A COMPARATIVE STUDY OF MIN-SUM BASED DECODING ALGORITHMS FOR LOW DENSITY PARITY CHECK CODES

by
Dhaval Shah

A Thesis
Presented to Lakehead University
in Partial Fulfilment of the Requirement for the Degree of
Master of Science
in
Electrical and Computer Engineering

Thunder Bay, Ontario, Canada

December 2009

Abstract

The demand for large scale broadband networks is gaining immense popularity for convenient access of information. A common concern of transmitting data through a wireless medium is the effects of noise on the signal. Maintaining the reliability of the data becomes crucial. Employing Low Density Parity Check (LDPC) codes specified by the IEEE 802.16e (WiMAX) standard simplifies the encoding and decoding structure within a digital communication system, making it attractive for the premise of this study.

The focus of the study is to develop LDPC decoding algorithms that require a simple decoding structure. All examined decoding algorithms are based on an approximation of the Belief Propagation (BP) decoding algorithm known as Min-Sum (MS) and Min-Sum based decoding. For this study, three new Min-Sum based decoding algorithms will be proposed and compared to three existing MS based decoding algorithms through software simulations. The objective of each proposed MS based decoding algorithm is to further simplify the decoding structure of already existing Min-Sum based decoding algorithms.

Acknowledgements

To begin with, I would like to thank my thesis supervisor Dr. Nam Yul Yu for his consistent assistance and patience shown throughout my graduate studies.

Secondly, I would like to thank my colleagues for their support and help during my graduate studies.

Lastly, I would like to thank my parents and sister for their continued support of my during my graduate studies.

Table of Contents

Abstract 2
1 Introduction 8
1.1The motivations and premise of this Study 8
1.2Applications of LDPC codes 8
1.3Introduction to the IEEE 802.16e (WiMAX) Standard 8
1.4Thesis Overview 10
1.5The original contribution of this thesis 10
2 Description of system model 11
3 LDPC Codes 13
3.1Introduction to Low Density Parity Check (LDPC) Codes 13
3.2General Description of LDPC codes 13
3.2.1LDPC codes Definitions 13
3.2.2Parity-check Matrix 14
3.2.3Tanner (bipartite) Graphs 15
3.2.4Belief Propagation (BP) 16
3.3Example of LDPC Decoding 18
3.4LDPC codes of the IEEE802.16e standard (WiMAX) 22
4 Min-Sum Based Decoding Algorithms for LDPC codes 27
4.1Known LDPC Decoding Algorithms 27
4.1.1Min-Sum (MS) Decoding Algorithm 27
4.1.2Normalized Min-Sum decoding algorithm (NM-MS) 28
4.1.3Offset Min-Sum decoding algorithm (OFF-MS) 28
4.1.4Self Corrected Min-Sum decoding algorithm (SC-MS) 29
4.2Proposed LDPC Decoding Algorithms 29
4.2.1Normalized and Self-Corrected Min-Sum Decoding Algorithm (NM-SC-MS) 30
4.2.2Offset and Self-Corrected Min-Sum Decoding Algorithm (OFF-SC-MS) 30
4.2.3Variable node Offset Min-Sum Decoding (V-OFF-MS) 31
4.3Comparison of known decoding versus proposed decoding algorithms 32
4.4Parameters Employed for the LDPC Decoding Algorithms 32
5 Simulations and Discussions 33
5.1Convergence of block error rate 33
5.2Average Iteration Numbers 40
5.3Block Error Rate Performance 47
6 Conclusions and Future Work 54
6.1Conclusions 54
6.2Future Work 54
References 56

Appendix A. LDPC Parity Check Matrices for Various Code Rates and Block Sizes 58
Appendix B. C++ source code of decoding algorithms 62
Appendix C. Simulation time 64

List of Figures

Figure 2.1: System model....11
Figure 2.2: System channel....12
Figure 3.1: Tanner (bipartite) graph....16
Figure 3.2: Selecting an edge for Check node message [24]....17
Figure 3.3: Selecting an edge for Variable node message [24]....18
Figure 3.4: Variable node update....20
Figure 3.5: Check node update after the first iteration....20
Figure 3.6: LLR update after the first iteration....21
Figure 5.1: Convergences of block error rate (BLER) at R= 1/2, N= 2304 and Eb/No at 1.5 dB....33
Figure 5.2: Convergences of block error rate (BLER) at R= 1/2, N= 2304 and Eb/No at 1.8 dB....34
Figure 5.3: Convergences of block error rate (BLER) at R= 1/2, N= 1728 and Eb/No at 1.7 dB....34
Figure 5.4: Convergences of block error rate (BLER) at R= 1/2, N= 1728 and Eb/No at 2.0 dB....35
Figure 5.5: Convergences of block error rate (BLER) at R= 1/2, N= 1152 and Eb/No at 1.9 dB....35
Figure 5.6: Convergences of block error rate (BLER) at R= 1/2, N= 1152 and Eb/No at 2.2 dB....36
Figure 5.7: Convergences of block error rate (BLER) at R= 2/3, N= 2304 and Eb/No at 2.2 dB....36
Figure 5.8: Convergences of block error rate (BLER) at R= 2/3, N= 2304 and Eb/No at 2.5 dB....37
Figure 5.9: Convergences of block error rate (BLER) at R= 3/4, N= 2304 and Eb/No at 2.9 dB....37
Figure 5.10: Convergences of block error rate (BLER) at R= 3/4, N= 2304 and Eb/No at 3.1 dB....38
Figure 5.11: Average iteration numbers for various Imax at R= 1/2, N= 2304, and Eb/No at 1.5 dB....40
Figure 5.12: Average iteration numbers for various Imax at R= 1/2, N= 2304, and Eb/No at 1.8 dB....40
Figure 5.13: Average iteration numbers for various Imax at R= 1/2, N= 1728, and Eb/No at 1.7 dB....41
Figure 5.14: Average iteration numbers for various Imax at R= 1/2, N= 1728, and Eb/No at 2.0 dB....41
Figure 5.15: Average iteration numbers for various Imax at R= 1/2, N= 1152, and Eb/No at 1.9 dB....42
Figure 5.16: Average iteration numbers for various Imax at R= 1/2, N= 1152, and Eb/No at 2.2 dB....42
Figure 5.17: Average iteration numbers for various Imax at R= 2/3, N= 2304, and Eb/No at 2.2 dB....43
Figure 5.18: Average iteration numbers for various Imax at R= 2/3, N= 2304, and Eb/No at 2.5 dB....43
Figure 5.19: Average iteration numbers for various Imax at R= 3/4, N= 2304, and Eb/No at 2.9 dB....44
Figure 5.20: Average iteration numbers for various Imax at R= 3/4, N= 2304, and Eb/No at 3.1 dB....44
Figure 5.21: Block error rate performance at R= 1/2, N= 2304, and Imax= 50....47
Figure 5.22: Block error rate performance at R= 1/2, N= 2304, and Imax= 500....47
Figure 5.23: Block error rate performance at R= 1/2, N= 1728, and Imax= 50....48
Figure 5.24: Block error rate performance at R= 1/2, N= 1728, and Imax= 500....48
Figure 5.25: Block error rate performance at R= 1/2, N= 1152, and Imax= 50....49
Figure 5.26: Block error rate performance at R= 1/2, N= 1152, and Imax= 500....49
Figure 5.27: Block error rate performance at R= 2/3, N= 2304, and Imax= 50....50
Figure 5.28: Block error rate performance at R= 2/3, N= 2304, and Imax= 500....50
Figure 5.29: Block error rate performance at R= 3/4, N= 2304, and Imax= 50....51
Figure 5.30: Block error rate performance at R= 3/4, N= 2304, and Imax= 500....51

Index of Tables

Table 1.1: Coverages of various telecommunication standards....................9
Table 4.1: Advantages of proposed decoding algorithms....................32
Table A.1: Code rate and block size adjustment 59
Table C.1: Single iteration duration for studied decoding algorithms 64

List of Abbreviations

Abbreviation	Definition
AWGN	Additive White Gaussian Noise
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
GF(2)	Galois Field over 2 elements
LDPC	Low Density Parity Check
LLR	Log-Likelihood Ratio
LLR-BP	Log-Likelihood Ratio Belief Propagation
ML	Maximum-Likelihood
MS	Min-Sum
NM-MS	Normalized Min-Sum
NM-SC-MS	Normalized and Self-Corrected Min-Sum
OFF-MS	Offset Min-Sum
OFF-SC-MS	Offset and Self-Corrected Min-Sum
SC-MS	Self-Corrected Min-Sum
V-OFF-MS	Variable node Offset Min-Sum

1 Introduction

1.1 The motivations and premise of this Study

For this study, an in-depth investigation of Min-Sum based decoding algorithms for LDPC codes will be examined. Until now, established Min-Sum based decoding algorithms include: *Normalized* Min-Sum decoding [1], *Offset* Min-Sum decoding [2], and *Self-Corrected* Min-Sum decoding [3]. The established decoding algorithms will be classified as *known* decoding algorithms throughout this study. For this study, three new decoding algorithms will be proposed.

The objective of this thesis is to develop simplified Min-Sum based decoding algorithms with simple decoding structures, and compare the results to known Min-Sum based decoding algorithms. For this study, the IEEE 802.16e standard [4] serves as a practical scenario, suitable for investigating the performances of all Min-Sum based algorithms at various lengths and code rates.

1.2 Applications of LDPC codes

LDPC codes are applicable to situations other than the IEEE 802.16e standard. Other applications include IEEE 802.3an-2006 Standard (10GBase-T) [5]. This standard provides 10 Gbps connections over unshielded or shielded twisted pair cables up to distances of 100 meters. Another application of LDPC coding is the G.hn Standard [6]; a standard developed by International Telecommunication Unit for networking (at a rate of 1 Gbps) over power lines, phone lines, and coaxial cables. Lastly, DVB-S2 (Digital Video Broadcasting) [6] uses LDPC coding. The DVB-S2 allows MPEG video and audio to be streamed over a channel.

1.3 Introduction to the IEEE 802.16e (WiMAX) Standard

The IEEE 802.16e is the successor to the IEEE standard 802.16-2004. The IEEE 802.16e standard builds on its predecessor by supporting subscriber stations moving at vehicular speeds, and thus specifies a system for combined fixed and mobile broadband wireless access as mentioned in [7]. The significance of this standard is that it will close the gap between very high data rate wireless local area networks (LANs) and very high mobility cellular systems. For this thesis, this standard will serve as basis for the encoding procedure, code rates, and block lengths are specified by the IEEE 802.16e standard.

WiMAX based applications are better suited for larger geographical networks compared to the IEEE 802.11 standard [8] (Wi-Fi). Table 1.1 shows the range of various wireless communication standards in comparison with the IEEE 802.16e standard. Traditionally, wide area networks (WANs) are composed of two wireless technologies, national mobile networks and satellite providers. In contrast, WiMAX standards are not as widely employed as Wi-Fi based networks, rather limited to mobile networks. Wide LAN (WLAN) technologies, such as those underlying the Wi-Fi standard, are capable of delivering data over a range of 150 meters and the Bluetooth standard provides access to users within 10 meters.

Table 1.1: Coverages of various telecommunication standards

Standard	Coverage
IEEE 802.15 (Bluetooth)	10 m
IEEE 802.11 (a, b, g) (WiFi)	150 m
IEEE 802.16- 2004 (WiMax)	50 km
IEEE 802.16e (WiMax)	Nationwide (e.g. United States of America)

Mobile wireless is scalable in both radio access technology and network architecture. Thus, it provides flexibility in network deployment and service offerings. Mobile WiMAX supports several key features mentioned in [9]. To begin with, mobile WiMAX has a flexible spectrum allocation in that it is scaled to work at various bandwidths from 1.25 to 20 MHz (varies for country requirements). Furthermore, new authentication methods results in enhanced security features. High data rates with the multiple input multiple output (MIMO) antenna techniques and flexible sub-channelization schemes are features of the IEEE 802.16e standard, which can support peak rates of 63 Mb/s in downlink and 28 Mb/s in uplink per sector.

For mobile applications power consumption is a critical factor. Mobile WiMAX provides two modes for power efficient operations; sleep and idle modes. Sleep mode aims to minimize mobile user power consumption and also provides flexibility that allows a mobile user to scan base stations (BS) to collect hand-off related information. Whereas, in idle mode the user can transverse multiple BSs and periodically capture downlink broadcast messages without registering to a particular BS.

1.4 Thesis Overview

Chapter 2 shows the digital communication system model this study employs. Also, a description of the changes for each signal (or vector) is described as it passes through the system.

Chapter 3 reviews general concept of LDPC coding, introduces the Belief Propagation decoding algorithm, followed with a numerical example of LDPC decoding, and the structure of the parity-check matrix defined by the IEEE 802.16e standard and LDPC encoding using the IEEE802.16e standard.

Chapter 4 describes existing and newly proposed Min-Sum based decoding algorithms. Also, the merits of the newly proposed decoding algorithms will be discussed.

Chapter 5 shows simulation results and provides a discussion of the results. The discussion compares and contrasts the performances of existing and newly proposed LDPC decoding algorithms.

Chapter 6 will discuss the merits of the proposed LDPC decoding algorithms in comparison to existing Min-Sum based decoding algorithms. Lastly, future work extending this study will be discussed.

1.5 The original contribution of this thesis

Three new Min-Sum based decoding algorithms will be proposed for LDPC coding. All proposed decoding algorithms are an improvement of known Min-Sum based decoding algorithms. Thus, each proposed decoding algorithm is expected to show better performance results than the known Min-Sum based decoding algorithms. The new decoding algorithms are obtained by a combination or modification of known Min-Sum based decoding algorithms, while maintaining the simple Min-Sum decoding structure. Software based simulations will be performed to compare the performances of proposed Min-Sum based decoding algorithms versus the known Min-Sum based decoding algorithms.

2 Description of system model

Figure 2.1 shows the system that will be employed for this study. The data is generated in the form of binary symbols. These information symbols are sent serially. The key parameter of information is the code rate, R, which is the minimum number of bits per second needed to represent the output signal.

The channel encoder code accepts the information bits as inputs. The encoder data is then organized into blocks of k-bits [10] each denoted by $\underline{u} = (u_0\, u_1,\ldots,u_{k-1})$. The LDPC encoder then maps the $\underline{u}$ linear block code into a coded sequence of n-bits, $\underline{c} = (c_0\, c_1,\ldots,c_{n-1})$, with $n > k$. For this system, the $\underline{u}$ blocks represents the information bits and $\underline{c}$ represents the codeword. The codeword contains both message and parity-check bits. This system will assume that the messages are composed of statistically independent [11] bits.

The output, $\underline{c}$, from the encoder is fed into the input of the modulator. The system will use binary shift phase keying (BPSK) modulation [11]. Once BPSK modulation is applied, the signal is redefined as vector $\underline{x} = (x_0\, x_1,\ldots,x_{n-1})$. The modulated signal is sent through the channel. Equation 2.1 shows the process of BPSK modulation.

$$x_i=(-1)^{c_i} \quad \text{, where} \quad 0 \leq i \leq n-1 \tag{2.1}$$

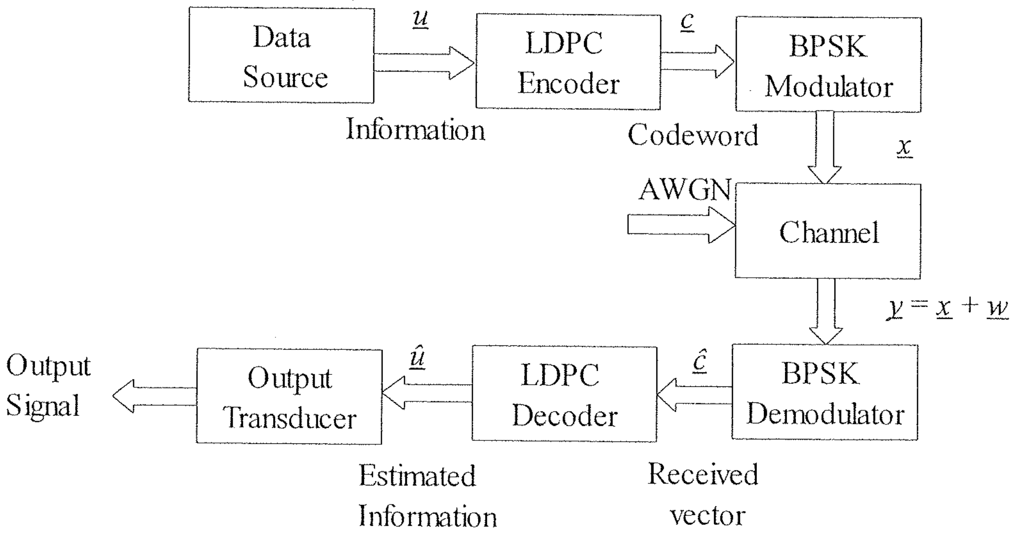

Figure 2.1: System model

The channel adds noise to the modulated vector. For this study, Additive White Gaussian Noise [13] (AWGN) is added to the vector and is denoted by $\underline{w} = (w_0\, w_1,\ldots,w_{n-1})$. Vector $\underline{w}$ and its elements follow the Gaussian distribution with mean of 0 and variance of $\sigma^2 = \frac{N_0}{2}$. Figure 2.2 shows the resultant vector, $\underline{y} = (y_0\, y_1, \ldots, y_{n-1})$, when AWGN is added to the modulated vector. Equation 2.2 shows the process of adding AWGN to a modulated signal.

$$y_i = x_i + w_i \quad \text{, where} \quad 0 \le i \le n-1 \tag{2.2}$$

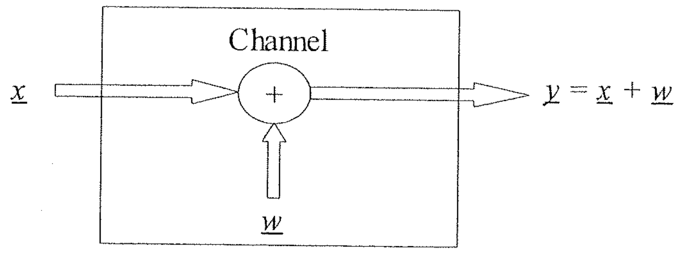

Figure 2.2: System channel

The system takes the $\underline{y}$ vector and employs an antipodal binary signaling scheme using a *maximum-likelihood detection,* or ML detection [10]. In ML detection, for this detector to be optimal [13], it is assumed that the probability of 0 or 1 transmission is equiprobable. The reconstructed codeword is represented by $\hat{\underline{c}}$ and is passed into the LDPC decoder.

The LDPC decoder uses a unique decoding algorithm. LDPC decoding uses a specialized algorithm known as *Belief Propagation* (BP) [12] decoding. For this study, an approximation of the BP algorithm known as the *Min-Sum* (MS) [13] decoding will be examined. LDPC decoding algorithms consist of applying a series of mathematical equations which will be discussed thoroughly in Chapter 3. Once the LDPC decoding procedure has been applied to the signal the resultant vector becomes $\hat{\underline{u}}$. A final operation is performed to verify the reliability of the information data $\hat{\underline{u}}$. This operation is known as an orthogonality check.

3 LDPC Codes

3.1 *Introduction to Low Density Parity Check (LDPC) Codes*

LDPC codes were first introduced by Robert Gallager in 1963 in his PhD dissertation [12]. In his dissertation, Gallager introduced the *Sum Product* algorithm. In order to study the applications of LDPC codes powerful computers, virtually non-existent, were required. In 1998, David MacKay [14] revisited LDPC codes by developing the Belief Propagation (BP) decoding algorithm [15].

LDPC codes are a class of linear block codes used as an iteration based error-correction coding method. LDPC codes are studied using long codewords, usually greater than 1000 bits. LDPC codes are characterized by their *parity-check matrix* which has a sparse amount of 1's and a dense amount of 0's. If the number of 1's is fixed in each column and row of the parity check matrix, then the LDPC code is known as *regular* [16]. Otherwise the LDPC code will be classified as *irregular* [17].

The overall effectiveness of LDPC coding has allowed for data transmission rates to reach close to the theoretical maximum, determined by the Shannon limit [18]. From [17] very long irregular LDPC codes have been designed to operate within 0.0045 dB of the Shannon limit. LDPC codes have the following characteristics: better performances compared to using Turbo codes when the block length is large, greater flexibility, simple description and resulting theoretical verifiability, lower decoding complexity compared to using Turbo codes, parallel capability, which facilitates hardware implementation, and higher throughput, which results in high-speed decoding. Furthermore, LDPC codes perform extremely well on Binary Symmetric Channels (BSC) [19] and AWGN [11] channels.

3.2 *General Description of LDPC codes*

3.2.1 LDPC codes Definitions

This section defines the notations that will be throughout this thesis for describing messages. For this study each message is represented by log-likelihood ratios (LLR). An LLR is a statistical test for making a decision between two hypotheses based on the value of this ratio. The definitions introduced in this section will discussed in detail in section 3.2.3.

- n is a variable node, a vertex on a bipartite graph, corresponding to each bit in codeword of length N.

- m is a check node, a vertex on a bipartite graph, corresponding to each check equation in a parity check matrix **H** of M rows.
- *M(n)* is a set of check nodes connected to a variable node n.
- *N(m)* is a set of variable nodes connected to a check node m ($m = n - k$).
- $r_{m,n}^{(i)}$ is a message from a check node m to a variable node n at the ith iteration on the bipartite graph. In this case a message serves as information for decoding.
- $q_{n,m}^{(i)}$ is a message from a variable node n to a check node m at the ith iteration on the bipartite graph. In this case a message serves as information for decoding.
- $l_n^{(i)}$ is an LLR output of a coded bit n at the ith iteration and $l_n^{(0)}$ is an initial LLR of the bit from the channel observation.

3.2.2 Parity-check Matrix

Longer block lengths result in using larger parity-check and generator matrices. The complexity of multiplying a codeword with a matrix depends on the density of 1's in the matrix. If there are a greater amount of 1's than 0's in a given matrix, the matrix will be considered dense. For linear block codes the parity-check matrix, $\boldsymbol{H}$, is put in the form of $\boldsymbol{H} = [\mathbf{P}^T \mid \mathbf{I}_{(n-k)}]$ via Gaussian elimination of the generator matrix, $\boldsymbol{G}$, this computation will be quite complicated for longer code lengths. For this parity matrix, $\boldsymbol{H}$, structure $\mathbf{P}^T$ of $(n - k)$ by k dimensions , and represents the parity check bits and $\mathbf{I}_{(n-k)}$ is the $(n - k)$ by $(n - k)$ identity matrix. Also, '|' denotes a concatenation. The generator matrix is determined by $\boldsymbol{G} = [\mathbf{I}_k \mid \mathbf{P}]$. The sub-matrix, $\boldsymbol{P}$, is generally dense, so the encoding complexity will be quite high. By employing LDPC codes, a sparse parity-check matrix is used instead of a generator matrix. The structure of the parity-check matrix used for LDPC coding is shown below (3.1). The parity-check matrix is defined by N X M dimensions.

$$\boldsymbol{H} = [P_A \mid P_B] = \left[\begin{array}{cccc|cccc} P_{A_{11}} & P_{A_{12}} & \cdots & P_{A_{i,j+1}} & P_{B_{11}} & P_{B_{12}} & \cdots & P_{B_{i,j+1}} \\ P_{A_{21}} & P_{A_{22}} & \cdots & P_{A_{i,j+1}} & P_{B_{21}} & P_{B_{22}} & \cdots & P_{B_{i,j+1}} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ P_{A_{il}} & P_{A_{i2}} & \cdots & P_{A_{i,j+1}} & P_{B_{il}} & P_{B_{i2}} & \cdots & P_{B_{i,j+1}} \end{array}\right] \qquad (3.1)$$

The parity matrix contains two major sub-divisions. When multiplied by a column vector codeword, $\underline{c}^T$, using modulo-2 operations, the result should be a column vector of 0's. Otherwise the encoded codeword is not valid for that particular parity-check matrix. Equation (3.2) shows algebraic relationship between a valid codeword and its parity-check matrix. The codeword contains a block of parity check bits, which has the same length as the number of rows as the parity-check matrix. The rate of the code (R) is the number of information bits, k, divided by the number of columns in the matrix, n.

$$H\underline{c}^T = 0 \quad where\ \underline{c}^T = [k_1 k_2 \ldots k_n | k_{1(n-k)} k_{2(n-k)} \ldots k_{n(n-k)}]^T \tag{3.2}$$

The significance of the two major sub-divisions P_A and P_B of the parity matrix is that something is multiplied with the information bits, k, and parity bits, $n - k$, of the codeword, respectively. The process of encoding is determining the set of parity-check bits that, when concatenated with the data bits, will multiply with the parity check matrix and create a column of zeros. The decoding process determines the set of bits that were transmitted with this property based on the received packet.

3.2.3 Tanner (bipartite) Graphs

Tanner graphs [20] are a visual representation that shows the relationship between the parity-check matrix and message nodes. Tanner graph nodes are partitioned into two classes, *check nodes* and *variable nodes*. For LDPC codes the check nodes denote the rows of the parity check matrix, $\boldsymbol{H}$. The check nodes correspond to the parity-check bits of a codeword. The variable nodes denote the columns of the parity-check matrix. The variable nodes correspond to the codeword vector $\underline{c}$. An edge connects a check node to a variable node if a nonzero entry exists in the intersection of the corresponding row (check node) and column (variable node).

Let the degree of every variable node be the degree at every check node. For LDPC coding the amount of connected check nodes to variable (and vice-versa) nodes are very small compared to the code length. The Tanner graph corresponds to a sparse parity-check matrix. Equation (3.3) shows a sample parity-check matrix. The first row of the parity-check matrix, m_1, corresponds to the first check node of the Tanner graph shown in Figure 3.1. The one-to-one mapping shown in Figure 3.1 shows the intersection of the variable node for a given check node. For every check node, an intersection exists between columns (variable nodes) in the parity-check matrix. For example, the row of m_1 is intersected with the columns of n_1, n_2, and n_4. Equivalently, on the Tanner graph the first check node (m_1) are connected to first (n_1), second (n_1), and fourth (n_1) variable nodes.

$$H=\begin{array}{c|cccccc|c} & n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & \\ \hline & 1 & 1 & 0 & 1 & 0 & 0 & m_1 \\ & 0 & 1 & 1 & 0 & 1 & 0 & m_2 \\ & 1 & 0 & 1 & 0 & 0 & 1 & m_3 \end{array} \tag{3.3}$$

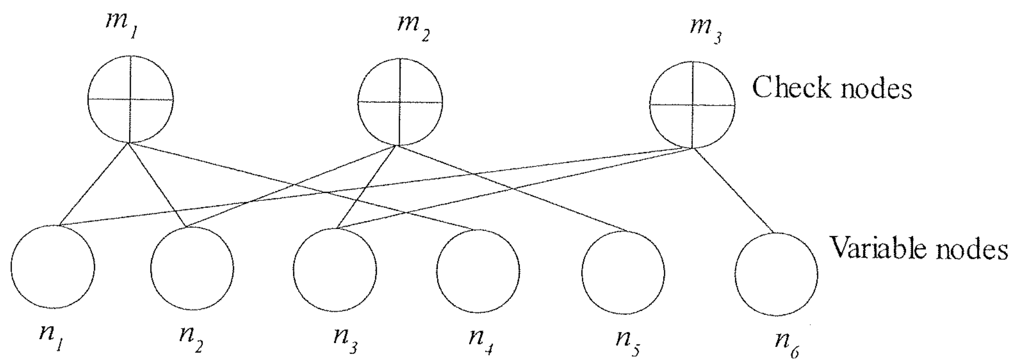

Figure 3.1: Tanner (bipartite) graph

3.2.4 Belief Propagation (BP)

As mentioned in [1], LLR domain messages are probabilistically advantageous, since multiplications are replaced by addition operations and the normalization step is eliminated. The decoding procedure has five steps: initialization, check node update, variable node update and LLR update, hard decision, and orthogonality check.

The LLR-BP decoding algorithms are based on the tanh rule. Other LLR-BP based decoding algorithms include the Gallager [12] and Jacobian approaches [1]. Following the LLR-BP [21][22] based on the tanh rule [21], the algorithm is summarized as follows.

Initialization: each symbol node n is assigned an *a posteriori* LLR $l_n^{(0)}$. For every position *(m,n)* such that $H_{m,n} = 1$, $q_{m,n}^{(0)}=l_n^{(0)}$ and $i = 0$. In LLR-BP decoding, $l_n^{(0)}=\frac{2}{\sigma^2}y_n$, where $y_n=\pm 1+w_n$ and w_n is the AWGN with mean of 0 and variance of $\sigma^2=\frac{N_0}{2}$.

- Step 1) *Check-node update:* For each *m,* and for each $n\in N(m)$, compute

$$r_{m,n}^{(i)} = \ln\left(\frac{1+\prod_{n'\in N(m)\backslash n} \tanh\left(\frac{q_{n',m}^{(i)}}{2}\right)}{1-\prod_{n'\in N(m)\backslash n} \tanh\left(\frac{q_{n',m}^{(i)}}{2}\right)}\right) \tag{3.4}$$

Figure 3.2 shows a pictorial representation of edge of the check node [23] update process (also denoted by equation (3.4)). Every check node represents a single parity-check bit. In every iteration, the output of the check node update messages in LLRs have to be passed to the variable nodes by exclusion of the incoming message from the tanh operation of every outgoing message.

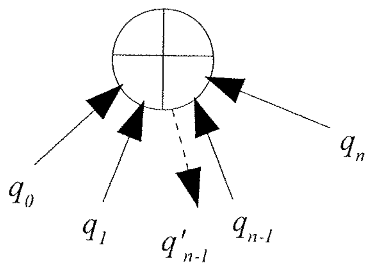

Figure 3.2: Selecting an edge for Check node message [24]

- Step 2a) *Variable-node update*: For each n, compute

$$q_{n,m}^{(i+1)} = q_{n,m}^{(0)} + \sum_{m'\in M(n)\backslash m} r_{m',n}^{(i)} \tag{3.5}$$

The variable node receives LLR values from the AWGN channel (q_n in the Figure 3.3) and from the corresponding check nodes where the variable nodes are connected. In every iteration the messages in LLRs have to be passed to the check nodes by exclusion of the incoming message from the sum of every outgoing message.

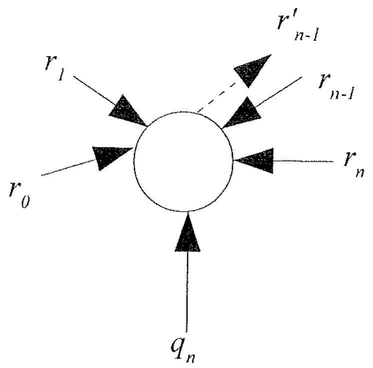

Figure 3.3: Selecting an edge for Variable node message [24]

Step 2b) *LLR update:* for each $m \in \boldsymbol{M}(n)$

$$l_n^{(i+1)} = l_n^{(0)} + \sum_{m \in M(n)} r'^{(i)}_{m,n} \tag{3.6}$$

- Step 3) *Hard decision of LLR outputs:* Quantize $\hat{\underline{c}} = [\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_N]^T$ such that $c_n = 0$ if $l_n^{(i+1)} \geq 0$, and $c_n = 1$ if $l_n^{(i+1)} < 0$.
- Step 4) *Parity check operation:* if $H\hat{\underline{c}}^T = 0$, the orthogonality is verified meaning the decoder has detected zero errors in the reconstructed codeword. Otherwise the decoding algorithm must return to Step 1), and $i = i + 1$. If the algorithm does not halt within I_{max}, a predefined number of maximum iterations, a decoder failure is declared. The block error rate (BLER) is increased by one.

3.3 Example of LDPC Decoding

Consider the following parity check matrix:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

Suppose that the incoming information vector is $\underline{u} = [1 \quad 0 \quad 1]$ (*1* by *k* array). Then taking the first three bits into as the information bits, the corresponding encoded word is $\underline{c} = [1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1]$ (*1* by *n* array). The transmitted codeword is then $\underline{x} = [-1 \quad -1 \quad 1 \quad 1 \quad -1 \quad -1]$ (*1* by *n* array) using BPSK modulation. Due to the effects of the AWGN channel the received vector is

$\underline{y}=[0.5 \quad -1 \quad 1 \quad 1 \quad -1 \quad 0]$ (*1* by *n* array). The extrinsic probability with respect to the AWGN decoder is given by $l_n^{(0)}=\frac{2}{\sigma^2}y_n$, which goes through the detector to give the intrinsic probabilities with respect to the LDPC decoder:

$$l_n^{(0)}=\frac{2}{\sigma^2}y_n \ . \tag{3.8}$$

For AWGN channel the assumed variance is $\sigma^2=1$, the input to the decoder is denoted by $\underline{y}$. The input to the LDPC decoder at the 0th iteration is denoted by vector $\underline{l}^{(0)}$, containing $l_n^{(0)}$ elements, and has a log-likelihood ratio of

$$\underline{l}^{(0)}=[1 \quad -2 \quad 2 \quad 2 \quad -2 \quad 0] \ . \tag{3.9}$$

At the 0th iteration vector $\underline{q}^{(0)}$, containing $q_{n,m}^{(0)}$ elements, is equivalent to $\underline{l}^{(0)}$.

Using the parity-check matrix, ***H***, the log-domain version of the message-passing algorithm is applied. Initially, $r^{(0)}_{m,n}$ is set to zero. Applying the check node message update from equation (3.5) solves for the edge of $q^{(1)}_{1,1}$ shown in (3.10). Figure 3.4 shows the computation tree of the variable node update for first iteration.

$$\begin{aligned} q_{n,m}^{(i+1)}&=q_{n,m}^{(0)}+\sum_{m'\in M(n)\backslash m} r_{m',n}^{(0)} \\ q_{1,1}^{(0+1)}&=1+0 \\ q_{1,1}^{(1)}&=1 \end{aligned} \tag{3.10}$$

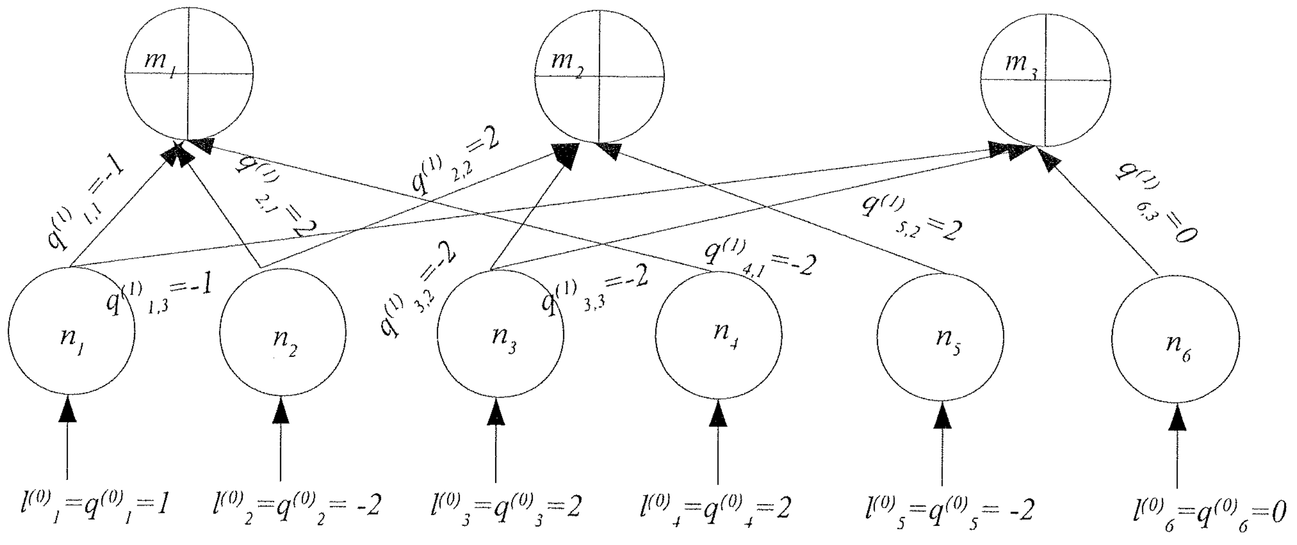

Figure 3.4: Variable node update

The edge of the variable node message update is computed by $r^{(1)}_{m,n}$. For example, an entry of the first row ($r^{(1)}_{1,1}$) is computed by

$$r^{(1)}_{1,1}=\ln\left(\frac{1+\prod_{n'\in N(m)\backslash n}\tanh\left(\frac{-2}{2}\right)\tanh\left(\frac{2}{2}\right)}{1-\prod_{n'\in N(m)\backslash n}\tanh\left(\frac{-2}{2}\right)\tanh\left(\frac{2}{2}\right)}\right)=-1.325\ . \tag{3.11}$$

Using the same formula from (3.11) the remaining check node message updates are determined the same way. Figure 3.5 shows the computation tree of the check node update after the first iteration.

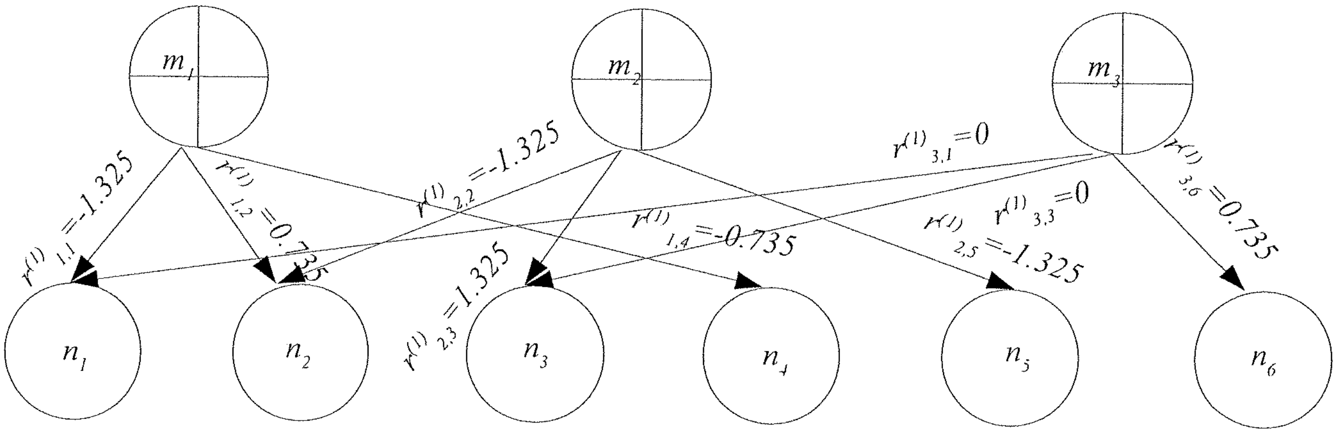

Figure 3.5: Check node update after the first iteration

Using equation (3.6) gives the extrinsic output for the first iteration of the decoder. Adding the extrinsic output for the first iteration of the decoder with the intrinsic input $l^{(0)}_n$ from the variable node message update, denoted by equation (3.6), gives the posterior (LLR update) information for $l^{(i)}_n$.

The remainder of the $\underline{l}^{(1)}$ array is shown in (3.13). In equation (3.12), an example calculating $l_1^{(1)}$ is shown as

$$\begin{aligned} l_n^{(i+1)} &= l_n^{(0)} + \sum_{m \in M(n)} r_{m,n}^{(i)} \\ l_1^{(0+1)} &= 1 + (-1.325) \\ l_1^{(1)} &= -0.325 \end{aligned} \tag{3.12}$$

$$\underline{l}^{(1)} = [-0.325 \quad -2.590 \quad 3.250 \quad 1.265 \quad -3.325 \quad 0.735] \tag{3.13}$$

Figure 3.6 shows the computation tree for the LLR update after the first iteration.

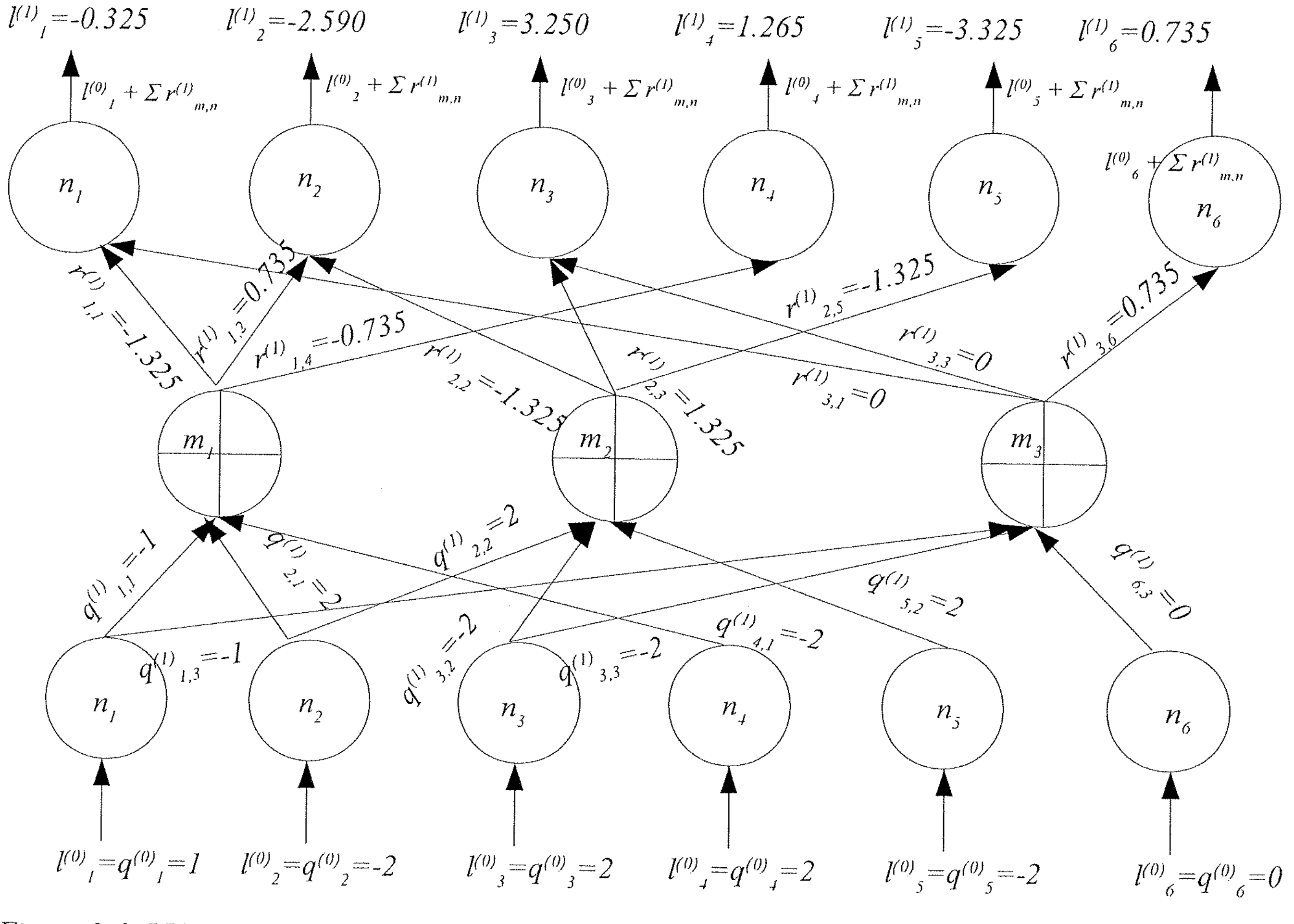

Figure 3.6: LLR update after the first iteration

Once the LLR outputs have been updated the next step is the hard decision. To reiterate, the decision rules are: if $l_n^{(i+1)} \geq 0$ then $c_n = 0$, and if $l_n^{(i+1)} < 0$ then $c_n = 1$. Using these decision rules the reconstructed codeword after one iteration is $\hat{c} = [1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0]$. Next, an orthogonality check (or syndrome) verifies the validity decoded codeword. This operation is performed below

$$H\hat{\underline{c}}^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} . \tag{3.14}$$

The syndrome does not result in an all zero vector. Thus, an error exists in the decoded codeword ($\hat{\underline{c}}^T$) and the iterative decoding procedure must repeated from the check node message update until the codeword is verified or the predefined number of iterations, I_{max} , has been reached. At this point an error will be declared. Usually after an error is declared, for the next iteration the magnitudes of the messages passed along the edges increases. In other words, $|q_{n,m}^{(i+1)}|>|q_{n,m}^{(i)}|$ and $|q_{m,n}^{(i+1)}|>|q_{m,n}^{(i)}|$. As the magnitudes increase the probability of detecting an error decreases for next iteration.

3.4 LDPC codes of the IEEE802.16e standard (WiMAX)

When examining the IEEE 802.16e standard, the parity-check matrix is used for both the encoding and decoding procedures. For the encoding algorithm, using a parity-check matrix rather than a generator simplifies the structure of the algorithm. This standard can accommodate for various code rate and block size adjustments (specified in the A.1). Each LDPC code in the set of LDPC codes is defined by a parity check matrix, $\boldsymbol{H}$, of size m by n, where n is the length of the code and m is the number of parity check bits in the code. The number of systematic (message length) bits is defined by $k = n - m$. The $\boldsymbol{H}$ matrix is defined as

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{P}_{0,0} & \boldsymbol{P}_{0,1} & \boldsymbol{P}_{0,2} & \cdots & \boldsymbol{P}_{0,n_b-2} & \boldsymbol{P}_{0,n_b-1} \\ \boldsymbol{P}_{1,0} & \boldsymbol{P}_{1,1} & \boldsymbol{P}_{1,2} & \cdots & \boldsymbol{P}_{1,n_b-2} & \boldsymbol{P}_{1,n_b-1} \\ \boldsymbol{P}_{2,0} & \boldsymbol{P}_{2,1} & \boldsymbol{P}_{2,2} & \cdots & \boldsymbol{P}_{2,n_b-2} & \boldsymbol{P}_{2,n_b-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{P}_{m_b-1,0} & \boldsymbol{P}_{m_b-1,1} & \boldsymbol{P}_{m_b-1,2} & \cdots & \boldsymbol{P}_{m_b-1,n_b-2} & \boldsymbol{P}_{m_b-1,n_b-1} \end{bmatrix} = \boldsymbol{P}^{H_b} \tag{3.15}$$

where $\mathbf{P}_{ij}$ is one set of z by z matrices. The parity-check matrix is expanded from a binary case matrix $\boldsymbol{H}_b$ of size m_b by n_b, where $n = z \cdot n_b$ and $m = z \cdot m_b$, with z an integer defined in A.1. The base matrix is expanded by replacing each existing 1 in the base matrix with a z by z permutation matrix, and each 0 with a zero matrix of z by z dimension. The base matrix size n_b is an integer equaling 24. For an example of an expansion of the $\boldsymbol{H}_b$ matrix see Appendix A.

The permutations used are circular right shifts, and the set of permutation matrices contains the z by z identity matrix and circular right shifted versions of the identity matrix. Since each permutation matrix is specified by a single circular right shift, the binary base matrix information and permutation replacement information can be combined into a single compact model $\boldsymbol{H}_{bm}$. The $\boldsymbol{H}_{bm}$ matrix model is the same as the binary base matrix $\boldsymbol{H}_b$, with each binary entry (i,j) of the base matrix $\boldsymbol{H}_b$ replaced to create the model matrix $\boldsymbol{H}_{bm}$. Each 0 in $\boldsymbol{H}_b$ is replaced by a negative (e.g. -1) to represent a z by z all-zero matrix, and each 1 in $\boldsymbol{H}_b$ is replaced by a circular shift size $p(i,j) \geq 0$. The model matrix $\boldsymbol{H}_{bm}$ can expand directly to $\boldsymbol{H}$.

The $\boldsymbol{H}_b$ matrix is portioned into two sections, where $\boldsymbol{H}_{b1}$ corresponds to the systematic bits and $\boldsymbol{H}_{b2}$ corresponds to the parity-check bits, such that

$$H_b = \left[(H_{b1})_{m_b x K_b} \middle| (H_{b2})_{m_b x K_b} \right] \tag{3.16}$$

Furthermore, $\boldsymbol{H}_{b2}$ is partitioned into two sections, where $\boldsymbol{h}_b$ vector has odd weight, and $\boldsymbol{H'}_{b2}$ has a dual-diagonal structure with matrix elements at row i, column j equal to 1 for $i = j$, 1 for $i = j + 1$, and 0 elsewhere. The base matrix has $h_b(0) = 1$, $h_b(m_b - 1) = 1$, and a third value $h_b(j)$, $0 < j < (m_b - 1)$ equivalent to 1. The base matrix structure avoids having multiple weight -1 columns in the expanded matrix.

$$H_{b2} = [h_b | H'_{b2}] = \left[\begin{array}{c|cccc} h_b(0) & 1 & & & \\ h_b(1) & 1 & 1 & 0 & \\ . & & 1 & & \\ . & & & 1 & \\ . & & 0 & 1 & 1 \\ h_b(m_b - 1) & & & & 1 \end{array} \right] \tag{3.17}$$

Specifically, the non-zero sub-matrices are circularly right shifted by a particular circular shift value. Each 1 in $\boldsymbol{H'}_{b2}$ is assigned a circular shift of 0, resulting in z by z identity matrix when expanding to $\boldsymbol{H}$. From equation (3.17), the two located at the top and bottom of $\boldsymbol{h}_b$ are assigned equal shift sizes, and the third 1 in the middle of $\boldsymbol{h}_b$ is given an unpaired shift size.

The base model matrix is defined for the largest code length supported by the standard (n = 2304) of each code rate. The set of shifts $\{p(i, j)\}$ in the base model matrix are utilized to determine the shift sizes for all other code lengths of the same code rate. Each base model matrix has $n_b = 24$ columns, and the expansion factor z_f equivalent to $n/24$ for length n. The standard specifies f as the index for a given code rate for, $f = 0, 1, 2, \ldots 18$. For example, code length $n = 2304$ the expansion factor is assigned $z_0 = 96$.

For code rates 1/2, 3/4 A and B, 2/3 B code, and 5/6 code, the shift sizes $\{p(f, i, j)\}$ for a code size corresponding to expansion factor z_f are derived from $\{p(i, j)\}$ by scaling $p(i, j)$ proportionally,

$$P(f,i,j)=\begin{cases} p(i,j),\ p(i,j)\leq 0 \\ \lfloor \dfrac{p(i,j)z_f}{z_0} \rfloor,\ p(i,j)>0 \end{cases} \tag{3.18}$$

where $\lfloor x \rfloor$ denotes a flooring function. For code rate 2/3 A, the shift sizes $\{p(f, i, j)\}$ for a code size corresponding to expansion factor z_f are derived from $\{p(i, j)\}$, using a modulo function

$$P(f,i,j)=\begin{cases} p(i,j),\ p(i,j)\leq 0 \\ mod(p(i,j),z_f),\ p(i,j)>0 \end{cases} \tag{3.19}$$

By using the parity-check matrix structure from equation 3.1, the LPDC code is flexible in that it can accommodate various code rates as well as packet sizes. The encoding of a data packet during transmission generates parity-check bits represented as $\mathbf{p} = (p_0, \ldots, p_{m-1})$ based on an information block $\mathbf{x} = (x_0, \ldots, x_{m-1})$, and transmits the parity-check bits along with the information block. Due to the current symbol set to be encoded and transmitted is contained in the transmitted codeword, the information block is known as systematic (message) bits. The encoder receives the information block $\mathbf{s} = (s_0, \ldots, s_{m-1})$ and utilizes matrix $\boldsymbol{H}_{\mathrm{bm}}$ to determine the parity-check bits. The expanded matrix $\boldsymbol{H}$ is determined from $\boldsymbol{H}_{\mathrm{bm}}$. Since the expanded matrix $\boldsymbol{H}$ contains only binary elements, encoding of a packet can be performed with vector or matrix operations conducted over the Galois field of 2 elements GF(2) [11].

The classical method of encoding requires determining a generator matrix $\boldsymbol{G}$ from $\boldsymbol{H}$ such that $\boldsymbol{G}\boldsymbol{H}^T=0$. A k-bit information block $\mathbf{s}_{1\mathrm{x}k}$ can encoded by the code generator matrix G_{kxn} using the operation $\mathbf{x} = \mathbf{s}\ \mathrm{G}$ to become an n-bit codeword $\mathbf{x}_{1xn}$, with codeword $\mathrm{x} = [\mathbf{s}\ \mathbf{p}] = [s_0, s_1, \ldots, s_{k-1}, p_0, p_1, \ldots, p_{m-1}]$, where $p_0, \ldots, p_{m-1}$ are the parity-check bits; and $s_0, \ldots, s_{m-1}$ are the systematic bits. Encoding an LDPC code from G can be quite complex; whereas, LDPC codes using the IEEE 802.16e standard require very low complexity such that encoding directly from $\boldsymbol{H}$ is possible. The remaining paragraphs of this section will discus direct encoding algorithm.

Encoding is the process of determining a parity sequence **p** associated with a given block of information sequence **s**. To encode, the information block **s** is divided into $k_b = n_b - m_b$ groups of z bits. The grouped **s** will be denoted by **u**,

$$u=[u(0)u(1)...u(k_b-1)] \tag{3.20}$$

where each element of **u** is a column vector as follows

$$u(i)=[s_{iz}s_{iz+1}...s_{(i+1)z-1}]^T \tag{3.21}$$

Utilizing the model matrix $\boldsymbol{H}_{\mathrm{bm}}$, the parity sequence **p** is determined in groups of z. The grouped parity sequence **p** will be denoted by **v**,

$$v=[v(0)v(1)...v(m_b-1)] \tag{3.22}$$

where each element **v** is a column vector as follows

$$v(i)=[p_{iz}P_{iz+1}...P(i+1)z-1]^T \tag{3.23}$$

The encoding process occurs in two steps, (1) initialization, which determined **v**(0), and (2) recursion, which determines **v**(i + 1) from **v**(i), $0 \le i \le m_b - 2$. An expression for **v**(0) can be derived from summing over the rows to $\boldsymbol{H}_{\mathrm{bm}}$ obtain

$$P_{p(x,k_b)}v(0)=\sum_{j=0}^{k_b-1}\sum_{i=0}^{m_b-1}P_{p(i,j)}u(j) \tag{3.24}$$

where x, $0 \le x \le m_b - 2$, is the row index of $\boldsymbol{h}_{\mathrm{bm}}$ where the entry is nonnegative and unpaired, and $\mathbf{P}_i$ represents a z by z identity matrix circularly right shifted by size i. Equation is solved for $v(0)$ by multiplying $P^{-1}_{p(x,k_b)}$, and, $P^{-1}_{p(x,k_b)}=P_{z-p(x,k_b)}$ since $p(x, k_b)$ represents a circular shift. Considering the structure of $\boldsymbol{H}'_{\mathrm{b2}}$, the recursion can be derived as follows,

$$v(1)=\sum_{j=0}^{k_b-1}P_{p(i,j)}u(j)+P_{p(i,k_b)}v(0),\ i=0 \tag{3.25}$$

$$v(i+1)=v(i)+\sum_{j=0}^{k_b-1}P_{p(i,j)}u(j)+P_{p(i,k_b)}v(0),\ i=1,...,m_b-2 \tag{3.26}$$

where

$$P_{-1} = 0_{z \times z} \tag{3.27}$$

Hence all parity bits not in $v(0)$ are determined by evaluation of equation (3.25) for $0 \leq i \leq m_b - 2$. The encoding algorithm is completely described by equations (3.25) to (3.27). These equations are straightforward in terms of standard digital logic architectures. Since the non-zero elements of $p(i, j)$ of $\boldsymbol{H}_{\mathrm{bm}}$ represent circular shift sizes of a vector, all products of the form $\mathbf{P}_{p(i, j)}u(j)$ is implemented by a size-z barrel shifter.

4 Min-Sum Based Decoding Algorithms for LDPC codes

4.1 *Known LDPC Decoding Algorithms*

4.1.1 Min-Sum (MS) Decoding Algorithm

The MS [13] decoding algorithms simplify the check node update (Step 1) in the BP based algorithm. Rather than a computationally involved hyperbolic tangent function, MS decoding check node update selects the minimum input value. The modified Step 1) from the check node update from the BP based algorithm is shown below. All other steps from the LLR-BP algorithm remains intact. The purpose of the MS decoding algorithm is that simplifies the check node update computation. By modifying MS several decoding algorithms can be developed.

Initialization: each symbol node n is assigned an *a posteriori* LLR $l_n^{(0)}$. For every position (m,n) such that $\mathbf{H}_{m,n} = 1$, $q_{m,n}^{(0)}=l_n^{(0)}$ and $i = 0$. From this simplification, $l_n^{(0)}=y_n$ rather than $l_n^{(0)}=\frac{2}{\sigma^2}y_n$ in all MS and MS based decoding algorithms described by removing $\frac{2}{\sigma^2}$, where $\frac{2}{\sigma^2}$ is considered as a factor that does not affect the overall performances and thus can be removed at the decoding procedure.

To calculate the magnitude of a particular outgoing message of a check node, the minimum magnitudes of all the other incoming messages need to be calculated (i.e. minimum of all the magnitude of the messages, except the message which corresponds to the outgoing message). For the sign of the outgoing message of a check node, the product of the signs of all other incoming messages has to be computed. This can be done by first computing the product of the signs of all inputs and then multiplying every outgoing message with the sign of the associated incoming message as shown in equation (4.1).

- Step 1) *Check-node update:* For each m, and for each $n \in N(m)$, compute

$$r_{m,n}^{(i)}=\left(\prod_{n' \in N(m)\setminus n} sign(q_{n',m}^{(i)})\right)\cdot\left(\min_{n' \in N(m)\setminus n} |q_{n',m}^{(i)}|\right) \quad (4.1)$$

- Step 2a) *Variable-node update*: For each n, compute

$$q_{n,m}^{(i+1)} = q_{n,m}^{(0)} + \sum_{m' \in M(n) \setminus m} r_{m',n}^{(i)} \tag{4.2}$$

  Step 2b) *LLR update:* for each $m \in M(n)$

$$l_n^{(i+1)} = l_n^{(0)} + \sum_{m \in M(n)} r_{m,n}^{(i)} \tag{4.3}$$

- Step 3) *Hard decision of LLR outputs:* Quantize $\hat{\underline{c}} = [\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_N]^T$ such that $c_n = 0$ if $l_n^{(i+1)} \geq 0$, and $c_n = 0$ if $l_n^{(i+1)} < 0$.

- Step 4) *Parity check operation:* if $H \hat{\underline{c}}^T = 0$, the orthogonality is verified meaning the decoder has detected zero errors in the reconstructed codeword. Otherwise the decoding algorithm must return to Step 1), and $i = i + 1$. If the algorithm does not halt within I_{max}, a decoder failure is declared. The BLER is increased by one.

4.1.2 Normalized Min-Sum decoding algorithm (NM-MS)

If the check node message output of MS decoding is compared to that of BP for the same inputs, their signs are identical, but the magnitude of MS decoding is larger than that of BP, which implies the overestimation of check node outputs in MS decoding [1]. To alleviate the overestimation, normalizing check node outputs are applied using a normalization factor α (0 < α < 1). The values of α were obtained through Monte Carlo simulations performed in [1]. The modified check node update from MS decoding is shown below.

- Step 1') *Check node update (NM-MS):*

$$r_{m,n}^{(i)} = \alpha \cdot \left(\prod_{n' \in N(m) \setminus n} sign(q_{n',m}^{(i)}) \right) \cdot \left(\min_{n' \in N(m) \setminus n} |q_{n',m}^{(i)}| \right) \tag{4.4}$$

4.1.3 Offset Min-Sum decoding algorithm (OFF-MS)

In OFF-MS decoding [2], the magnitudes of the check node outputs are reduced by subtracting it from an offset factor β from the outputs. The modified check node update from MS decoding is shown below.

- Step 1') *Check node update (OFF-MS):*

$$r_{m,n}^{(i)} = \left(\prod_{n' \in N(m) \setminus n} sign(q_{n',m}^{(i)}) \right) \cdot \max \left(\min_{n' \in N(m) \setminus n} |q_{n',m}^{(i)}| - \beta, 0 \right) \tag{4.5}$$

For the OFF-MS decoding algorithm, a check node output is erased $(r_{m,n}^{(i)} = 0)$ if its magnitude is less than or equal to the offset factor (β).

4.1.4 Self Corrected Min-Sum decoding algorithm (SC-MS)

The SC-MS is an alternative method to NM-MS and OFF-MS. The proposed SC-MS [3] decoding detects unreliable information by the sign fluctuation of the variable node update. Precisely, any variable node update changing its sign between two consecutive iterations is erased, meaning that any such fluctuating message is set to zero. For SC-MS, the check node update is the same as classical MS. However, for the variable node update is modified as shown below, but the LLR update remains the same as MS decoding.

- Step 2a') *Variable node update (SC-MS):*

$$\begin{gathered} q_{n,m}^{(i+1)} = q_{n,m}^{(0)} + \sum_{m' \in M(n) \setminus m} r_{m',n}^{(i)} \\ \text{if } q_{n,m}^{(i)} \neq 0 \text{ and } sign(q_{n,m}^{(i+1)}) \neq sign(q_{n,m}^{(i)}), \text{ then } q_{n,m}^{(i+1)} = 0 \end{gathered} \tag{4.6}$$

For the SC-MS variable node procedure, first the new extrinsic LLR for the current iteration $q_{n,m}^{(i+1)}$ is computed. However, unlike classical MS decoding, this value is stored as an intermediary value. At this moment it should be noted that the message $q_{n,m}^{(i)}$ still contains the value of the previous iteration. Next, the signs of $q_{n,m}^{(i+1)}$ and $q_{n,m}^{(i)}$ that was sent by the variable node, *n*, to the check node, *m,* at the previous iteration are compared. If the two signs are equal the variable node message is updated by $q_{n,m}^{(i+1)} = q_{n,m}^{(i)}$ and this value is sent to the check node, *m*. Though, if the two signs differ, the variable node, *n*, sends an erasure to the check node, *m*, meaning that the variable node message $q_{n,m}^{(i)}$ is set to zero. It should be noted that zero messages are considered to have both negative and positive signs. In other words, whenever the old message $q_{n,m}^{(i)} = 0$, the new variable node message is updated by $q_{n,m}^{(i)} = q_{n,m}^{(i+1)}$.

4.2 Proposed LDPC Decoding Algorithms

In this section, three new decoding algorithms are proposed. Each of these algorithms are an improvement of basic Min-Sum decoding. These new algorithms are obtained by combination and/or modifications of known Min-Sum based algorithms. Advantageously, these proposed decoding algorithms preserve the simple structure of Min-Sum based decoding algorithms.

4.2.1 Normalized and Self-Corrected Min-Sum Decoding Algorithm (NM-SC-MS)

This algorithm combines self-corrected and normalized Min-Sum based decoding algorithms. Thus, both the check and variable node messages updates require modification from the original Min-Sum decoding algorithm. The changes are shown as follows.

- Step 1') *Check node message update for NM-SC-MS decoding:*

$$r_{m,n}^{(i)} = \alpha_s \cdot \left(\prod_{n' \in N(m) \backslash n} sign\left(q_{n',m}^{(i)}\right) \right) \cdot \left(\min_{n' \in N(m) \backslash n} \left| q_{n',m}^{(i)} \right| \right) \tag{4.7}$$

- Step 2a') *Variable node message update for NM-SC-MS decoding:*

$$\begin{gathered} q_{n,m}^{(i+1)} = q_{n,m}^{(0)} + \sum_{m' \in M(n) \backslash m} r_{m',n}^{(i)} \\ if\, q_{n,m}^{(i)} \neq 0 \text{ and } sign(q_{n,m}^{(i+1)}) \neq sign(q_{n,m}^{(i)}), \text{ then } q_{n,m}^{(i+1)} = 0 \end{gathered} \tag{4.8}$$

Compared to normalized Min-Sum decoding, an erasure operation is performed at the variable node message update due to the self-correcting process. As a result, in NM-SC-MS decoding a normalization factor α_s greater than the α employed in the normalized MS decoding is used to avoid too much reduction of the messages in the decoding process.

4.2.2 Offset and Self-Corrected Min-Sum Decoding Algorithm (OFF-SC-MS)

This algorithm is a combination of offset and self-corrected Min-Sum based decoding algorithms. Compared to NM-SC-MS, a subtracting offset factor β_s from each check node output of MS decoding rather than normalizing it. Again, to avoid too much reduction of messages during decoding, the offset factor β_s should be less than β in the offset MS decoding. The following changes are shown (with respect the MS decoding).

- Step 1') *Check node message update for OFF-SC-MS decoding:*

$$r_{m,n}^{(i)}=\left(\prod_{n'\in N(m)\backslash n} sign(q_{n',m}^{(i)})\right)\cdot \max\left(\min_{n'\in N(m)\backslash n}\left|q_{n',m}^{(i)}\right|-\beta_s,0\right) \tag{4.9}$$

- Step 2a') *Variable node message update for OFF-SC-MS decoding:*

$$\begin{gathered}q_{n,m}^{(i+1)}=q_{n,m}^{(0)}+\sum_{m'\in M(n)\backslash m} r_{m',n}^{(i)}\\ if\, q_{n,m}^{(i)}\neq 0 \text{ and } sign(q_{n,m}^{(i+1)})\neq sign(q_{n,m}^{(i)}), \text{ then } q_{n,m}^{(i+1)}=0\end{gathered} \tag{4.10}$$

4.2.3 Variable node Offset Min-Sum Decoding (V-OFF-MS)

This decoding method is a variation of the offset MS decoding, where an offset factor is subtracted from each variable node output. Furthermore, if the magnitude of a variable node output is less than the offset, it goes through another decision process before being erased. The idea behind another decision process is adopted from the erasure process of SC-MS decoding. Although, slight modifications are made to the variable node process there is no need for any additional memory, by keeping the sign of a variable node output over each edge. Precisely, if the magnitude of a variable node output over an edge is less than the offset factor, the algorithm checks whether the sign is identical to the sign of the variable node input over the same edge. If the output is different, the output is erased. By making an erasure over an edge, both conditions of small magnitudes and different signs must be met, which is expected to generate fewer erasures than the SC-MS decoding algorithm.

The decoding structure of the V-OFF-MS based decoding is simpler than SC based decoding algorithms, since this method requires no additional memory to store the messages in the previous iteration. The following shows the check node and variable node processes updates for the V-OFF-MS algorithm.

- Step 1') *Check node message update for OFF-SC-MS decoding:*

$$r_{m,n}^{(i)}=\left(\prod_{n'\in N(m)\backslash n} sign(q_{n',m}^{(i)})\right)\cdot\left(\min_{n'\in N(m)\backslash n}\left|q_{n',m}^{(i)}\right|\right) \tag{4.11}$$

- Step 2a') *Variable node message update for OFF-SC-MS decoding:*

$$
\begin{gathered}
q_{tmp} = q_{n,m}^{(0)} + \sum_{m' \in M(n) \setminus m} r_{m',n}^{(i)} \\
q_{sgn} = sign(q_{tmp}) \\
If\ |q_{tmp}| > |\beta_v|,\ then\ q_{mag} = |q_{tmp}| - \beta_v, \\
\text{else if } sign(r_{m,n}^{(i)}) \neq 0 \text{ and } q_{sgn} \neq sign(r_{m,n}^{(i)}),\ then\ q_{mag} = 0 \\
\rightarrow q_{n,m}^{(i+1)} = q_{sgn} \cdot q_{mag}
\end{gathered}
\tag{4.12}
$$

4.3 Comparison of known decoding versus proposed decoding algorithms

Table 4.1: Advantages of proposed decoding algorithms

Proposed decoding algorithm	Advantages
Normalized and Self Corrected Min-Sum (NM-SC-MS)	- Self-correcting process will result in lower block error rate than NM-MS. - Faster converge than NM-MS
Offset and Self Corrected Min-Sum (OFF-SC-MS)	- Expected to converge faster than NM-SC-MS since simpler operations (i.e. check node outputs are reduced by subtraction rather than multiplication). - Lower block error rate and faster converge than OFF-MS
Variable node Offset Min-Sum (V-OFF-MS)	- Check magnitudes and signs over variable node edges without the addition of memory. - Has a simpler decoding structure than SC decoding algorithms.

4.4 Parameters Employed for the LDPC Decoding Algorithms

To make the simulations appear practical, irregular LDPC codes are applied to the IEEE802.16e standard. The simulation parameters are listed below.

- Code rates R= 1/2, 2/3, 3/4. Codeword lengths N= 1152, 1728, and 2304.
- BPSK modulation over AWGN channel.
- $\alpha = 0.8$ for NM-MS and $\beta = 0.15$ OFF-MS.
- $\alpha_s = 0.92$ for NM-SC-MS and $\beta_s = 0.08$ for OFF-SC-MS.
- $\beta_v = 0.15$ and for V-OFF-MS.
- Maximum Iteration number $I_{max} \leq 500$.

5 Simulations and Discussions

In this section several simulations are shown to compare the performances of known and proposed LDPC decoding algorithms. For this study, known and proposed algorithms will be compared by decoding structure. Moreover, SC-MS will be compared with OFF-MS-SC and NM-MS-SC, and NM-MS and OFF-MS will be compared with V-OFF-MS.

5.1 Convergence of block error rate

For all simulations shown in this chapter all plot legends are organized as follows: the known decoding algorithms are the three listed in the top of each graph legend, and the proposed algorithms are three listed at the bottom of each graph legend. Also, each proposed decoding algorithm is asterisked.

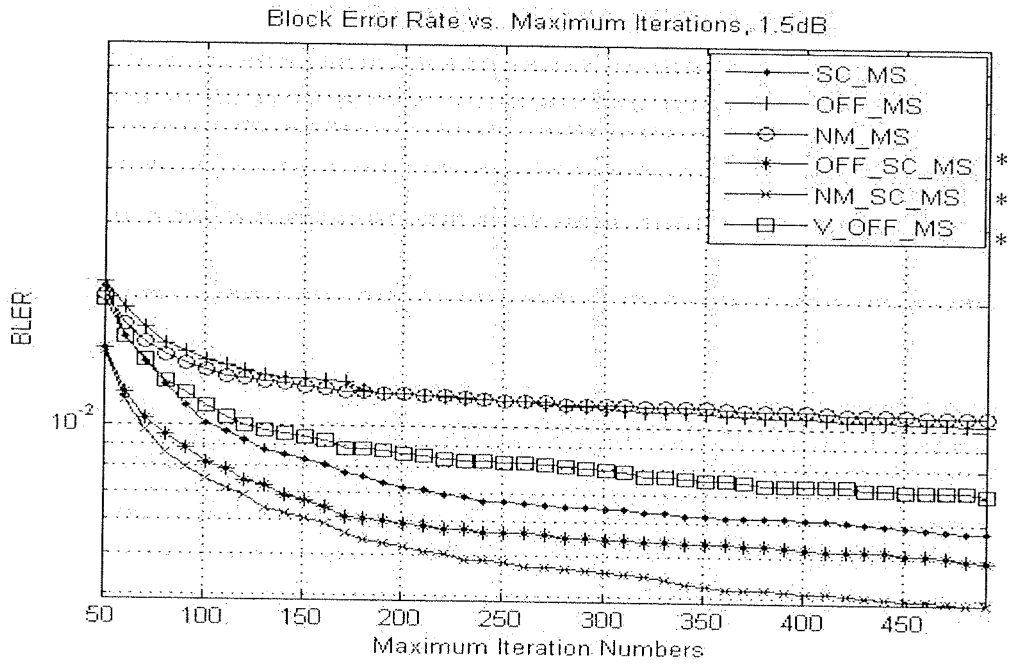

Figure 5.1: Convergences of block error rate (BLER) at R= 1/2, N= 2304 and E_b/N_o at 1.5 dB

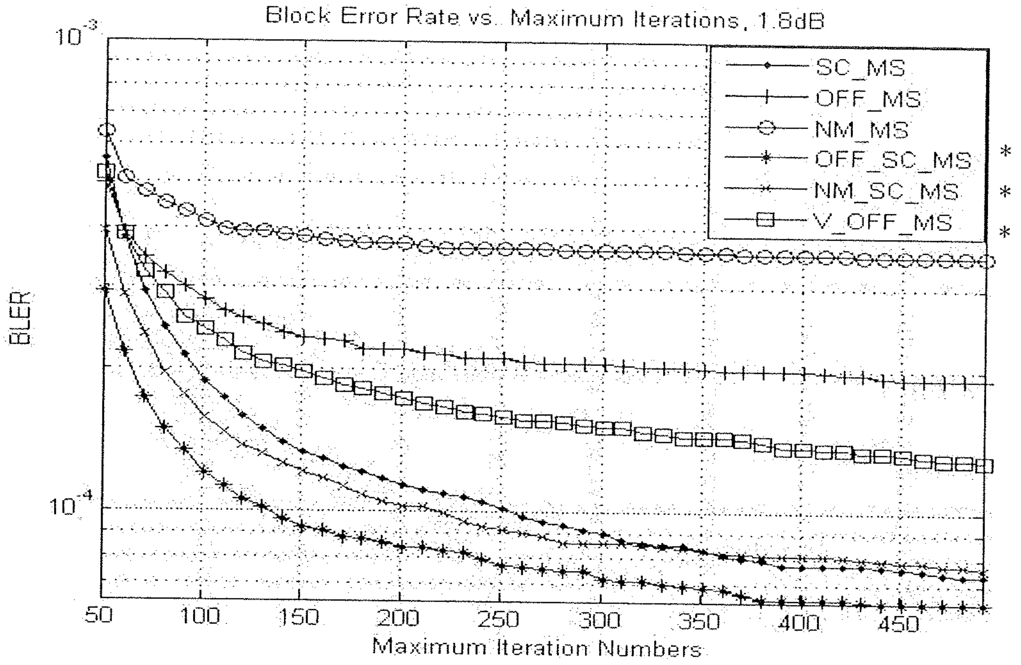

Figure 5.2: Convergences of block error rate (BLER) at R= 1/2, N= 2304 and E_b/N_o at 1.8 dB

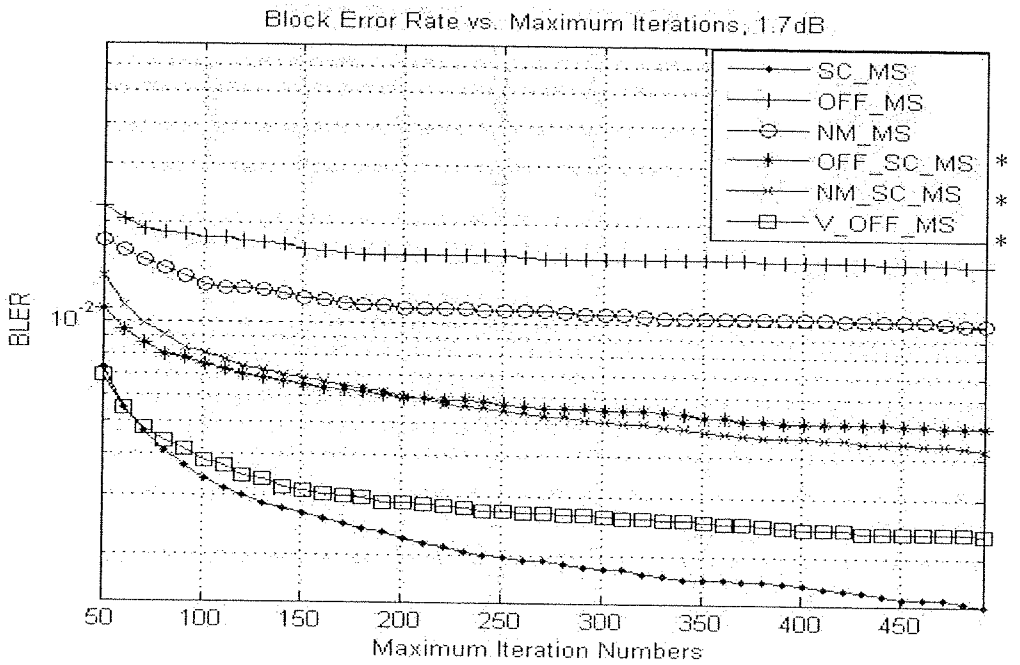

Figure 5.3: Convergences of block error rate (BLER) at R= 1/2, N= 1728 and E_b/N_o at 1.7 dB

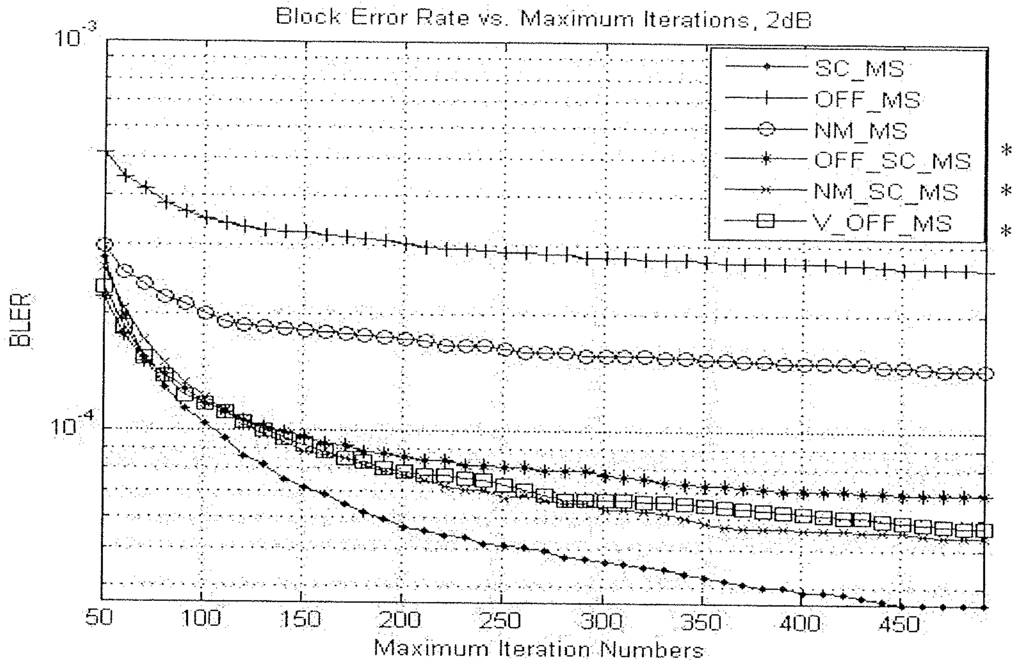

Figure 5.4: Convergences of block error rate (BLER) at R= 1/2, N= 1728 and E_b/N_o at 2.0 dB

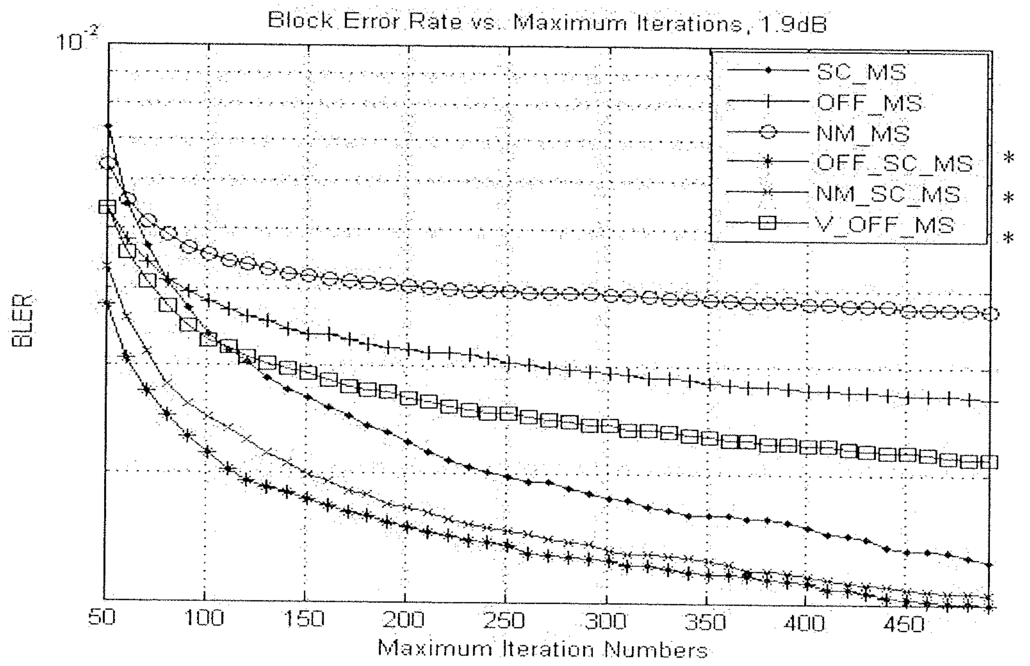

Figure 5.5: Convergences of block error rate (BLER) at R= 1/2, N= 1152 and E_b/N_o at 1.9 dB

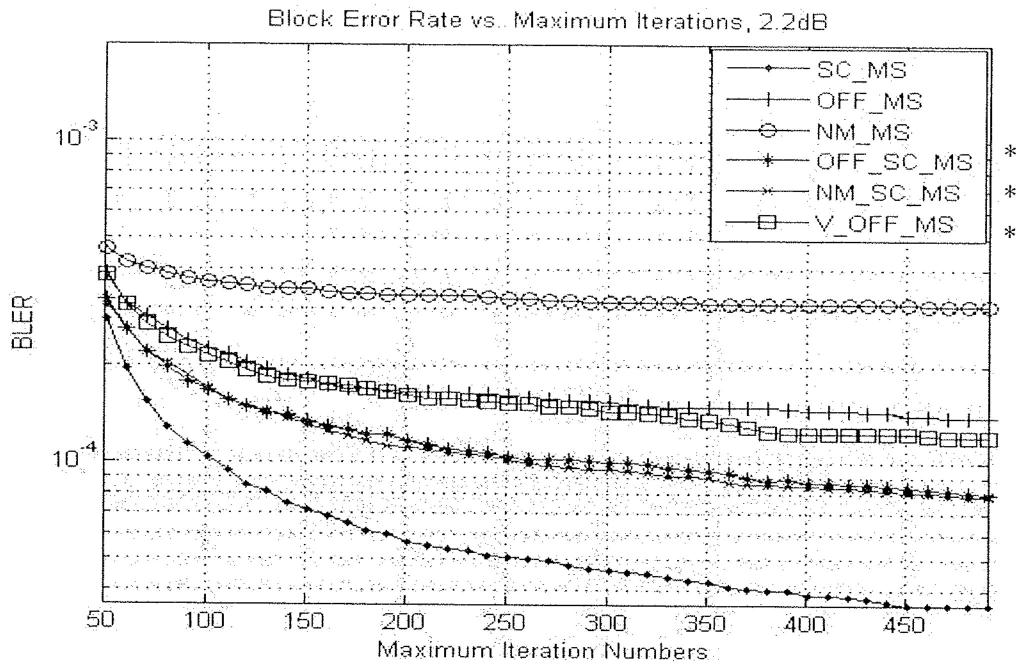

Figure 5.6: Convergences of block error rate (BLER) at R= 1/2, N= 1152 and E_b/N_o at 2.2 dB

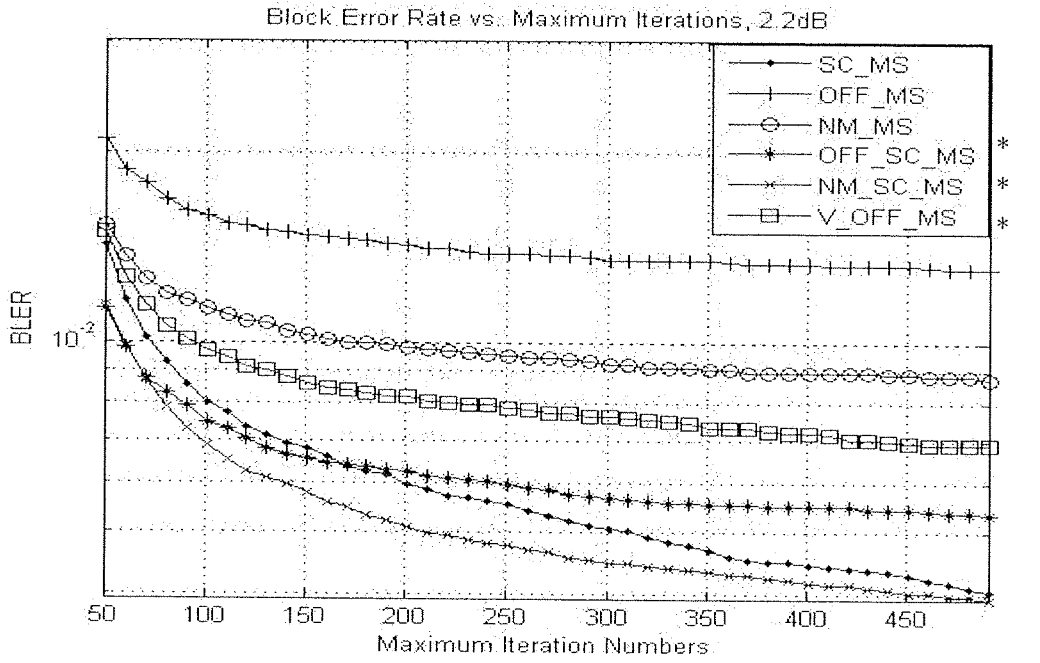

Figure 5.7: Convergences of block error rate (BLER) at R= 2/3, N= 2304 and E_b/N_o at 2.2 dB

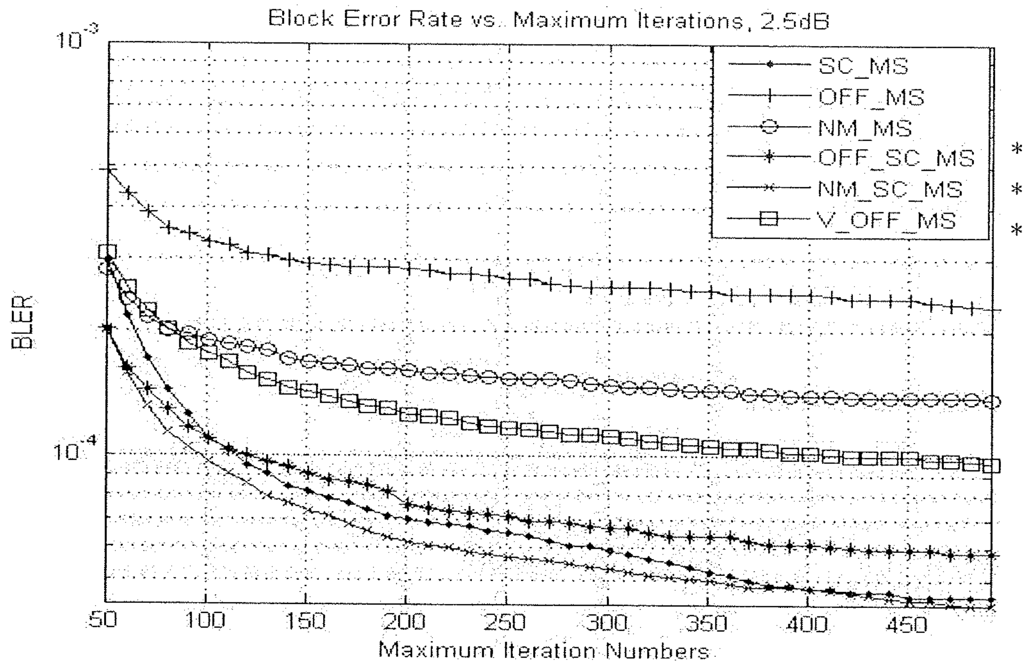

Figure 5.8: Convergences of block error rate (BLER) at R= 2/3, N= 2304 and E_b/N_o at 2.5 dB

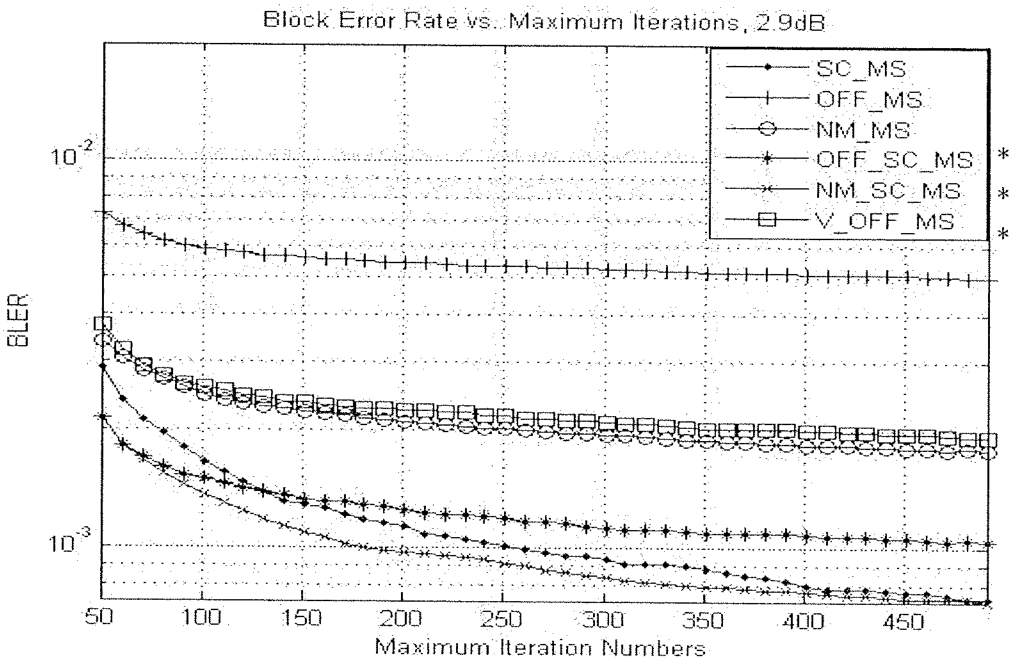

Figure 5.9: Convergences of block error rate (BLER) at R= 3/4, N= 2304 and E_b/N_o at 2.9 dB

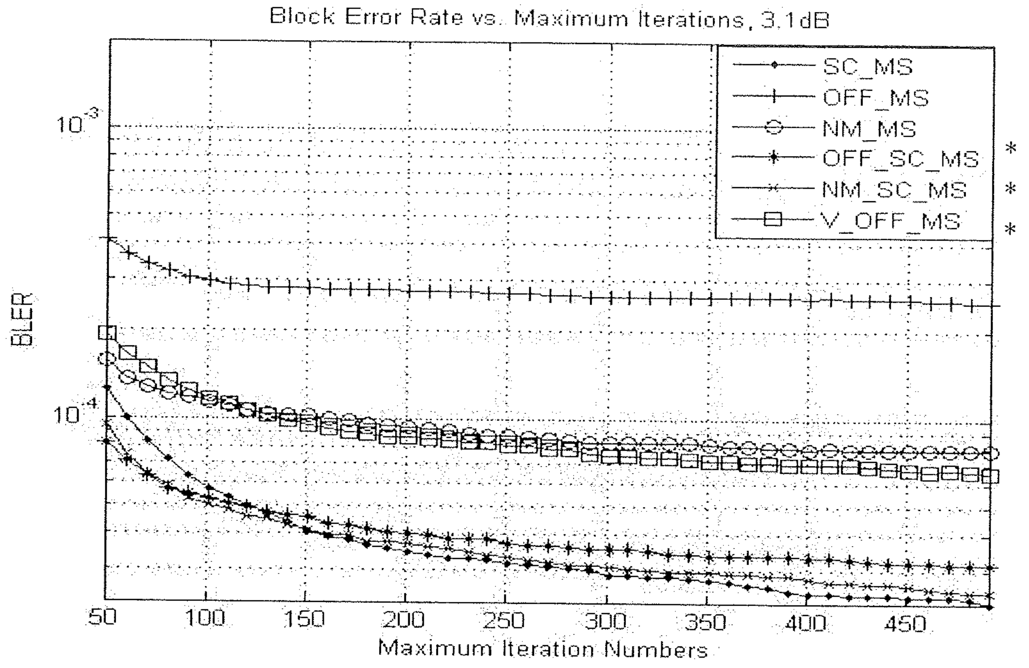

Figure 5.10: Convergences of block error rate (BLER) at R= 3/4, N= 2304 and E_b/N_o at 3.1 dB

Figures 5.1 and 5.2 showed the convergences of the BLER of the MS based decoding algorithms at a medium (1.5 dB) and high (1.9 dB) E_b/N_o, respectively. For this simulation R is equal to 1/2 and N= 2304. Figure 5.1 showed that SC based Min-Sum decoding algorithms improved as I_{max} increased. For the SC based Min-Sum decoding algorithms both NM-SC-MS and OFF-SC-MS converged faster than SC-MS between $50 \leq I_{max} \leq 150$. For the case on non-SC based MS decoding algorithms V-OFF-MS converged the fastest. In Figure 5.2, SC based MS decoding algorithms improved as I_{max} increased. OFF-SC-MS decoding converged faster than than SC-MS. However, SC-MS converged faster than NM-SC-MS decoding. For non-SC based Min-Sum decoding V-OFF-MS converged the fastest.

Figure 5.3 and 5.4 showed the convergences of the BLER of the MS based decoding algorithms at a medium (1.7 dB) and high (2.0 dB) E_b/N_o, respectively. For this simulation R is equal to 1/2 and N= 1728. In Figure 5.3, SC-MS decoding converged the fastest. However, V-OFF-MS converged faster than both the OFF-SC-MS and NM-SC-MS decoding algorithms. Although, both OFF-SC-MS and NM-SC-MS converged faster than the OFF-MS and NM-MS decoding algorithms. In Figure 5.4, SC-MS converged the fastest. NM-SC-MS converged faster than V-OFF-MS decoding. Although, V-OFF-MS converged faster than OFF-SC-MS decoding. All proposed decoding algorithms converged faster than both OFF-MS and NM-MS decoding.

Figures 5.5 and 5.6 showed the convergences of the BLER of the MS based decoding algorithms at a medium (1.9 dB) and high (2.2 dB) E_b/N_o, respectively. The code rate, R, is equal to 1/2 and block length, N, is equal to 1152. A medium size block length from the IEEE802.16e standard is considered for this case. Figure 5.5 showed that SC-based decoding improved as I_{max} increased. This is particularly true for SC-MS which had the steepest slope. For SC based Min-Sum decoding both OFF-SC-MS and NM-SC-MS converged faster than SC-MS decoding. Between $50 \leq I_{max} \leq 100$, V-OFF-MS converged faster than SC-MS decoding. For non-SC based MS decoding algorithms V-OFF-MS converged the fastest. Figure 5.6 showed that SC-MS converged faster than both OFF-SC-MS and NM-SC-MS decoding. Though, for non-SC based Min-Sum decoding V-OFF-MS converged the fastest.

Figures 5.7 and 5.8 showed the convergences of the BLER of the MS based decoding algorithms at a medium (2,2 dB) and high (2.5 dB) E_b/N_o, respectively. The code rate, R, is equal to 2/3 and block length, N, is equal to 2304. Figure 5.7 showed that SC based Min-Sum decoding converged the fastest. NM-SC-MS converged faster than SC-MS decoding. However, SC-MS converged faster than OFF-SC-MS. For non-SC based Min-Sum decoding algorithms V-OFF-MS converged the fastest. The algorithmic performances in Figure 5.8 exhibited the exact same trend from Figure 5.7.

Figures 5.9 and 5.10 showed the convergences of the BLER of the MS based decoding algorithms at a medium (2.9 dB) and high (3.1 dB) E_b/N_o, respectively. The code rate, R, is equal to 3/4 and block length, N, is equal to 2304. From Figure 5.9, the SC based MS decoding algorithms converged the fastest. SC-MS converged fastest at $I_{max} = 500$. For non-SC based MS decoding NM-MS converged the fastest. In Figure 5.10, the exact same trend can be observed for SC based Min-Sum decoding algorithms from Figure 5.9. Though, for non-SC based Min-Sum decoding V-OFF-MS converged the fastest.

5.2 Average Iteration Numbers

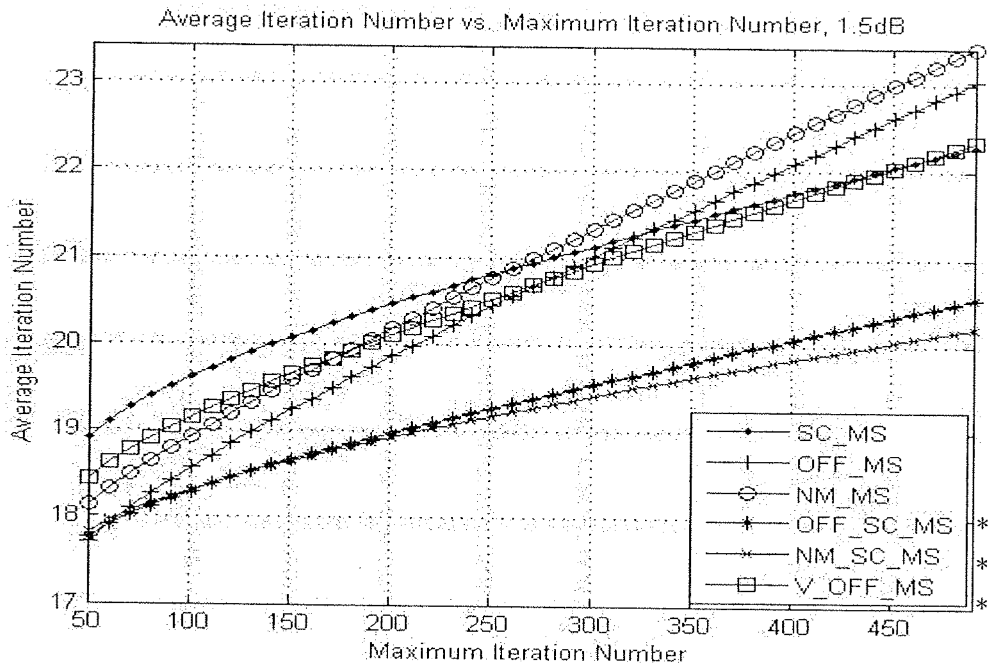

Figure 5.11: Average iteration numbers for various I_{max} at R= 1/2, N= 2304, and E_b/N_o at 1.5 dB

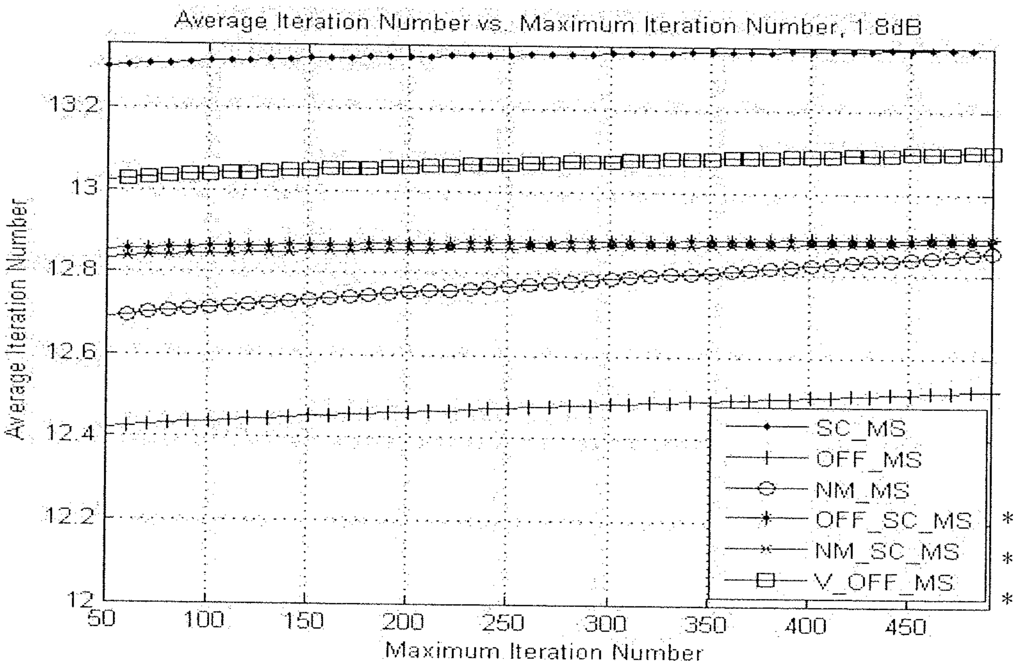

Figure 5.12: Average iteration numbers for various I_{max} at R= 1/2, N= 2304, and E_b/N_o at 1.8 dB

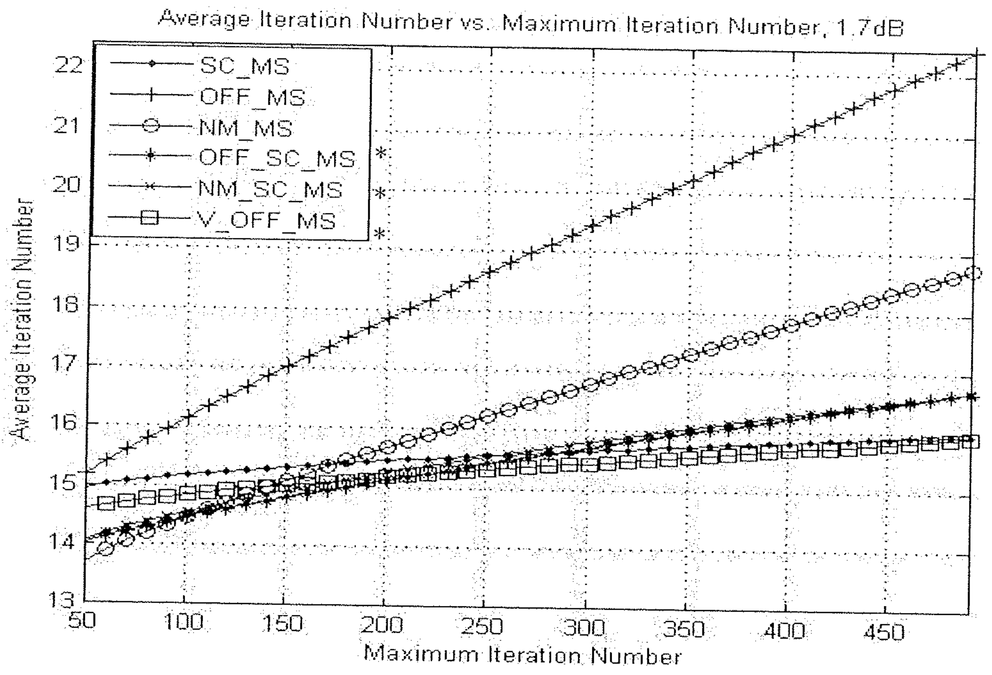

Figure 5.13: Average iteration numbers for various I_{max} at R= 1/2, N= 1728, and E_b/N_o at 1.7 dB

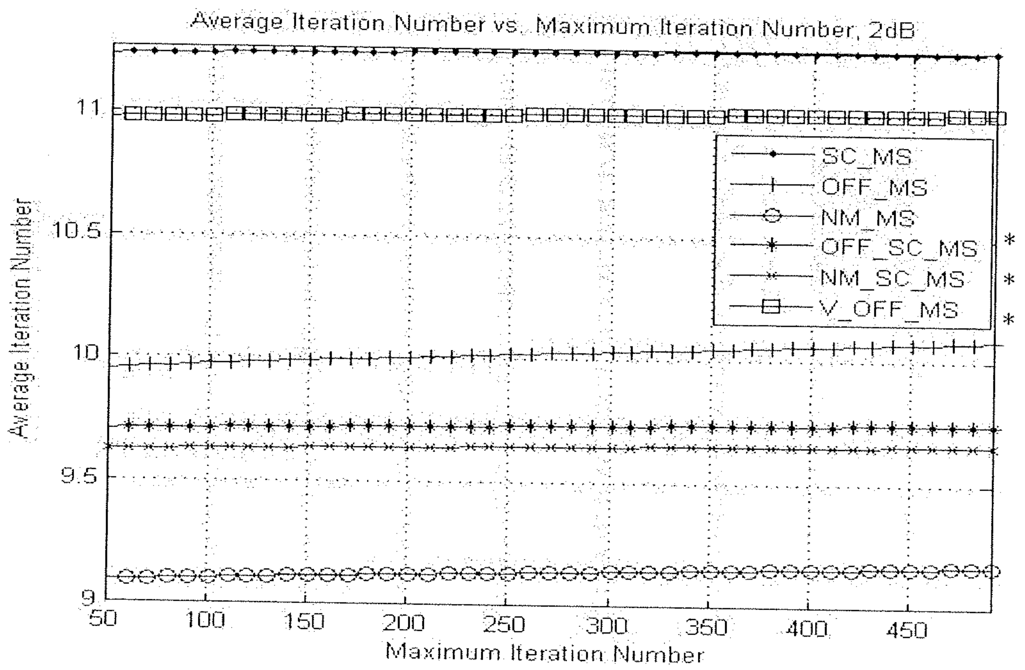

Figure 5.14: Average iteration numbers for various I_{max} at R= 1/2, N= 1728, and E_b/N_o at 2.0 dB

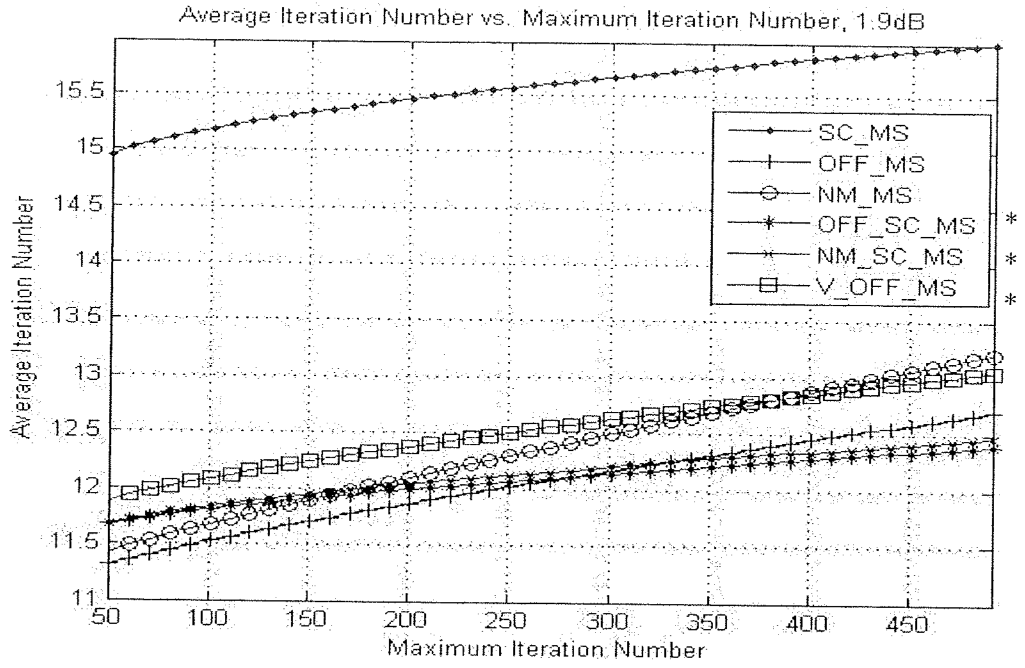

Figure 5.15: Average iteration numbers for various I_{max} at $R= 1/2$, $N= 1152$, and E_b/N_o at 1.9 dB

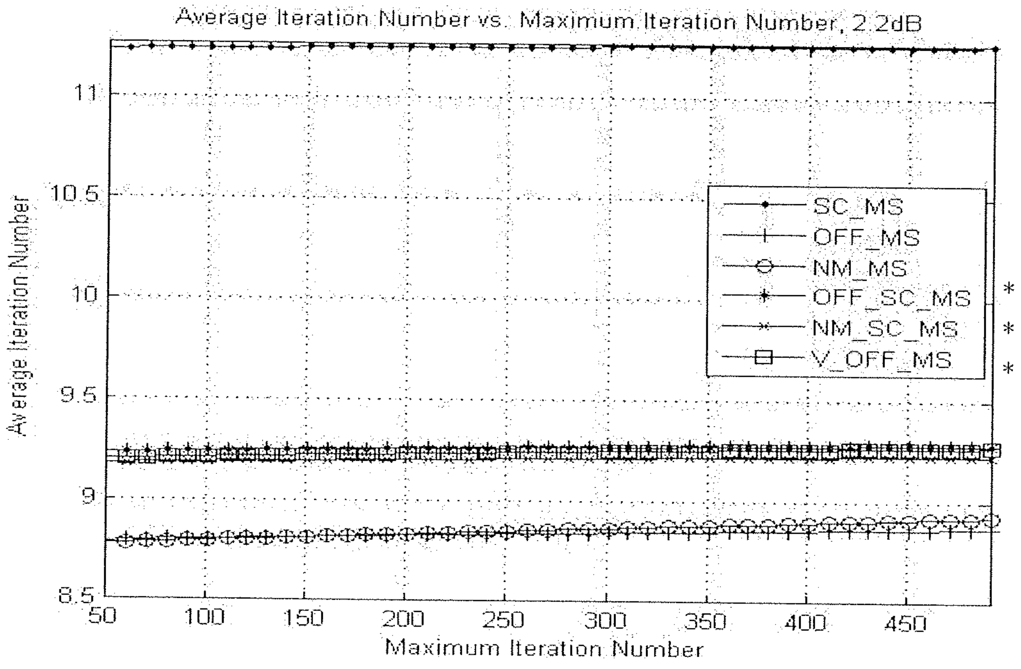

Figure 5.16: Average iteration numbers for various I_{max} at $R= 1/2$, $N= 1152$, and E_b/N_o at 2.2 dB

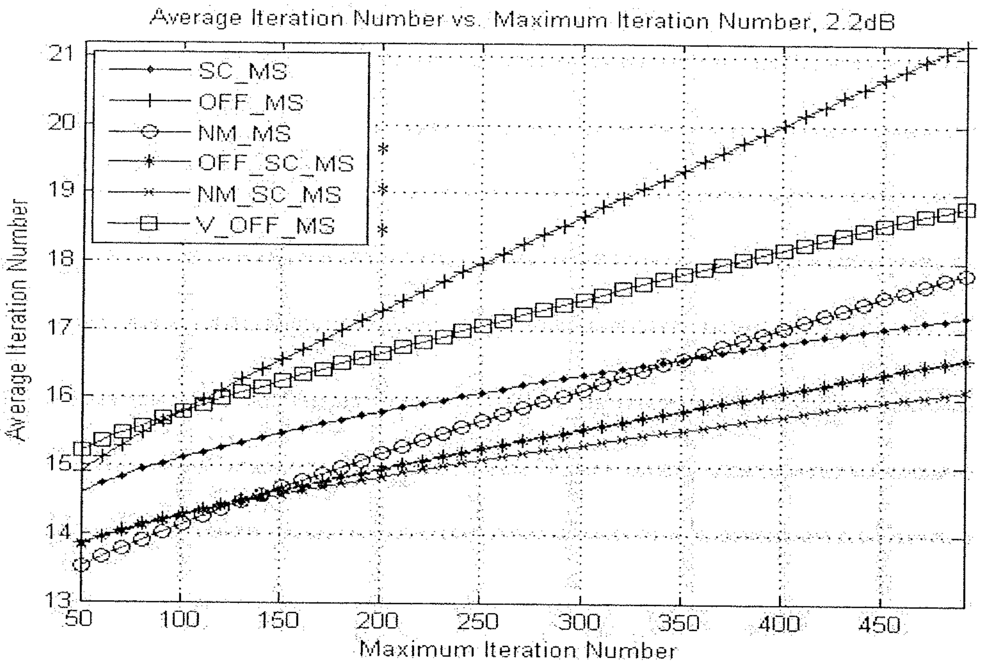

Figure 5.17: Average iteration numbers for various I_{max} at $R= 2/3$, $N= 2304$, and E_b/N_o at 2.2 dB

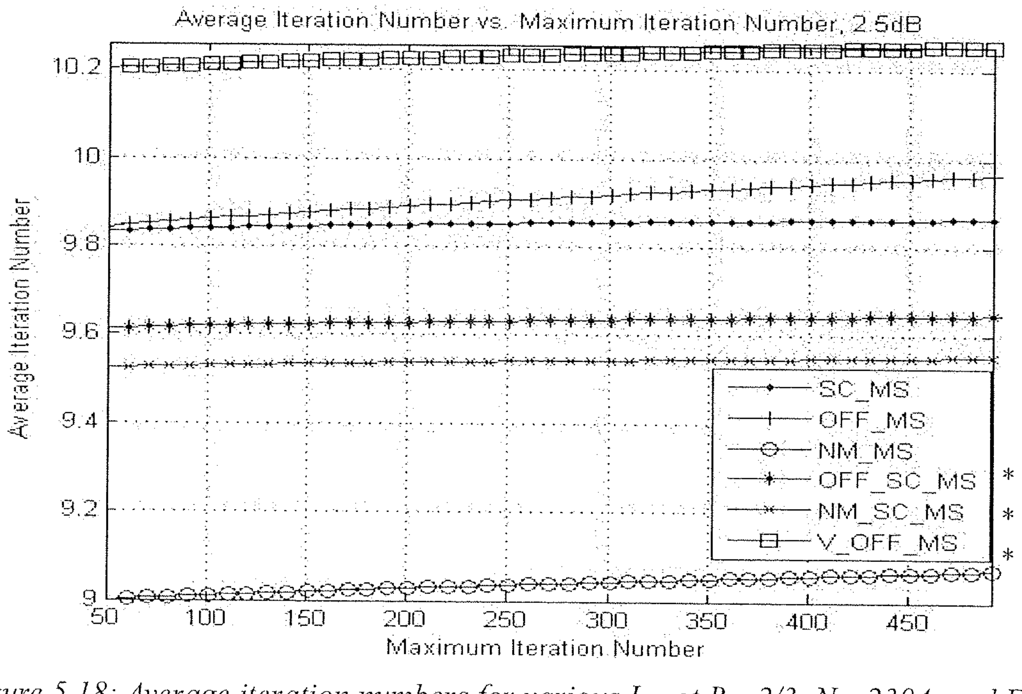

Figure 5.18: Average iteration numbers for various I_{max} at $R= 2/3$, $N= 2304$, and E_b/N_o at 2.5 dB

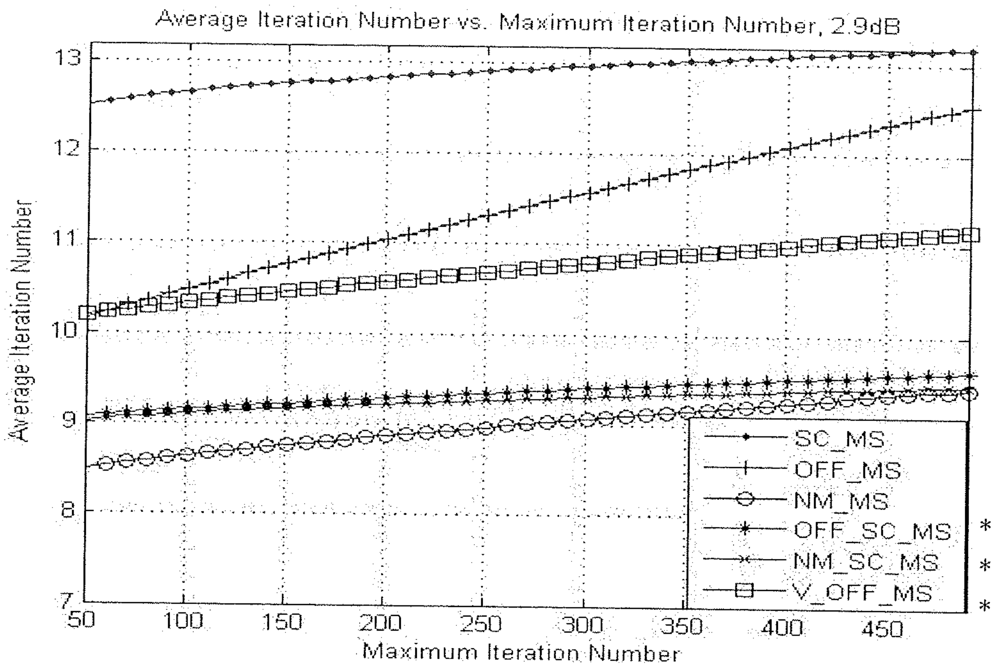

Figure 5.19: Average iteration numbers for various I_{max} at $R= 3/4$, $N= 2304$, and E_b/N_o at 2.9 dB

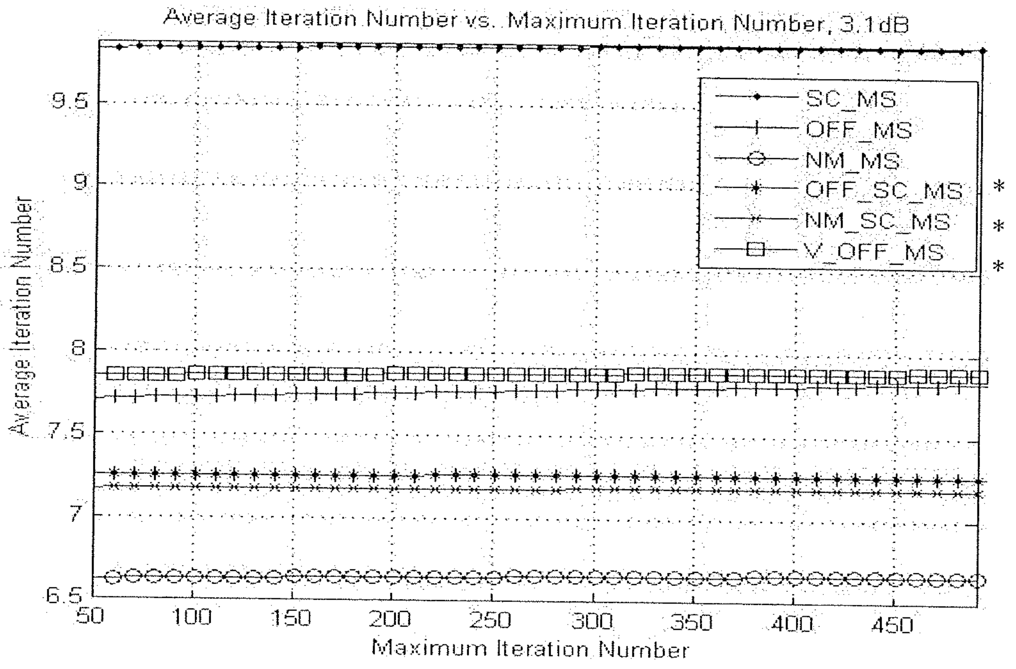

Figure 5.20: Average iteration numbers for various I_{max} at $R= 3/4$, $N= 2304$, and E_b/N_o at 3.1 dB

Figures 5.11- 5.20 shows the average iteration numbers. For this calculation, both corrected and erroneous blocks are considered for determining the mean. Equation 5.1 shows how to calculate the average iteration number

$$AIN = \frac{\sum_{i=1}^{T} I_i(I_{max})}{T} \quad where\ 1 \geq I_i \geq I_{max} \quad . \tag{5.1}$$

From equation 5.1 T denotes the total number of blocks transmitted, and I_i indicates the iteration number where the error was resolved.

This information gives an idea how well each algorithm may fare on a fixed point simulation, using low level computer, based off this software model. Similar to figures 5.11 – 5.20, the average iteration numbers will be taken at the medium and high E_b/N_o values, respectively.

Figures 5.11 and 5.12 showed the average iteration number for R is 1/2 and N= 2304. For Figure 5.11, between $50 \leq I_{max} \leq 250$, SC-MS decoding required the greatest number of iterations. For SC based MS decoding algorithms both OFF-SC-MS and NM-SC-MS had the two lowest average iteration numbers. For non-SC MS based decoding algorithms V-OFF-MS had the lowest number of average iterations. In Figure 5.12, both OFF-MS and NM-MS decoding required the lowest average number of iterations. NM-SC-MS, OFF-SC-MS, and V-OFF-MS decoding required, on average, required lower number of iterations than SC-MS decoding.

Figures 5.13 and 5.14 showed the average iteration number for R is 1/2 and N= 1728. In Figure 5.13, V-OFF-MS decoding had the lowest average number of iterations for any SC and non-SC Min-Sum based decoding algorithm. SC-MS did have a lower number of iterations, on average, than the NM-SC-MS and OFF-SC-MS decoding algorithms. In Figure 5.14, NM-MS required less number of average iterations than both the NM-SC-MS and OFF-SC-MS decoding algorithms. V-OFF-MS had the second highest average number of iterations, only less than the SC-MS decoding algorithm.

Figures 5.15 and 5.16 showed the average iteration number for R is 1/2 and N= 1152. In Figure 5.15, once $I_{max} \geq 250$, both OFF-SC-MS and NM-SC-MS decoding algorithms had the lowest average number of iterations. However, OFF-MS had a lower number of average iterations than the V-OFF-MS decoding algorithm. V-OFF-MS had a lower number of iterations than both the NM-MS and SC-MS decoding algorithms. In Figure 5.16, SC-MS decoding required the greatest average number of iterations. V-OFF-MS decoding had a lower number of average iterations than OFF-SC-MS, but higher than NM-SC-MS decoding. However, NM-MS and OFF-MS required, on average, the lowest number of iterations.

Figures 5.17 and 5.18 showed the average iteration number for R is 2/3 and N= 2304. In Figure 5.17, at I_{max} = 500, SC based Min-Sum decoding algorithms had the lowest average iteration numbers. OFF-SC-MS and NM-SC-MS had lower iteration numbers than MS-SC decoding. For non-SC based MS decoding algorithms V-OFF-MS had a higher average number of iteration number than NM-MS. In Figure 5.18, NM-MS decoding required the least number of average iterations. The NM-SC-MS and OFF-SC-MS decoding algorithms had the second and third lowest average iteration numbers, respectively. However, V-OFF-MS decoding required the highest average number of iterations. Though, on average, 1.6 iterations more than NM-MS decoding between $50 \leq I_{max} \leq 500$.

Figures 5.19 and 5.20 showed the average iteration number for R is 3/4 and N= 2304. In Figure 5.19, NM-MS decoding required the lowest average iteration number. After that, SC-MS and OFF-SC-MS decoding algorithms had the second and third lowest average iteration numbers, respectively. In terms of the V-OFF-MS decoding performance, it had a lower average iteration number compared to both the OFF-MS and SC-MS decoding algorithms. In Figure 5.20, the NM-MS, OFF-SC-MS, NM-SC-MS, and SC-MS decoding algorithms had the same rank from Figure 5.19. The differences between Figure 5.20 and Figure 5.19 is OFF-MS required a lower average number than V-OFF-MS decoding.

5.3 Block Error Rate Performance

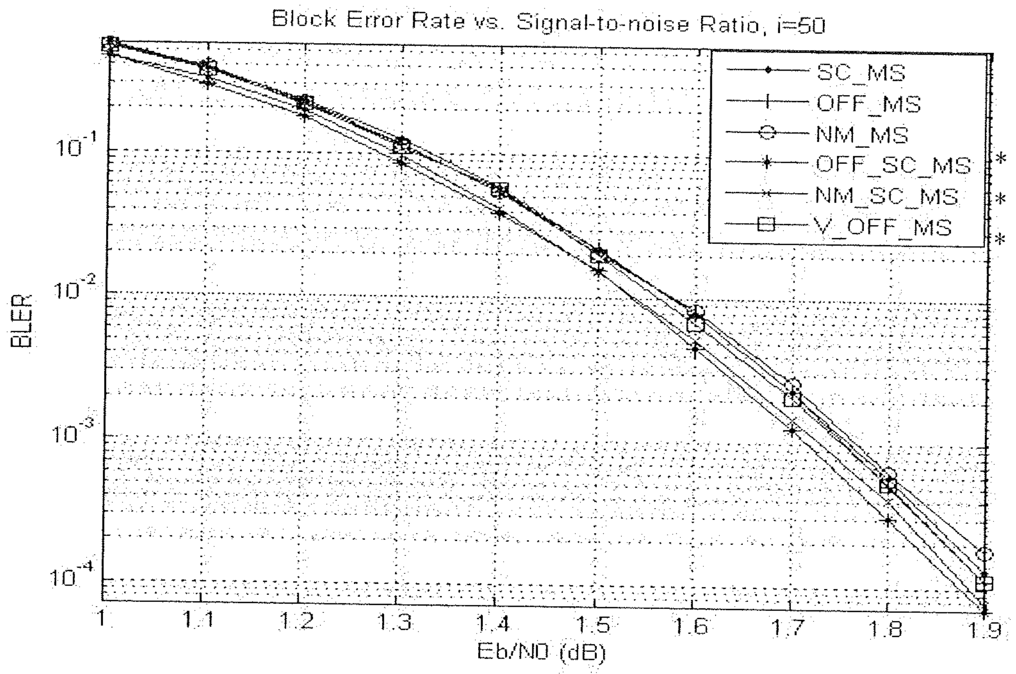

Figure 5.21: Block error rate performance at R= 1/2, N= 2304, and I_{max}= 50

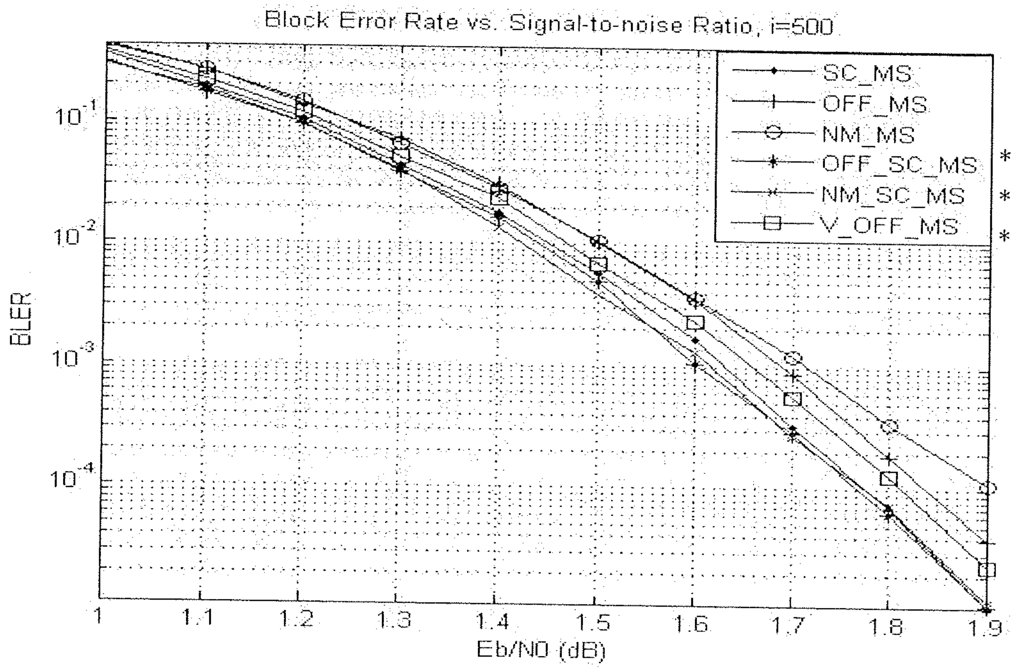

Figure 5.22: Block error rate performance at R= 1/2, N= 2304, and I_{max}= 500

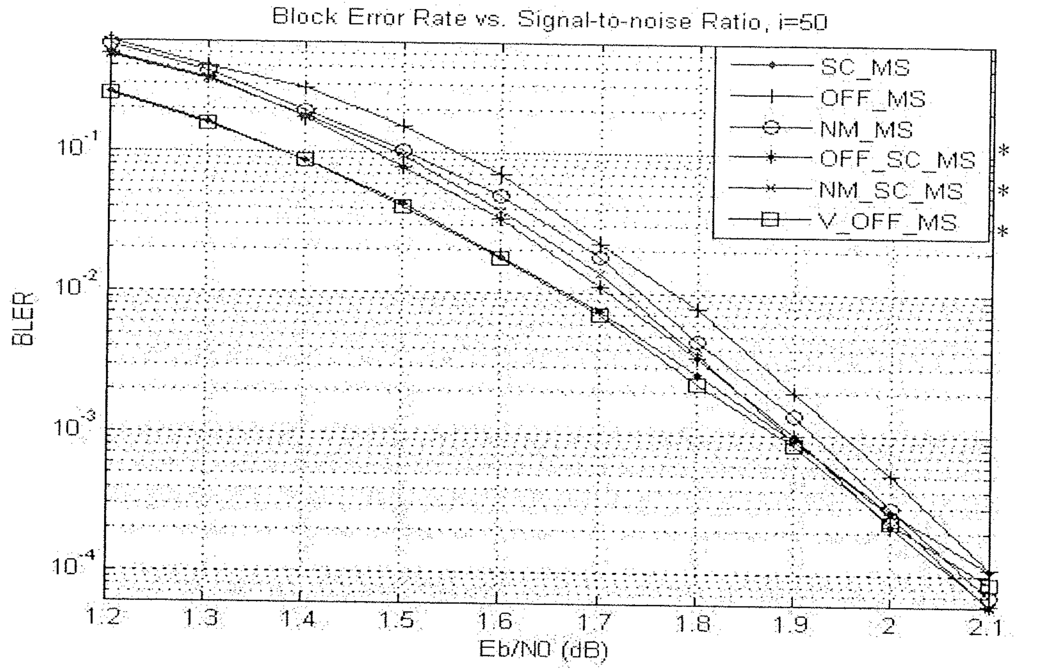

Figure 5.23: Block error rate performance at R= 1/2, N= 1728, and I_{max}= 50

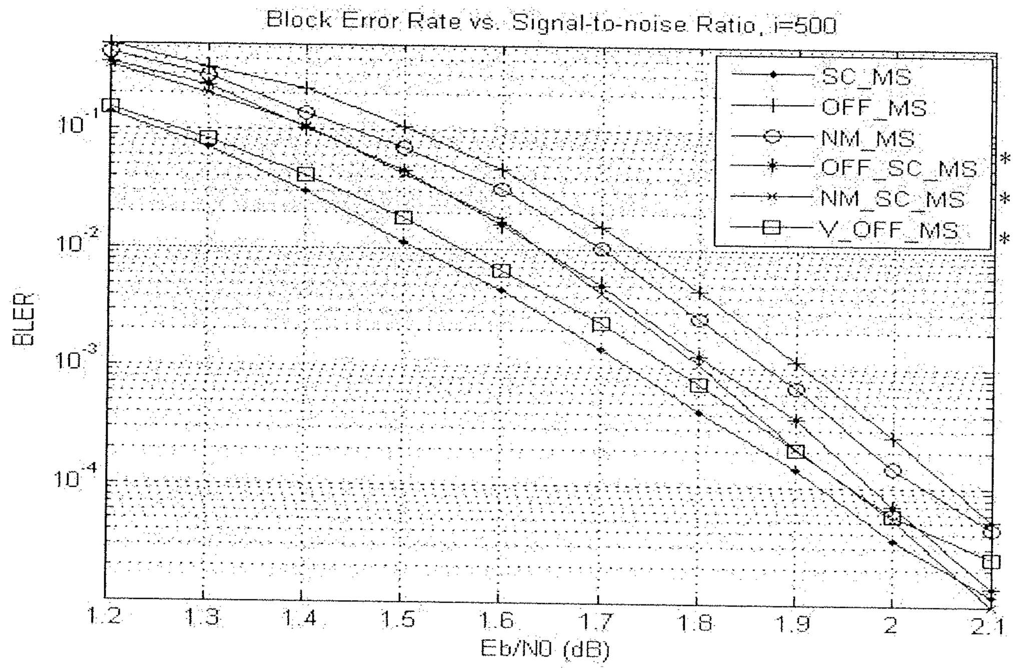

Figure 5.24: Block error rate performance at R= 1/2, N= 1728, and I_{max}= 500

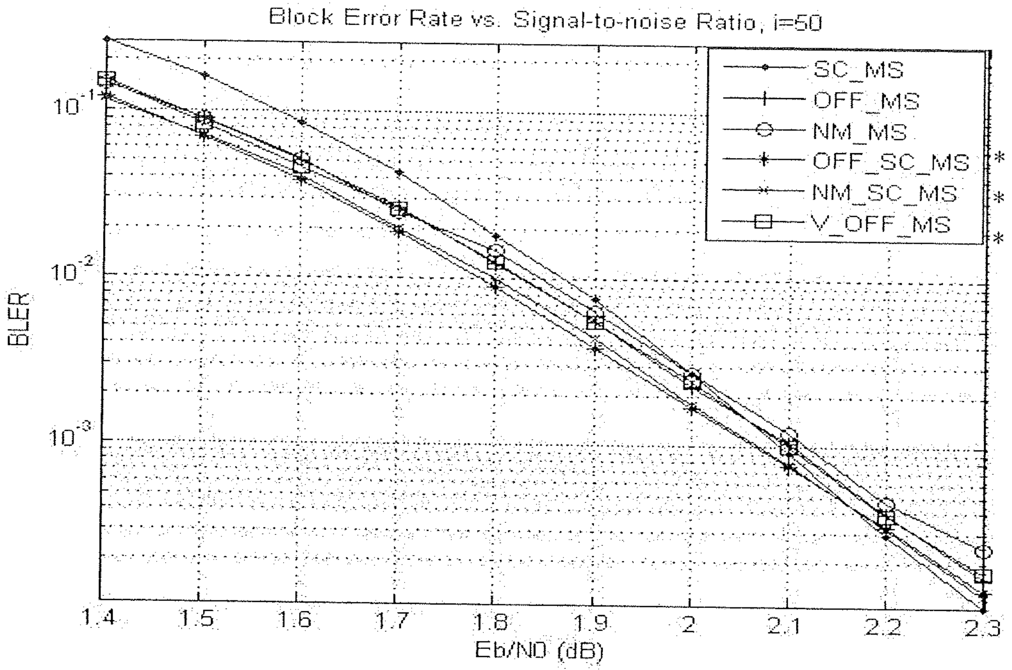

Figure 5.25: Block error rate performance at R= 1/2, N= 1152, and I_{max}= 50

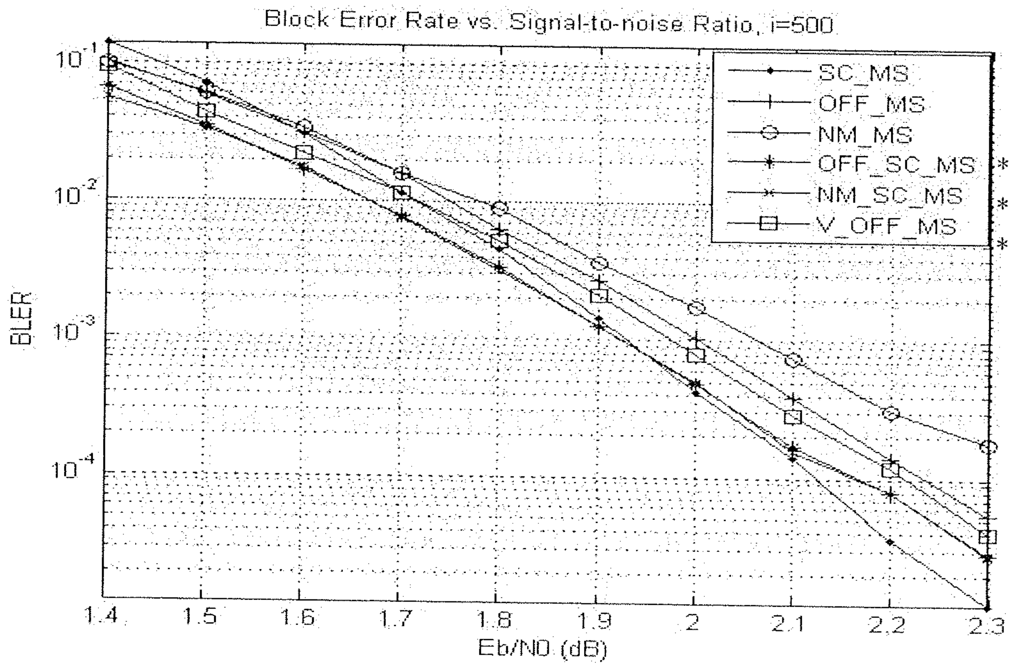

Figure 5.26: Block error rate performance at R= 1/2, N= 1152, and I_{max}= 500

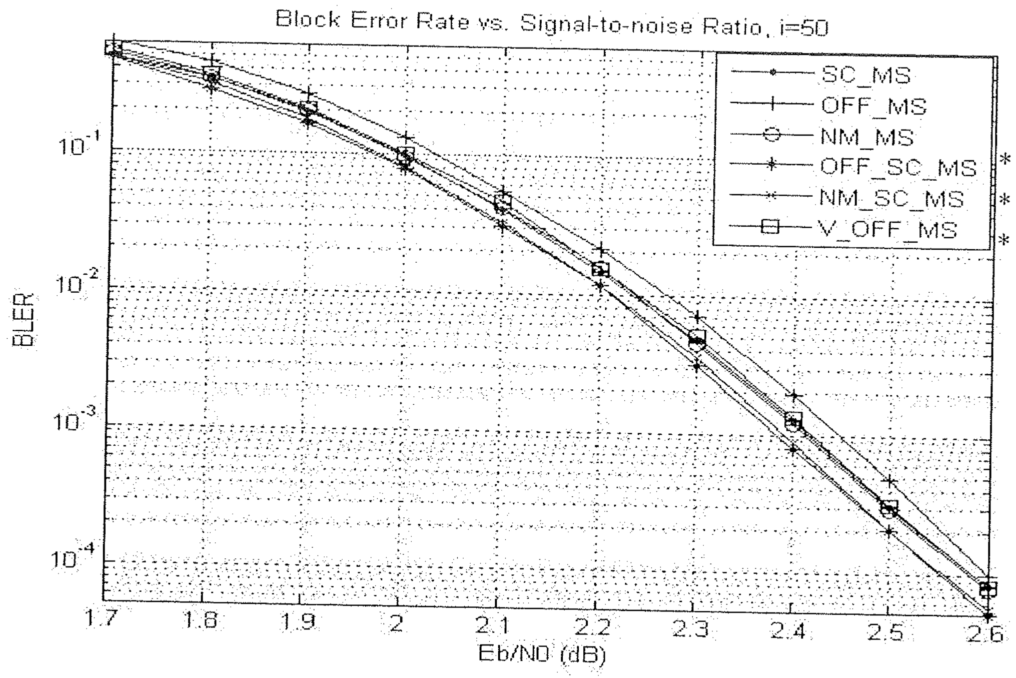

Figure 5.27: Block error rate performance at R= 2/3, N= 2304, and I_{max}= 50

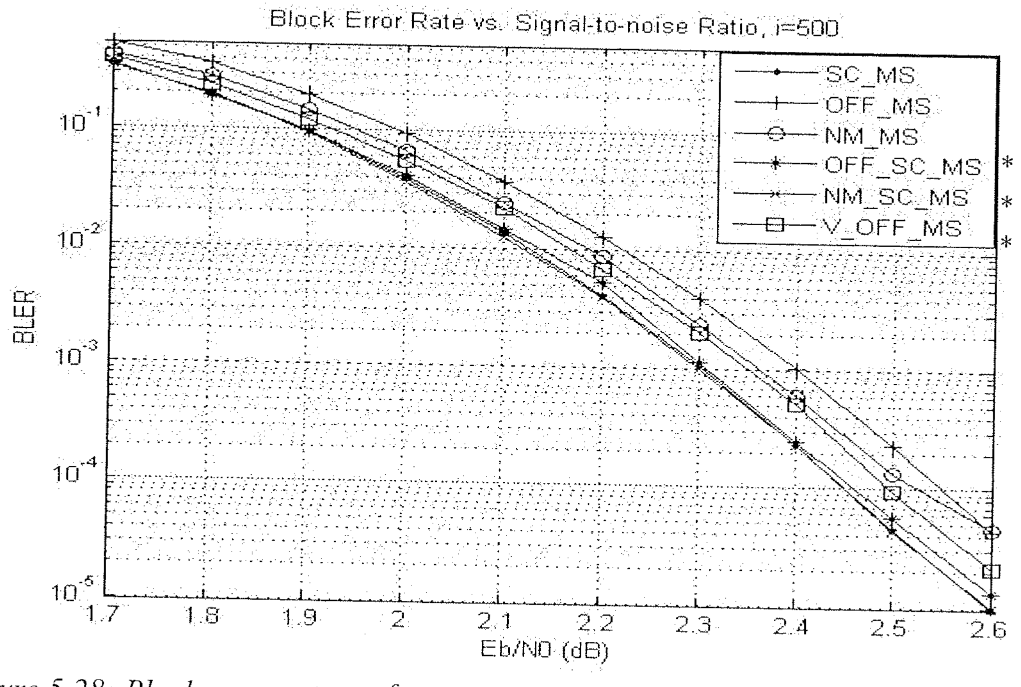

Figure 5.28: Block error rate performance at R= 2/3, N= 2304, and I_{max}= 500

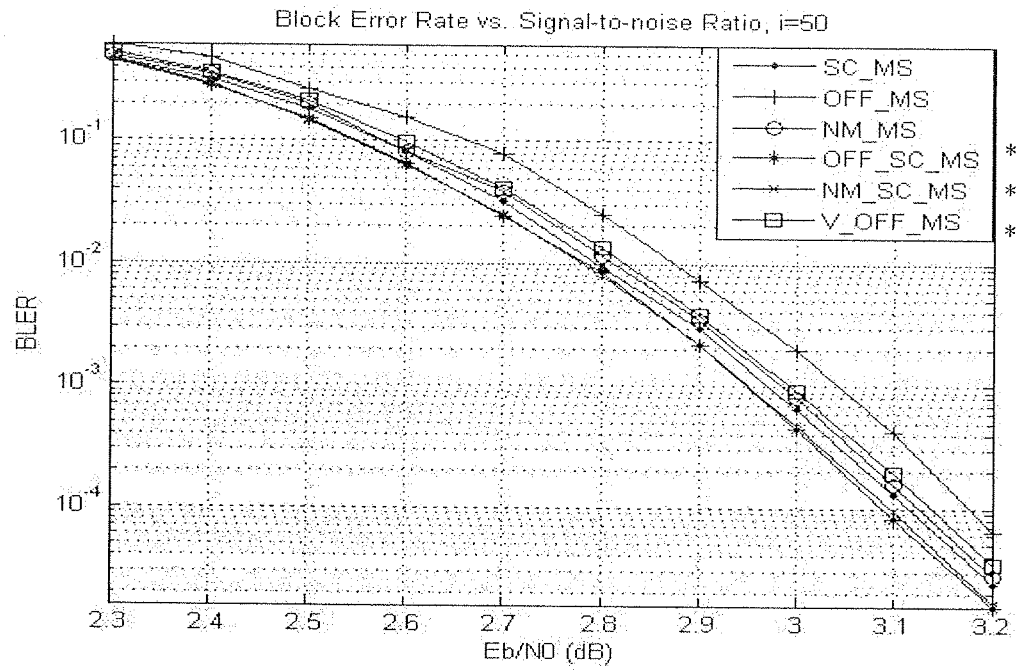

Figure 5.29: Block error rate performance at R= 3/4, N= 2304, and I_{max}= 50

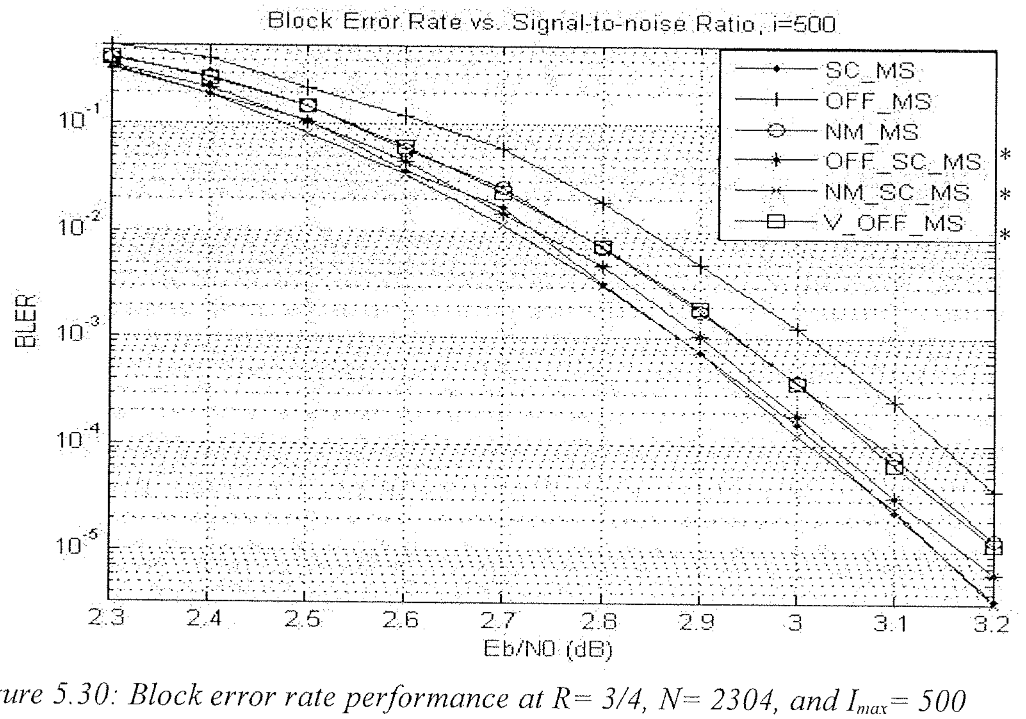

Figure 5.30: Block error rate performance at R= 3/4, N= 2304, and I_{max}= 500

This section discusses block error rate (BLER) versus E_b/N_0 (dB). Each simulation case consists of a range of 10 distinct E_b/N_0 values for each MS based decoding algorithm. The E_b/N_0 range were chosen specifically to obtain a BLER approaching 10^{-5} by the 10th value, occurring at $I_{max} = 500$. The BLER is determined by: the number of erroneous divided by umber of blocks transmitted. For these simulations, the BLER vs E_b/N_0 plots are obtained for $I_{max} = 50$ and 500.

Figures 5.21 and 5.22 showed the case for R= 1/2 and N= 2304. In Figure 5.21, OFF-MS and OFF-SC-MS had the two best BLERs. The performance of the NM-M-SC and V-OFF-MS decoding algorithms are identical, and are better than the SC-MS and NM-MS decoding algorithms. In Figure 5.22, the SC based MS decoding algorithms performed better than the non-SC based MS decoding algorithms. For SC based MS decoding NM-SC-MS had a better BLER than both the SC-MS and OFF-SC-MS. SC-MS had a lower BLER than OFF-SC-MS decoding. For non-SC based MS decoding V-OFF-MS had the best BLER.

Figures 5.23 and 5.24 showed the case for R= 1/2 and N= 1728. In Figure 5.23, the SC decoding algorithms performed the best. All SC based Min-Sum decoding algorithms showed identical results at 2.1 dB. Of the non-SC based MS algorithms, V-OFF-MS had the best performance. In Figure 5.24, the exact same trend occurred from Figure 5.23.

Figures 5.25 and 5.26 showed the case for R= 1/2 and N= 1152. In Figure 5.25, SC based MS decoding algorithms performed better than the non-SC based MS decoding algorithms. From this Figure, SC-MS decoding had best performance. For non-SC based MS decoding V-OFF-MS and OFF-MS had identical results. Both methods performed better than NM-MS decoding. In Figure 5.26, the same trend for SC based MS decoding algorithms were intact from Figure 5.25. However, for non-SC Min-Sum based decoding V-OFF-MS had the best performance.

Figures 5.27 and 5.28 showed the case for R= 2/3 and N= 2304. In Figure 5.27, the SC based Min-Sum decoding algorithms performed better than the non-SC Min-Sum based decoding algorithms. From this Figure, NM-SC-MS decoding had the best performance. SC-MS outperformed OFF-SC-MS decoding. In terms of non-SC based Min-Sum decoding algorithms NM-MS outperformed V-OFF-MS. In turn, V-OFF-MS decoding performed better than OFF-MS decoding. In Figure 5.28, the SC based Min-Sum decoding algorithms performed better than the non-SC based Min-Sum decoding algorithms. In this Figure, both SC-MS and NM-SC-MS showed identical and had the best decoding results. For non-SC based Min-Sum decoding V-OFF-MS had the best performance.

Figures 5.29 and 5.30 showed the case for R= 2/3 and N= 2304. In Figure 5.29, the SC Min-Sum based decoding algorithms performed better than non-SC based Min-Sum decoding algorithms. Both OFF-SC-MS and NM-SC-MS had better results than SC-MS. For non-SC Min-Sum based decoding algorithms NM-MS had the best result. Although, V-OFF-MS outperformed OFF-MS. In Figure 5.30, the SC Min-Sum based decoding algorithms performed better than non-SC Min-Sum based decoding algorithms. Moreover, both SC-MS and NM-SC-MS showed identical and the best results. For non-SC Min-Sum based decoding V-OFF-MS performed better than both NM-MS and OFF-MS.

6 Conclusions and Future Work

6.1 Conclusions

This thesis presented three new Min-Sum based decoding algorithms for LDPC coding. The performances of known and newly proposed Min-Sum based decoding algorithms were analyzed closely through intensive simulations for irregular LDPC codes outlined in the IEEE802.16e standard. The studied decoding algorithms were compared on the criteria of convergence characteristics, average iteration numbers, and block error rates.

For various code rates, R, and block-lengths, N, simulations have shown that all three proposed decoding algorithms performed competitively compared to the known decoding algorithms. V-OFF-MS did not require any additional storage (memory) the cost of hardware is reduced as a result of a simpler decoding structure. However, V-OFF-MS decoding had a higher average iteration number compared to the SC based Min-Sum decoding algorithms, but the reduced complexity of the decoding algorithm allows for faster computations, reducing latency. Compared to other non-SC based Min-Sum decoding algorithms V-OFF-MS consistently showed faster convergence and a lower block error rate than both NM-MS and OFF-MS. For practical implementation V-OFF-MS decoding algorithm is recommended, as reflected in the simulation results

For the proposed SC-based Min-Sum decoding algorithms, both NM-SC-MS and OFF-SC-MS consistently outperformed SC-MS for average iteration numbers. Throughout the study, regardless of E_b/N_o, NM-SC-MS and OFF-SC-MS always had lower average iteration numbers than SC-MS decoding. From this study it can be concluded if any SC Min-Sum based algorithms were to be implemented, decoding using NM-SC-MS and OFF-SC-MS algorithms would be preferred over SC-MS.

6.2 Future Work

Up to this point an in-depth floating point simulation was studied. The proposed theory in this thesis extends beyond this floating point simulation. The next step is to implement a fixed point simulation.

This fixed point simulation will consist of eliminating all float pointing variables from the C++

code, replacing them with integer variables. Furthermore, the C++ code must be optimized. Then the C++ code is ready to be implemented on a microcontroller or low level microprocessor that is not capable of floating point unit (FPU) operations.

References

[1] J. Chen and M.P.C. Fossorier, "Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity-Check Codes," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406 – 414, Mar. 2002.

[2] J. Chen and M.P.C. Fossorier, "Density Evolution for Two Improved BP-Based Decoding Algorithms of LDPC Codes," *IEEE Communication Letters*, vol. 6, no. 5, pp. 208 – 210, May 2002.

[3] V. Savin, "Self-Corrected Min-Sum decoding of LDPC codes," *IEEE International Symposium on Information Theory 2008 (ISIT08)*, Toronto, Canada, Mar. 2008.

[4] IEEE802.16e-2005, IEEE Standard for Local and Metropolitan Area Networks, Part 16 – Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands.

[5] IEEE802.2an, IEEE Standard for Information technology - Telecommunications and information exchange between systems - Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Corrigendum 2: IEEE Std 802.an™-2006 10GBASE-T Correction.

[6] Z. Tu and Z. Zhang "Overview of LDPC Codes" *Seventh International Conference on Computer and Information Technology,"* pp. 469 – 474.

[7] S.J. Vaughn-Nichols, "Achieving Wireless Broadband with WiMax," *Industry Trends*, pp. 10 -13, June 2004.

[8] IEEE 802.11-2007, IEEE Standard for Information Technology – Telecommunications and information exchange between systems – Local and area metropolitan networks – Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[9] B.Li, Y.Qin, C.P. Low, and C.L Gwee, "A survey on Mobile WiMax," *IEEE Communications Magazine,* pp. 70 – 75, Dec. 2007.

[10] R.W. Wells, "Applied Coding and Information Theory for Engineers," First Ed., Upper Saddle River, NJ: Prentice Hall, pp. 48 – 49.

[11] J. Proakis and M. Saheli, "Digital Communications", Fifth Ed., Singapore: McGraw Hill, 2008,

pp. 40 – 44 , 101 – 103, 161 – 166.

[12] R.G. Gallager, "Low-Density Parity-Check Codes," M.I.T. Press, Cambridge, MA: 1963.

[13] N. Wiberg, Codes and Decoding on General Graphs, Ph. D Dissertation, Linköping University, Sweden,1996.

[14] A. Shokrollahi, "LDPC Codes: An Introduction," Digital Fountain Inc., Apr. 2003.

[15] F. R. Kschischang, B. J. Frey, and H-A. Loeliger, "Factor Graphs and the Sum Product Algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498 – 519, Feb. 2001.

[16] L.Nassib, H.Omessad, B.Ammar, "Decoding Algorithm of Low Density Parity Check Code," *Springer-Verlag Berlin Heidelberg 2008,* pp. 220 – 227, 2008.

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

[18] C.E. Shannon, "Communication in the presence of noise," *Proc. Institute of Radio Engineers,* vol.37, no.1, pp.10 – 21, Jan. 1949. Reprint as classic paper in: *Proc. IEEE*, vol. 86, no.2, Feb. 1998.

[19] L.H.C. Lee, "Error-Control Block Codes for Communications Engineers," First Ed., Norwood, MA: Artech House, pp. 39 – 41, 2000.

[20] R.Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory,* vol.27, no. 5, pp. 533 – 547, Sept. 1981.

[21] J. Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier, and X.-Y. Hu, "Reduced-Complexity Decoding of LDPC codes," *IEEE Trans Commun.*, vol. 53, no. 8, pp. 1288 – 1299, Aug. 2005.

[22] S. Gounai, T. Ohtsuki, and T. Kaneko, "Modified Belief Propagation Decoding Algorithm for Low-Density Parity Check Code Based on Oscillation," *IEEE Vehicular Technology Conference 2006 (VTC20006-Spring)*, pp. 1467 – 1471, May 2006.

[23] "IEEE Std. 802.16e Decoder on XPP-III" XPP Technologies, Aug. 2006.

[24] N. Yu, *Research on LDPC Codes*, Group Seminar, University of Waterloo, Canada, 2004.

Appendix A. LDPC Parity Check Matrices for Various Code Rates and Block Sizes

Parity Check Matrices

Rate 1/2:

$$\begin{bmatrix}
-1 & 94 & 73 & -1 & -1 & -1 & -1 & -1 & 55 & 83 & -1 & -1 & 7 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 27 & -1 & -1 & -1 & 22 & 79 & 9 & -1 & -1 & -1 & 12 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 24 & 22 & 81 & -1 & 33 & -1 & -1 & -1 & 0 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
61 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & 65 & 25 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 39 & -1 & -1 & -1 & 84 & -1 & -1 & 41 & 72 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 46 & 40 & -1 & 82 & -1 & -1 & -1 & 79 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 95 & 53 & -1 & -1 & -1 & -1 & -1 & 14 & 18 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & 11 & 73 & -1 & -1 & -1 & 2 & -1 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
12 & -1 & -1 & -1 & 83 & 24 & -1 & 43 & -1 & -1 & -1 & 51 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 94 & -1 & 59 & -1 & -1 & 70 & 72 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & 7 & 65 & -1 & -1 & -1 & -1 & 39 & 49 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
43 & -1 & -1 & -1 & -1 & 66 & -1 & 41 & -1 & -1 & -1 & 26 & 7 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}$$

Rate 2/3A:

$$\begin{bmatrix}
3 & 0 & -1 & -1 & 2 & 0 & -1 & 3 & 7 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & 36 & -1 & -1 & 34 & 10 & -1 & -1 & 18 & 2 & -1 & 3 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 12 & 2 & -1 & 15 & -1 & 40 & -1 & 3 & -1 & 15 & -1 & 2 & 13 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & -1 & 19 & 24 & -1 & 3 & 0 & -1 & 6 & -1 & 17 & -1 & -1 & -1 & 8 & 39 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
20 & -1 & 6 & -1 & -1 & 10 & 29 & -1 & -1 & 28 & -1 & 14 & -1 & 38 & -1 & -1 & 0 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & 10 & -1 & 28 & 20 & -1 & -1 & 8 & -1 & 36 & -1 & 9 & -1 & 21 & 45 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
35 & 25 & -1 & 37 & -1 & 21 & -1 & -1 & 5 & -1 & -1 & 0 & -1 & 4 & 20 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
-1 & 6 & 6 & -1 & -1 & -1 & 4 & -1 & 14 & 30 & -1 & 3 & 36 & -1 & 14 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}$$

Rate 3/4A:

$$\begin{bmatrix}
6 & 38 & 3 & 93 & -1 & -1 & -1 & 30 & 70 & -1 & 86 & -1 & 37 & 38 & 4 & 11 & -1 & 46 & 48 & 0 & -1 & -1 & -1 & -1 \\
62 & 94 & 19 & 84 & -1 & 92 & 78 & -1 & 15 & -1 & -1 & 92 & -1 & 45 & 24 & 32 & 30 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
71 & -1 & 55 & -1 & 12 & 66 & 45 & 79 & -1 & 78 & -1 & -1 & 10 & -1 & 22 & 55 & 70 & 82 & -1 & -1 & 0 & 0 & -1 & -1 \\
38 & 61 & -1 & 66 & 9 & 73 & 47 & 64 & -1 & 39 & 61 & 43 & -1 & -1 & -1 & -1 & 95 & 32 & 0 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & -1 & -1 & 32 & 52 & 55 & 80 & 95 & 22 & 6 & 51 & 24 & 90 & 44 & 20 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
-1 & 63 & 31 & 88 & 20 & -1 & -1 & -1 & 6 & 40 & 56 & 16 & 71 & 53 & -1 & -1 & 27 & 26 & 48 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}$$

Rate 5/6:

$$\begin{bmatrix}
1 & 25 & 55 & -1 & 47 & 4 & -1 & 91 & 84 & 8 & 86 & 52 & 82 & 33 & 5 & 0 & 36 & 20 & 4 & 77 & 80 & 0 & -1 & -1 \\
-1 & 6 & -1 & 36 & 40 & 47 & 12 & 79 & 47 & -1 & 41 & 21 & 12 & 71 & 14 & 72 & 0 & 44 & 49 & 0 & 0 & 0 & 0 & -1 \\
51 & 81 & 83 & 4 & 67 & -1 & 21 & -1 & 31 & 24 & 91 & 61 & 81 & 9 & 86 & 78 & 60 & 88 & 67 & 15 & -1 & -1 & 0 & 0 \\
50 & -1 & 50 & 15 & -1 & 36 & 13 & 10 & 11 & 20 & 53 & 90 & 29 & 92 & 57 & 30 & 84 & 92 & 11 & 66 & 80 & -1 & -1 & 0
\end{bmatrix}$$

Code Rate and Block Size Adjustment

Table A.1: Code rate and block size adjustment

n (bits)	*n* (bytes)	*z* factor	*k* bytes			
			R=1/2	R=2/3	R=3/4	R=5/6
576	72	24	36	48	54	60
672	84	28	42	56	63	70
768	96	32	48	64	72	80
864	108	36	54	72	81	90
960	120	40	60	80	90	10
1056	132	44	66	88	99	110
1152	144	48	72	96	108	120
1248	156	52	78	104	117	130
1344	168	56	84	112	126	140
1440	180	60	90	120	135	150
1536	192	64	96	128	144	160
1632	204	68	102	136	153	170
1728	216	72	108	144	162	180
1824	228	76	114	152	171	190
1920	240	80	120	160	180	200
2016	252	84	126	168	189	210
2112	264	88	132	176	198	220
2208	276	92	138	184	207	230
2304	288	96	144	192	216	240

Example expanding an element of a parity check matrix into expanded binary elements

If n= 576 $\rightarrow$ $z = n/24 = 24$. From section 3.4, each element of the parity check is expanded by a *z* factor. Employing this factor is necessary to insert binary elements into the parity check matrix. This section will show an example on how to expand a non-zero element, a zero element, and negative element.

Expanding a non-zero element: 7. As shown below (A.1) the binary expansion is an identity matrix circularly right shifted by 7. The expanded binary matrix is of *z* by *z* dimensions.

$$H_b=\left[\begin{array}{cccccccccccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right] \qquad (A.1)$$

Expanding a zero element. As shown below (A.2) the binary expansion is an identity matrix of z by z dimensions.

$$H_b=\left[\begin{array}{cccccccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \qquad (A.2)$$

Expanding a negative element: -1. As shown below (A.3) the binary expansion is an all zero matrix of z by z dimensions.

$$H=\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix} \tag{A.3}$$

Appendix B. C++ source code of decoding algorithms

```
// Calculate the minsum decoding pi and min terms without exclusion.
struct functor_r_ms_pi
{
	double min0, min1;	// The lowest and second-lowest minima, respectively
	int pi;			// The product term, without exclusion.
	inline void callback(double &q)
	{
		double qv = q;
		if (qv < 0)
			pi = -pi;		// This effectively does the sign function.
		qv = fabs(qv);
		if (min0 >= qv)		// Update both minima.
		{
			min1 = min0;
			min0 = qv;
		}
		else if (min1 > qv)		// Update the second-lowest minimum.
			min1 = qv;
	}
};

// Updates the R matrix with minsum decoding. Performs exclusion.
template <DecodeMethod::Enum msMethod, bool sc>
struct functor_r_ms_update
{
	int pir0;
	double min0, min1;
	inline void callback(double &r, double &q)
	{
		int pir = pir0;	// The pi term to use.

		const double qv = q;				// The Q term to use.
		// Perform exclusion on the pi term.
		if (qv < 0)
			pir = -pir;
		// Perform exclusion on the min term.
		const double qvmin = (fabs(qv) == min0) ? min1 : min0;

		// Offset min sum calculation for r^(i)_(m,n)
		r = pir;
		if (msMethod == DecodeMethod::offms)
			r *= max((double)0.0, qvmin - (sc ? betasc : beta));
		else
			r *= qvmin;

		if (msMethod == DecodeMethod::nms)
			r *= sc ? alphasc : alpha;
	}
};

// Calculates the sigma term without exclusion, for use in updating qn and ln
// during decoding.
```

```
struct functor_sigmar
{
        double rsigma;
        inline void callback(double &r)
        {
                rsigma += r;
        }
};

// Update qn during decoding. Performs exclusion.
template <DecodeMethod::Enum method, bool sc>
struct functor_updateq
{
        // The value of qn for all elements in this column at iteration 0
        double q0;
        double rsigma;
inline void callback(double &q, double &r)
        {
                // Performs exclusion and sets Q.
                const double qnew = q0 + rsigma - r;
                if (method == DecodeMethod::v_off_ms)
                {
                        const bool qsgn = qnew < 0;
                        double qmag = fabs(qnew);
                        if (qmag > betav)
                        {
                                qmag -= betav;
                                if (qsgn) qmag = -qmag;
                                q = qmag;
                        }
                        else
                        {
                                if (r != 0 && (r < 0) != qsgn)
                                        q = 0;
                        }
                }
                else
                {
                        if (sc)
                        {
                                if (q != 0 && ((q >= 0) != (qnew >= 0)))
                                        q = 0;
                                else
                                        q = qnew;
                        }

                }
        }
};
```

Appendix C. Simulation time

Table C.1 shows a histogram showing the computation time of a single iteration for each studied decoding algorithm. The computation times from Table C.1 occurs at code rate R = 1/2 and blocklength N= 2304. The computation time of a single iteration is variant of available computation power. The simulations for this study were performed on an intel T6400 core 2 duo processor. Furthermore, the C++ code was multithreaded with two threads, using 100% CPU of processor.

Table C.1: Single iteration duration for studied decoding algorithms

LDPC Decoding Algorithm	Time for single iteration (μs)
SC-MS	183
OFF-MS	179
NM-MS	200
OFF-SC-MS	157
NM-SC-MS	180
V-OFF-MS	181