS	min-sum
PG	projective geometry(ies)
RBS	reliability based schedule
RBS-BP	reliability based scheduled A belief-propagation
RBS-MS	reliability based scheduled A min-sum
RBS-GA	reliability based scheduled A Gallager's algorithm A
RBS-SS	reliability based scheduled A Sipser-Spielman
SNR	signal to noise ratio
SS	Sipser-Spielman
TG	Tanner graph
WBF	weighted bit-flipping

Appendix B: LDPC Codes

This section contains more information about the LDPC codes that have been used throughout this work.

B.1 (273, 191) Regular Code; Column Weight = 3

This code [24] is a high rate code. It has the same block length and rate as B.2 and B.6 codes. The columns have weight three, while the rows have weight ten. Also, matrix is available online at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#l93>, and the code is referred to as 273.82.3.353.

B.2 (273, 191) Regular Code; Column Weight = 4

This code [24] is also a high rate code. The columns have weight four, while the rows have weights thirteen and fourteen. Matrix is available online at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#198>, and the code is referred to as 273.82.4.3065.

B.3 (1008, 504) Regular LDPC Code

This code [24] has a rate = $\frac{1}{2}$. The columns have weight three, while the rows have weights six. Matrix is available online at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#126>, and the code is referred to as 504.504.3.504.

B.4 (2640, 1320) Regular LDPC Code

This code [24] has a rate = $\frac{1}{2}$. The columns have weight three, while the rows have weights six. Matrix is available online at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#127>, and the code is referred to as Margulis2640.1320.3.

B.5 (8000, 4000) Regular LDPC Code

This code [24] has a rate = $\frac{1}{2}$. The columns have weight three, while the rows have weights six. Matrix is available online at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html#130>, and the code is referred to as 8000.4000.3.483.

B.6 (273, 191) PG LDPC Code

This code [16] is a high rate PG code, with cyclic $\mathbf{H}$. All columns and rows have weight 17. The first row's indices (where the non-zero entries are) are: 0 5 15 34 35 42 73 75 86 89 98 134 151 155 177 183 201. The parity-check matrix is constructed by cyclically shifting the first row 272 times.

B.7 (1057, 813) PG LDPC Code

This code [16] is a high rate PG code, with cyclic $\mathbf{H}$. All columns and rows have weight 33. The first row's indices are: 0 16 39 48 94 112 123 204 240 262 283 323 333 424 437 496 513 540 582 662 682 687 739 777 864 890 909 975 1008 1042 1043 1045 1049. The parity-check matrix is constructed by cyclically shifting the first row 1056 times.

B.8 (1000, 500) Irregular LDPC Code

This code [30] is optimized for AWGN channel and BP decoding algorithm. The code has parameters $(\mathbf{n}, \mathbf{k}) = (1000, 500)$. The columns have weights two, three, six and seven, while the rows have weights six and seven.

Appendix C: More Results

More simulation results are given here for RBS, HB and BS schemes. For all the simulation results in this dissertation, unless otherwise is mentioned, the maximum number of iterations is chosen to be 200, and enough code-words are simulated at each signal to noise ratio (SNR), to generate 100 code-word errors. The number of iterations is counted from receiving the channel output till decoding is done. Also, for every code, the received vectors are decoded simultaneously using all algorithms reported for that code.

C.1 Reliability Based Schedule

Figure C.1.1 BER and MER Curves for B.1 Code, Decoded by MS, GA, SS, RBS-MS, RBS-GA and RBS-SS Algorithms.

Figure C.1.2 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by MS (—) and RBS-MS (---) Algorithms.

Figure C.1.3 Effect of a on BER Given for Selected SNRs for B.1 Code Decoded by GA (—) and RBS-GA (---) Algorithms.

Figure C.1.4 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by SS (—) and RBS-SS (---) Algorithms.

Table C.1.1 Statistics of the Number of Decoding Iterations for B.1 Code for Converged Cases for MS, GA, RBS-MS, RBS-GA and RBS-SS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		RBS Version	
		Mean	Standard Deviation	Mean	Standard Deviation
MS	2.5	14.18	23.19	16.33	27.55
	3	8.36	14.63	8.66	16.09
	3.5	5.26	8.32	5.06	8.57
GA	5	2.39	0.90	2.59	1.15
	5.5	2.29	0.77	2.20	1.11
	6	2.13	0.69	1.73	1.12
SS	5	3.03	1.39	2.49	1.22
	5.5	2.77	3.43	2.13	2.41
	6	2.33	1.04	1.68	1.09

Table C.1.2 The Percentage of Wrongly-Converged Code-Words for B.1 Code for MS, SS, RBS-MS and RBS-SS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional	RBS Version
MS	2.5	0.617%	0.565E%
	3	6.02E-2%	0.168%
	3.5	6.07E-2%	5.71E-2%
SS	5	0.00%	0.00%
	5.5	0.134%	6.46E-2%
	6	0.00%	1.63E-2%

Figure C.1.5 BER and MER Curves for B.6 Code, Decoded by BP, MS, GA, SS, RBS-BP, RBS-MS, RBS-GA and RBS-SS Algorithms.

Figure C.1.6 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by BP (—) and RBS-BP (---) Algorithms.

Figure C.1.7 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by MS (—) and RBS-MS (---) Algorithms.

Figure C.1.8 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by GA (—) and RBS-GA (---) Algorithms.

Figure C.1.9 Effect of a on BER Given for Selected SNRs for B.6 Code Decoded by SS (—) and RBS-SS (---) Algorithms.

Table C.1.3 Statistics of the Number of Decoding Iterations for B.6 Code for Converged Cases for BP, MS, GA, SS, RBS-BP, RBS-MS RBS-GA and RBS-SS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		RBS Version	
		Mean	Standard Deviation	Mean	Standard Deviation
BP	2.5	6.12	13.75	5.97	15.70
	3	3.74	7.53	3.21	7.08
	3.5	2.69	3.04	2.25	3.33
MS	2.5	3.27	0.80	3.33	6.47
	3	2.99	1.30	2.79	8.17
	3.5	2.61	1.71	2.28	2.88
GA	5	3.04	0.68	2.41	1.69
	5.5	2.66	0.77	2.07	1.45
	6	2.47	0.80	1.48	1.18
SS	4	3.63	2.42	2.27	1.21
	4.5	2.88	1.24	2.03	0.56
	5	2.48	0.70	1.81	0.48

Table C.1.4 The Percentage of Wrongly-Converged Code-Words for B.6 Code for BP, MS, SS, RBS-BP, RBS-MS and RBS-SS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional	RBS Version
BP	2.5	1.53E%	1.22%
	3	9.72E-2%	0.129%
	3.5	1.09E-2%	9.62E-3%
MS	2.5	0.1%	0.4%
	3	0.00%	0.00%
	3.5	0.00%	0.00%
SS	4	5.37%	1.5%
	4.5	0.739%	0.149%
	5	0.139%	7.46E-3%

C.2 Hybrid (Multistage) Decoding

Figure C.2.1 BER Curves for B.1 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Figure C.2.2 MER Curves for B.1 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Table C.2.1 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.1 Code.

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
WBF	3	0	10
	4	0	7.34
	5	0	5.47
1BP-WBF	3	1	4.12
	4	1	1.91
	5	1	0.58
2BP-WBF	3	2	1.65
	4	2	0.32
	5	1.40	0
3BP-WBF	3	3	0.64
	4	2.21	0
	5	1.40	0
1MS-WBF	3	1	4.15
	4	1	1.97
	5	1	0.60
2MS-WBF	3	2	2.30
	4	2	0.42
	5	1.42	0
3MS-WBF	3	3	2.05
	4	2.30	0
	5	1.41	0

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
BP	3	5.98	0
	4	3.36	0
	5	2.40	0
MS	3	8.36	0
	4	3.59	0
	5	2.42	0

Figure C.2.3 BER Curves for B.2 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Figure C.2.4 MER Curves for B.2 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Table C.2.2 Statistics on the Complexity of BP, MS, WBF, (iBP,WBF) and (iMS, WBF) Decoding Algorithms for B.2 Code.

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
WBF	3	0	8.35
	4	0	7.32
	5	0	5.52
1BP-WBF	3	1	4.85
	4	1	2.41
	5	1	0.62
2BP-WBF	3	2	1.94
	4	2	0.39
	5	1.39	0
3BP-WBF	3	3	0.75
	4	2.26	0
	5	1.38	0
1MS-WBF	3	1	4.55
	4	1	2.42
	5	1	0.66
2MS-WBF	3	2	2.54
	4	2	0.67
	5	1.42	0
3MS-WBF	3	3	1.36
	4	2.50	0
	5	1.41	0

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
BP	3	7.51	0
	4	3.43	0
	5	2.38	0
MS	3	14.01	0
	4	4.05	0
	5	2.41	0

Figure C.2.5 BER Curves for B.5¹ Code, Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.

¹ For this code, the maximum number of iterations is set to 500, for all decoding algorithms.

Figure C.2.6 MER Curves for B.5 code, Decoded by BP, MS, GA, (iBP, GA) and (iMS, GA) Algorithms.

Table C.2.3 Statistics on the Complexity of BP, MS, GA, (iBP, GA) and (iMS, GA) Decoding Algorithms for B.5 Code.

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
GA	5	0	15.37
	5.2	0	12.40
	5.4	0	10.48
1BP-GA	4.15	1	14.68
	4.3	1	12.20
	4.5	1	9.91
2BP-GA	3.7	2	14.91
	3.85	2	12.00
	4	2	10.11
3BP-GA	3.25	3	18.26
	3.5	3	12.42
	3.75	3	8.69
1MS-GA	4.15	1	15.07
	4.3	1	12.51
	4.5	1	11.06
2MS-GA	3.7	2	15.58
	3.85	2	12.63
	4	2	10.47
3MS-GA	3.25	3	18.36
	3.5	3	12.57
	3.75	3	8.33

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
BP	1.4	24.13	0
	1.5	19.36	0
	1.6	16.51	0
MS	1.8	36.87	0
	2	17.11	0
	2.1	13.86	0

Figure C.2.7 BER Curves for B.6 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Figure C.2.8 MER Curves for B.6 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Table C.2.4 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.6 Code.

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
WBF	2	0	19.29
	2.5	0	16.76
	3	0	13.89
1BP-WBF	2	1	12.05
	2.5	1	8.42
	3	1	5.21
2BP-WBF	2	2	6.82
	2.5	2	3.78
	3	2	1.86
3BP-WBF	2	3	5.26
	2.5	3	2.76
	3	3	1.52
1MS-WBF	2	1	15.33
	2.5	1	10.92
	3	1	6.50
2MS-WBF	2	2	14.61
	2.5	2	8.42
	3	2	3.73
3MS-WBF	2	3	14.05
	2.5	3	8.38
	3	3	3.36

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
BP	2	9.93	0
	2.5	6.12	0
	3	3.74	0
MS	2	3.68	0
	2.5	3.27	0
	3	2.99	0

Figure C.2.9 BER Curves for B.7 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Figure C.2.10 MER Curves for B.7 Code, Decoded by BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Algorithms.

Table C.2.5 Statistics on the Complexity of BP, MS, WBF, (iBP, WBF) and (iMS, WBF) Decoding Algorithms for B.7 Code.

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
WBF	3	0	35.46
	3.5	0	32.83
	4	0	26.96
1BP-WBF	3	1	25.81
	3.5	1	14.19
	4	1	5.16
2BP-WBF	3	2	12.18
	3.5	2	2.61
	4	2	1.06
3BP-WBF	3	3	4.29
	3.5	3	1.18
	4	3	1.01
1MS-WBF	3	1	25.86
	3.5	1	18.32
	4	1	7.73
2MS-WBF	3	2	13.16
	3.5	2	5.17
	4	2	1.71
3MS-WBF	3	3	2.55
	3.5	3	1.78
	4	3	1.11

Algorithm	E_b/N_0 (dB)	Average Number of Soft Iterations	Average Number of Hard Iterations
BP	3	6.84	0
	3.5	3.39	0
	4	2.80	0
MS	3	3.77	0
	3.5	3.30	0
	4	2.87	0

C.3 Bootstrap Decoding Algorithms

Figure C.3.1 BER and MER Curves for B.1 Code Decoded by BP, MS, WBF, GA, SS, BWBF, BGA-MS, BGA-MR and BSS Algorithms.

Figure C.3.2 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by WBF (—) and BWBF (---) Algorithms.

Figure C.3.3 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by GA (—) and BGA-MS (---) Algorithms.

Figure C.3.4 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by GA (—) and BGA-MR (---) Algorithms.

Figure C.3.5 Effect of α on BER Given for Selected SNRs for B.1 Code Decoded by SS (—) and BSS (---) Algorithms.

Table C.3.1 Statistics of the Number of Decoding Iterations for B.1 Code for Converged Cases for WBF, GA, SS, BWBF, BGA-MS and BSS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
WBF	5	5.47	2.00	2.41	1.40
	5.5	4.37	1.75	1.80	1.04
	6	3.42	1.55	1.45	0.75
GA	5	2.39	0.90	1.92	1.22
	5.5	2.29	0.77	1.62	1.06
	6	2.13	0.69	1.35	0.91
SS	5	3.03	1.39	2.11	2.70
	5.5	2.77	3.43	1.71	1.02
	6	2.33	1.04	1.29	0.79

Table C.3.2 Statistics of the Number of Decoding Iterations for B.1 Code for Converged Cases for GA and BGA-MR Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MR-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	2.39	0.90	2.32	1.05
	5.5	2.29	0.77	2.21	1.11
	6	2.13	0.69	1.66	1.04

Figure C.3.6 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by BP, MS, WBF, GA, BWBF, BGA-MS and BGA-MR Algorithms.

Figure C.3.7 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by WBF (—) and BWBF (---) Algorithms.

Figure C.3.8 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by GA (—) and BGA-MS (---) Algorithms.

Figure C.3.9 Effect of α on BER Given for Selected SNRs for B.2 Code Decoded by GA (—) and BGA-MR (---) Algorithms.

Table C.3.3 Statistics of the Number of Decoding Iterations for B.2 Code for Converged Cases for WBF, GA, BWBF and BGA-MS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
WBF	5	5.52	2.03	2.72	1.52
	5.5	4.41	1.82	1.92	1.11
	6	3.46	1.55	1.45	0.76
GA	5	3.50	1.95	3.25	7.06
	5.5	2.94	1.63	2.23	4.63
	6	2.40	1.08	1.56	1.38

Table C.3.4 Statistics of the Number of Decoding Iterations for B.2 Code for Converged Cases for GA and BGA-MR Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MR-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	3.50	1.95	3.67	7.68
	5.5	2.94	1.63	2.54	4.35
	6	2.40	1.08	1.84	1.65

Figure C.3.10 BER and MER Curves for B.3 Code Decoded by BP, GA, BGA-MS and BGA-MR Algorithms.

Figure C.3.11 Effect of a on BER Given for Selected SNRs for B.3 Code Decoded by GA (—) and BGA-MS (---) Algorithms.

Figure C.3.12 Effect of α on BER Given for Selected SNRs for B.3 Code Decoded by GA (—) and BGA-MR (---) Algorithms.

Table C.3.5 Statistics of the Number of Decoding Iterations for B.3 Code for Converged Cases for GA and BGA-MS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	10.14	3.29	7.74	2.70
	5.5	7.62	2.85	5.47	1.77
	6	5.44	1.98	4.15	1.31

Table C.3.6 Statistics of the Number of Decoding Iterations for B.3 Code for Converged Cases for GA and BGA-MR Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MR-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	10.14	3.29	9.58	3.62
	5.5	7.62	2.85	6.60	2.27
	6	5.44	1.98	5.12	1.79

Figure C.3.13 BER and MER Curves for B.4 Code Decoded by GA, BGA-MS and BGA-MR Algorithms.

Figure C.3.14 Effect of a on BER Given for Selected SNRs for B.4 Code Decoded by GA (—) and BGA-MS (---) Algorithms.

Figure C.3.15 Effect of a on BER Given for Selected SNRs for B.4 Code Decoded by GA (—) and BGA-MR (---) Algorithms.

Table C.3.7 Statistics of the Number of Decoding Iterations for B.4 Code for Converged Cases for GA and BGA-MS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	12.57	3.81	8.89	2.07
	5.5	8.78	2.07	6.57	1.38
	6	6.44	1.37	5.34	1.12

Table C.3.8 Statistics of the Number of Decoding Iterations for B.4 Code for Converged Cases for GA and BGA-MR Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MR-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
GA	5	12.57	3.81	11.23	3.30
	5.5	8.78	2.07	8.19	1.88
	6	6.44	1.37	6.44	1.37

Figure C.3.16 BER and MER Curves for B.5 Code Decoded by BP, MS, WBF and BWBF Algorithms.

Figure C.3.17 Effect of α on BER Given for Selected SNRs for B.5 Code Decoded by WBF (—) and BWBF (---) Algorithms.

Table C.3.9 Statistics of the Number of Decoding Iterations for B.5 Code for Converged Cases for WBF and BWBF Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
WBF	5	--	--	125.82	14.21
	5.5	232.87	15.18	85.40	10.87
	6	183.76	13.79	46.09	7.72

Figure C.3.18 BER and MER Curves for B.6 Code Decoded by BP, MS, WBF, SS, BWBF and BSS Algorithms.

Figure C.3.19 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by WBF (—) and BWBF (---) Algorithms.

Figure C.3.20 Effect of α on BER Given for Selected SNRs for B.6 Code Decoded by SS (—) and BSS (---) Algorithms.

Table C.3.10 Statistics of the Number of Decoding Iterations for B.6 Code for Converged Cases for WBF, SS, BWBF and BSS Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
WBF	4	9.32	2.91	3.92	2.28
	4.5	7.42	2.51	2.91	1.57
	5	5.83	2.18	2.69	1.32
SS	4	3.63	2.42	2.08	0.87
	4.5	2.88	1.24	1.85	0.49
	5	2.48	0.70	1.76	0.48

Figure C.3.21 BER and MER Curves for B.7 Code Decoded by BP, MS, WBF and BWBF Algorithms.

Figure C.3.22 Effect of α on BER Given for Selected SNRs for B.7 Code Decoded by WBF (—) and BWBF (---) Algorithms.

Table C.3.11 Statistics of the Number of Decoding Iterations for B.7 Code for Converged Cases for WBF and BWBF Algorithms

Algorithm	E_b/N_0 (dB)	Conventional		BS (MS-Based) Version	
		Mean	Standard Deviation	Mean	Standard Deviation
WBF	3.5	32.83	5.43	21.24	6.66
	4	26.96	5.11	13.43	5.41
	4.5	20.73	4.40	7.91	3.32

References

- [1] “Concatenated coding techniques and iterative decoding: sailing toward channel capacity,” Special issue *IEEE J. Selected Areas Comm.*, vol. 16, no. 2, Feb. 1998.
- [2] “Codes on graphs and iterative algorithms,” Special issue *IEEE Trans. Inform. Theory*, vol. 47, no. 2, Feb. 2001.
- [3] “The turbo principle: from theory to practice I,” Special issue *IEEE J. Selected Areas Comm.*, vol. 19, no. 5, May 2001.
- [4] “The turbo principle: from theory to practice II,” Special issue *IEEE J. Selected Areas Comm.*, vol. 19, no. 9, Sept. 2001.
- [5] A. Anastasopoulos, “A Comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution,” in Proc. *IEEE Global Telecom. Conf. (Globecom)*, vol. 2, pp. 1021–1025, 2001.
- [6] A.H. Banihashemi, A. Nouh and P. Zarrinkhat, “Hybrid (multistage) decoding of low-density parity-check (LDPC) codes,” in Proc. *40th Annual Allerton*

- Conference on Communication, Control, and Computing*, Allerton, Illinois, USA, Oct. 2002.
- [7] A.H. Banihashemi and A. Noh, "Hybrid (multistage) decoding of low-density parity-check (LDPC) codes," to be submitted to *IEEE Comm. Lett.*
- [8] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding," in Proc. *IEEE Int. Conf. Comm. (ICC)*, pp. 1067–1070, 1993.
- [9] J.W. Bond, S. Hui, and H. Schmidt, "Constructing low-density parity-check codes," in Proc. *IEEE/AFCEA, Inform. Systems for Enhanced Public Safety and Security (EUROCOMM)*, pp.260-262, 2000.
- [10] A.M. Chan and F.R. Kschischang, "A simple taboo-based soft-decision decoding algorithm for expander codes," *IEEE Comm. Lett.*, vol. 2, issue 7, pp.183–185, July 1998.
- [11] J. Chen and M.P.C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Comm.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.
- [12] Sae-Young Chung, G.D. Forney, T.J. Richardson and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the shannon limit," *IEEE Comm. Lett.*, vol. 5, issue 2, pp.58–60, Feb. 2001.
-

-
- [13] Sae-Young Chung, T.J. Richardson and R.L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans Inform Theory*, vol. 47, no. 2, pp.657–670, Feb. 2001.
 - [14] G.D. Forney Jr, "On iterative decoding and the two-way algorithm," in Proc. *Int. Symp. on Turbo Codes and Related Topics*, Brest, France, pp. 12-25, Sept. 1997.
 - [15] R.G. Gallager, *Low-Density Parity-Check Codes*. Cambridge MA: MIT Press, 1963.
 - [16] Y. Kou, S. Lin and M.P.C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, vol. 47, no. 7, pp. 2711-2736, Nov. 2001.
 - [17] F.R. Kschischang and B.J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 219–230, Feb. 1998.
 - [18] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in Proc. *Annual ACM Symp. Theory of Computing*, pp. 249–258, 1998.
 - [19] 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, issue 2, pp.585-598, Feb. 2001.
-

-
- [20] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proc. Annual ACM Symp. Theory of Computing*, pp. 150–159, 1997.
- [21] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 569–584, Feb. 2001.
- [22] R. Lucas, M.P.C. Fossorier, Yu Kou and Shu Lin, "Iterative decoding of one-step majority logic deductible codes based on belief propagation," *IEEE Trans. Comm.*, vol. 48, no. 6, pp. 931–937, Jun. 2000.
- [23] D.J.C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.
- [24] D.J. MacKay, "Encyclopedia of sparse graph codes," <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html>.
- [25] D. MacKay, S. Wilson, and M. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Comm.*, vol. 47, pp. 1449–1454, Oct. 1999.
- [26] Y. Mao and A.H. Banihashemi, "Decoding low-density parity-check codes with probabilistic scheduling," *IEEE Comm. Lett.*, vol. 5, no. 10, pp. 414–416, Oct. 2001.
-

-
- [27] A. Nouh and A.H. Banihashemi, "A new decoding algorithm for low-density parity-check codes," in *Proc. Biennial Symp. on Comm. Conf.*, Kingston, Canada, pp. 2–4, May 2002.
 - [28] A. Nouh and A.H. Banihashemi, "Bootstrap decoding of low-density parity-check codes," *IEEE Comm. Lett.*, Sep. 2002.
 - [29] A. Nouh and A.H. Banihashemi, "Reliability based scheduling for low-density parity-check codes," to be submitted to *IEEE Comm. Lett.*
 - [30] T.J. Richardson, *private communication*.
 - [31] 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, no. 2, pp. 599–618, Feb. 2001.
 - [32] T.J. Richardson and R.L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.
 - [33] T.J. Richardson, M.A. Shokrollahi and R.L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, issue 2, pp. 619–637, Feb. 2001.
 - [34] M. Sipser and D.A. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, issue 6, pp. 1710–1722, Nov. 1996.
 - [35] R.M. Tanner, "A recursive approach to low-complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981.
-

-
- [36] H. Xiao and A.H. Banihashemi, "New schedules for decoding LDPC codes," in *Proc. Biennial Symp. on Comm. Conf.*, Kingston, Canada, pp. 78–81, May 2002.
 - [37] F. Zarkeshvari and A.H. Banihashemi, "On implementation of min-sum algorithm for decoding low-density parity-check (LDPC) codes," in *Proc. IEEE Global Telecom. Conf. (Globecom)*, Taipei, Taiwan, Nov. 2002.
 - [38] P. Zarrinkhat and A.H. Banihashemi, "Density evolution and convergence properties of bit-flipping algorithms for decoding LDPC codes," in *Proc. 40th Annual Allerton Conference on Communication, Control, and Computing*, Allerton, Illinois, USA, Oct. 2002.
-