Performance Estimation, Code Construction and Decoding of Finite-Length Low-Density Parity-Check Codes

By

Hua Xiao

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

Ottawa-Carleton Institute for Electrical and Computer Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-40543-7
Our file Notre référence
ISBN: 978-0-494-40543-7

Abstract

While much is known about the asymptotic performance of Low Density Parity Check (LDPC) codes under iterative decoding algorithms, the finite-length analysis of LDPC codes is still an active area of research with many open problems. At finite block length, trapping sets in the Tanner graph representing an LDPC code affect the performance of the code under iterative decoding algorithms, especially at high signal-to-noise ratio (SNR) region. In this thesis, we address several issues of finite-length LDPC codes related to trapping sets in the graph. In particular, we are concerned with performance analysis, decoding algorithms and constructions of finite-length LDPC codes.

We study the structure of trapping sets under different hard-decision iterative decoding algorithms. We propose a general method to estimate the error performance of LDPC codes for hard-decision iterative decoding algorithms based on direct enumeration on the smallest error patterns the decoder fails to correct. The proposed method can provide accurate estimates for both the frame error rate (FER) and the bit error rate (BER) over the whole range of crossover probabilities of practical interest. We then propose cycle enumeration algorithm which reduces the computational

complexity of the direct enumeration by several orders of magnitude without any sizable degradation in the accuracy of the estimates, making the estimation method applicable to almost any LDPC code of practical interest.

We extend the estimation method for hard-decision iterative decoding algorithms to soft-decision iterative decoding algorithms. Modified cycle enumeration algorithm is introduced to reduce the computational complexity. We also propose an algorithm which estimates the performance of a decoder with large number of quantization bits based on the estimation result for small number of quantization bits. This reduces the computational complexity.

We also propose a modification to the progressive-edge-growth (PEG) algorithm for the construction of irregular LDPC codes. The proposed modification reduces the number of small trapping sets and improves the minimum distance of the code, which results in considerable improvement of the performance in high signal-to-noise ratio (SNR) region without sacrificing low-SNR performance.

Moreover, we investigate the application of successive relaxation (SR) in different domains for the iterative decoding of LDPC codes. We show by simulations that, in general, for finite-length codes, the SR algorithms outperform their conventional successive substitution (SS) counterparts. The improvement in performance increases as the maximum number of iteration becomes larger.

Acknowledgements

I would like to express my sincerest gratitude to my thesis supervisor, Dr. Amir H. Banihashemi, for his continuous support, encouragement and patience throughout this thesis and my education at Carleton University. I am deeply influenced by his passion for research. His knowledgeable suggestions and editorial efforts have greatly improved the quality of this thesis.

I am thankful to Dr. William Ryan from University of Arizona, Dr. Claude D'Amours of the University of Ottawa, Dr. Calvin Plett of the Department of Electronics, Dr. Ian Marsland, and Dr. Halim Yanikomeroglu of the Department of Systems and Computer Engineering, for being my examining committee members, and for their helpful comments and advice.

I also thankfully acknowledge the financial support of my work provided by Ontario Graduate Scholarship in Science and Technology (OGSST), Ontario Graduate Scholarship (OGS), Nortel Networks and Carleton University.

I would also like to thank all my friends at Carleton, especially Saied Hemati, Chan-Tong Lam, Xiaoning Li, Angie Low, Jinshi Qiu, Lily Yuan, Pirouz Zarrinkhat, and Jenny Zhu, all the helpful staff members of the Department of Systems and Computer Engineering, especially Anna Lee, Blazenka Power, and Narendra Mehta.

Last, but not least, I am truly grateful to my dear parents, sister and brother-in-law for their unconditional love and support.

Ottawa, May 12, 2008

Hua Xiao

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACE:	Approximate cycle extrinsic message degree
APP:	A posterior probability
AWGN:	Additive white Gaussian noise
BEC:	Binary erasure channel
BER:	Bit error rate
BIAWGN:	Binary input additive white Gaussian noise
BP:	Belief propagation
BPSK:	Binary phase shift keying
BSC:	Binary symmetric channel
EXIT:	Extrinsic information transfer
FER:	Frame error rate
GA:	Gallager's decoding algorithm A
GB:	Gallager's decoding algorithm B
LD:	Likelihood difference
LDPC:	Low-density parity-check
LLR:	Log-likelihood ratio
LR:	Likelihood ratio
MB:	Majority-based

ML:	Maximum-likelihood
MS:	Min-sum
PEG:	Progressive edge-growth
SR:	Successive relaxation
SS:	Successive substitution
SNR:	Signal-to-noise ratio
TG:	Tanner graph

List of Symbols

$\mathcal{C}$:	A linear block code
$\mathbf{c}$:	A codeword of a linear block code
c:	A check node
$\boldsymbol{H}$:	Parity-check matrix of a linear block code
$\mathbf{0}$:	All zero vector
n:	Block length of a linear block code
k:	Dimension of a linear block code
$\mathcal{M}$:	Message alphabet
v:	A variable node
$\mathbf{m}_v$:	Initial message of variable node v
$\mathbf{m}_{v\to c}$:	Message from node v to node c
$N(x)$:	Set of neighboring nodes of node x
$\lceil x \rceil$:	Smallest integer $\geq x$
$\lfloor x \rfloor$:	Largest integer $\leq x$
R:	Code rate
$G(S)$:	Induced subgraph of node set S
$C_{odd}(S)$:	Check nodes with odd degree in $G(S)$.
J:	The smallest weight of error patterns the decoder fails to correct

S_i:	Set of weight-i error patterns
E_i:	Set of weight-i error patterns that the decoder fails to correct
g:	Girth of a graph
C_l:	Set of cycles of length l
c_{th}:	Clipping threshold
I_{max}:	Maximum number of iterations
q:	Number of quantization bit
$\mathcal{A}_q$:	Quantization alphabet
$\mathcal{V}$:	Set of input vectors to a decoder
$\vec{R}$:	Most reliable input vector
S_d:	Set of input vectors with Euclidean distance d to $\vec{R}$
E_d:	Subset of S_d, which contains input vectors that the decoder fails to correct
$\underline{\mathcal{V}}(\vec{r})$:	Set of vectors which are below vector $\vec{r} \in \mathcal{V}$
T_k:	Union of sets $C_g, \cdots, C_{g+2k}$
$N_{v_j}^l$:	Set of check nodes reached by a tree spreading from variable node v_j within depth l.
$\vec{V}_0$:	Vector of initial messages
$\vec{V}_t$:	Output message vector from variable nodes at time t

Chapter 1

Introduction

Low Density Parity Check (LDPC) codes and the iterative algorithms that can be used to decode these codes were first introduced by Gallager [1–3] in his Ph.D. thesis in the early sixties. However, in spite of Gallager's promising results on the strength and the flexibility of LDPC codes and iterative decoding algorithms, they were almost completely forgotten after their invention mainly due to the limited computational capabilities of the computers at that time which made the associated iterative decoding algorithms look impractical. For the next thirty years, only a few works, such as [4] and [5], had been performed in this area. Inspired by the success of turbo-codes [6], LDPC codes were rediscovered [7]. Since then, there has been a great amount of research devoted to LDPC codes and their encoding and decoding (see, e.g., [8–10] and the references therein). Near Shannon limit error performance and low decoding complexity have been shown for LDPC codes [7, 11–13]. It has been also observed that LDPC codes with long block length outperform the best known turbo codes [13]. Nowadays, LDPC codes are considered a serious candidate in almost all the practical

systems where error control coding is needed (e.g., data storage systems, satellite communication systems, mobile communication systems, optical communication systems, etc.)

LDPC codes are linear block codes and thus are specified by their parity-check matrices. Iterative decoding algorithms are the efficient tools for decoding LDPC codes. The decoding process is performed by iteratively exchanging messages between the nodes and through the edges in the Tanner graph [4] representing the code. There is a wide range of iterative algorithms for decoding LDPC codes, each offering a particular tradeoff between error performance and decoding complexity.

Under iterative decoding algorithms, the performance of a finite block length LDPC code is closely related to the *trapping sets* in the code's Tanner graph, especially in the high signal-to-noise ratio (SNR) region (or the so-called error floor region) [14]. In general, a trapping set is defined as a set of variable nodes that cannot be eventually corrected by the decoder [14]. The effect of trapping sets to iterative decoding is similar to that of low-weight codewords to maximum-likelihood (ML) decoding. However, unlike low-weight codewords, trapping sets depend not only on the code's structure but also on the specific decoding algorithm and the channel. In this thesis, we first study the trapping sets of LDPC codes under different iterative decoding algorithms and propose a combinatorial method to estimate the error performance of finite-length LDPC codes under different hard-decision iterative decoding algorithms. We then extend this method to the case of soft-decision (quantized) iterative decoding algorithms. We propose a modification to a well-known construction method, progressive-edge-growth [15], which reduces the number of small trapping

sets in the Tanner graph and improves the code's performance in the error floor region. We also investigate the application of successive relaxation (SR) in different domains to the iterative decoding of finite-length LDPC codes.

1.1 Thesis Motivations

The asymptotic performance analysis of LDPC codes under iterative decoding algorithms is now a rather mature subject. Analysis tools such as *density evolution* [12] and EXIT charts [16] provide a rather complete picture of the asymptotic performance of LDPC codes at infinite block lengths. Density evolution is applied to the ensembles of LDPC codes to derive the *thresholds* of the ensembles under a variety of decoding algorithms, see, e.g., [12]. Concentration results [12] then guarantee that, for sufficiently large block lengths, the performance of the individual codes from the ensemble is close to the average performance predicted by the asymptotic analysis and derived based on the assumption that the Tanner graph of the codes is cycle-free.

Our knowledge of iterative decoding algorithms at finite block length however is not nearly as complete. This is despite the fact that in practice, we always have to deal with finite-length codes. The difficulty in analyzing LDPC codes with finite block length under iterative message-passing algorithms originates from the fact that the Tanner graph representation of these codes includes many short cycles (practical codes don't have cycle-free Tanner graph representations [17]). These cycles, even for memoryless channels where the initial messages are independent, create dependencies among the messages after the first few iterations have passed. This makes the performance of LDPC codes deviate from the asymptotic results. In fact, the finite-length

analysis of LDPC codes is still an active area of research with many challenging open problems. In this thesis, we mainly focus on practical LDPC codes and address several issues related to finite block length codes, including error performance estimation, code construction and decoding.

1.2 Thesis Contributions

The results of this thesis have been partly published or submitted for publication in a number of conference proceedings and journals [18–28].

In this thesis, we study the structures of trapping sets for finite block length LDPC codes under different iterative decoding algorithms. We also investigate the relationship between trapping sets and the error performance of the iterative algorithms. Based on the study on trapping sets, we derive upper and lower bounds on the frame error rate (FER) of LDPC codes decoded by hard-decision iterative algorithms over binary symmetric channels (BSC). The bounds, which are tight for sufficiently small crossover probabilities of the channel, provide a good estimate of FER in this region. These results have been published in [18].

A general combinatorial method for estimating the error performance of LDPC codes decoded by hard-decision iterative decoding algorithms on BSCs is proposed. Based on the direct enumeration of the smallest weight error patterns that cannot be all corrected by the decoder, this method estimates both the FER and the bit error rate (BER) of a given LDPC code with very good precision for all crossover probabilities of practical interest. This method can be effectively applied to both

regular and irregular LDPC codes and to a variety of hard-decision iterative decoding algorithms. Compared with the conventional Monte Carlo simulations, the proposed method has a much smaller computational complexity, particularly for lower error rates. These results have been published in [19] and [20].

The computational complexity associated to the direct enumeration can be still very high for many practical cases. Using cycle enumeration, we reduce the complexity of direct enumeration by several orders of magnitude, making it possible to estimate the error rates for almost any practical LDPC code. To obtain the error rate estimates, we propose an algorithm that progressively improves the estimates as larger cycles are enumerated. These results have been published in [21] and submitted for possible publication in *IEEE Trans. Comm.* [22].

We then extend the estimation technique to the case of soft-decision iterative decoding algorithms. The decoders under consideration are quantized versions of soft-decision decoding algorithms. To reduce the computational complexity, we propose a modified cycle enumeration algorithm. We also propose an algorithm which estimates the performance of decoder with large number of quantization bits based on the results for small number quantization bits. These results have been published in [23] and submitted for possible publication in *IEEE JSAC.* [24].

As another contribution, we propose a very simple modification to one of the best known methods for constructing LDPC codes at short and intermediate block lengths, the progressive-edge-growth (PEG) algorithm [15]. This modification considerably enhances the performance at high SNR region without any degradation in the low-SNR performance by improving both the minimum distance and the trapping sets of

the code. These results have been published in [25, 26].

We also investigate the application of successive relaxation (SR) in different domains to the iterative decoding of LDPC codes. We show by simulations that improvement in error performance offered by SR is quite different when it is applied in different domains. We observe that, in general, the SR algorithms outperforms their conventional successive substitution (SS) counterparts and the improvement in performance increases with increasing the maximum number of iterations. These results have been published in [27] and submitted for possible publication in *IEEE Trans. Comm.* [28].

1.3 Thesis Organization

Chapter 2 gives a background review on basic concepts and related topics involved in the rest of the thesis. We review the basics of LDPC codes, such as, Tanner graph representation and code construction methods. We survey some well-known iterative decoding algorithms. We also review some important analysis tools for infinite block lengths, such as density evolution.

In Chapter 3, we focus on hard-decision iterative decoding algorithms. We first derive upper and lower bounds on the FER of LDPC codes decoded by hard-decision iterative decoding algorithms over a BSC. We then investigate the trapping set structures for hard-decision iterative algorithms and propose our error performance estimation technique.

A method is presented in Chapter 4 to reduce the computational complexity of

the estimation technique proposed in Chapter 3 using cycle enumeration. We also propose an algorithm which progressively improves the estimates.

In Chapter 5, we propose a technique for the performance estimation of LDPC codes decoded by quantized (soft-decision) iterative decoding algorithms. We introduce modified cycle enumeration algorithm in this chapter. We also introduce an algorithm which generates the estimate for decoders with large number of quantization bits based on the estimation results for small number of quantization bits.

Chapter 6 presents our modification to PEG algorithm. We show that this modification improves the performance considerably in high SNR region with no sacrifice in low SNR performance.

Chapter 7 presents our results on SR for decoding of LDPC codes. We show that the application of SR in different domains results in different error correcting performance.

Chapter 8 concludes the thesis and provides topics for future research.

Chapter 2

Background Review

2.1 Low-Density Parity-Check codes and Tanner Graph Representation

LDPC codes are a class of linear block codes. Like any other linear block code, an LDPC code $\mathcal{C}$ can be fully described by its parity check matrix $\boldsymbol{H}$, through the parity-check equations $\mathbf{c}\boldsymbol{H}^T = \mathbf{0}$, $\mathbf{c} \in \mathcal{C}$, where $\boldsymbol{H}^T$ is the transpose of matrix $\boldsymbol{H}$. For an LDPC code, the parity check matrix is sparse. That is, the parity check matrix consists of many zeros and only a small fraction of elements are nonzero. In other words, for a binary (in this thesis, we assume that the codes are binary) LDPC code, the density of ones in the parity check matrix is low. Hence, it is named "low density". (The sparseness makes the complexity of the corresponding iterative decoding algorithms increase only linearly with the code's block length.)

A (d_v, d_c)-*regular* LDPC code has a parity check matrix which has the same weight (number of ones) d_v per column and the same weight d_c per row (the original Gal-

lager's codes introduced in [1–3] are regular LDPC codes). Otherwise, it's called *irregular*. In irregular LDPC codes, column and row weights are not constant, and chosen according to two distributions [13]. Irregular LDPC codes were introduced in [29] and were further studied in [30–32]. It has been shown that irregular LDPC codes in general have better performance than regular ones (in the waterfall region). It has also been observed that irregular LDPC codes with long block length even outperform the best known turbo codes [13].

Like any other linear block code, an LDPC code can be represented by a *Tanner graph* [4] which is constructed based on the parity check matrix of the code. A Tanner graph is a *bipartite graph* which contains only two types of nodes: variable (bit) nodes and check (constraint) nodes. Variable nodes correspond to the columns of the parity check matrix (i.e., bits of the codeword). Check nodes correspond to the rows of the parity check matrix (i.e., parity-check equations of the code). Variable node j is connected to check node i by an edge if and only if there is a "1" located at position (i, j) in the parity check matrix (i.e., the jth code bit participates in the ith parity-check equation). Note that, we can also construct a parity-check matrix, and therefore a code, based on a Tanner graph, if it does not contain parallel edges.

Example 2.1.1 The parity-check matrix of the (7, 4) Hamming code is given as:

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

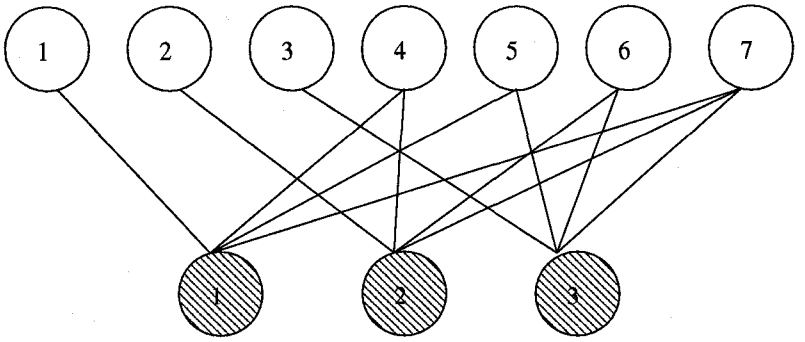

Figure 2.1: Tanner graph representation of the code given in Example 2.1.1.

The Tanner graph constructed from $\boldsymbol{H}$ is shown in Figure 2.1. In the figure (and also in the rest of this thesis), variable nodes are represented by empty circles and check nodes are represented by shaded circles. For example, in the graph, check node 1 is connected to variable node 5 by an edge, which means that variable node 5 participates in the 1st parity-check equation and a "1" is located in the (1, 5) position of $\boldsymbol{H}$. □

Note that the degree (number of edges connected to a node in the graph) of a variable node equals the weight of the corresponding column in the parity check matrix and the degree of a check node equals the weight of the corresponding row in the parity check matrix. Therefore, in the Tanner graph of a (d_v, d_c)-regular code, all the variable nodes and all the check nodes have the same degree d_v and d_c, respectively. The Tanner graph in this case is called a regular graph. For an irregular LDPC code, the degree of each set of nodes is chosen according to some distribution. The Tanner graph in this case is called an irregular graph.

Suppose that we have an irregular graph with some maximum variable node degree D_v and some maximum check node degree D_c. We may specify our irregular graph

(or equivalently, irregular code) by degree sequences $(\lambda_1, \cdots, \lambda_{D_v})$ and $(\rho_1, \cdots, \rho_{D_c})$, where λ_i and ρ_i are the fraction of edges emanating from variable nodes and check nodes of degree i, respectively [13, 29]. (Note that, $\sum_i \lambda_i = \sum_i \rho_i = 1$.) Usually, we require the degree of each node in the Tanner graph to be greater than or equal to 2. Equivalently, the variable (check) node degree distribution [13] can be specified by the polynomial:

$$\lambda(x) = \sum_{i=2}^{D_v} \lambda_i x^{i-1} \left(\rho(x) = \sum_{i=2}^{D_c} \rho_i x^{i-1} \right). \tag{2.1.1}$$

For (d_v, d_c)-regular LDPC codes, $\lambda(x) = x^{d_v - 1}$ and $\rho(x) = x^{d_c - 1}$. As we will see later, graphical representations of LDPC codes are extremely useful in studying and analyzing LDPC codes and the associated iterative decoding algorithms.

Cycles[1] in the Tanner graph are known to be problematic structures to iterative decoding algorithms since they create dependencies between messages passed around the graph by the iterative algorithms. This causes the iterative algorithms to perform sub-optimally. It is well accepted that a good Tanner graph for iterative algorithms should not have too many small cycles (e.g., cycles of length 4) and should preferably have a relatively large *girth* (the length of the smallest cycle in the graph).

For a given degree distribution pair (λ, ρ), the *ensemble* of LDPC codes of length n is defined in the following way. In the Tanner graphs, there are n variable nodes and $m = \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx} n$ check nodes. The number of edges is: $|E| = \frac{n}{\int_0^1 \lambda(x)dx}$. The node degree distributions are defined as: $\tilde{\lambda}_i = \frac{\lambda_i / i}{\int_0^1 \lambda(x)dx}$ and $\tilde{\rho}_j = \frac{\rho_j / j}{\int_0^1 \rho(x)dx}$, i.e., $\tilde{\lambda}_i$ fraction of the n variable nodes have i sockets and $\tilde{\rho}_j$ fraction of check nodes have j sockets.

[1] A cycle is a closed walk in which all the nodes except for the start and the end nodes are distinct.

Both the variable node sockets and the check node sockets are labeled $\{1, 2, ..., |E|\}$. A random Tanner graph in this ensemble is generated by a random permutation $\pi : \{1, 2, ..., |E|\} \rightarrow \{1, 2, ..., |E|\}$, i.e., the ith variable node socket is connected to the $\pi(i)$th check node socket. By definition, all elements in this ensemble are equiprobable.

2.2 Iterative Decoding Algorithms

In the context of decoding LDPC codes, iterative algorithms are naturally described using the Tanner graphs representing the codes. Iterative decoding algorithms, also called "message-passing algorithms," are performed by exchanging messages (or information) between check nodes and variable nodes through the edges of the graph, in both directions and iteratively. The complexity per iteration of the algorithms is then proportional to the number of edges in the Tanner graph. For LDPC codes, the complexity of the algorithms is low due to the sparseness of $\boldsymbol{H}$ (i.e., the sparseness of edges in the Tanner graph). The messages passing through the graph could be "hard" (bits) and/or "soft" (real numbers) information, and the operations performed in variable and check nodes to generate messages depend on the nature of the decoding algorithm and the message type(s).

Generally speaking, an iterative decoding algorithm starts by creating initial (local) messages (weights) for each variable node from the observations at the output of the channel and channel characteristics. As the initial step, it then passes these initial messages to check nodes through the edges of the Tanner graph. Iterations

then start, by each iteration consisting of two steps: 1) check nodes processing the incoming information and passing new (extrinsic) messages to the variable nodes. These messages usually reflect the estimate of check nodes from the value and/or reliability of variable nodes. 2) variable nodes processing the incoming messages sent by check nodes and sending updated (extrinsic) information about their value and/or the associated reliability to check nodes. Here, "extrinsic" (following turbo-code terminology [6]) means that the outgoing message sent along one edge does not depend on the incoming message sent along the same edge. Allowing only extrinsic messages passing along edges of the Tanner graph is a necessary condition of having good message-passing decoding algorithms. In each iteration, a hard decision on the value of each bit (e.g., "0" or "1") is made at variable nodes. The algorithms stops if the hard-decision assignments for bits satisfy all the check equations (i.e., the algorithm converges to a codeword), or a maximum number of iterations is reached, in which case a "decoder failure" is declared.

There are a wide variety of iterative algorithms for decoding LDPC codes, each offering a particular tradeoff between error performance and decoding complexity. In general, these algorithms can be divided into two classes, namely, hard-decision and soft-decision, based on whether the messages passing through the graph are "hard" or "soft" information, respectively. Also, there are combined hard/soft decision algorithms such as weighted bit-flipping algorithm mentioned in [33]. In the following sub-sections, we will briefly review some well-known iterative decoding algorithms.

2.2.1 Hard-Decision Iterative Decoding Algorithms

The main advantage of hard-decision algorithms with hard messages is their simplicity which makes them fast and easy to implement. In hardware, this translates to higher speed and lower area and power consumption. In addition, hard-decision algorithms are the only way to decode LDPC codes if only hard-decisions are available at the receiver.

Gallager's Decoding Algorithm A (GA)

GA was first introduced by Gallager in [3] and can be used over BSC. Let $\mathcal{M}$ denote the message alphabet space of GA. For example, $\mathcal{M} = \{-1, +1\}$ if each bit c_i in a codeword $\mathbf{c}$ is mapped to bipolar signal $x_i = (-1)^{c_i}$ and transmitted over BSC. Then at the initial step (at iteration $l = 0$), a variable node v obtains the initial message $\mathbf{m}_v \in \mathcal{M}$ from the channel output. In iteration $l \geq 1$, for a check node c, the message passed from c to its adjacent variable node v is the product of the incoming messages of all c's adjacent variable nodes except v in iteration $l - 1$,

$$\mathbf{m}_{c \to v} = \prod_{v' \in N(c) \setminus v} \mathbf{m}_{v' \to c}, \tag{2.2.2}$$

where, $N(x)$ denotes the set of neighboring nodes of node x and $N(x) \setminus y$ means that node y is excluded from set $N(x)$ (This guarantees that only the extrinsic information will be sent out). A variable node v passes its initial message $\mathbf{m}_v$ to its adjacent check node c unless all of its incoming messages in iteration l from check nodes in set $N(v) \setminus c$

disagree with this, in which case, $-\mathbf{m}_v$ is sent out,

$$\mathbf{m}_{v\to c} = \begin{cases} -\mathbf{m}_v, & \text{if } \left|\{c' \in N(v) \setminus c : \mathbf{m}_{c'\to v} = -\mathbf{m}_v\}\right| = d_v - 1 \\ \mathbf{m}_v, & \text{otherwise} \end{cases} . \tag{2.2.3}$$

Gallager's Decoding Algorithm B (GB)

GB was also introduced by Gallager in his thesis. Compared to GA, GB is more complex but more efficient. The message passing operations of GB are similar to those for GA, except that the message sent by a variable node v to its adjacent check node c $\mathbf{m}_{v\to c}$, in iteration l, is changed to:

$$\mathbf{m}_{v\to c} = \begin{cases} -\mathbf{m}_v, & \text{if } \left|\{c' \in N(v) \setminus c : \mathbf{m}_{c'\to v} = -\mathbf{m}_v\}\right| \geq b_l \\ \mathbf{m}_v, & \text{otherwise} \end{cases} . \tag{2.2.4}$$

In other words, in GB, a variable node passes its initial message unless at least b_l extrinsic incoming messages disagree with the initial message. Note that, unlike GA, the message passing operations of GB depends on the iteration number. The value of b_l in general is a function of variable node degree, check node degree and iteration number l. The optimal choice of b_l was determined in [1].

Among binary message-passing algorithms, GB with the optimal value of b_l is shown to have the minimum message error-rate in each iteration and the maximum threshold [3, 34–37] .

Majority-Based (MB) Decoding Algorithms

MB decoding algorithms are studied in detail in [38]. MB algorithm of order ω, denoted by MB^{ω}, has message passing operations the same as those of GA, except that message $\mathbf{m}_{v\to c}$ sent in iteration l from variable node v to check node c is given by:

$$\mathbf{m}_{v\to c} = \begin{cases} -\mathbf{m}_v, & \text{if } \left|\{c' \in N(v)\setminus c:\ \mathbf{m}_{c'\to v} = -\mathbf{m}_v\}\right| \geq \lceil d_v/2\rceil + \omega \\ \mathbf{m}_v, & \text{otherwise} \end{cases}, \quad (2.2.5)$$

where $0 \leq \omega \leq d_v - 1 - \lceil d_v/2\rceil$ and $\lceil x\rceil$ is the smallest integer which is greater than or equal to x. Note that, GA is a special case of MB algorithms. In fact, GA is the MB algorithm with the maximum order of $\omega = d_v - 1 - \lceil d_v/2\rceil$. It has been shown in [38] that other MB algorithms can offer great advantages over GA including faster convergence and better performance.

2.2.2 Soft-Decision Iterative Decoding Algorithms

Soft-decision iterative decoding algorithms process soft information (real numbers) through the code graph. Therefore, they in general offer better error performance than that of the hard-decision algorithms with higher computational complexity. In the following, we briefly review two important soft-decision iterative decoding algorithms, namely sum-product algorithm and min-sum algorithm.

Sum-Product Decoding Algorithm

Sum-product decoding algorithm belongs to a class of the so-called "two-way algorithms" [39]. Two-way algorithms have a long and convoluted history and have different variants. Besides coding, they have applications in many other fields, such as statistics, optimization, expert systems, artificial intelligence and etc. In coding literature, the earliest known application of two-way algorithms is Gallager's *a posterior probability* (APP) decoding algorithms for LDPC codes [1–3]. MacKay and others [40, 41] realized that the sum-product decoding algorithm is a version of the so-called belief propagation (BP) algorithm in Bayesian networks in the fields of expert systems and artificial intelligence [42, 43]. Examples of applications of the sum-product algorithm in other fields include maximum APP (MAP) symbol detection on inter-symbol interference (ISI) channels [44], the well-known forward-backward algorithm for tracking hidden Markov models [45], the well-known BCJR algorithm in coding literature for minimizing bit error probability [46], to name a few. It is known that sum-product algorithm converges to the optimal bit-wise APP decoding on cycle-free Tanner graphs [39, 47, 48].

Conventionally, in the sum-product algorithm, at the initial step, each variable node is assigned two initial weights $w(0)$ and $w(1)$, corresponding to the reliability that "0" and "1" are sent, respectively. $w(0)$ and $w(1)$ are given by the following likelihood functions (assuming equally likely transmission of zeros and ones, and a memoryless channel):

$$w(0) = p(r|0),\ w(1) = p(r|1), \tag{2.2.6}$$

where $p(r|0)$ and $p(r|1)$ are the conditional probability density functions of receiving value r at that variable node given zero and one are transmitted, respectively. These initial messages can be computed when the characteristics of the channel are known. Due to the nature of the initial weights, each message in the graph also has two components, $\mathbf{m}(0)$ and $\mathbf{m}(1)$ corresponding to zero and one, respectively.

In the first step of each iteration, the message passing from a check node c to a variable node v is as follows:

$$\mathbf{m}_{c\to v}(0) = \sum_{\substack{\phi:\text{configurations} \\ \text{in which node } v \text{ is } 0}} \left(\prod_{v' \in N(c)\setminus v} \mathbf{m}^{\phi}_{v'\to c} \right), \tag{2.2.7}$$

$$\mathbf{m}_{c\to v}(1) = \sum_{\substack{\phi:\text{configurations} \\ \text{in which node } v \text{ is } 1}} \left(\prod_{v' \in N(c)\setminus v} \mathbf{m}^{\phi}_{v'\to c} \right), \tag{2.2.8}$$

where, $\mathbf{m}^{\phi}(.)$ means one of the two components of a message which is in configuration ϕ. A configuration of a check node c is a combination of the values (e.g., 0 and 1 in our case) of all the variable nodes in the set $N(c)$ which satisfies the parity-check constraint represented by check node c.

In the second step of each iteration, the message passing from a variable node v to a check node c is:

$$\mathbf{m}_{v\to c}(0) = w_v(0) \left(\prod_{c' \in N(v)\setminus c} \mathbf{m}_{c'\to v}(0) \right), \tag{2.2.9}$$

$$\mathbf{m}_{v\to c}(1) = w_v(1) \left(\prod_{c' \in N(v)\setminus c} \mathbf{m}_{c'\to v}(1) \right), \tag{2.2.10}$$

where, $w_v(x)$ represents the initial message corresponding to value x of variable node v.

A hard decision is made in each iteration at a variable node v based on the value of $w_v(0) \cdot \prod_{c' \in N(v)} \mathbf{m}_{c' \to v}(0)$ and $w_v(1) \cdot \prod_{c' \in N(v)} \mathbf{m}_{c' \to v}(1)$. If the first term is greater than the second term, the declared value of variable node v is zero, otherwise, the declared value of v is one.

What we described above is the implementation of sum-product algorithm in the probability domain. There are other ways to implement the sum-product algorithm in other domains (see, e.g., [49]). Examples include likelihood ratio (LR) domain, log-likelihood (LLR) domain and likelihood difference (LD) domain, where the messages are of the form $w(0)/w(1)$, $\ln(w(0)/w(1))$ and $w(0) - w(1)$, respectively. Clearly, the variable node and check node operations are changed when the algorithm is implemented in different domains.

For example, when the sum-product algorithm is implemented in the LLR domain, the initial message of variable node v is of the form: $\mathbf{m}_v = \ln\left(\frac{w_v(0)}{w_v(1)}\right)$. It can be shown [12, 49] that the operations performed in check nodes and variable nodes are given by the following equations, respectively,

$$\mathbf{m}_{c \to v} = 2 \cdot \tanh^{-1}\left(\prod_{v' \in N(c) \setminus v} \tanh \frac{\mathbf{m}_{v' \to c}}{2}\right), \tag{2.2.11}$$

$$\mathbf{m}_{v \to c} = \mathbf{m}_v + \sum_{c' \in N(v) \setminus c} \mathbf{m}_{c' \to v}. \tag{2.2.12}$$

A hard decision is made for each variable node v in each iteration based on the value

$\mathbf{m}_v + \sum_{c \in N(v)} \mathbf{m}_{c \to v}$. If this value is greater than zero, v is decoded as 0, otherwise, it is decoded as 1.

If the values 0 and 1 are mapped to alphabet +1 and -1, respectively, over a binary input additive white Gaussian noise (AWGN) channel, it is easy to see that the initial message of a variable node v is:

$$\mathbf{m}_v = \frac{2}{\sigma^2} y, \tag{2.2.13}$$

where y is the received value of variable node v and $\sigma^2 = \frac{N_0}{2}$ is the power spectral density of the AWGN. Thus, the sum-product algorithm requires the knowledge of the channel.

Min-Sum Decoding Algorithm

Min-sum algorithm is another well-known soft-decision iterative algorithm for decoding LDPC codes. Other names for min-sum are "max-sum" and "max-product" depending on the type of messages and the corresponding operations performed in variable and check nodes [39, 50]. The Min-sum algorithm converges to the ML solution for codewords on cycle-free Tanner graphs [39, 47, 48]. It can also be considered as an approximation to the sum-product decoding algorithm [49]. In general, the error performance of min-sum algorithm is inferior to that of sum-product by a few tenths of a dB[2]. However, compared to sum-product, it is much simpler to implement and does not require the estimate of noise power.

[2]It should be noted that although the performance of min-sum algorithm is worse in the "waterfall" region, it's been observed that min-sum can outperform sum-product algorithm in the error floor region [51].

In the LLR domain, the operation performed at the variable nodes is the same as Equation (2.2.12), and the operation performed at the check nodes is changed as follows:

$$\mathbf{m}_{c\to v} = \text{sign}\left(\prod_{v'\in N(c)\setminus v} \mathbf{m}_{v'\to c}\right) \cdot \min_{v'\in N(c)\setminus v} \{|\mathbf{m}_{v'\to c}|\}. \tag{2.2.14}$$

If the initial message is of the form given by (2.2.13), the constant factor σ^2 can be extracted from Equations (2.2.12) and (2.2.14) without changing the final result of the algorithm. Thus, the min-sum algorithm does not require the information about the channel SNR.

2.2.3 Analysis of Iterative Algorithms

The dynamics of iterative algorithms on Tanner graphs can be analyzed by several techniques.

Density Evolution [12] is an efficient method to predict the asymptotic behavior of iterative algorithms. It is used to track the evolution of the probability density functions of the messages passed on the Tanner graph and therefore to determine the *decoding threshold* for a given ensemble of LDPC codes and a given iterative algorithm. The decoding threshold is defined as the worst-case channel parameter such that for any channel parameter which is better than the threshold, a typical code in the ensemble is capable of reliable communication on this channel (in other words, the average fraction of incorrect messages in the graph tends to zero as the number of iterations increases [12]). Concentration results [12] guarantee that, for sufficiently large block lengths, the performance of the individual codes from the

ensemble is close to the average performance predicted by the asymptotic analysis and derived based on the assumption that the Tanner graph of the codes is cycle-free. The computational complexity associated with density evolution is very high for complex decoding algorithms such as soft-decision algorithms. Several techniques such as Gaussian approximation [52,53] have been proposed to reduce the complexity with a cost of less accuracy.

Another analytical tool for iterative decoding algorithms is *extrinsic information transfer* (EXIT) chart [16,54–56]. Instead of tracking the evolution of the probability density functions of the messages, the EXIT chart analysis tracks the evolution of a single parameter, i.e., the mutual information between the messages and the transmitted bits in each iteration. Compared to density evolution, the method of EXIT chart is less complex with a small degradation in the accuracy. It provides a way to visualize the behavior of iterative decoding algorithms which is important in both performance analysis and code design.

It is worth mentioning that all of the asymptotic analysis techniques are based on the assumption that the block length tends to infinity and thus the corresponding Tanner graph is cycle-free, which is not the case for practical codes [17]. Therefore, the above mentioned techniques provide more accurate analysis for large block lengths (say, $\geq 10^4$) than for short and intermediate block lengths.

2.3 Design of LDPC Codes

Based on the relationship between LDPC codes and Tanner graphs, the obvious way to construct an LDPC code is via the construction of a low-density Tanner graph (i.e., a low-density parity-check matrix). There are many design techniques with different design criteria, such as efficient encoding/decoding, near-capacity performance, good performance in the error floor region, etc. In this section, we briefly review several important design/construction techniques.

2.3.1 Random/Semi-Random Constructions

In random/semi-random constructions, the parity check matrices (or Tanner graphs) are obtained by computer random search with certain constraints. One common constraint is to avoid small cycles in the graph such that the girth of the graph is relatively large [57]. For example, most of the LDPC codes in the online code database [58], which are mostly regular, are constructed by computer random search with some constraints such as avoiding cycles of length 4 [11]. To design irregular LDPC codes, the choice of the degree distribution pair (λ, ρ) is important. As mentioned before, for a given ensemble with distribution pair (λ, ρ) and a given iterative algorithm, density evolution determines the decoding threshold. The authors of [13,31] showed how to optimize these distribution sequences for a variety of channels in the sense that, the optimal degree sequences give the best decoding threshold. After obtaining the optimal (λ, ρ), LDPC codes can be constructed by computer search. Using this approach, an LDPC code was designed with error performance within

0.0045 dB of the capacity for a binary-input AWGN channel [59]. This code has length $n = 10^7$ and rate $R = 1/2$. As mentioned before, since the asymptotic analysis tools are based on the cycle-free assumption, it is generally true that the design using density evolution is best applied to codes with low rate and large block length (e.g., $\geq 10^4$). It has been shown in [60,61] that when such optimized distribution pairs are used for short and medium block lengths, the codes tend to have high error floors.

One drawback of randomly generated codes is that they lack sufficient structure to have low-complexity encoding. Actually, encoding complexity has been one major practical objection to LDPC codes. In [32], to resolve this issue, the authors suggested a lower triangular structure for the parity-check matrix. An efficient encoding technique which does not restrict the structure of the parity-check matrix was presented in [62].

2.3.2 Geometric and Algebraic Constructions

In [33], the authors designed two classes of regular LDPC codes whose constructions are based on finite geometries. The resulting codes belong to the cyclic and quasi-cyclic classes of block codes such that simple encoder implementation using shift-register circuits is possible. One drawback of these classes of LDPC codes is that the column and row weights are relatively large which is undesirable since the decoding complexity per iteration is proportional to these values.

Margulis proposed a Caylay graph construction of (3, 6)-regular LDPC codes with rate 1/2 [5]. The performance of these codes was investigated by the authors of [63],

who also proposed a similar "Ramanujan-Margulis" code. It has been shown that these two classes of LDPC codes have significant weaknesses of having high error floors [64].

2.3.3 PEG Construction

Among the existing methods for constructing good LDPC codes at short and intermediate block lengths, one of the most successful approaches is the progressive-edge-growth (PEG) algorithm proposed by Hu *et al* [15]. PEG construction builds up a Tanner graph (or equivalently a parity-check matrix), for an LDPC code on an edge-by-edge basis and by maximizing the local girth at variable nodes in a greedy fashion. (Local girth of a node is defined as the length of the smallest cycle passing the node.) PEG algorithm is simple and also flexible in that it can construct codes with arbitrary length and rate and can construct both regular and irregular LDPC codes. In addition, PEG algorithm can be used to construct linear-time encodable LDPC codes. In Chapter 5, we propose a modification to PEG algorithm which improves a code's high SNR performance significantly.

Chapter 3

Performance Estimation of LDPC Codes under Hard-Decision Iterative Algorithms

3.1 Introduction

The asymptotic performance analysis of LDPC codes under iterative decoding algorithms is now a rather mature subject. As we mentioned in Chapter 2, tools such as density evolution [12] are applied to ensembles of codes to derive the thresholds of the ensembles under a variety of decoding algorithms, see, e.g., [12]. Concentration results [12] then guarantee that, for sufficiently large block lengths, the performance of an individual code from the ensemble is close to the average performance predicted by the asymptotic analysis and derived based on the assumption that the Tanner graph of the codes is cycle-free. In practice however, LDPC codes have finite length and their Tanner graphs contain cycles. This makes the performance of LDPC codes deviate from the asymptotic results. In fact, the finite-length analysis of LDPC codes is still an active area of research with many open problems. In this chapter, our focus

is on practical LDPC codes with block lengths in the range of a few hundred to a few thousand bits. We indiscriminately refer to such block lengths as short. Short LDPC codes are important since many delay and/or complexity sensitive applications cannot accommodate larger block lengths. Examples of such applications are wireless communications, magnetic recording, high-speed optical links and power-line communication systems, see, e.g., [65–68]. In working with short LDPC codes, especially in applications that require low error floors, one important issue is to estimate the performance of the code at low error rates. This however would be computationally expensive, or even infeasible, using the conventional Monte Carlo software simulation. Even with hardware implementations of iterative decoders, very deep error floors remain undetected. It is therefore of interest to develop methods that can efficiently estimate the performance of LDPC codes. This is the subject of this chapter. The iterative algorithms of interest in this chapter are hard-decision iterative decoding algorithms. Hard-decision iterative decoding algorithms such as, GA, GB and MB algorithms introduced in Chapter 2 are of practical interest because they are simple, fast and easy to implement. Because of their extremely low complexity compared to soft-decision algorithms, they are attractive for high-rate data communications and/or low-power applications.

In this chapter, we first study the graphical objects which cause failures for hard-decision algorithms. We then provide simple upper and lower bounds on the FER of LDPC codes decoded by hard-decision iterative decoding algorithms over a BSC in Section 3.3. In Section 3.4, we propose a general method to estimate the FER and the BER for hard-decision iterative decoding algorithms.

3.2 Trapping Sets of Hard-Decision Iterative Decoding Algorithms

As mentioned in Chapter 2, hard-decision iterative decoding algorithms are initiated by the outputs from the BSC. Hard messages are then passed between the variable nodes and the check nodes in the Tanner graph of the code through the edges of the graph in both directions and iteratively. The messages are generated at variable nodes and check nodes according to the updating rules of a specific iterative decoding algorithm. A decision is made in each iteration on the values of the variable nodes. In this work, for each variable node, the decision is made based on the majority of the initial message of the node and the outgoing messages passed along the edges emanating from the node. In other words, if the majority indicates that the value of the variable node is 1, then this bit is decoded as 1. Otherwise, it is decoded as 0. If there is a tie, the bit is decoded as its initial message. The algorithm is stopped if a codeword is detected (i.e., if the decisions on the values of the variable nodes satisfy all the check equations) or a maximum number of iterations is reached. Decoding fails if the decoder converges to a wrong codeword (undetected errors) or if the decoder can not converge to any codeword within the maximum number of iterations (detected errors). We assume that the decoder is symmetric [12]. To evaluate the performance, we can then assume that the all-zero codeword, or the all-one codeword if we consider $\{+1,-1\}$ as the alphabet space instead of $\{0,1\}$, is transmitted.

It is well-known that decoder failures of iterative algorithms can be related to graphical objects, particularly at low error rates. The authors of [69] showed that on

a binary erasure channel (BEC) all decoder failures of the belief propagation algorithm are related to graphical objects called *stopping sets*. No similar results are available for other channels and iterative decoding algorithms. Richardson [14] defined a more general graphical structure, called *trapping set*, which is related to the failures of iterative decoders in a more general context. A trapping set is defined as a set of variable nodes that cannot be eventually corrected by the decoder. We use (a, b) to denote a trapping set with a variable nodes and b unsatisfied parity-check equations. Trapping sets with both small values of a and b are main contributors to the total error rates, especially in the high SNR region. In general, trapping sets depend not only on the structure of the graph but also on the decoder inputs (channel) and the decoding algorithm. For example, while for the sum-product algorithm over the BEC, the trapping sets are precisely the stopping sets, for ML decoding over the BEC or any other channel, the trapping sets are the nonzero codewords.

The structure of the trapping sets of hard-decision iterative decoding algorithms on a BSC is in general unknown. The following theorem identifies the structure of certain trapping sets for majority-based (MB) algorithms. This theorem is in fact a generalization of Facts 3 and 4 of [14]. Facts 3 and 4 of [14] only apply to GA and GB on regular Tanner graphs with variable node degree 3 and check node degree 6. Recall that GA is a special case of MB algorithms with maximum possible order. Also note that, for every variable node in the Tanner graph, GB eventually switches to GA as the number of iterations tends to infinity. The following theorem extends Facts 3 and 4 of [14] to irregular codes with arbitrary degree distributions where different nodes are allowed to perform different MB algorithms. (For more details on MB algorithms,

refer to [38].)

Theorem 3.2.1 For an LDPC code with degree distribution pair (λ, ρ), suppose that variable node v performs MB of order ω_v, denoted by $\text{MB}^{\omega_v} (0 \leq \omega_v \leq d_v - 1 - \lceil d_v/2 \rceil)$, where d_v is the degree of variable node v. Consider a set of variable nodes S and denote its induced subgraph in the code's Tanner graph by $G(S)$. Also denote the check nodes which have an odd degree in $G(S)$ by $C_{odd}(S)$. If each variable node v in the Tanner graph has at most $\lceil d_v/2 \rceil + \omega_v - 1$ neighbors from $C_{odd}(S)$, then S is a trapping set.

Proof: We show that if all the variable nodes in S are initially in error and all the other nodes are initially correct, then the decoder fails to correct the errors and the initial error pattern remains unchanged throughout the iterations.

In the first iteration, the outgoing messages of a variable node v is the initial message $\mathbf{m}_v \in \{-1, 1\}$ of node v. Based on the updating rule for MB algorithms in the check nodes, it is easy to see that node v receives $-\mathbf{m}_v$ along all the edges connected to $C_{odd}(S)$ and $\mathbf{m}_v$ along all the edges connected to set $C \backslash C_{odd}(S)$, where C is the set of all the check nodes in the Tanner graph. This is regardless of whether node v is in the set S or not. Due to the structure of S, node v will therefore have at most $\lceil d_v/2 \rceil + \omega_v - 1$ incoming messages with their values equal to $-\mathbf{m}_v$. This implies that for each outgoing message of node v at the start of the second iteration, the number of extrinsic incoming messages with value $-\mathbf{m}_v$ is also at most $\lceil d_v/2 \rceil + \omega_v - 1$. Hence, the value of the outgoing message will remain $\mathbf{m}_v$. This is because, for the outgoing message to change, MB^{ω_v} requires at least $\lceil d_v/2 \rceil + \omega_v$ extrinsic incoming

messages with value $-\mathbf{m}_v$. This means that all the messages exchanged between variable nodes and check nodes, remain unchanged throughout the iterations. This implies that the decoded values of the variable nodes remain the same as the initial messages regardless of the number of iterations. ■

Theorem 3.2.1 describes an instance of one type of trapping sets which we refer to as *fixed-pattern*. For hard-decision algorithms in general, however, this is not the only type of trapping set. In our experiments with different LDPC codes and different hard-decision iterative decoding algorithms, we have observed the following types of decoder failures corresponding to different types of trapping sets. The observations are made by tracking the error positions at the output of the decoder throughout iterations.

1. *Fixed-pattern*: After a finite number of iterations, the error positions at the output of the decoder remain unchanged.

2. *Oscillatory-pattern*: After a finite number of iterations, the error positions at the output of the decoder oscillate periodically within a small set of variable nodes.

3. *Random-like*: Error positions change with iterations in a seemingly random fashion. The errors seem to propagate in the Tanner graph and result in a larger number of errors at the output of the decoder even if the initial error pattern has only a small weight.

Note that in the first two cases, there could be a transition phase, through which

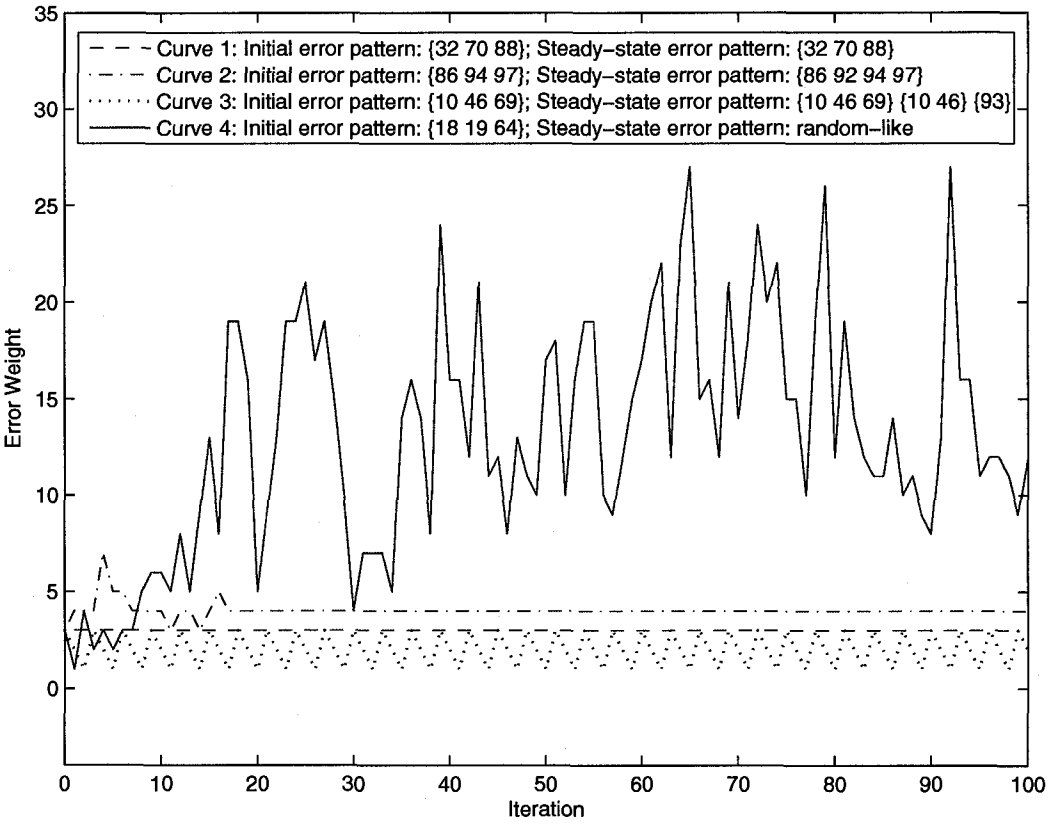

Figure 3.1: Example 3.2.1 : Different types of decoder failures.

the error pattern settles into the steady-state. The following example illustrates the properties of different decoding failures.

Example 3.2.1 We construct a random (100,50) LDPC code with $\lambda(x) = x^2$ and $\rho(x) = x^5$ with no cycle of length 4 in its Tanner graph. This code is referred as Code 1 in the rest of this chapter. For decoding, we use GA over BSC. By simulations, we observe all three types of decoder failures. Four examples are given in Figure 3.1, where we have plotted the number of errors (error weight) at the decoder output versus iteration number. The position of initial errors are also given as well as the position of steady-state errors for fixed-pattern and oscillatory-pattern trapping

sets. Two examples (Curves 1 and 2) illustrate fixed-pattern trapping sets. In the first case (Curve 1), the decoder is trapped into the steady-state error pattern right from the start, while in the second case (Curve 2), it goes through a transition phase before settling into the trapping set. The third example (Curve 3) demonstrates an oscillatory-pattern trapping set, where the error pattern oscillates within the trapping set {10,46,69,93} periodically with period 3. Finally the fourth example (Curve 4) illustrates a random-like trapping set, where an initial error pattern of weight 3 propagates through the graph as iterations progress. The number of errors increases and the error patterns are seemingly random. In fact in this example and all the other random-like cases that we have observed, the trapping set includes all the codeword bits, i.e., no bit can be eventually corrected by the decoder.

In our experiments we also observed that the percentage of random-like failures out of the total failures increases on average as the weight of the initial error patterns increases. For the Code 1, this percentage is 64.5% for the initial error patterns of weight 3, and 95.6% for the initial error patterns of weight 4. □

The relationship between the trapping sets and the structure of the Tanner graph is in general complicated. With the exception of Theorem 3.2.1, which is a generalization of Facts 3 and 4 of [14], we are not aware of any other such results. Our experiments show that in many cases cycles in the Tanner graph are part of the trapping sets. The relationships among the cycles, the trapping sets and the initial error patterns that trap the decoder in the trapping sets however seem to be complex. The following example provides some insight into this complexity.

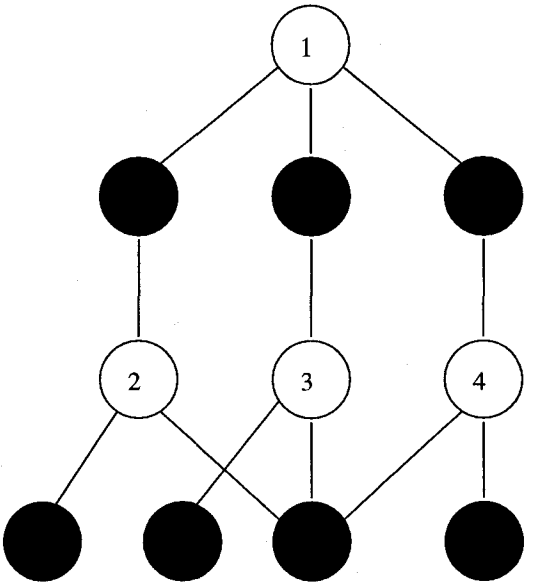

Figure 3.2: Tanner graph of Example 3.2.2.

Example 3.2.2 Consider a Tanner graph with girth 6 and the set of variable nodes V . Theorem 3.2.1 implies that a set $\mathcal{V} \subset V$ of 3 variable nodes in a cycle of length 6 is a trapping set for GA applied to this Tanner graph over a BSC. This is if all the other variable nodes $v \in V$ are connected to at most $d_v - 2$ induced odd-degree check nodes in the subgraph induced by $\mathcal{V}$. Now consider the graph structure presented in Figure 3.2 as part of this Tanner graph, where variable nodes and check nodes are represented by empty and full circles, respectively. It is easy to see that the variable set $V_1 = \{1, 2, 3\}$ is a trapping set based on Theorem 3.2.1. In particular variable node 4 has only one induced odd-degree neighboring check node. (Note that we assume all the other variable nodes, not shown in the figure, also satisfy the condition of Theorem 3.2.1.) In fact, one can see that sets $V_2 = \{1, 2, 4\}$ and $V_3 = \{1, 3, 4\}$ are also fixed-pattern trapping sets. However, if the initial error pattern is $V_1 \cup V_2 \cup V_3 = \{1, 2, 3, 4\}$, one can see that after two iterations GA can correct all the errors. □

It is noteworthy that while cycles of length 6 can be trapping sets for GA, other MB algorithms may be able to correct them.

3.3 Bounds on the FER

Several work has been performed to investigate the performance of finite-length LDPC codes based on the trapping sets. In [69], relating the decoding failures of the belief propagation algorithm over a BEC to the stopping sets (a special case of trapping sets), the authors showed failures in this case have a fully combinatorial characteristic which enables one to derive the average bit and frame erasure probabilities for an ensemble of LDPC codes. For channels other than the BEC, however, no similar results are available. Another step in analyzing the performance of finite-length LDPC codes was taken by Richardson [14], where the performance of a given LDPC code, particularly in the error floor region, was related to the trapping sets of the underlying graph. In particular, Richardson obtained very good estimates for the frame error rate (FER) floors of LDPC codes on additive white Gaussian noise (AWGN) channels by identifying and enumerating the most relevant trapping sets. This rather ad hoc process however is tedious and relies partly on heuristics and partly on the algebraic structure of the graphs. Using the same general approach, most recently, Cole *et al.* [51], devised an efficient algorithm for estimating the FER floors of soft-decision LDPC decoders on the AWGN channel using importance sampling. Importance sampling was also used to estimate error rates of short LDPC codes (block length less than 100) in [70, 71]. A simple ensemble estimate of the level of error floor of irregular

repeat-accumulate (IRA) codes (a subclass of LDPC codes) was presented in [72]. Estimations of the FER of GA and GB for certain categories of LDPC codes were also presented in [73, 74].

In this section, we propose a simple method to estimate the performance of LDPC codes on the BSC decoded by hard-decision iterative algorithms. The estimate is obtained through an upper and a lower bound on the frame error rate (FER) of the code. The gap between the bounds vanishes for sufficiently small channel crossover probabilities ε. To derive the bounds, we enumerate all the possible error patterns of Hamming weight less than or equal to some small integer K at the input to the decoder. These are the main contributors to the performance of the code at small values of ε. The larger the value of K, the tighter the bounds. Unlike the methods of [14, 51, 73, 74] which first identify the most relevant trapping sets and then evaluate their contribution to the error floor, our approach does not require the identification of the trapping sets and is thus much simpler.

The method presented in this section is based on elementary probability and is conceptually quite general in that it can be applied to any hard-decision decoding algorithm (not just that of LDPC codes). In practice, however, it is the computational complexity of this algorithm which makes it attractive for iterative decoding of LDPC codes, and for example not for the hard-decision ML decoding of LDPC or other codes.

Consider a hard-decision decoder for a code of block length n which can correct all the error patterns of weight $J-1$ and smaller. The complexity of the proposed method for such a decoder increases exponentially with J, essentially as n^J . For ML decoding, $J = \lceil (d_{min} - 1)/2 \rceil$, which is usually a large number for good codes

with the block lengths of interest in this section. This makes the proposed algorithm computationally infeasible for ML decoding. Our observations however show that for the hard-decision iterative decoding of a large number of popular short LDPC codes, $J = 2, 3$,or 4. For example, we tested all the codes with rate $1/2$ and $n \leq 2048$ in [58] under GA, and they all have $J \leq 4$. Later in this chapter, we show through examples of various regular and irregular LDPC codes decoded by different hard-decision iterative algorithms that our bounds are tight at small values of ε and provide good estimates for the FER in this region.

3.3.1 Lower and Upper Bounds on FER

Consider a given LDPC code with block length n decoded by a given hard-decision iterative algorithm over a BSC with crossover probability ε. The FER can be written as:

$$\begin{aligned} FER &= \sum_{i=1}^{K} P(e|i)p_i + P(e|i > K)P(i > K) \\ &= P(K) + P(e|i > K)P(i > K), \end{aligned} \tag{3.3.1}$$

where e is the event of having a frame decoded erroneously, i denotes the weight of error patterns at the input to the decoder, and p_i is the probability of having i errors at the output of the channel (or the input of the decoder), given by the binomial distribution $p_i = \binom{n}{i}\varepsilon^i(1-\varepsilon)^{n-i}$. From Equation (3.3.1), it is easy to see that,

$$P(K) \leq FER \leq P(K) + \left(1 - \sum_{i=0}^{K} p_i\right). \tag{3.3.2}$$

The probabilities $P(e|i=j)$, $1 \leq j \leq K$ in $P(K)$ are calculated by enumerating all the $\binom{n}{j}$ error patterns of weight j at the input of the decoder, performing the iterative decoding for each of them and finding the ratio of decoding failures (non-convergence to all-zero codeword).

One can see that the bounds in (3.3.2) become tighter as K increases. In fact the lower and upper bounds coincide for $K = n$, and we have $FER = P(n)$. For a given value of K, where $P(K) \neq 0$, the bounds become tighter as ε tends to zero. In the limit, both bounds converge to the same value $P(e|i=J)p_J$, where J is the smallest integer less than or equal to K for which $P(e|i=J) \neq 0$. This implies that for sufficiently small values of ε, the FER tends to zero as a function of ε^J. To be more precise, the dominant (largest) term of FER is equal to $N_J\varepsilon^J(1-\varepsilon)^{n-J}$, where N_J is the number of error patterns of weight J that are decoded erroneously.

We will show later in this chapter, for the block lengths and ε values of interest, that the computational complexity of the proposed method can be much smaller than that of Monte Carlo simulations.

3.4 Error Rate Estimation

The bounds derived in the previous section are tight only for sufficiently small crossover probabilities of the channel and thus provide good estimates for the FER in this region. In this section, we propose a method to estimate the performance of LDPC codes with very good accuracy over a much wider range of crossover probabilities. The method is applicable to any given regular or irregular LDPC code, decoded by

any hard-decision iterative algorithm. Using certain assumptions, we relate the FER and the BER of a given LDPC code, to the size J and the number $|E_J|$ of the smallest error patterns that the decoder fails to correct. For short LDPC codes, as J appears to be a small number, usually ≤ 4, the complexity of the method is manageable, and can be much smaller than that of Monte Carlo simulations especially for low error rates. Despite its simplicity, the proposed method is very effective in providing very accurate estimates of both the FER and the BER over the whole range of crossover probabilities of interest. Unlike the approach of [69] and [72], the proposed method is applied to a *given* LDPC code, not to an ensemble of codes. From a practical standpoint, this is more desirable, as there can be a large difference between the ensemble average and the performance of a given code at short block lengths, especially for low error rates [69]. Like the bounds presented in the previous section, compared to the methods of [14, 51, 73, 74] which first identify the most relevant trapping sets and then evaluate their contribution to the error floor, our approach does not require the identification of the trapping sets and is thus much simpler. Moreover, our approach is not only applicable to the error floor region, but also provides very accurate estimates in the waterfall region. In addition, we provide estimates for the BER, which is not even discussed in [14, 51, 73, 74]. In fact one of the important contributions of this work is to shed some light on the generally complex structure of the error events for iterative decoding, and to be able to establish simple approximate relationships between these complex events and the smallest initial error patterns that the decoder fails to correct. These relationships are the key in deriving the estimates.

3.4.1 FER Estimation

We denote the set of all the error patterns of weight i by S_i, and those that cannot be corrected by the decoder by E_i. Clearly, $|S_i| = \binom{n}{i}$. Suppose that the decoder can correct all the error patterns of weight $J-1$ and smaller, i.e., $|E_i| = 0$, $\forall i < J$. Also suppose that there are $|E_J| \neq 0$ weight-J error patterns that the decoder fails to correct. The FER is then equal to:

$$FER = \sum_{i=J}^{n} P(e|i)p_i = \sum_{i=J}^{n} \frac{|E_i|}{|S_i|} p_i = \sum_{i=J}^{n} |E_i| \varepsilon^i (1-\varepsilon)^{n-i}, \tag{3.4.3}$$

The first term of the summation in (3.4.3) is denoted by $P(J)$ and is equal to $|E_J|\varepsilon^J(1-\varepsilon)^{n-J}$. To estimate the FER, we enumerate E_J and calculate $P(J)$ precisely. For the other terms in (3.4.3), we estimate $|E_i|$ as a function of $|E_J|$ as follows.

For a given $i > J$, we partition the set S_i as $S_i = S_i' \cup \overline{S_i'}$, where S_i' is the set of error patterns of weight i, each containing at least one element of E_J as its subpattern, and $\overline{S_i'}$ is the complement set of S_i' in S_i. The set $\overline{S_i'}$ thus contains the elements of S_i that do not have any elements of E_J as their sub-patterns. We make the following assumptions:

Assumption (a): No error pattern in the set S_i' can be corrected by the decoder.

Assumption (b): Every error pattern in the set $\overline{S_i'}$ can be corrected by the decoder.

By the above assumptions, we have $|E_i \cap S_i'| = |S_i'|$ and $|E_i \cap \overline{S_i'}| = 0$, and therefore $|E_i| = |E_i \cap S_i'| + |E_i \cap \overline{S_i'}| = |S_i'|$. The key in estimating $|E_i|$, $i > J$, is that $|S_i'|$ can be approximated as a function of $|E_J|$ using the following combinatorial arguments. Consider the probability $\mathcal{P}$ that a randomly selected weight-i error pattern contains

at least one element of E_J as its sub-pattern. We then have

$$\mathcal{P} = \frac{|S_i'|}{|S_i|} \approx 1 - \left(1 - \frac{|E_J|}{|S_J|}\right)^{\binom{i}{J}}. \tag{3.4.4}$$

The first equality follows from the fact that there are $|S_i|$ equally likely possibilities out of which $|S_i'|$ are favorable cases. The second part of the equation is a consequence of approximating the random experiment by a sequence of $\binom{i}{J}$ independent and identically distributed Bernoulli trials, each involving the selection of a weight-J error pattern with "success" defined as the pattern being in E_J. The probability of success is then $|E_J|/|S_J|$, and the probability of having at least one success in $\binom{i}{J}$ trials is $1 - (1 - |E_J|/|S_J|)^{\binom{i}{J}}$ following the binomial distribution. From (3.4.4), we have

$$|S_i'| \approx |S_i| \left[1 - (1 - \frac{|E_J|}{|S_J|})^{\binom{i}{J}}\right] \approx |S_i| \frac{|E_J|\binom{i}{J}}{|S_J|} = |E_J| \binom{n-J}{i-J}, \tag{3.4.5}$$

where the second approximation is obtained by considering only the first two terms in the binomial expansion.

Note that Assumption (a) is valid for the belief propagation decoder on a BEC. For hard-decision iterative decoding algorithms on a BSC, although the two assumptions are not necessarily correct, they are approximately valid, especially for smaller values of i. The graph structure of Figure 3.2, as explained in Example 3.2.2, provides a counter-example for Assumption (a) as the set of variables $\{1, 2, 3, 4\}$ is in S_4' but can be corrected by the decoder. The following example however illustrates that

Assumptions (a) and (b) are statistically viable and result in good approximations for $|E_i|$, $i > J$.

Example 3.4.1 We construct a (200, 100) irregular LDPC code and use the GA algorithm for decoding. The degree distributions for the code, which are optimized for the BSC and GA, are given by $\lambda(x) = 0.1115x^2 + 0.8885x^3$ and $\rho(x) = 0.26x^6 + 0.74x^7$ [75]. (We refer to this code as Code 2.) The Tanner graph of the code is free of cycles of length 4, and the maximum number of iterations is 100. Under these conditions, $J = 3$ and $|E_3| = 1434$. We also have found that $|S_4'| = 280064$ and $|\overline{S_4'}| = 64404886$. The approximate value of $|S_4'|$ from Equation (3.4.5) is 282035.6. By passing the error patterns of S_4' and $\overline{S_4'}$ through the GA decoder, we obtain $|E_4 \cap S_4'| = 278799$ and $|E_4 \cap \overline{S_4'}| = 32109$. This means that only $(280064 - 278799)/280064 \approx 0.45\%$ of the error patterns in the set S_4' can be corrected by the decoder. It also demonstrates that almost all, i.e., $(64404886 - 32109)/64404886 \approx 99.95\%$, of the error patterns in the set $\overline{S_4'}$ can be corrected by the decoder. We can therefore see that, $|S_4'|$ and its estimated value from (3.4.5), i.e., 282035.6, provide close approximations for $|E_4|$, which is equal to $|E_4 \cap S_4'| + |E_4 \cap \overline{S_4'}| = 310908$. □

In general our observations show that the estimation $|E_i| \approx |S_i'|$ for a given value of i improves as the error correcting capability of the code improves, say for example, by increasing the block length for a given rate or by reducing the rate for a given block length. For example, for $i = 4$, while Example 3.4.1 demonstrates that the estimation works very well for Code 2, for Code 1 used in Example 3.2.1, we have $|S_4'| = 104812$ and $|E_4| = 710980$. It is noteworthy that even for cases such as Code

1, where the estimation $|E_i| \approx |S_i'|$ is rather poor, our FER estimate FER_U, given in Equation (3.4.7), provides a good approximation for the FER.

Combining Equations (3.4.3) and (3.4.5) with $|E_i| \approx |S_i'|$, $i > J$, we derive the following estimates for the FER:

"Lower" estimate,

$$FER_L(N) = P(J) + \sum_{i=J+1}^{N} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i}, \tag{3.4.6}$$

"Upper" estimate,

$$FER_U(N) = P(J) + \sum_{i=J+1}^{N} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i} + \sum_{i=N+1}^{n} p_i, \tag{3.4.7}$$

where $N \in \{J+1, J+2, \ldots, n\}$ is a parameter to be selected for the best accuracy of the estimates. It is clear that $FER_L(N) \leq FER_U(N)$, and the equality holds if and only if $N = n$. In this case, both estimates have the same value: $P(J) + \sum_{i=J+1}^{n} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i}$. The difference between the two estimates is the probability that the input error patterns to the decoder have a weight larger than N. While $FER_L(N)$ is derived based on the assumption that error patterns of weight larger than N occur with negligible probability, $FER_U(N)$ is obtained based on the assumption that for such input error patterns the decoder fails with probability one. Our observations show that in practice there exists a certain threshold N_0 for error weights, around which a relatively abrupt change in the percentage of failures occurs. This is such that for error weights larger than N_0, the probability of failure

goes to one very rapidly. The following example illustrates this.

Example 3.4.2 Consider two regular LDPC codes with parameters (210, 35) and (1008, 504). The first code, referred to as Code 3, has variable node degree 5 and check node degree 6, while for the second code, referred to as Code 4, these values are 3 and 6, respectively. Code 3 is randomly constructed and Code 4 is taken from [58]. Both codes have no cycles of length 4 in their Tanner graphs. Code 3 is decoded by the MB algorithm of order zero (MB^0). For the degree distributions of this code, MB^0 has a better threshold than GA does [38]. Code 4 is decoded by GA. The maximum number of iterations for both codes is 100.

For Code 3, and also Codes 1 and 2, described in Examples 3.2.1 and 3.4.1, respectively, we randomly generate 10^7 or $\binom{n}{i}$, whichever is smaller, error patterns of each weight N_0-2, N_0-1, N_0, N_0+1, N_0+2. For Code 4, we generate only 10^6 error patterns of each weight due to the slower simulations per error pattern. The value N_0 for each code corresponds to the error-weight threshold around which the percentage of failures changes rather abruptly. The generated error patterns are passed through the corresponding decoders and the percentage of decoding failures are recorded. The results are summarized in Table 3.1. These results demonstrate the rapid change of the decoder failure rate with the initial error weight in the vicinity of the error weight N_0. □

Later in Section 3.6, we will see that $FER_U(N_0)$ provides a very accurate estimate of the FER for all channel crossover probabilities of interest. This suggests the following practical approach for determining N_0. We perform Monte Carlo simulations

Table 3.1: Percentage of decoding failures R_e caused by error patterns with different weights i. N_e: number of simulated error patterns.

Code 1		Code 2, $N_e = 10^7$		Code 3, $N_e = 10^7$		Code 4, $N_e = 10^6$	
i	R_e	i	R_e	i	R_e	i	R_e
2	0%, $N_e = 4950$	7	9.69%	11	0.094%	36	4.48%
3	0.69%, $N_e = 161700$	8	17.15%	12	0.745%	37	17.47%
4	18.13%, $N_e = 3921225$	9	35.51%	13	5.61%	38	39.34%
5	76.32%, $N_e = 10^7$	10	66.86%	14	26.14%	39	57.75%
6	97.84%, $N_e = 10^7$	11	95.99%	15	60.15%	40	79.16%

at high FER values, say around $0.01 - 0.1$. We then choose N_0 such that the estimate $FER_U(N_0)$ is the closest to the simulated FER. As the Monte Carlo simulations are performed at high FER values, their complexity is low and easily manageable.

3.4.2 BER Estimation

At the first glance, the problem of BER estimation may seem very complicated as from Example 3.2.1, we observe that even for the initial error patterns of the same weight, the number of bits in error at the end of the decoding can be quite different. Our further study into this however reveals that despite this variety of possibilities, one can estimate the average number of bit errors depending on the initial weight i of the error patterns. In general, we expect the conditional average BER given the initial error weight i to be an increasing function of i. To simplify the analysis however, we partition the range of $J \leq i \leq n$ into two subsets, $J \leq i < N_0$ and $N_0 \leq i \leq n$. For these partitions, we estimate the average number of bit errors by J and M, respectively, where M is the estimate of the average number of bit errors for error patterns of weight N_0, obtained by Monte Carlo simulations. We thus derive

our estimate for the BER based on the FER estimate $FER_U(N_0)$, proposed in the previous subsection, as follows,

$$BER \approx \frac{J}{n}\left[P(J) + \sum_{i=J+1}^{N_0-1} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i}\right] + \frac{M}{n}\left[|E_J| \binom{n-J}{N_0-J} \epsilon^{N_0}(1-\epsilon)^{n-N_0} + \sum_{i=N_0+1}^{n} p_i\right]. \quad (3.4.8)$$

In the following we provide the rationale behind the derivation of Equation (3.4.8). We recall that $|E_i| \approx |S_i'|$. We partition S_i' further as $S_i' = S_i'' \cup \overline{S_i''}$, where S_i'' is the set of error patterns of weight i that have one and only one element of the set E_J as their subset. Using similar discussions as those used for the derivation of Equation (3.4.5), we have

$$|S_i''| \approx |S_i| \binom{i}{J} \left(1 - \frac{|E_J|}{|S_J|}\right)^{\binom{i}{J}-1} \frac{|E_J|}{|S_J|}. \quad (3.4.9)$$

It appears that for the error patterns of weight $J \leq i < N_0$, the ratio of $|S_i''|/|\overline{S_i''}|$ is a large number and thus a large majority of the error patterns in the set E_i belong to the set S_i''. This means that, for this range of error weights, many of the error patterns that result in decoding failures have one and only one element of the set E_J as their sub-pattern. For these error patterns we now make the following assumption:

Assumption (c): For the error patterns of weight $J \leq i < N_0$, the number of bit errors at the end of decoding is approximately J.

Furthermore, we assume:

Assumption (d): For the error patterns of weight $N_0 \leq i \leq n$, the number of bit errors at the end of decoding is on average M, where M is the estimate of the average

number of bit errors for error patterns of weight N_0.

To determine M, we simulate a given number N_e (say $10^3 - 10^4$) of randomly generated error patterns of weight N_0.

Example 3.4.3 Consider Codes 1-4, described previously, with the corresponding decoding algorithms. Figure 3.3 shows the ratio of $|S_i''|/|\overline{S_i''}|$, obtained based on Equation (3.4.9), versus the initial error pattern weight i in the range $[J+1, N_0]$. For Code 1, $N_0 = J+1 = 4$, and thus we only have one point. For the other codes, the ratio is a decreasing function of i. For all codes, the ratio is relatively large over the considered range of i values. Over this range therefore, one can assume that $|E_i| \approx |S_i'| \approx |S_i''|$.

For each code and decoder pair, we also perform the following experiments and record the average number of bit errors: i) Test all the error patterns in E_J; ii) Test a given number of randomly generated error patterns in the set S_i'', where $i = \lfloor (J+N_0)/2 \rfloor$; iii) Test a given number of randomly generated error patterns in the set S_{N_0}; iv) Test a given number of randomly generated error patterns in the set S_i, where $i = \lfloor (N_0+n)/2 \rfloor$. The results are presented in Table 3.2. In the table, the number of tested patterns in each experiment is denoted by N_e, and the average number of bit errors by $\overline{B}$. For Code 1, experiments i and ii are identical and thus only one entry is given. As can be seen, for all 4 codes, $\overline{B} \approx J$ in experiments i and ii. This indicates that even midway through the interval $[J, N_0]$, the average conditional BER is very close to J. The value of $\overline{B}$ in experiment iii is in fact M. Experiment iv examines the average conditional BER given i initial errors, where i is the midpoint

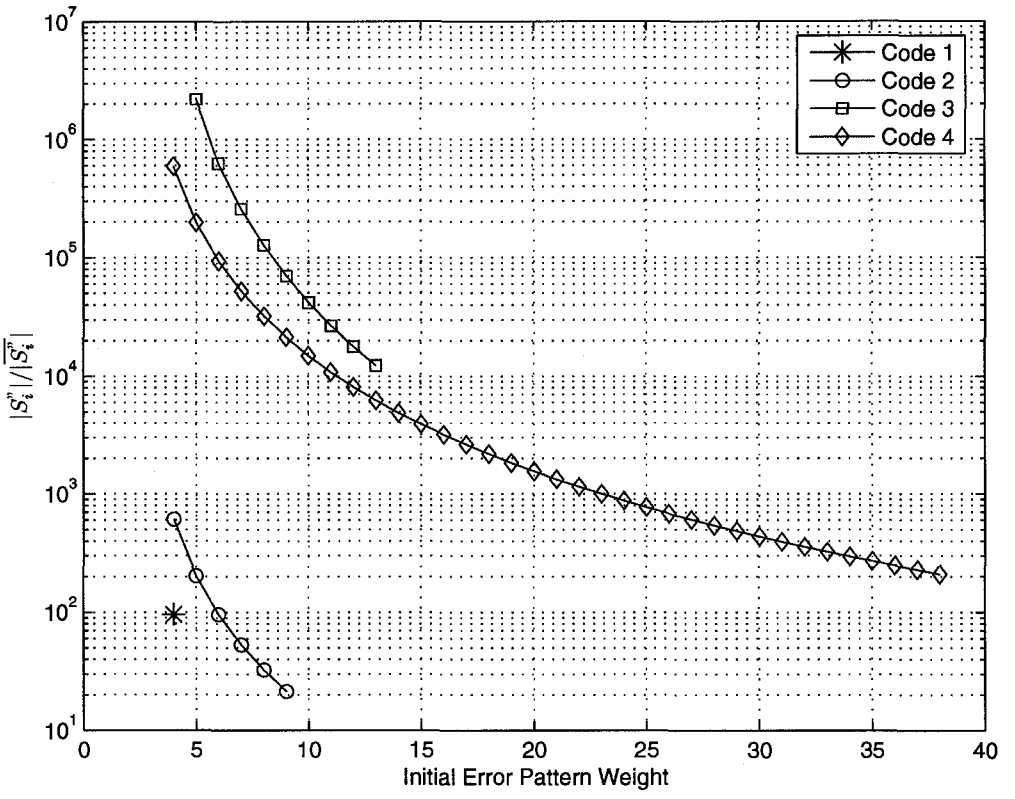

Figure 3.3: $|S_i''|/|\overline{S_i''}|$ versus initial error pattern weight i for Codes 1 - 4, over the range $J \leq i \leq N_0$.

of the interval $[N_0, n]$. As expected, the results of Table 3.2 for this experiment show that the average conditional BER is larger, and in some cases much larger, than M. In general $\overline{B}$ increases with i. Although $\overline{B}$ is close to M for i values closer to the start of the interval, it deviates from M for the larger values of i. Nevertheless as the contribution of the former values to the total BER is more than that of the latter values, Assumption (d) still provides a very good estimate of the total BER. □

Table 3.2: Average number of bit errors $\overline{B}$ for initial error patterns with different weights i. N_e: number of simulated error patterns.

	Code 1			Code 2			Code 3			Code 4		
Set	i	N_e	$\overline{B}$	i	N_e	$\overline{B}$	i	N_e	$\overline{B}$	i	N_e	$\overline{B}$
E_J	3	1115	10.02	3	1434	3.01	4	18	3.94	3	192	2.92
$S''_{\lfloor (J+N_0)/2 \rfloor}$	-	-	-	6	10^6	3.66	8	10^6	4.08	20	10^6	3.11
S_{N_0}	4	10^6	13.91	9	10^6	7.73	13	10^6	46.95	38	10^6	143.93
$S_{\lfloor (N_0+n)/2 \rfloor}$	52	10^6	51.62	104	10^6	104.98	111	10^6	108.8	523	10^6	518.63

3.5 Computational Complexity

To obtain the bounds on FER or to obtain an estimate of FER, one needs to enumerate $\sum_{i=1}^{J} \binom{n}{i}$ error patterns. For different error patterns, the iterative decoder performs the decoding and counts the number of decoding failures. Suppose that we are interested in estimating the performance of the LDPC code at an FER of p using Monte Carlo simulations and would like to observe at least m codeword errors for a reliable result. Assuming that the average number of computations required for iterative decoding in the two cases are the same, the ratio of the computational complexities of the Monte Carlo simulation and the proposed estimation method is:

$$\eta = \frac{m}{p \sum_{i=1}^{J} \binom{n}{i}}. \tag{3.5.10}$$

For BER estimation, we need to perform an extra N_e iterative decoding to estimate M, and about $100m$ iterative decoding to obtain N_0. These are usually negligible compared to $\sum_{i=1}^{J} \binom{n}{i}$.

It can be seen that η is a function of n, J, p and m. It increases with increasing m and with decreasing p, n and J. It often appears that J is a small number (≤ 4). For example, as mentioned before, we have tested all the codes with rate $1/2$ and $n \leq 2048$ in [58] under GA, and they all have $J \leq 4$. The value of m is often selected in the range of a few tens to a few hundreds. With given values for J, m and n, the proposed estimation method is more efficient than the Monte Carlo simulation ($\eta > 1$) if $p < m/\sum_{i=1}^{J} \binom{n}{i}$. In fact, for the block length and ε values of interest, the computational complexity of the proposed methods (bounds and estimations) can be much smaller than that of the Monte Carlo simulations.

One should also note that the value of η in (3.5.10) only reflects the saving in complexity compared to *one* Monte Carlo simulation point. Unlike our estimations that can be easily calculated for different values of ε once we obtain values of J and $|E_J|$, in Monte Carlo simulations, for each simulation point, a new set of input vectors has to be generated and simulated. This makes the proposed method even more attractive from the complexity viewpoint.

3.6 Simulation Results

To show that our methods can be applied to both regular and irregular LDPC codes and to a variety of hard-decision algorithms, in this section, we perform experiments on four pairs of code/decoding algorithm. These are Codes 2-4, described in previous sections, and Code 5, which is a (1998, 1776) high rate (4, 36)-regular LDPC code taken from [58]. For convenience, the code's parameters and the corresponding

Table 3.3: Code's parameter

Code Number:	Code 2	Code 3	Code 4	Code 5
(n, k)	(200,100)	(210,35)	(1008,504)	(1998,1776)
Decoding Algo.	GA	MB^0	GA	GA
N_0	9	13	38	10
M	7.73	46.95	143.93	133.57

Table 3.4: Categorization of error patterns of weight J for different LDPC codes.

	Code 2	Code 3	Code 4	Code 5
J	3	4	3	3
$\|E_J\|$	1434	18	192	200479
Undetected Error	0	0	0	0
Fixed-pattern	1303(90.87%)	14(77.8%)	165(85.9%)	200438(99.98%)
Oscillatory-pattern	131(9.13%)	4(22.2%)	27(14.1%)	41(0.02%)
Random-like	0	0	0	0

decoding algorithms are listed in Table 3.3. In Table 3.3, we also list the values of N_0 and M used in the FER and BER estimates for each code. In our simulations, the maximum number of iterations is 100 for all the decoders and for each crossover probability, we simulate until we obtain 100 codeword errors ($m = 100$).

Table 3.4 shows the values of J and $|E_J|$ for the four codes. It also shows the number and the percentages of different types of decoding failures for E_J. As can be seen, for all the codes, most of the decoding failures caused by initial error weight J have fixed patterns and there is no random-like failure.

The lower and upper bounds of Equation (3.3.2) on the FER of Codes 2-4 for different values of K are given in Figures 3.4 and 3.5, respectively. For each code, the simulated FER is also plotted.

From the figures, it can be seen that the simulated FER is always located between

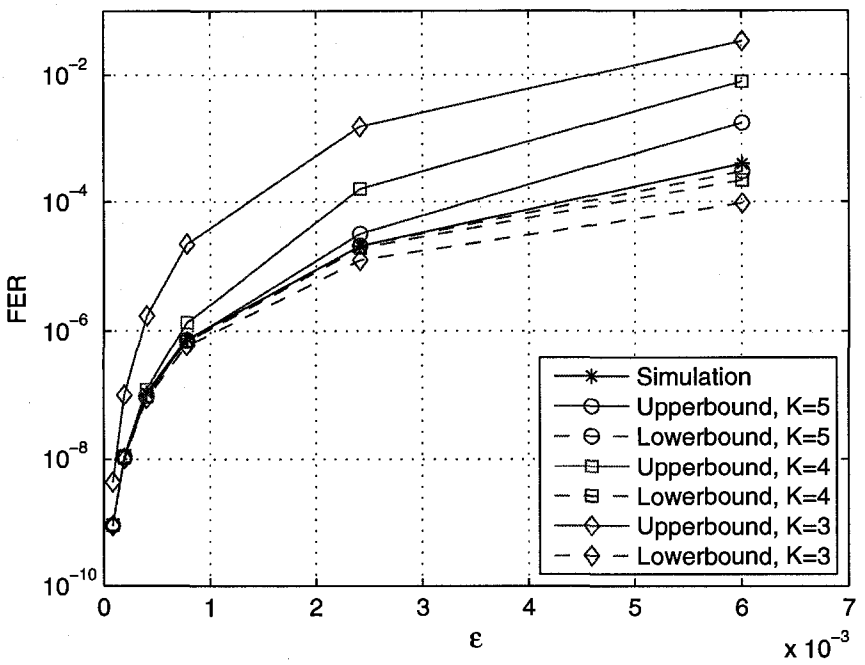

Figure 3.4: FER simulation and bounds for Code 2 under GA.

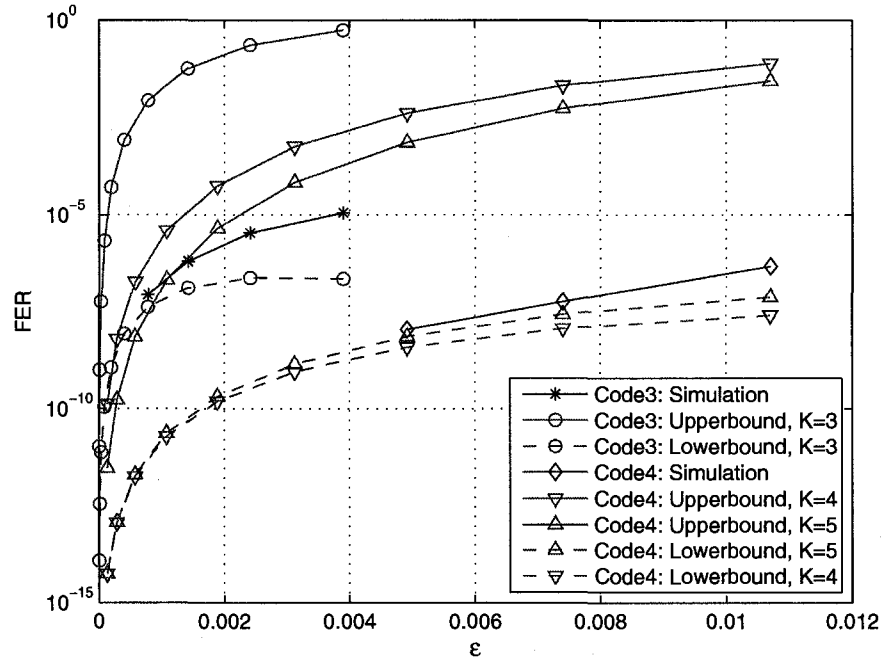

Figure 3.5: FER simulations and bounds for Code 3 under MB^0 and Code 4 under GA.

the proposed lower and upper bounds. The bounds are tighter for larger values of K and for smaller values of ε. In particular, for any value of K the bounds are tight for sufficiently small values of ε and provide good estimates of FER. For all three codes, the lower bound provides a better estimate of FER. The tightness of the lower bound implies that $P(E \mid i > K)P(i > K)$ is much smaller than $P(K)$.

For Codes 2 and 4, $J = 3$ and for Code 3, under MB^0, $J = 4$. The FER of codes 2 and 4 thus tends to zero as a function of ε^3 as ε tends to zero. For code 3, the asymptotic rate of decrease for FER is ε^4. Comparison of the bounds in Figures 3.4 and 3.5 shows that for given values of K and ε the gap between the lower and upper bounds increases with block length. This is justifiable as the difference between the bounds in Equation (3.3.2) is equal to $1 - \sum_{i=0}^{K} p_i$, which is an increasing function of block length n.

Figures 3.6 - 3.9 show the FER and BER performances of Codes 2 - 5, obtained by simulations, respectively. In these figures, we have also given the upper and the lower estimates of the FER for different values of N, as well as the estimates for the BER. For the BER estimates, the values of N_0 and M used in the estimates for Codes 2 - 5 are given in Table 3.3. From the figures, it can be seen that for each code, the lower estimate $FER_L(N)$ improves by increasing N. However, even the best estimate $FER_L(n)$ is only good at sufficiently small values of crossover probability. It fails to provide an accurate estimate at higher error rates. On the other hand, for all the codes, $FER_U(N_0)$ provides an impressively accurate estimate of the FER over the whole range of crossover probabilities of practical interest. It can also be seen that the BER estimates for all the codes follow the simulated BER curves very closely.

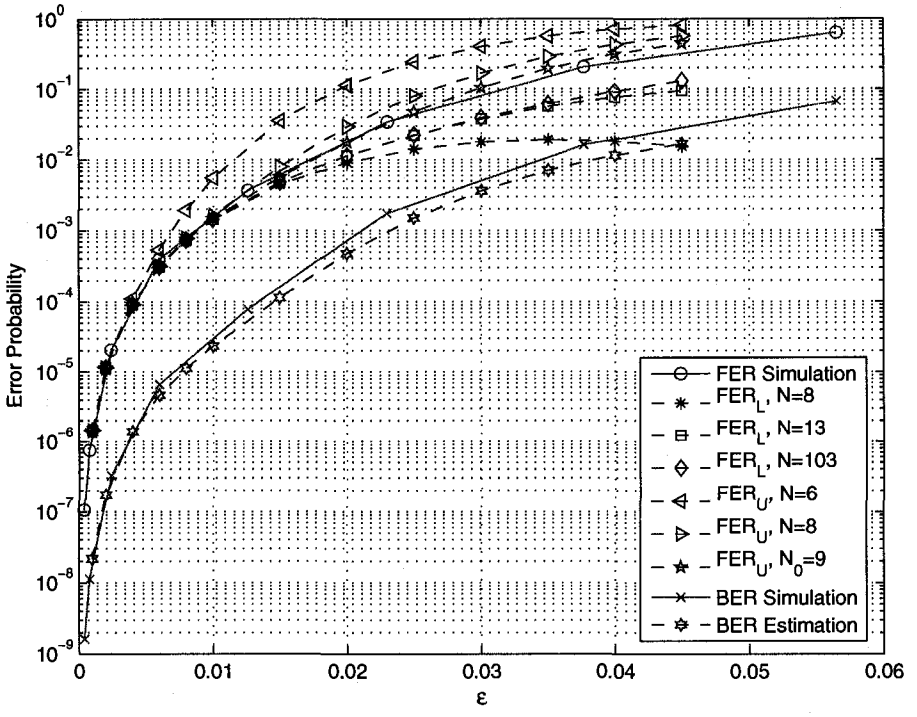

Figure 3.6: FER and BER estimations and simulations of Code 2 decoded by GA

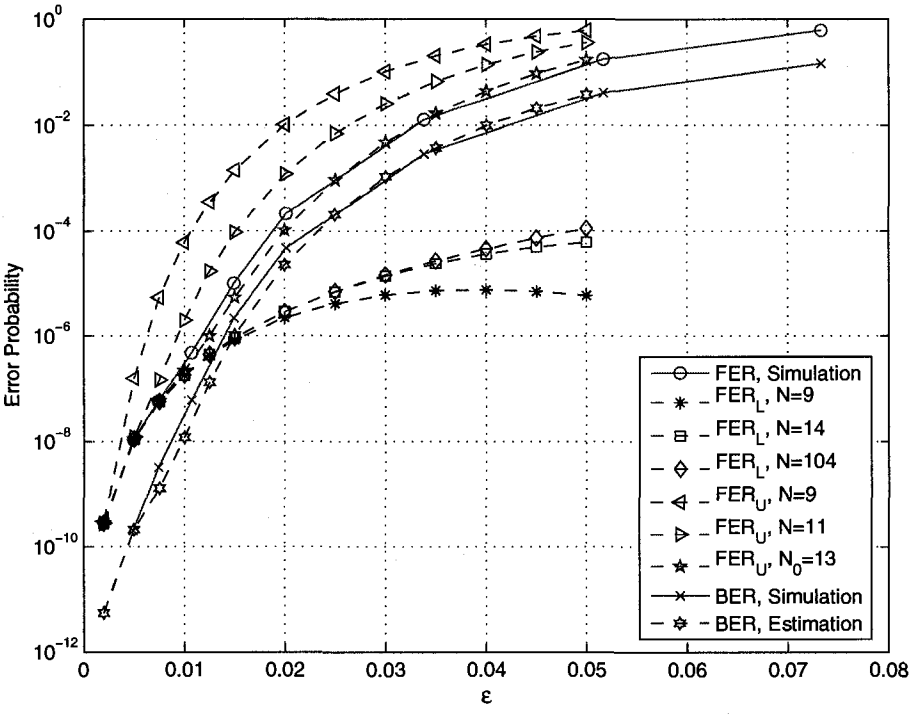

Figure 3.7: FER and BER estimations and simulations of Code 3 decoded by MB^0.

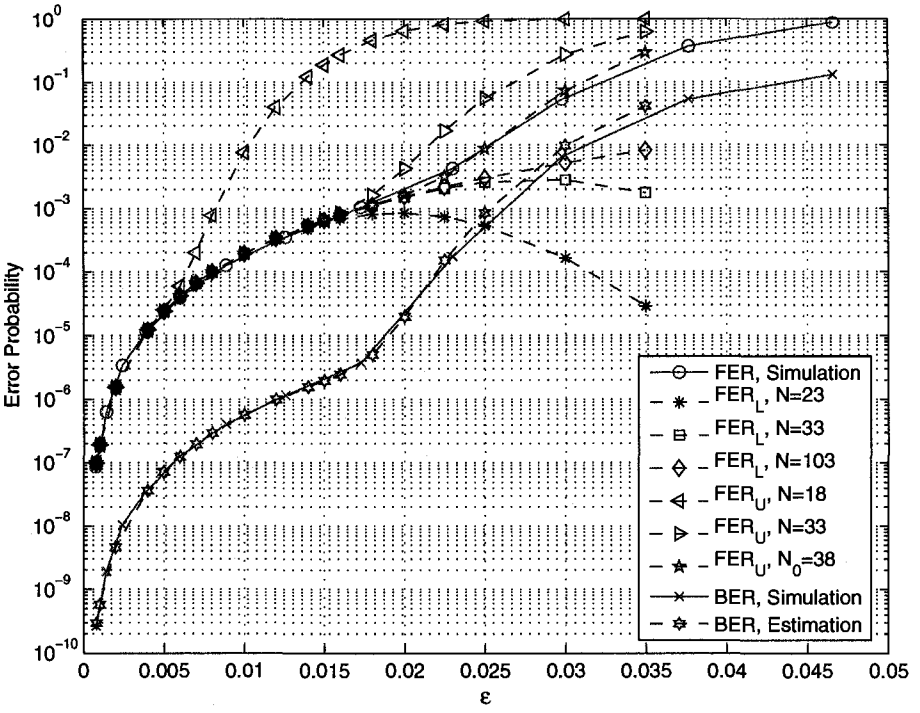

Figure 3.8: FER and BER estimations and simulations of Code 4 decoded by GA.

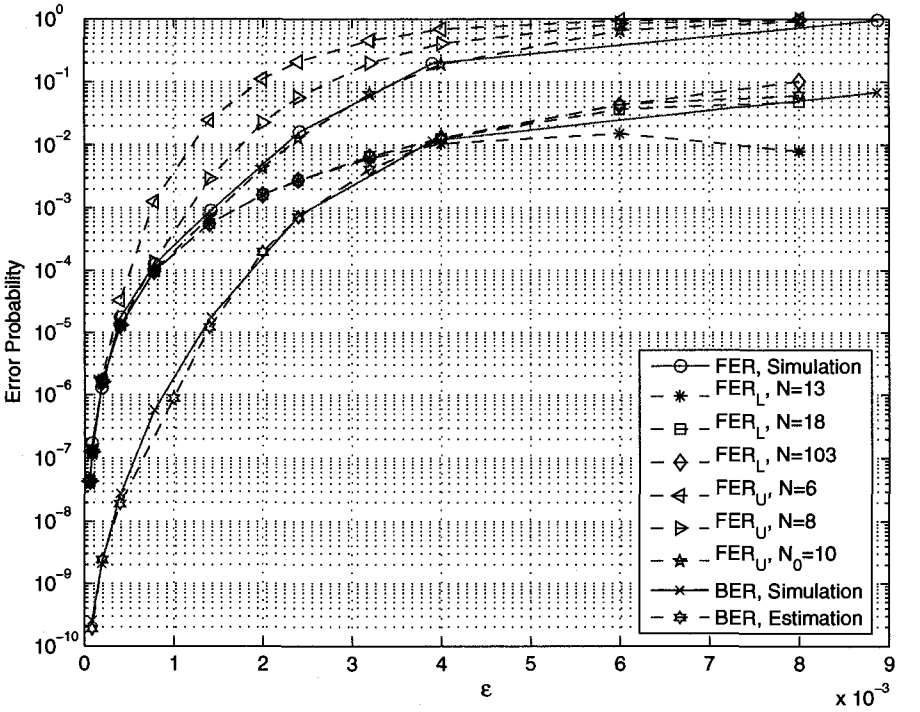

Figure 3.9: FER and BER estimations and simulations of Code 5 decoded by GA.

To compare the computational complexity of our proposed method with that of the conventional Monte Carlo simulations, we consider the following example.

Example 3.6.1 We consider estimating the performance of Code 2 at $p = 10^{-7}$. In this case, $J = 3$, $n = 200$ and we assume $m = 100$ as is the case for the simulation results presented in Fig. 3.6. With these values, we have $\eta = 750$. If we were to add another simulation point at $p = 10^{-8}$, the complexity of Monte Carlo simulations would increase to 8250 times that of our proposed method. For the larger block length of 1008, our proposed method is more efficient than Monte Carlo simulations if the target FER $p < 6 \times 10^{-7}$.

One should note that, as we discussed in Section 3.5, for any given n and J, there exists a FER p', for which our method is more efficient than the Monte Carlo simulations in estimating the FER over the range $p < p'$. □

3.7 Summary

In this Chapter, we studied the trapping sets for hard-decision iterative decoding algorithms. We derived upper and lower bounds on the FER and proposed a method to estimate the FER and BER of LDPC codes decoded by hard-decision iterative decoding algorithms over a BSC. In our approach, we only focused on the smallest weight error patterns that cannot be all corrected by the decoder. By only enumerating these error patterns, the bounds were tight for sufficiently small crossover probabilities of the channel and provide good estimates for the FER in this region. Our estimation method provided estimates for both the FER and the BER for a

given LDPC code over the whole crossover probability region of interest with very good accuracy. The proposed method is universal in that it is applicable in principle to both regular and irregular LDPC codes of arbitrary degree distributions and to any hard-decision iterative algorithm. This universality is partly due to the fact that, unlike previous approaches, our method is not based on identifying the trapping sets and their relationship with the Tanner graph structure of the code and the decoding algorithm, which is shown to be complex in Section 3.2.

Although the complexity of our proposed method is much less than that of the Monte Carlo simulations for many cases of interest, it still increases exponentially with J, essentially as n^J, where n is the block length. It would be interesting to find (to estimate) the number of error patterns of weight J that cannot be corrected by the decoder, using methods other than direct enumeration. In the next chapter, we present an approach to address this problem, making the estimation method proposed in this chapter applicable to almost any practical LDPC code.

Chapter 4

Performance Estimation of LDPC Codes Using Cycle Enumeration

In this chapter, we efficiently approximate J and $|E_J|$ used in the estimation equations in Chapter 3 by enumerating and testing the error patterns that are subsets of short cycles in the code's Tanner graph. This reduces the computational complexity by several orders of magnitude compared to direct enumeration, making it possible to estimate the error rates for almost any practical LDPC code. To obtain the error rate estimates, we propose an algorithm that progressively improves the estimates as larger cycles are enumerated. Through a number of examples, we demonstrate that the proposed method can accurately estimate both the BER and the FER of regular and irregular LDPC codes decoded by a variety of hard-decision iterative decoding algorithms.

4.1 Introduction

In Chapter 3, a method was proposed to provide very accurate estimates of both the FER and the BER of LDPC codes decoded by hard-decision iterative decoding

algorithms over a BSC. The BER and FER of a given LDPC code were related to the size J and the number $|E_J|$ of the smallest error patterns that the decoder fails to correct. The proposed method provided very accurate estimates of both the FER and the BER over the whole range of crossover probabilities of practical interest. To obtain the values of J and $|E_J|$, direct enumeration was used in Chapter 3. The computational complexity of direct enumeration increases exponentially with J, effectively as n^J, where n is the block length of the code. Although this computational complexity can be much less than that of conventional Monte Carlo simulations for many cases of interest as shown in Chapter 3, it can still be very high or even infeasible for large values of n and J. This limited the application of the proposed estimation method to codes with small n and J.

The goal of this chapter is to reduce the computational complexity of the estimation method proposed in Chapter 3, with no sizable degradation in the accuracy of estimates. Based on the observation that the majority of problematic low-weight error patterns are subsets of short cycles in the code's Tanner graph, we obtain approximations of J and $|E_J|$ by enumerating and testing such subsets. We also modify the estimation equations derived in Chapter 3 to include further steps for improving the proposed estimates and for identifying their convergence. Through a number of examples, we show that the proposed method still provides very good estimates of both the FER and the BER. This is while the computational complexity is reduced by several orders of magnitude compared to the direct enumeration, making it possible to estimate the performance of almost any LDPC code of practical interest.

As an example, consider a (4000, 2000) regular LDPC code with variable and

check node degrees, 3 and 6, respectively [58]. To estimate the FER of this code under GA, direct enumeration requires 1.06×10^{10} instances of iterative decoding. This complexity can be easily prohibitive even for the state-of-the-art computers. By using the method proposed in this chapter, the number of iterative decoding instances required to obtain the estimation is reduced to 1197, about 8.9×10^6 times less than that of the enumeration method. This is while the proposed method is still very accurate in estimating the code's performance.

4.2 Error Rate Estimations Using Cycle Enumeration

4.2.1 Problem Formulation and Motivation

From Chapter 3, we know that after determining the values of N_0 and M, the FER an BER can be estimated by the following two estimates, respectively,

$$FER \approx P(J) + \sum_{i=J+1}^{N_0} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i} + \sum_{i=N_0+1}^{n} p_i, \tag{4.2.1}$$

$$\begin{aligned} BER \approx & \frac{J}{n}\left[P(J) + \sum_{i=J+1}^{N_0-1} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i}\right] \\ & + \frac{M}{n}\left[|E_J| \binom{n-J}{N_0-J} \varepsilon^{N_0} (1-\varepsilon)^{n-N_0} + \sum_{i=N_0+1}^{n} p_i\right]. \end{aligned} \tag{4.2.2}$$

From Equations (4.2.1) and (4.2.2), one can see that the values of J and $|E_J|$ are essential for the estimations. If these values are obtained by direct enumeration,

one thus needs to perform $\sum_{i=1}^{J} \binom{n}{i}$ instances of iterative decoding. To reduce the computational complexity, it is natural to search for methods of finding J and $|E_J|$ without enumeration. It is well-known that the existence of problematic low-weight error patterns, such as those in E_J, is closely related to the existence of cycles in the code's Tanner graph. However, the relationships among these error patterns, their trapping sets and the cycle structure of the Tanner graph are in general complex and unknown as we discussed in Chapter 3. Such relationships depend not only on the graph structure but also on the decoding algorithm. Our key observation is that the majority of the low-weight error patterns that cannot be corrected by hard-decision iterative decoders are subsets of short cycles in the Tanner graph. These cycles and their subsets can be enumerated efficiently as their multiplicity is rather small compared to $\sum_{i=1}^{J} \binom{n}{i}$ error patterns that need to be tested in direct enumeration. As an example, for the (4000, 2000) code, described in Section 4.1, the number of cycles of length 6 and 8 are only 171 and 1220, respectively. This corresponds to 1197 and 18300 variable node subsets, respectively. Compared to 1.06×10^{10} candidate error patterns of direct enumeration, these numbers are minuscule.

4.2.2 Proposed Algorithm

Let g be the *girth* of the Tanner graph. Let $C_{l/2}$ denote the set of error patterns of weight $l/2$ each forming a cycle of length l ($l = g, g+2, \cdots$). Our proposed algorithm first enumerates all the error patterns in $C_{g/2}$ and their sub-patterns as initial error patterns to the decoder. These error patterns have weights 1, 2, $\cdots$, $g/2$. If at least

Algorithm 1 Cycle Enumeration

step 1. $l = g$, $\widetilde{J} = +\infty$ and $|\widetilde{E}_i| = 0, (i = 1, 2, ...)$.
step 2. Enumerate the elements of the set $C_{l/2}$.
step 3. Decode all the error sub-patterns of weight 1,2, $\cdots$, $l/2$ of each element in $C_{l/2}$.
step 4. For each weight-i $(i = 1, 2, \cdots, l/2)$ error pattern found in Step 3 which cannot be corrected by the decoder, $\widetilde{J} = min(\widetilde{J}, i)$ and $|\widetilde{E}_i| = |\widetilde{E}_i| + 1$.
step 5. if $\widetilde{J} == +\infty$, $l = l+2$, go to Step 2; else, generate the error rate estimates.
step 6. if the estimates converge, stop; else, $l = l + 2$, go to Step 2.

one of the error patterns is not corrected by the decoder, the algorithm estimates J by the minimum weight of such error patterns, denoted by $\widetilde{J}$. The algorithm also updates the values $|\widetilde{E}_{\widetilde{J}}|, \cdots, |\widetilde{E}_{g/2}|$ by the number of error patterns of weight $\widetilde{J}, \cdots, g/2$, that cannot be corrected by the decoder, respectively. These numbers are used as estimates for $|E_J|, \cdots, |E_{g/2}|$ in modified versions of estimation Equations (4.2.1) and (4.2.2). If all the error patterns in $C_{g/2}$ and their sub-patterns are corrected by the decoder, $\widetilde{J}$ is set to $+\infty$. The algorithm continues by enumerating all the error patterns in $C_{(g+2)/2}$ and their sub-patterns. After testing these error patterns, the value of $\widetilde{J}$, as the size of the smallest error patterns that cannot be all corrected by the decoder, as well as the values $|\widetilde{E}_{\widetilde{J}}|, \cdots, |\widetilde{E}_{(g+2)/2}|$ are updated. If $\widetilde{J} \neq +\infty$, the updated values are used to obtain new estimates for error rates. The algorithm continues by enumerating the next larger set of cycles until $\widetilde{J} \neq +\infty$ and at least one set of estimates for error rates is available. As more sets of cycles are examined the estimates improve. One can in fact stop the algorithm when two successive estimates are close together indicating a convergent behavior.

The algorithm is summarized as Algorithm 1.

4.2.3 Modified Estimation Equations

Consider the case where the algorithm of Section 4.2.2 is at Step 5 and for the first time $\widetilde{J} \neq +\infty$. Also, assume that $|\widetilde{E}_i| = 0$, $\forall i \neq \widetilde{J}$. The error rate estimates are then obtained by replacing J and $|E_J|$ in Equations (4.2.1) and (4.2.2) by $\widetilde{J}$ and $|\widetilde{E}_{\widetilde{J}}|$, respectively. By continuing the algorithm at Step 6 and enumerating the next set of larger cycles, we would come across error patterns of weight $\widetilde{J}+1$ that cannot be corrected by the decoder. At the end of Step 4, we would obtain $|\widetilde{E}_{\widetilde{J}+1}|$ and also an updated value for $|\widetilde{E}_{\widetilde{J}}|$. Assuming that at this stage $|\widetilde{E}_i| = 0$, $\forall i \notin \left\{\widetilde{J}, \widetilde{J}+1\right\}$, we generate new estimates for FER and BER as follows:

$$FER \approx P(\widetilde{J}) + \sum_{i=\widetilde{J}+1}^{N_0} |E_i|_{est}\varepsilon^i(1-\varepsilon)^{n-i} + \sum_{i=N_0+1}^{n} p_i, \tag{4.2.3}$$

$$\begin{aligned} BER \;\approx\; & \frac{\widetilde{J}}{n}\left\{P(\widetilde{J}) + \sum_{i=\widetilde{J}+1}^{N_0-1} |E_i|_{est}\varepsilon^i(1-\varepsilon)^{n-i}\right\} \\ & + \frac{M}{n}\left\{|E_{N_0}|_{est}\varepsilon^{N_0}(1-\varepsilon)^{n-N_0} + \sum_{i=N_0+1}^{n} p_i\right\}. \end{aligned} \tag{4.2.4}$$

In these equations, $|E_i|_{est}$, $\widetilde{J}+1 \leq i \leq N_0$, is the estimate for $|E_i|$, given by

$$|E_i|_{est} = |\widetilde{E}_{\widetilde{J}}|\binom{n-\widetilde{J}}{i-\widetilde{J}} + \left(|\widetilde{E}_{\widetilde{J}+1}| - |E'|\right)\binom{n-\widetilde{J}-1}{i-\widetilde{J}-1}, \tag{4.2.5}$$

where the set E' consists of all the error patterns in $\widetilde{E}_{\widetilde{J}+1}$ which have at least one element of $\widetilde{E}_{\widetilde{J}}$ as their sub-pattern. Since the contributions of the error patterns of

E' in estimating $|E_i|$ $(\widetilde{J}+1 \leq i \leq N_0)$ are already included in the first term of the summation in Equation (4.2.5), we need to subtract them in the second term of the summation to avoid over-counting. Note that the estimate (4.2.5) for $|E_i|$ has replaced the estimate $|E_J|\binom{n-J}{i-J}$ used in Equations (4.2.1) and (4.2.2) for this term. The rationale and justifications for estimate (4.2.5) is similar to those used in Chapter 3 to derive Equations (4.2.1) and (4.2.2), and are not repeated here.

As the algorithm progresses and more information about sets E_i become available, we will have new nonzero values $|\widetilde{E}_i|$ and more refined $|\widetilde{E}_i|$ values for the existing sets. These can be easily incorporated into the estimation equations following the same approach used in deriving Equation (4.2.5) to obtain more accurate estimates for error rates.

4.3 Computational Complexity

Since the value l in each round of the algorithm is a small number, say between g and $g+4$, Step 2 can be performed efficiently. In fact it is easy to see that in a breadth-first algorithm, the complexity of enumerating cycles of length l for a length n LDPC code with variable node degree d_v and check node degree d_c is only $O\left(n(d_v d_c)^{\lceil \frac{l}{4} \rceil}\right)$, where $\lceil x \rceil$ is the smallest integer which is greater than or equal to x. The complexity of Step 2 is thus negligible compared to that of Step 3 which contributes the most to the total computational complexity of the proposed algorithm.

In Step 3, for each element in $C_{l/2}$, we need to perform $\sum_{i=1}^{l/2} \binom{l/2}{i} = 2^{l/2} - 1$ instances of iterative decoding. Suppose that the algorithm stops at $l = L$. In total,

we need to perform $\sum_{i=g/2}^{L/2} |C_i| (2^i - 1)$ instances of iterative decoding. Assuming that the average number of iterations required for decoding are the same, the ratio of the computational complexities of the direct enumeration method of Chapter 3 and the proposed algorithm for a given LDPC code with block length n is:

$$\tau = \frac{\sum_{i=1}^{J} \binom{n}{i}}{\sum_{i=g/2}^{L/2} |C_i| (2^i - 1)}. \tag{4.3.6}$$

As we will see in the next section, for practical codes , $\tau \gg 1$.

It is shown in Chapter 3 that there is always a FER p below which the enumeration method is more efficient than the Monte Carlo simulation in generating the error rates. Similarly, the proposed algorithm is more efficient than the Monte Carlo simulation if the target FER is less than p which is given as:

$$p = \frac{m}{\sum_{i=g/2}^{L/2} |C_i| (2^i - 1)} \tag{4.3.7}$$

where m is the number of codeword errors we would like to observe in the Monte Carlo simulations to have a reliable result. The value of m is often selected in the range of a few tens to a few hundreds. We will show later in the next section that, unlike the method of Chapter 3 where p is usually very small, for the proposed algorithm, p is usually in the high or medium error rate region, which means that the proposed algorithm is more efficient than the Monte Carlo simulation for almost all

the crossover probabilities of practical interest.

4.4 Simulation Results

To demonstrate the generality of the proposed algorithm, we perform experiments on 8 regular and irregular LDPC codes with different rates. The block lengths of these codes are from a few hundreds to a few thousands. The codes are decoded by different hard-decision iterative decoding algorithms. The codes block lengths, dimension, the corresponding decoding algorithms and the values of N_0 and M used in the estimates are listed in Table 4.1. Codes 1, 2, 3 and 7 are Codes 2, 3, 4 and 5 used in Chapter 3, respectively. Code 6 is an irregular code and has the same degree distributions as Code 1, which are optimized for the BSC and GA and are given by $\lambda(x) = 0.1115x^2 + 0.8885x^3$ and $\rho(x) = 0.26x^6 + 0.74x^7$ [75]. Codes 2-5 are taken from [58]. Codes 2, 3 and 5 have variable node and check node degrees (3, 6), (4, 36) and (3, 6), respectively. Code 4 is constructed by the progressive-edge-growth (PEG) algorithm [15] and has a variable node degree 3. Codes 7 and 8 both have variable node degree 5 and check node degree 6. Codes 7 and 8 are decoded by MB algorithm of order zero (MB^0), while all the other codes are decoded by GA. For the degree distributions of Codes 7 and 8, MB^0 has a better threshold than GA algorithm does [38]. The Tanner graphs of the codes do not have cycles of length 4. In all cases, the maximum number of iterations is 100 and Monte Carlo simulations run until 100 codeword errors are observed for each simulated point.

Different parameters of the 8 LDPC codes, as related to the error rate estimates,

Table 4.1: Summary of different LDPC codes.

Code No. :	1	2	3	4	5	6	7	8
$\frac{n}{k}$	$\frac{200}{100}$	$\frac{1008}{504}$	$\frac{1998}{1776}$	$\frac{1008}{504}$	$\frac{4000}{2000}$	$\frac{4000}{2000}$	$\frac{210}{35}$	$\frac{600}{100}$
Dec. Algo.	GA	GA	GA	GA	GA	GA	MB^0	MB^0
N_0	9	38	10	39	156	338	13	34
M	7.73	143.93	133.57	145.62	575.43	97.82	46.95	134.51

are listed in Table 4.2. The values for J and $|E_J|$ are obtained by enumeration. For Codes 4, 5, 6 and 8, the computational complexity of obtaining $|E_J|$ is too high. However, for these codes, we still can find the exact value of J by testing all the error patterns of weight less than J and observing that they can all be corrected by the decoder. Using the proposed algorithm, we are also able to find error patterns of weight J that cannot be corrected by the decoder for these codes. Girth g of the Tanner graph for each code is given in the third row. The number of cycles of length g, $|C_{g/2}|$, for each code is given in the fourth row of the table. The following row contains the number of error patterns in set $C_{g/2}$ that cannot be corrected by the decoder. This is denoted by $|T_{g/2}|$. By comparison of the two rows, it can be seen that while for GA, none of the error patterns in $C_{g/2}$, with the exception of one error pattern for Code 1, can be corrected, for MB^0 applied to Codes 7 and 8, they can all be corrected.

For codes 1-6, the proposed algorithm obtains an estimate for $\widetilde{J}$ in the first round. For codes 7 and 8, an estimate for $\widetilde{J}$ is available at the second round when the algorithm enumerates cycles of length $g+2$. The values of $\widetilde{J}$ are given in the sixth

Table 4.2: Parameters of different LDPC codes related to the estimates .

Code No.:	1	2	3	4	5	6	7	8
J	3	3	3	4	3	3	4	4
$\lvert E_J \rvert$	1434	192	200479	-	-	-	18	-
g	6	6	6	8	6	6	6	6
$\lvert C_{g/2} \rvert$	1293	165	200438	2	171	1312	1424	1353
$\lvert T_{g/2} \rvert$	1292	165	200438	2	171	1312	0	0
$\widetilde{J}$	3	3	3	4	3	3	4	4
$\lvert \widetilde{E}_{\widetilde{J}} \rvert$	1292+119	165+12	200438	2	171+2	1312+7	17	2
$\lvert C_{(g+2)/2} \rvert$	18358	1258	-	11238	1220	18172	20225	20396
$\lvert C_{(g+4)/2} \rvert$	-	-	-	-	-	-	318070	320180
$\lvert \widetilde{E}_{\widetilde{J}+1} \rvert$	18311	1258	-	11238	1220	18172	69	3
$\lvert E' \rvert$	1307	6	-	0	1	65	38	2

row of the table. As can be seen for all the 8 LDPC codes, we have $\widetilde{J} = J$. For codes 1-6, after the first round, the algorithm reaches the estimates 1292, 165, 200438, 2, 171 and 1312, for $|\widetilde{E}_{\widetilde{J}}|$, respectively. These estimates for codes 1, 2, 5 and 6 are improved in the second round of the algorithm, as shown by the added values in the seventh row of the table. In the second round, estimates for $|\widetilde{E}_{\widetilde{J}+1}|$ are also obtained as listed in the tenth row of the table. Note that for Code 3, we have only enumerated cycles of length g. Since this code has a high rate, its Tanner graph is relatively dense and contains many short cycles. The task of enumerating and testing the cycles of larger length is thus tedious. Nevertheless, since $|\widetilde{E}_{\widetilde{J}}|$ for this code is large, the estimates obtained by substituting $|\widetilde{E}_{\widetilde{J}}|$ in Equations (4.2.1) and (4.2.2) are very accurate as shown in Figure 4.4. For Codes 7 and 8, in the second round of the algorithm, estimates 17 and 2 are obtained for $|\widetilde{E}_{\widetilde{J}}|$. In the third round of

the algorithm, these estimates remain unchanged and new estimates of 69 and 3 for $|\widetilde{E}_{\widetilde{J}+1}|$ are obtained for Codes 7 and 8, respectively. For all the codes, the values for $|E'|$ required in Equations (4.2.3) and (4.2.4) are given in the last row of the table.

We provide the following discussions to relate our results to the results of [73]. In [73], to estimate the error rate, one needs to identify and enumerate the trapping set of a given LDPC code under a hard-decision iterative decoder.

Discussion 4.4.1 Consider Code 2. For this code, the 165 cycles of length 6 are all (3, 3) fixed-pattern trapping sets defined in Chapter 3. Out of the 1258 cycles of length 8, six of them are (4, 2) trapping sets with the structure depicted in Figure (4.1) (Fig. 1 (c) of [73]). In the figure, variable nodes and check nodes are represented by empty and full circles, respectively. These trapping sets have *critical number* 3 and *strength* 4 [73]. Out of the 24 failure sets corresponding to these trapping sets, 12 are cycles of length 6 (fixed-pattern trapping sets) and the other 12 are the so-called oscillatory-pattern trapping sets defined in the previous chapter. The latter are the 12 additional error patterns contributing to $|\widetilde{E}_{\widetilde{J}}|$ in the table. Our investigation shows that the remaining $192 - (165 + 12) = 15$ error patterns in E_J are all failure sets of the (5, 3) trapping sets whose structure is shown in Figure (4.2) (Fig. 1 (b) of [73]). The (5, 3) trapping set is oscillatory, where the error locations oscillate between the failure set and the set of the other two variables in the trapping set.

Discussion 4.4.2 The 12 extra error patterns of weight 3 in $\widetilde{E}_{\widetilde{J}}$ that were discovered for Code 2 by examining cycles of length 8, have all the same graphical structure as shown in Figure 4.1. Unlike those patterns, the 119 extra error patterns in $\widetilde{E}_{\widetilde{J}}$ for Code

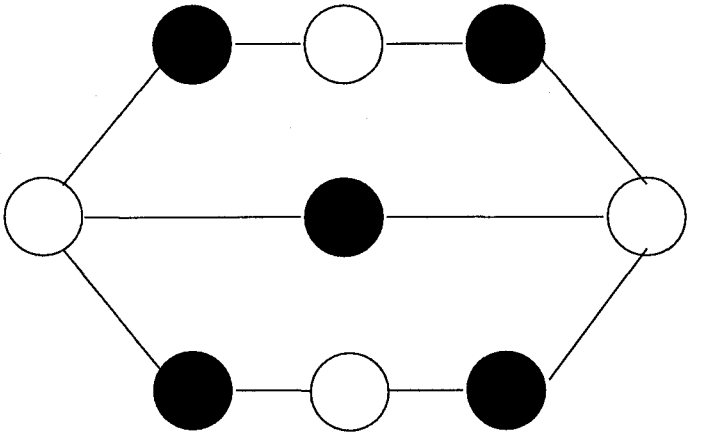

Figure 4.1: Tanner graph of a (4, 2) trapping set.

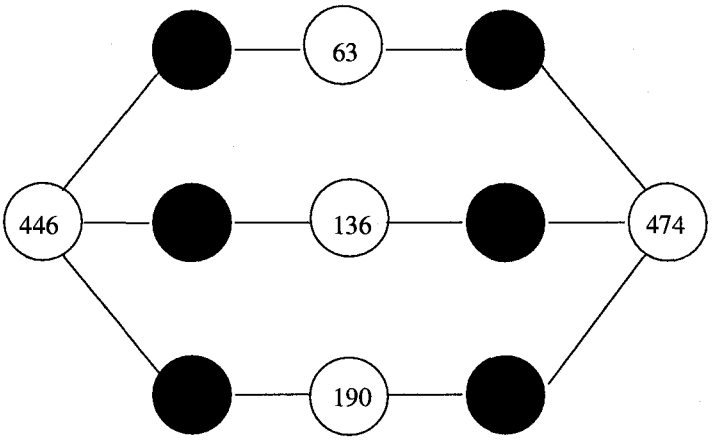

Figure 4.2: Tanner graph of a (5, 3) trapping set.

1 have a wide variety of graphical structures and correspond to both oscillatory- and fixed-pattern trapping sets. The example of Code 1 demonstrates that the approach of identifying and enumerating dominant trapping sets based on the graphical structure of these sets, as adopted, e.g., in [73], may not be very effective for estimating E_J for general random codes, particularly the irregular ones.

For Codes 1-3 and 5-8, the initial estimates of error rates obtained by substituting $\widetilde{J}$ and the first estimate of $|\widetilde{E}_{\widetilde{J}}|$ in Equations (4.2.1) and (4.2.2) are accurate in

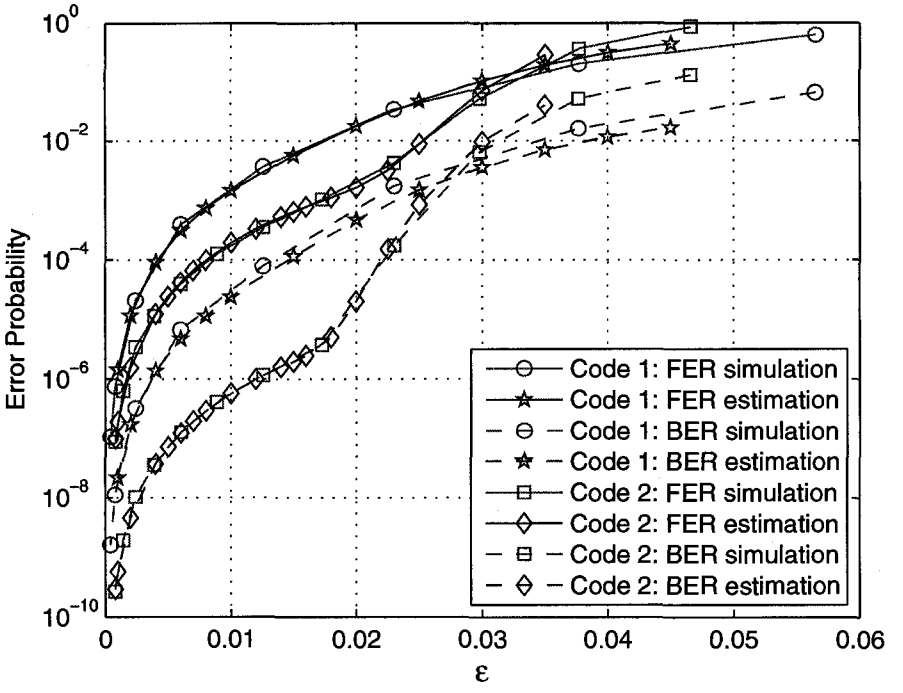

Figure 4.3: FER and BER estimations and simulations of Codes 1 and 2.

the whole range of crossover probabilities of interest. These estimates are shown in Figures 4.3, 4.4, 4.6 and 4.7 along with Monte Carlo simulation results. The values of N_0 and M used in the estimates for each code are given in Table 4.1. For Code 4, however, as shown in Figure 4.5, although the initial estimates are accurate for small and large values of crossover probability, they are not so accurate in between. The improved estimates obtained at the second round of the algorithm however are accurate over the whole range.

Table 4.3 shows the values of τ, defined in Equation (4.3.6), for the 8 codes and for the estimates given in Figures 4.3-4.7 (for Code 4, τ corresponds to the improved estimates). As can be seen the proposed method is less complex than direct enumeration by several orders of magnitude. The reduction in computational

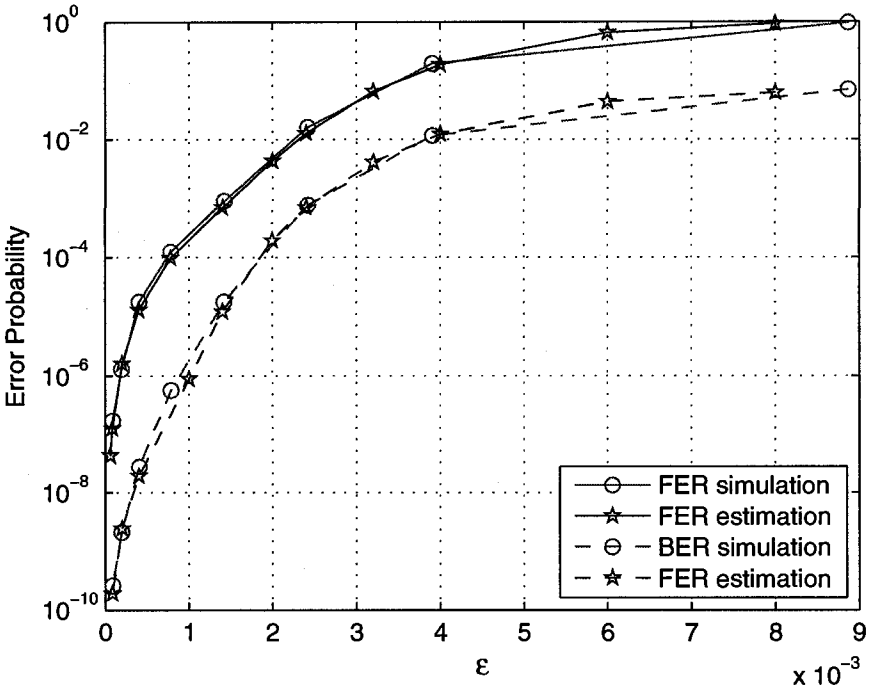

Figure 4.4: FER and BER estimations and simulations of Code 3.

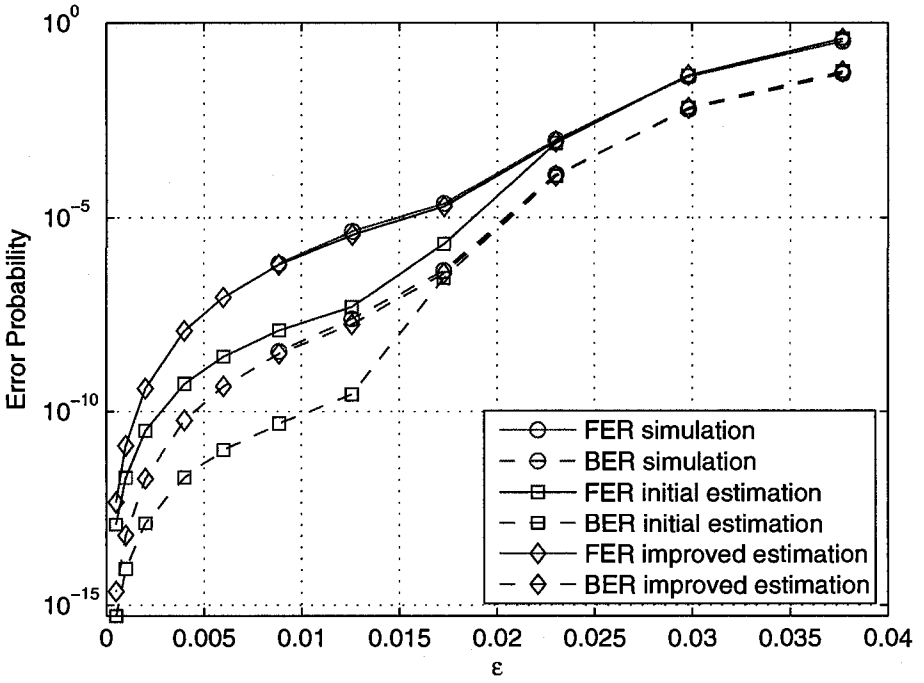

Figure 4.5: FER and BER estimations and simulations of Code 4.

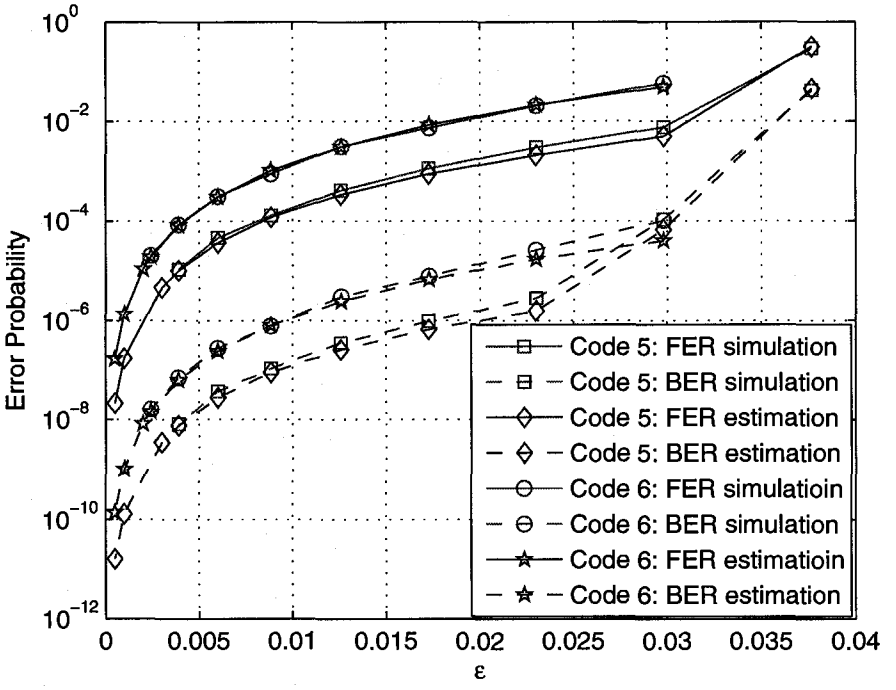

Figure 4.6: FER and BER estimations and simulations of Codes 5 and 6.

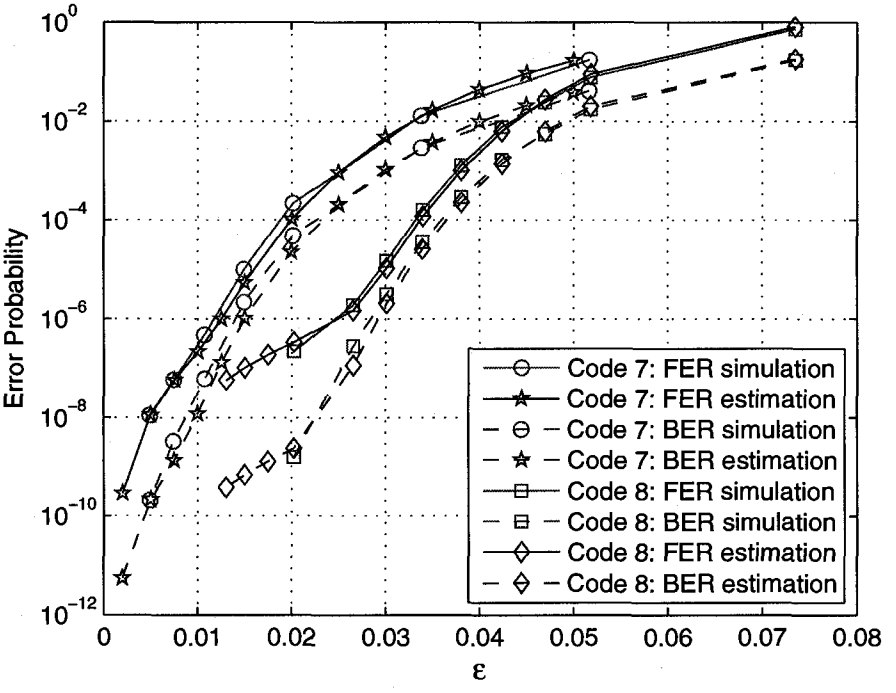

Figure 4.7: FER and BER estimations and simulations of Codes 7 and 8.

Table 4.3: Computational Complexity Comparison Between the Proposed Method and Direct Enumeration.

Code:	1	2	3	4	5	6	7	8
τ	147.33	1.5×10^5	947.45	1.2×10^5	8.9×10^6	1.1×10^5	253.9	1.7×10^4

Table 4.4: Values of p Defined in Equation (4.3.7) for Different Codes.

Code:	1	2	3	4	5	6	7	8
p	1.1×10^{-2}	8.7×10^{-2}	7.1×10^{-5}	2.9×10^{-4}	8.4×10^{-2}	1.1×10^{-2}	3.2×10^{-4}	3.1×10^{-4}

complexity is particularly significant for the cases where the block length and/or the value of J is large. In such cases, direct enumeration can easily be too complex to apply. In all cases, the substantial reduction in complexity comes at practically no cost in terms of the accuracy of the estimates. The values of p defined in Equation (4.3.7) are given in Table 4.4. As can be seen that in many cases the proposed method is more efficient than the Monte Carlo simulation for almost all ε region of practical interest and the increase in efficiency becomes more significant as the target error rate becomes smaller.

Among many LDPC codes that we have examined, we have been able to find only one code for which the estimate $\widetilde{J}$ obtained by the algorithm in the first two rounds is not equal to the true value of J. This is explained in the following example.

Example 4.4.1 Consider a PEG-constructed (504, 252) code with variable degree 3 [58] referred as Code 9. The code's Tanner graph has girth 8 and by applying the proposed algorithm for GA, we obtain $\widetilde{J} = 4$, $|\widetilde{E}_{\widetilde{J}}| = 802$ and $|\widetilde{E}_{\widetilde{J}+1}| = 11279$. Using direct enumeration, however, we find that $J = 3$ and $|E_J| = 14$. By further

examination of the 14 error patterns of weight 3, we discover that they all have the same graphical structure and correspond to the failure set of the (5, 3) trapping set shown in Figure (4.2). If the initial error pattern is the variable node set {63, 136, 190}, then the variable node set {63, 136, 190, 446, 474} forms a so-called oscillatory-pattern trapping set defined in Chapter 3 and the error locations oscillate between the sets {63, 136, 190} and {446, 474}. The set {63, 136, 190} is not part of any cycle of length 8 or 10. That is why our algorithm is not capable of detecting it in the first two rounds. By allowing the algorithm to continue enumerating larger cycles, we are able to detect this structure as the set {63, 136, 190} is part of a cycle of length 14. This is also the case for all the other 13 error patterns.

For this code, we have obtained the error rate estimates by substituting $\widetilde{J} = 4$ and $|\widetilde{E}_{\widetilde{J}}| = 802$ in Equations (4.2.1) and (4.2.2). These estimates are given in Figure 4.8 ($N_0 = 19$ and $M = 72.42$). Improved estimates based on Equations (4.2.3) and (4.2.4) are almost identical to those of Figure 4.8. Comparison with simulation results show that the estimates are still very accurate at higher values of crossover probability. It is only for the lower values that the difference between the estimates and simulations is non-negligible. In Fig. 7, we have also given the error rate estimates obtained by using $J = 3$ and $|E_J| = 14$ in Equations (4.2.1) and (4.2.2). These estimates are accurate over the whole range of crossover probabilities. □

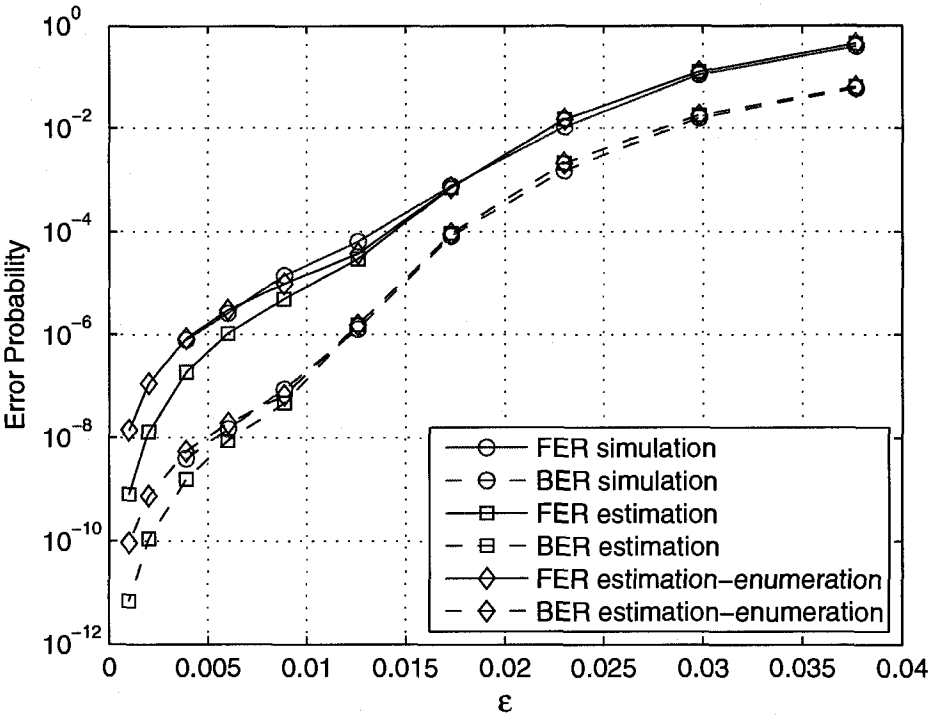

Figure 4.8: FER and BER estimations and simulations of Code 9.

4.5 Summary

In this chapter, we have proposed an algorithm that can efficiently approximate J and $|E_J|$, the size and the number of the smallest error patterns that a hard-decision iterative decoder for LDPC codes over a BSC fails to correct. These approximations are used to estimate the error rate performance of LDPC codes over the whole range of channel crossover probabilities of interest. The proposed algorithm examines the subsets of short cycles in the code's Tanner graph as candidate error patterns that can trap the decoder in a trapping set, and is proposed as an alternative for Monte Carlo simulations or direct enumeration. We show, through a number of examples involving regular and irregular LDPC codes with different rates decoded by different hard-decision algorithms, that while the estimates provided by the proposed algorithm are

practically as accurate as those obtained by direct enumeration, their computational complexity is substantially lower. Using the proposed algorithm, we can generate FER and BER curves for a given LDPC code and a given hard-decision iterative decoding algorithm in a matter of minutes instead of days or even weeks which would take by Monte Carlo simulations or direct enumeration. The complexity advantage is particularly significant for larger block lengths and lower code rates.

While for most codes, the proposed algorithm converges very fast, in some cases, such as Example 4.4.1, to obtain an accurate estimate of error rate over the whole range of crossover probabilities of interest, a large number of cycle sets may need to be examined. In such cases, it would be beneficial to search for dominant trapping sets based on the graphical structure of such sets instead of blindly searching the subsets of individual cycles. To develop an efficient systematic approach for such a search on a general graph would be an interesting topic for future research.

Chapter 5

Performance Estimation of LDPC Codes Decoded by Soft-Decision Iterative Algorithms

In this chapter, we describe a combinatorial approach to estimate the error rate performance of LDPC codes decoded by (quantized) soft-decision iterative decoding algorithms. This approach can be seen as an extension of the techniques proposed in Chapters 3 and 4. The extension is by no means trivial and involves a number of important changes to the machinery, the estimation equation and the search algorithm. The method is based on efficient enumeration of input vectors with small distances to a reference vector whose elements are selected to be the most reliable values from the input alphabet. Several techniques, including modified cycle enumeration, are employed to reduce the complexity of the enumeration. The error rate estimate is derived by testing the input vectors of small distances followed by estimating the con-

tribution of larger distance vectors. We demonstrate by a number of examples that the proposed method provides accurate estimates of error rate with computational complexity much lower than that of Monte Carlo simulations, especially at the error floor region.

5.1 Introduction

In Chapter 3, we proposed a method to estimate the performance of LDPC codes decoded by hard-decision iterative decoding algorithms over a binary symmetric channel (BSC). By using combinatorial arguments, we related the FER and the BER of a given LDPC code to the size J and the number $|E_J|$ of the smallest error patterns that the decoder fails to correct. This method provides very accurate estimates of both the FER and the BER over the whole range of crossover probabilities of practical interest. To reduce the computational complexity of direct enumeration used in Chapter 3 for finding the values of J and $|E_J|$, we proposed cycle enumeration algorithm in Chapter 4 which reduces the computational complexity by several orders of magnitude and makes the estimation method applicable to almost any practical LDPC code.

Following a similar approach of Chapters 3 and 4, in this chapter, we extend the idea to the performance estimation of (quantized) soft-decision iterative decoding algorithms. Soft-decision iterative decoding algorithms, such as BP and MS, are of great interests in practice due to their good error performance. Several works [14,51] have been done to estimate the performance of a given LDPC code under soft-decision iterative decoding algorithms based on identifying and enumerating dominant trapping sets in the TG. Richardson [14] proposed a semi-analytical technique which

predicts the performance of a given code in the error floor region on the AWGN channel. Cole et al. [51] devised a three-step algorithm for estimating the FER floors of soft-decision LDPC decoders on the AWGN channel using importance sampling. Similar to the case of hard-decision algorithms, the structure of trapping sets for different codes and different soft-decision iterative decoding algorithms is in general complex and unknown. Thus, it is difficult to identify and enumerate them. Heuristics are used to identify and enumerate them [14, 51]. In another direction and inspired by statistical physics, the authors of [76–79] estimated the FER of min-sum (MS) decoding for a given code using the concept of "instanton", which is the most probable configuration of the noise to cause a decoding error.

The decoders of interest in this chapter are practical implementations of different soft-decision iterative decoding algorithms. In practice, iterative algorithms are implemented in (quantized) fixed-point arithmetic. There are different ways for implementing quantized soft-decision iterative decoding algorithms (e.g., [80]- [85]). It has been shown (see, e.g. [80,81]) that usually quantized soft decoders with 4 or 5 bits can perform very closely to their corresponding infinite precision implementations. It is well-known that the error floor behavior of coding systems depends strongly on the quantization scheme [14], [83]. It is therefore of great practical interest to efficiently estimate the performance of different quantization schemes in the error floor region. Also, from the operational point of view, our combinatorial approach is based on enumerating the input vectors to the decoder and quantization is required to transform a continuous alphabet space to a finite discrete one.

In the proposed method, the input vector set of a q-bit decoder is partitioned based

on each vector's Euclidean distance to the most reliable vector. Contributions of input vectors to the error rate are estimated based on the information of vectors with small distances that cannot be corrected by the decoder. The modified cycle enumeration algorithm is introduced to efficiently search for input vectors of small distance which make the decoder fail. To further reduce the computational complexity, we also introduce an algorithm which generates the estimate for a decoder with a larger number of quantization bits based on the estimation results for smaller number of quantization bits. Unlike the approaches of [14, 51], our method does not require the identification of trapping sets and is thus much simpler. Our approach is not only applicable to the error floor region, but also provides good estimates in the waterfall region. The computational complexity of our technique can be much less than that of the Monte Carlo simulation, especially at the error floor region.

The rest of this chapter is organized as follows. In the next section, we propose our estimation method for quantized soft decision decoders and discuss the similarities and differences with the method of Chapter 3. In the same section, we also introduce two algorithms to reduce the computational complexity and discuss the efficient calculation of probabilities of different sets required for the estimation. Experimental results are given in Section 5.3. Section 5.4 concludes the chapter.

5.2 Error Rate Estimation of LDPC Codes Decoded by Soft-Decision Iterative Algorithms

5.2.1 Error Rate Estimation

In this chapter, we assume the decoder is implemented in the log-likelihood ratio (LLR) domain. The proposed method can be, in principle, applied to implementations in other domains where there is only one message per bit, e.g., likelihood-ratio (LR) domain. The decoders considered here are symmetric [12] so that we assume that the all-zero codeword is transmitted. The channel model is the binary input AWGN channel with 0 and 1 mapped to +1 and -1 , respectively.

In general, for a q-bit quantized iterative decoder, the received values are first clipped symmetrically at a threshold c_{th} and then uniformly quantized into $2^q - 1$ quantization intervals within the range $[-c_{th}, c_{th}]$. Each interval is represented by q quantization bits. Integer numbers $\{-(2^{q-1} - 1), \cdots, 2^{q-1} - 1\}$ are assigned to the intervals. Thus, the input alphabet of the decoder is given by: $\mathcal{A}_q = \{-(2^{q-1} - 1), \cdots, 2^{q-1} - 1\}$. Operations in variable and check nodes are performed on integers. The outgoing messages passed along each edge of the TG are clipped to $-(2^{q-1} - 1)$ or $2^{q-1} - 1$, for negative and positive values, respectively. Iterative decoding continues until it converges to a codeword or the maximum number of iterations I_{max} is reached.

If the block length of the code is n, we use $\mathcal{V} = \mathcal{A}_q^n$ to denote the set of all possible input vectors of the decoder. For each input vector $\vec{r} = (r_1, \cdots, r_n) \in \mathcal{V}$, there is a corresponding probability $P(\vec{r})$ that $\vec{r}$ appears at the input of the decoder. Due

to the memoryless nature of the channel, the elements of the input vector to the decoder are independent and identically distributed (i.i.d) discrete random variables with probability mass function (pmf) $p(r)$. Thus, $P(\vec{r}) = \prod_{i=1}^{n} p(r_i)$. The pmf $p(r)$ depends on the signal-to-noise (SNR) and the quantization scheme. For example, for the quantization scheme described above, if the variance of the noise is σ^2 and if the integer $r \in \mathcal{A}_q$ corresponds to the interval $[x_1, x_2]$, then we have:

$$p(r) = Q(\frac{1 - x_2}{\sigma}) - Q(\frac{1 - x_1}{\sigma}), \tag{5.2.1}$$

where the Q function is defined by: $Q(z) \triangleq \frac{1}{\sqrt{2\pi}} \int_z^{\infty} e^{-x^2/2} dx$.

As described in Chapter 3, for hard-decision iterative decoding algorithms, the set of error patterns (input vector set) is partitioned into different sets according to the Hamming weight of the vectors, or equivalently, the Hamming distance between the vectors and the all-zero vector. In this work, for a q-bit ($q \geq 2$) iterative decoder, we use Euclidean distance instead of Hamming distance. For vectors $\vec{r}_1$, $\vec{r}_2 \in \mathcal{V}$, the Euclidean distance between them is defined by $d_E(\vec{r}_1, \vec{r}_2) = [\sum_{i=1}^{n} (r_{1i} - r_{2i})^2]^{1/2}$. Let $\vec{R} = (2^{q-1} - 1, \cdots, 2^{q-1} - 1)$. Since we assume that the all-zero codeword is transmitted, $\vec{R}$ is the most reliable input vector and intuitively, the larger the Euclidean distance between an input $\vec{r}$ and $\vec{R}$, the higher the probability that $\vec{r}$ cannot be corrected by the decoder.

Let S_d denote the set which contains all input vectors whose Euclidean distance to $\vec{R}$ is d, i.e., $S_d = \{\vec{r} \in \mathcal{V} : d_E(\vec{r}, \vec{R}) = d\}$, and E_d be the subset of S_d whose elements cannot be correctly decoded by the decoder. Since the set of possible values for d is

discrete and finite, the FER can be expressed as:

$$FER = \sum_{d=d_m}^{d_M} P(E_d) = \sum_{d=d_m}^{d_M} \frac{P(E_d)}{P(S_d)} P(S_d), \tag{5.2.2}$$

where d_m is the minimum Euclidean distance such that $|E_d| \neq 0$ (i.e., $|E_d| = 0$, $\forall d < d_m$, and $E_{d_m} \neq \emptyset$), and $d_M = (2^q - 2)\sqrt{n}$ is the maximum Euclidean distance between an input vector and $\vec{R}$. $P(S_d)$ is the probability of having an input vector with distance d to $\vec{R}$ and $\frac{P(E_d)}{P(S_d)}$ is the conditional probability that decoding fails given that the input vector has Euclidean distance d to $\vec{R}$. Following a similar approach to that of Chapter 3, we first obtain the set E_{d_m} as accurately as possible. We then estimate the contributions of vectors of distance larger than d_m to the total FER based on the information of E_{d_m}.

To achieve this, we first define a partial ordering of the input vectors.

Definition 5.2.1 An input vector $\vec{r}$ is *below* another vector $\vec{t}$ (denoted by $\vec{r} \leq \vec{t}$), iff $r_i \leq t_i$, $1 \leq i \leq n$.

From the definition, it's clear that if $\vec{r} \leq \vec{t}$, then $d_E(\vec{r}, \vec{R}) \geq d_E(\vec{t}, \vec{R})$. The following assumption is valid for a maximum likelihood (ML) decoder and our experiments show that it is also valid with high probability for iterative decoders under consideration.

Assumption (1): If an input vector $\vec{r}$ cannot be corrected by the decoder, then none of the vectors below $\vec{r}$ can be corrected either.

The following example shows that Assumption (1) is statistically viable.

Example 5.2.1 Consider the (7, 4) Hamming code decoded by a 2-bit MS decoder

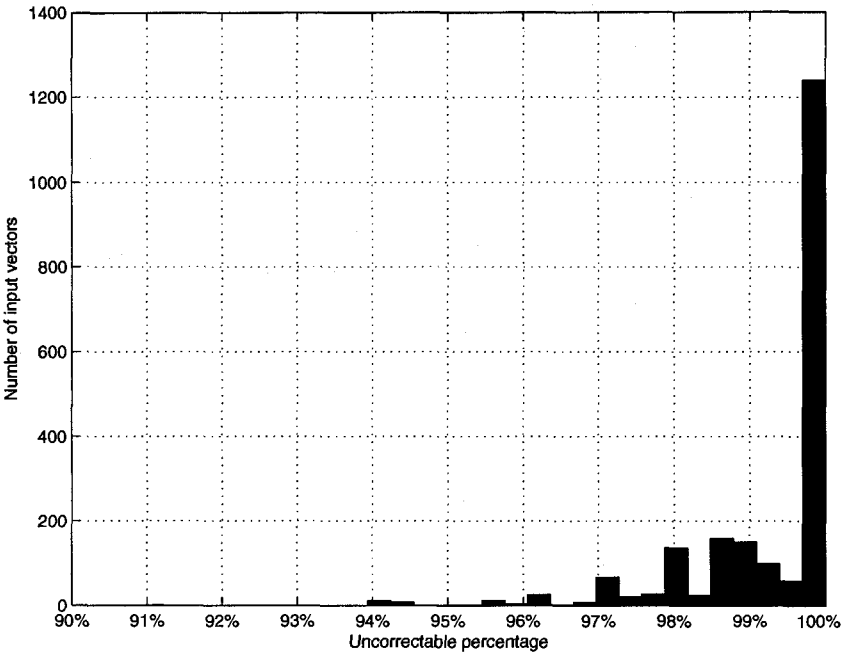

Figure 5.1: Histogram of Example 1.

($c_{th} = 2$, and $I_{max} = 30$). In this case, we have $|\mathcal{V}| = 3^7 = 2187$ input vectors. Out of this number, 2056 cannot be correctly decoded. For each of these 2056 input vectors, we decode all the vectors which are below the given vector and record the percentage of uncorrectable vectors. Figure 5.1 is the histogram of these results, where the horizontal axis is the percentage and the vertical axis is the number of vectors out of the 2056 vectors with this percentage. For example, Figure 5.1 shows that for about 1250 out of the 2056 vectors, all the vectors below them cannot be corrected. Only a few vectors have percentages around 91, and there is no vector with the percentage below 91. □

For each $\vec{r} \in \mathcal{V}$, we define $\underline{\mathcal{V}}(\vec{r}) = \{\vec{t} \in \mathcal{V} : \vec{t} \leq \vec{r}\}$, i.e., $\underline{\mathcal{V}}(\vec{r})$ contains all the vectors in $\mathcal{V}$ which are below vector $\vec{r}$. Assuming set E_{d_m} is known, we would like to use the

information of E_{d_m} to estimate sets E_d, $d > d_m$, so that we can use Equation (5.2.2) to obtain the FER.

Note that, from Definition 5.2.1, it is clear that no two distinct vectors $\vec{r}_1$ and $\vec{r}_2$ of E_{d_m} satisfy $\vec{r}_1 \leq \vec{r}_2$. Let D be the set of distances defined by $D = \{d : d_m \leq d \leq d_M\}$. For $d \in D \setminus d_m$, there are always vectors in the set S_d which are below a vector in E_{d_m}. We partition S_d as $S'_d \bigcup \overline{S'_d}$, where S'_d contains vectors in S_d which are below some vectors in E_{d_m} and $\overline{S'_d}$ is the complementary set of S'_d in S_d. By applying Assumption (1) to set S'_d, we have :

Assumption (2): All vectors in the set S'_d, $d \in D \setminus d_m$ cannot be corrected by the decoder.

We also make the following assumption about the set $\overline{S'_d}$.

Assumption (3): All vectors in the set $\overline{S'_d}$, $d \in D \setminus d_m$ can be corrected by the decoder.

Although the above assumptions are not always correct, our results show that they are statistically viable.

Assumptions (2) and (3) imply that $E_d = S'_d$, $d \in D \setminus d_m$. Thus, the FER is estimated based on Equation (5.2.2) by:

$$FER \approx \sum_{d \in D} P(S'_d) = P(\bigcup_{d \in D} S'_d) = P(\underline{U}(E_{d_m})), \tag{5.2.3}$$

where $\underline{U}(S) \triangleq \bigcup_{\vec{r} \in S} \underline{\mathcal{V}}(\vec{r})$, for a set S.

The following example describes the application of estimate (5.2.3) to the (7,4) Hamming code.

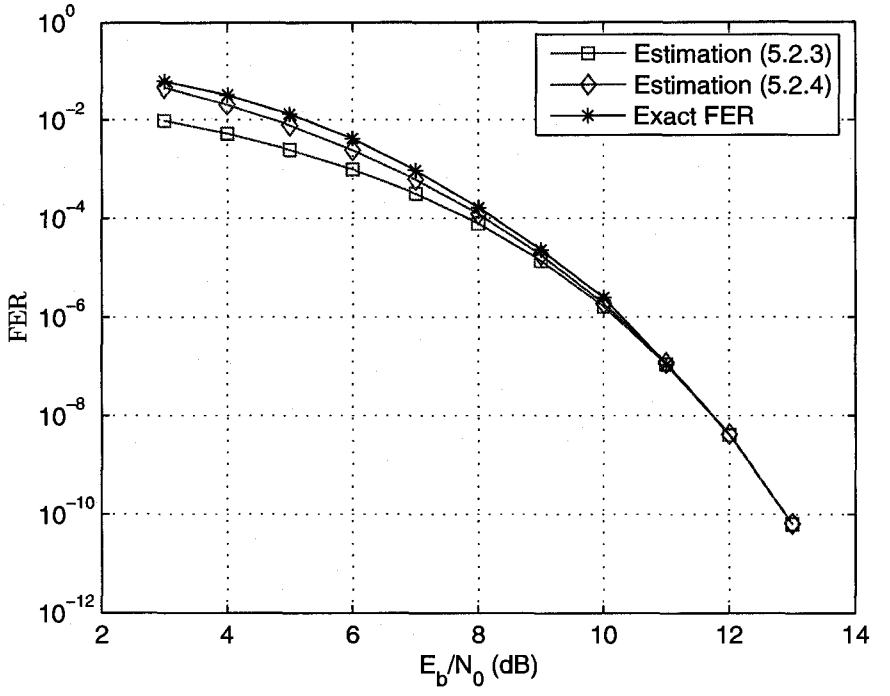

Figure 5.2: Exact FER and estimation results of the (7,4) Hamming code decoded by a 3-bit MS decoder.

Example 5.2.2 Consider the (7, 4) Hamming code decoded by the 3-bit MS decoder with $c_{th} = 2$ and $I_{max} = 30$. In this case, there are $|\mathcal{V}| = 7^7 = 823543$ input vectors. By enumerating the vectors in the set $\mathcal{V}$ and passing them to the decoder, we find out that there are 783606 vectors which cannot be correctly decoded. Investigating those vectors, we find d_m to be $\sqrt{24}$ and $|E_{\sqrt{24}}| = 3$. There are 282975 vectors in the set $\underline{U}(E_{\sqrt{24}})$, among which 279140 (98.6%) cannot be correctly decoded. The estimate of FER based on Equation (5.2.3) is given in Figure 5.2 (the curve with square symbols). In the figure, we have also shown the exact FER obtained by the sum of the probabilities of the 783606 incorrectly decoded input vectors. It can be seen that (5.2.3) provides a good estimation for the FER at the high SNR region. □

In Example 5.2.2, the probability $P(\underline{U}(E_{d_m}))$ in Equation (5.2.3) is calculated by first enumerating all the vectors in the set $\underline{U}(E_{\sqrt{24}})$ and then adding their probabilities. This approach easily becomes computationally infeasible as the block length and the number of quantization bits increase. In fact, $|\underline{U}(E_{d_m})|$ increases essentially proportional to 2^{qn}. Therefore, we need to find efficient ways to calculate this probability. For instance, by assuming that there are no overlaps between sets $\underline{\mathcal{V}}(\vec{r})$, $\vec{r} \in E_{d_m}$, we can calculate $P(\underline{U}(E_{d_m}))$ as $\sum_{\vec{r} \in E_{d_m}} P(\underline{\mathcal{V}}(\vec{r}))$, where $P(\underline{\mathcal{V}}(\vec{r}))$ can be calculated efficiently as shown later in Subsection 5.2.5. However, there does exist overlaps between different sets $\underline{\mathcal{V}}(\vec{r})$, $\vec{r} \in E_{d_m}$, and we cannot ignore the over-counting effects in some cases. Although it is not practically possible to remove all the overlaps between sets $\underline{\mathcal{V}}(\vec{r})$, later in the chapter, we will discuss how we can reduce the effect of overlaps to the FER estimate such that the accuracy of the estimate is not affected in any significant way.

As Example 5.2.2 demonstrated, Equation (5.2.3) usually only provides a good estimate at the high SNR region. To improve the estimate in the low SNR region, similar to the approach taken to derive Equation (3.4.7), we partition the input vectors according to their distances to $\vec{R}$. In general, if an input vector has a large enough distance to $\vec{R}$, it cannot be corrected with high probability regardless of whether it belongs to any set $\underline{\mathcal{V}}(\vec{r})$, $\vec{r} \in E_{d_m}$, or not. The following example illustrates this point for the (7,4) Hamming code.

Example 5.2.3 When the (7, 4) Hamming code is decoded by the 3-bit MS decoder ($c_{th} = 2$ and $I_{max} = 30$), the input vectors in the set $\mathcal{V}$ have distances from 0 to $\sqrt{252}$

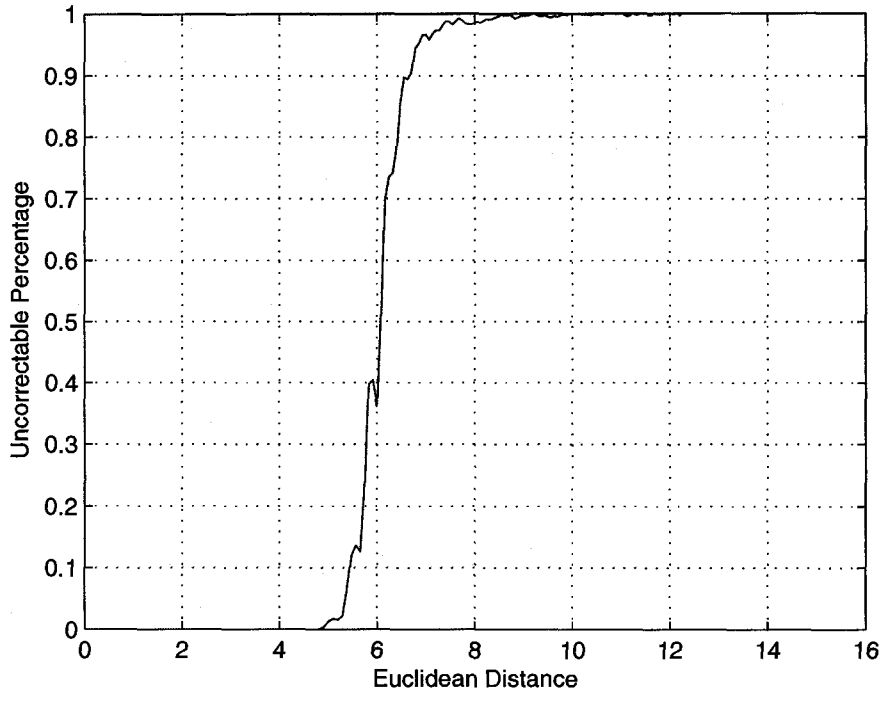

Figure 5.3: Uncorrectable percentage of the input vectors to a 3-bit MS decoder versus their distance to $\vec{R}$ for the (7,4) Hamming code.

to the vector $\vec{R}$. For each distance d, we decode all the vectors in the set S_d and obtain the set E_d by recording the uncorrectable input vectors. The uncorrectable percentage for each distance d is then calculated by $|E_d|/|S_d|$, which is shown in Figure 5.3 for each distance. It is clear that the uncorrectable percentage increases as distance becomes larger and for distances larger than about 7, the percentages are close to 100%. Figure 5.4 shows the ratio $P(E_d)/P(S_d)$ for two SNR values (4 and 8 dB). It can be seen that this ratio tends to one for distances larger than about 7. □

We thus assume that there exists a certain threshold d_{th} on d such that all input vectors with Euclidean distance larger than d_{th} to $\vec{R}$ cannot be correctly decoded. In fact, d_{th} plays a similar role as N_0 does in Equation (3.4.7). Let $S_{d>d_{th}} \triangleq \bigcup_{d>d_{th}} S_d$, the

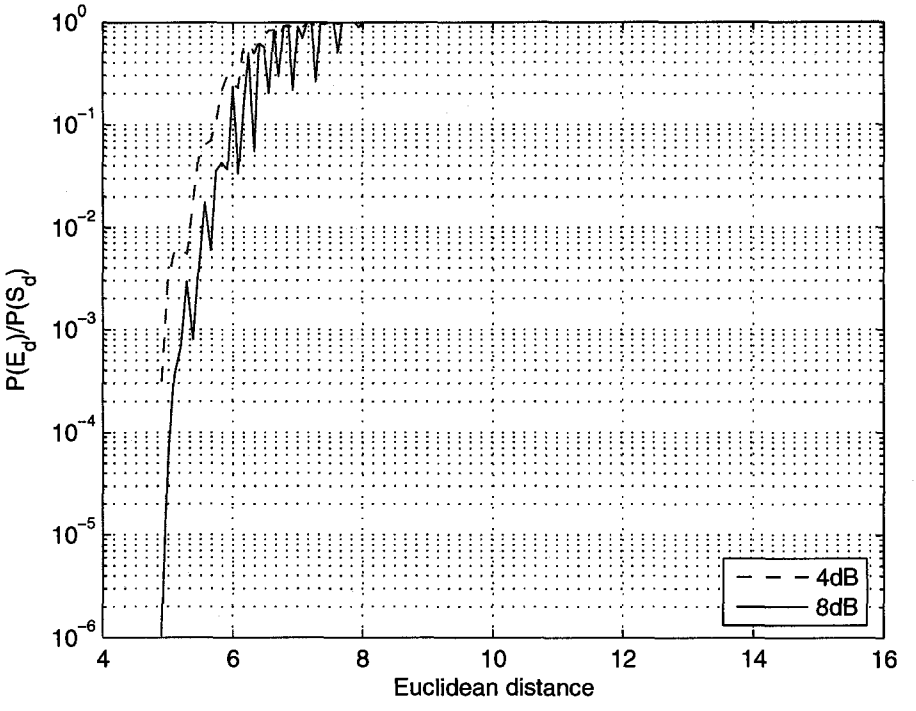

Figure 5.4: $P(E_d)/P(S_d)$ of the input vectors to a 3-bit MS decoder of the (7, 4) Hamming code versus their distances to $\vec{R}$ for SNR values 4dB and 8dB.

FER estimate can then be modified as:

$$FER \approx P(\underline{U}(E_{d_m})) + P(S_{d>d_{th}}). \tag{5.2.4}$$

Note that, there are overlaps between the two terms in (5.2.4), i.e., some vectors in the set $S_{d>d_{th}}$ are also in the set $\underline{U}(E_{d_m})$ and the contributions to the FER from those vectors are over-counted. Our experimental results show however that the probability of overlapped vectors is negligible. Similar to the approach taken in Chapter 3 for the determination of N_0 in Equation (3.4.7), the value of d_{th} can be determined by performing Monte Carlo simulations at the high FER region.

Example 5.2.4 The estimate based on Equation (5.2.4) is given in Figure 5.2 (the curve with diamond symbol) for the (7,4) Hamming code decoded by the 3-bit MS ($c_{th} = 2$ and $I_{max} = 30$). The parameter d_{th} in this case is found to be $\sqrt{45}$. It can be seen in Figure 5.3 (5.4) that the uncorrectable percentages ($P(E_d)/P(S_d)$) for distances larger than d_{th} are high and close to 1. Compared to the estimate based on Equation (5.2.3), which is only accurate at high SNR values, (5.2.4) provides an accurate estimate for the whole SNR region of interest. □

5.2.2 Relationship with the Estimation Method of Chapter 3

In this subsection, we discuss similarities and differences between the estimation technique presented in Chapter 3 and the one proposed in the previous subsection. We first show that estimate (3.4.7) is actually the hard-decision parallel of estimate (5.2.3), where the alphabet size is two and the Euclidean distance is replaced by the Hamming distance.

Fact 5.2.1 In the case of hard-decision algorithms, estimate (5.2.3) is identical to estimate (3.4.7) where N_0 is replaced by the code's block length n.

Proof: For $N_0 = n$, based on Equation (3.4.7), we have:

$$FER \approx P(J) + \sum_{i=J+1}^{n} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i} = \sum_{i=J}^{n} |E_J| \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i}. \tag{5.2.5}$$

In this case, J and E_J play the same roles as d_m and E_{d_m} do in (5.2.3) , respectively, and the all-zero vector acts as $\vec{R}$. For each $\vec{r} \in E_J$, the number of error vectors of

Hamming distance i, $J \leq i \leq n$, to the all-zero vector and below $\vec{r}$ is $\binom{n-J}{i-J}$ (note that the all-zero vector corresponds to the all-one vector at the input to the decoder). This means that there are $\binom{n-J}{i-J}$ vectors of Hamming distance i in the set $\underline{\mathcal{V}}(\vec{r})$ and in this case, they all have the same probability $\varepsilon^i(1-\varepsilon)^{n-i}$. Thus,

$$P(\underline{\mathcal{V}}(\vec{r})) = \sum_{i=J}^{n} \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i}, \vec{r} \in E_J.$$

Using Equation (5.2.3) and assuming that there is no overlap between the sets $\underline{\mathcal{V}}(\vec{r})$, $\vec{r} \in E_J$, we can estimate the FER by:

$$FER \approx \sum_{\vec{r} \in E_J} P(\underline{\mathcal{V}}(\vec{r})) = |E_J| \sum_{i=J}^{n} \binom{n-J}{i-J} \varepsilon^i (1-\varepsilon)^{n-i},$$

which is the same as (5.2.5). ■

Despite the similarity between (3.4.7) and (5.2.3) as stated in Fact 1, there are a few differences between the two cases. One difference, as stated in Section 5.2.1, is the fact that in Equation (5.2.3) the overlap between the sets $\underline{\mathcal{V}}(\vec{r})$, $\vec{r} \in E_{d_m}$, cannot be ignored and has to be accounted for. Another difference is explained in the following.

As shown in Chapter 3, in the case of hard-decision algorithms, as long as we have the values of J and $|E_J|$, we can have a very good estimate using Equation (3.4.7) at least in the very small crossover probability region. This is because the FER (3.4.3) tends to zero as a function of ε^J, for sufficiently small ε. To be more precise, the dominant (largest) term in Equation (3.4.3) for sufficiently small ε is equal to $|E_J|\varepsilon^J(1-\varepsilon)^{n-J}$. This, however, may not be the case for quantized algorithms.

In other words, $P(E_{d_m})$ in Equation (5.2.2) may not be the dominant term even at the high SNR region. As shown in Equation (5.2.2), $P(E_d)$ can be considered as the product of two terms: $P(S_d)$ and $P(E_d)/P(S_d)$. $P(S_d)$ is the probability of having an input vector of distance d from $\vec{R}$ and depends on the SNR and the quantization scheme. The conditional probability $P(E_d)/P(S_d)$ however depends on the error-correcting capability of the code and the decoder. The reason that $P(E_{d_m})$ may not be the dominant term in (5.2.2) is partly due to the fact that the distance d which maximizes $P(S_d)$ is usually much larger than d_m. In the q-bit quantization scheme considered in this chapter, at high SNR, with high probability the quantized values fall into the interval which contains the real value 1. If the integer representing this interval is $r \in \mathcal{A}_q$, at high SNR, the value r appears with high probability for all the n positions of the input vector, which translates to a large $P(S_d)$ for $d = \sqrt{n(2^{q-1} - 1 - r)^2}$ (the value of r depends on q and the clipping threshold c_{th}). The following example illustrates this for the (7,4) Hamming code.

Example 5.2.5 Consider the 3-bit MS decoder to decode the (7,4) Hamming code with $c_{th} = 2$ and $I_{max} = 30$. From Example 5.2.2, we know that $d_m = \sqrt{24}$. We calculate $P(S_d)$ for each possible distance d in the range of 0 to $\sqrt{252}$ and for different SNR values. The results are given in Figure 5.5. It can be seen that at high SNR, $P(S_{\sqrt{24}})$ is not the largest. In fact, in this example, since the real value 1 is contained in the quantization interval represented by integer 2, the set $S_{\sqrt{7}}$ is asymptotically dominant (at very high SNR). We also calculate $P(E_d)$ for several distances. The results are shown in Figure 5.6. It is clear that $P(E_{\sqrt{24}})$ is not the largest. □

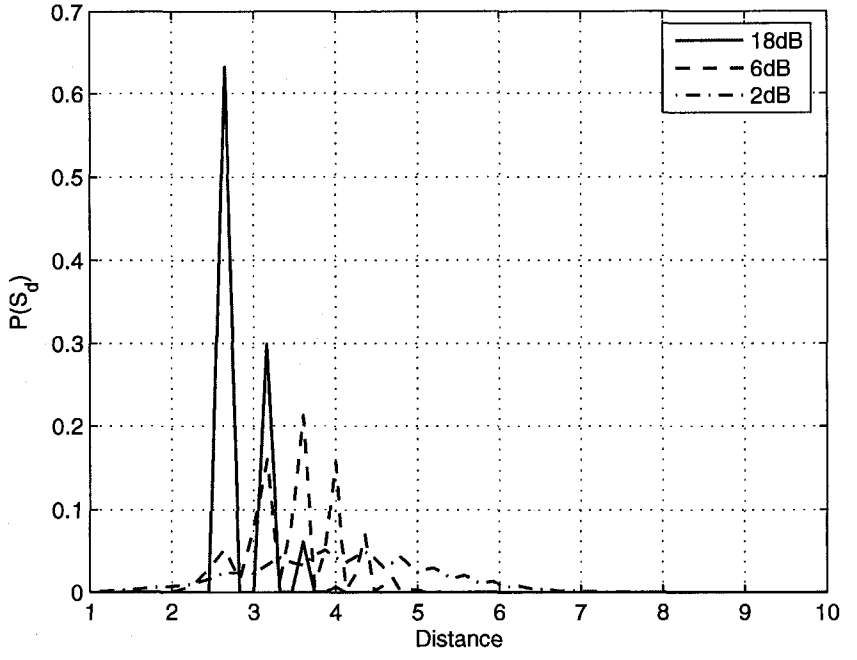

Figure 5.5: Probability of set S_d versus d for different SNR values.

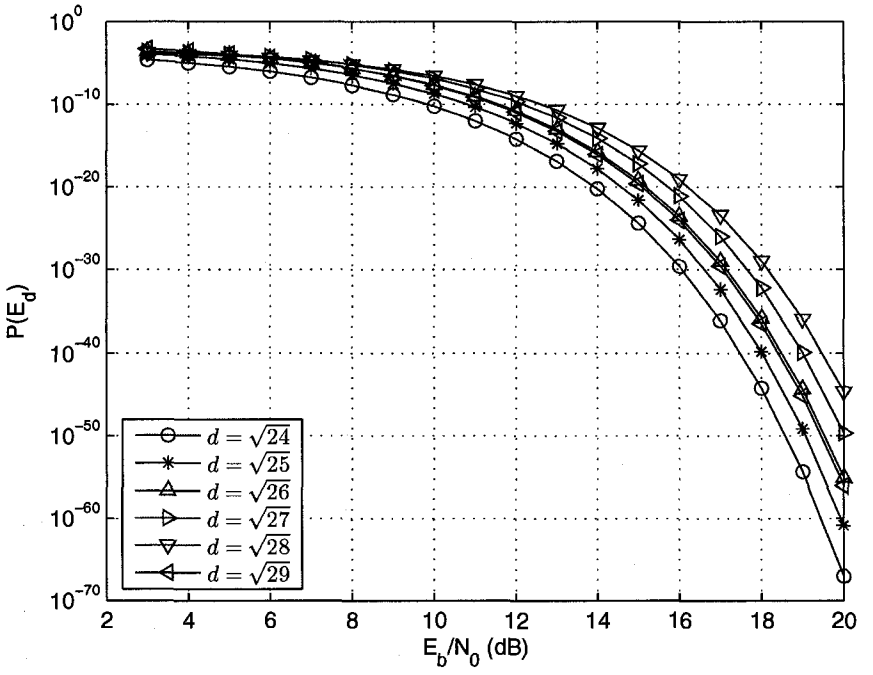

Figure 5.6: Probability of set E_d versus SNR for different values of d.

Although in Examples 5.2.2 and 5.2.4, it has been shown that the information of E_{d_m} is enough to provide a very good estimate for the FER, this is usually not the case for practical codes. In other words, vectors in the sets which have dominant values of $P(E_d)$ may not be included in sets $\underline{\mathcal{V}}(\vec{r})$, $\vec{r} \in E_{d_m}$. This requires us to search for the vectors in the sets E_d for $d > d_m$ to improve the estimates. Direct enumeration on input vectors is infeasible for practical codes. In the next subsection, we propose a search technique which looks for vectors in the sets E_d for small values of d. We also propose modified estimation equations which incorporate the information obtained by the search technique to improve the accuracy of the estimates.

5.2.3 Modified Cycle Enumeration Algorithm and Modified Estimation Equation

To have an accurate estimate for the error rate, we usually need information about sets E_d, $d > d_m$, in addition to the set E_{d_m}. The computational complexity of direct enumeration for obtaining such information is high and this approach can easily become impractical even for small block lengths and small values of q.

Cycle enumeration proposed in Chapter 4 has been shown to be a powerful tool in estimating the error rate of hard-decision iterative decoding algorithms which can reduce the computational complexity by several orders of magnitude compared to direct enumeration. Instead of exhaustive search, cycle enumeration focuses on input vectors whose error locations correspond to small cycles in the underlying TG, which are believed to be problematic structures for iterative decoding algorithms.

Let C_l denote the set of cycles of length l, $l = g, g+2, \ldots$ (g is the girth of the graph). In cycle enumeration proposed in Chapter 4, we enumerate cycles of different lengths starting from those in C_g. For a q-bit decoder, for each cycle, input vectors are generated by setting the vector elements equal to $2^{q-1}-1$ in all positions except those involved in the cycle. For the positions involved in the cycle, we consider all the permutations associated with the 2^q-1 possible values in each position. For a cycle of length l, this process generates $(2^q-1)^{l/2}$ test vectors. Note that, the $(2^q-1)^{l/2}$ test vectors correspond to different values of d. By passing the test vectors to the decoder, we obtain vectors in sets E_d for different distances.

Since we are interested in sets E_d with small values of d, we do not need to pass all the $(2^q-1)^{l/2}$ vectors to the decoder. In fact, instead of testing all possible vectors corresponding to one cycle set for a given length l and then going to the next larger cycle set (with length $l+2$), we take a different approach called *modified cycle enumeration algorithm.* This approach is summarized in Algorithm 2.

In the modified cycle enumeration algorithm, the input vectors are generated based on their distance to $\vec{R}$ in increasing order. For a given distance, we enumerate all possible vectors corresponding to cycles under consideration. We first generate cycle sets of different lengths, e.g., $C_g, C_{g+2}, \cdots$. We use the set $T_k \triangleq \{C_g, C_{g+2}, \cdots, C_{g+2k}\}$, $(k = 0, 1, \cdots)$ as possible error location sets. We then create another list PL_k, which stores error patterns corresponding to all possible number of locations in set T_k (i.e., $g/2, \cdots, (g+2k)/2$) in increasing order of the distance. Let D'_k denote the set of distances corresponding to the error patterns in PL_k. For a given distance $d \in D'_k$, PL_k stores error patterns of distance d with number of locations $g/2, \cdots, (g+2k)/2$.

Algorithm 2 Modified Cycle Enumeration

1. Create T_k, PL_k and D'_k.
2. $d = min\{x : x \in D'_k\}$, $d_m = \infty$.
3. Decode all patterns in PL_k of distance d to $\vec{R}$ and corresponding to locations in T_k.
4. If there are patterns that cannot be corrected, $d_m = min\{d, d_m\}$ and update set $\widetilde{E}_d$.
5. If $d_m = \infty$, $d =$ the next larger distance in D'_k, go to step 3;
   else generate error rate estimate.
6. If estimates converge, stop;
   else $d =$ the next larger distance in D'_k, go to step 3.

Example 5.2.6 Consider the case of a 2-bit decoder applied to a graph with $g = 6$. $T_0 = \{C_6\}$ contains all the cycles of length 6. In this case, PL_0 contains error patterns with $g/2 = 3$ locations. For example, PL_0 contains error patterns (-1, 1, 1), (1, -1, 1), (1, 1, -1) corresponding to distance 2. For the maximum distance $\sqrt{12}$ corresponding to 3 locations, PL_0 stores the only pattern with this distance to $\vec{R}$, i.e., (-1, -1, -1). The set D'_0 in this case contains distances 1, $\sqrt{2}$, $\sqrt{3}$, 2, $\cdots$, $\sqrt{12}$, where the first three values correspond to having the value zero at one, two and three locations, respectively. □

After creating T_k and PL_k, the algorithm starts generating input vectors to the decoder from the smallest distance $d_1 \in D'_k$. If there are patterns in PL_k of distance d_1 and corresponding to $g/2$ locations, the input vectors are generated by having $2^{q-1} - 1$ in all positions except those in C_g. The positions in C_g use corresponding error patterns in PL_k as their values. After testing all locations in C_g, the algorithm repeats this process for the locations corresponding to the set C_{g+2} and so on. If at

least one of the input vectors with distance d_1 cannot be corrected, the algorithm estimates d_m as d_1. The algorithm also updates the set E_{d_1}. If all the error patterns of distance d_1 are corrected by the decoder, d_m is set to ∞. The algorithm continues by repeating the process for patterns in PL_k with the next larger distance. If $d_m \neq \infty$, estimates for FER is obtained. The algorithm continues by examining the next larger distance error patterns in PL_k until $d_m \neq \infty$ and at least one estimate for the FER is available. As error patterns with larger distance are examined, the estimates improve. One can in fact stop the algorithm when two successive estimates are close together indicating a convergent behavior.

We use $\widetilde{E}_d$ to denote the estimate of the set E_d obtained by the algorithm. It is clear that, the more cycle sets are included in the set T_k, the more accurate the information about d_m and sets E_d will be. However, the computational complexity increases as well. Our experiments show that for codes with block lengths of several hundreds to around a thousand, the first 3 or 4 cycle sets, i.e., $C_g, \cdots, C_{g+6}$ are enough for an accurate estimate. Similar to the approach presented in Chapter 4, one can refine the estimates by increasing k and by stopping the algorithm when the estimates for two successive values of k are close enough to indicate convergence.

Consider the case where Algorithm 2 is at Step 5 and for the first time $d_m \neq \infty$ and is estimated as $d_1 \in D'_k$. The algorithm also returns the set $\widetilde{E}_{d_1}$. The error rate estimates are then obtained by replacing E_{d_m} in Equation (5.2.4) by $\widetilde{E}_{d_1}$. By continuing the algorithm at Step 6 and examining the patterns of the next larger distance $d_2 \in D'_k$, we would find vectors in the set $\widetilde{E}_{d_2} \subseteq E_{d_2}$. We should use the information of $\widetilde{E}_{d_2}$ to improve our estimates. Ideally, we would like to estimate the

FER by Equation (5.2.4) as: $FER \approx P(\underline{U}(\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2})) + P(S_{d>d_{th}})$. For practical codes however, it is very hard to enumerate vectors in the set $\underline{U}(\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2})$. Let $\mathcal{D}$ denote all the distances corresponding to the sets obtained by the algorithm (in this case, $\mathcal{D} = \{d_1, d_2\}$). Let L denote all the locations corresponding to the vectors in the set $\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2}$. Clearly, L is a subset of T_k. We refer to each element $l \in L$ as a *location*. It turns out that many vectors in the set $\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2}$ have the same locations. Let $\widetilde{E}_{d,l}$, $d \in \mathcal{D}$, $l \in L$ be the set of vectors of distance d that have the same location l. For each $l \in L$, $\widetilde{E}_{\mathcal{D},l} \triangleq \bigcup_{d \in \mathcal{D}} \widetilde{E}_{d,l}$ is the set of vectors in $\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2}$, which have the same location l. Set $\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2}$ can be partitioned based on the locations as: $\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2} = \bigcup_{l \in L} \widetilde{E}_{\mathcal{D},l}$. Thus, we have $\underline{U}(\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2}) = \underline{U}(\bigcup_{l \in L} \widetilde{E}_{\mathcal{D},l}) = \bigcup_{l \in L} \underline{U}(\widetilde{E}_{\mathcal{D},l})$. Therefore, $P(\underline{U}(\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2}))$ can be calculated as:

$$P(\underline{U}(\widetilde{E}_{d_1} \bigcup \widetilde{E}_{d_2})) = P(\bigcup_{l \in L} \underline{U}(\widetilde{E}_{\mathcal{D},l})) \approx \sum_{l \in L} P(\underline{U}(\widetilde{E}_{\mathcal{D},l})). \tag{5.2.6}$$

$P(\underline{U}(\widetilde{E}_{\mathcal{D},l}))$ can be calculated efficiently as shown later in the chapter. Note that, although there are overlaps between sets with different locations, the computational complexity of removing the overlaps is high. Moreover, our results show that the overlapped vectors have very small probabilities and do not affect the accuracy of the estimates in any significant way. We thus use the following modified FER estimate:

$$FER \approx \sum_{l \in L} P(\underline{U}(\widetilde{E}_{\mathcal{D},l})) + P(S_{d>d_{th}}). \tag{5.2.7}$$

As the algorithm progresses and vectors of larger distances are examined, we will

have new sets of $\widetilde{E}_d$ available. This information can be easily incorporated into the estimation equations following the same approach used in deriving (5.2.7) to obtain more accurate estimates for error rates.

5.2.4 Estimation for Decoders with Large q Based on the Results for Small q

The number of input vectors increases exponentially with the value of q as $|\mathcal{V}| = (2^q - 1)^n$. Although the modified cycle enumeration algorithm (Algorithm 2) greatly reduces the search space, the computational complexity is still high for many practical cases where the block length and the value of q are large. Due to the relatively low computational complexity for the cases with small q (e.g., 2, 3), we would like to obtain the estimates for larger values of q based on the estimates we already have for smaller values of q so that we do not need to repeat Algorithm 2 for different values of q.

Consider the application of Algorithm 2 to a q_1-bit decoder. The algorithm returns vector sets $\widetilde{E}^{q_1}_{d,l}$, $d \in D_{q_1}$, $l \in L$, where D_{q_1} is the set of distances and L is the set of locations. To obtain estimates for a q_2-bit decoder ($q_2 > q_1$), we use L as the location set. Since cycles in the set L are found to be problematic structures for the q_1-bit decoder, with high probability, they are problematic structures for the q_2-bit decoder as well. For each location $l \in L$, error patterns corresponding to l for the q_2-bit decoder are generated based on vectors in sets $\widetilde{E}^{q_1}_{d,l}$, $d \in D_{q_1}$. Each quantization interval in the q_1-bit case is divided into several intervals in the q_2-bit case, which

Algorithm 3 Performance Estimation of the q_2-bit Decoder Based on Results for the q_1-bit Decoder ($q_2 > q_1$)

Input: $\widetilde{E}_{d,l}^{q_1}$, $d \in D_{q_1}$, $l \in L$;
$\widetilde{E}_{d,l}^{q_2} = \varnothing$, $D_{q_2} = \varnothing$;
for each $\vec{r} \in \widetilde{E}_{d,l}^{q_1}$, $d \in D_{q_1}$, $l \in L$
  Generate set $S(\vec{r}, q_2)$;
    for each $\vec{t} \in S(\vec{r}, q_2)$
      if $\vec{t}$ is not included and below any vector in $\widetilde{E}_{d,l}^{q_2}$, $d \in D_{q_2}$, $l \in L$
        Decode $\vec{t}$ with q_2-bit decoder and update $\widetilde{E}_{d,l}^{q_2}$ and D_{q_2};
      end
    end
end
Generate the error rate estimate for the q_2-bit decoder.

means that, each integer $r \in \mathcal{A}_{q_1}$ corresponds to several integers in $\mathcal{A}_{q_2}$. Thus, for each $\vec{r} \in \widetilde{E}_{d,l}^{q_1}$, we generate a set of input vectors $S(\vec{r}, q_2)$ for the q_2-bit decoder. Each vector $\vec{t} \in S(\vec{r}, q_2)$ has value $2^{q_2-1} - 1$ in all positions except for those in l. The value of each position in l for vector $\vec{t}$ is an integer in $\mathcal{A}_{q_2}$ corresponding to the value (an integer in $\mathcal{A}_{q_1}$) in the same position of $\vec{r}$.

By decoding vectors in sets $S(\vec{r}, q_2)$, $\vec{r} \in \widetilde{E}_{d,l}^{q_1}$, we find vectors which cannot be corrected by the q_2-bit decoder and belong to sets $\widetilde{E}_{d,l}^{q_2}$, $d \in D_{q_2}$, $l \in L$ (note that, the location set L is the same as that of the q_1-bit case, the distance set D_{q_2} is however different). We then generate the error rate estimate based on $\widetilde{E}_{d,l}^{q_2}$ following the same approach used in deriving (5.2.7).

This approach is summarized in Algorithm 3.

If each integer in $\mathcal{A}_{q_1}$ corresponds to K integers in $\mathcal{A}_{q_2}$ and there are $|l|$ positions corresponding to the location l, the number of input vectors generated for the q_2-bit decoder based on a vector $\vec{r} \in \widetilde{E}_{d,l}^{q_1}$ is $|S(\vec{r}, q_2)| = K^{|l|}$. In practice, the values of K and $|l|$ are usually small, which translates to a small number of iterative decoding

instances for each vector $\vec{r}$. To further reduce the computational complexity, we do not need to decode all the vectors in $S(\vec{r}, q_2)$. We only decode input vectors which are not the same as or below any vector in sets $\widetilde{E}_{d,l}^{q_2}$, $d \in D_{q_2}$, $l \in L$, found in previous steps of the algorithm. For example, assume the algorithm has obtained some vectors in sets $\widetilde{E}_{d,l}^{q_2}$, $d \in D_{q_2}$, $l \in L$ at the current step, at the next step, the algorithm generates new test vector $\vec{t} \in S(\vec{r}, q_2)$. The new vector $\vec{t}$ may be already included in some set $\widetilde{E}_{d,l}^{q_2}$ or it may be below certain vector in set $\widetilde{E}_{d,l}^{q_2}$, for both cases, we don't pass vector $\vec{t}$ to the decoder. The following example shows the saving in computational complexity of this approach compared with the direct application of the modified cycle enumeration approach of Algorithm 2.

Example 5.2.7 A (273, 82) regular LDPC code from [58] with variable node degree 3 and check node degree 10 is decoded by a 3-bit MS decoder ($c_{th} = 2$ and $I_{max} = 100$). By enumerating the cycles in C_8, we find 24 input vectors which cannot be corrected by the decoder and they all have the minimum distance $\sqrt{38}$ to $\vec{R}$. A very good estimate of the FER of the 3-bit decoder can be obtained by only using these 24 vectors in the set $\widetilde{E}_{\sqrt{38},l}^{3}$, $l \in L$. Consider now the application of Algorithm 3 to estimate the performance of a 4-bit MS decoder based on the information of $\widetilde{E}_{\sqrt{38},l}^{3}$, $l \in L$. In this case, we have $K = 3$ and $|l| = 4$ (4 variable nodes in a cycle of length 8). Therefore, for each of the 24 vectors $\vec{r} \in \widetilde{E}_{\sqrt{38},l}^{3}$, $l \in L$, we have $|S(\vec{r}, q_2)| = 81$. The total number of input vectors generated for the 4-bit decoder is therefore: $24 \times 81 = 1944$. From this number, we only need to test 1124 vectors since the rest are either repetitions or are below some of the 1124 vectors. By decoding the 1124 vectors, we find 260 vectors

that cannot be corrected and belong to the union of sets $\widetilde{E}^4_{d,l}$, where $d \in D_4$ and $l \in L$. We also obtain $min\{D_4\} = \sqrt{198}$. Using the 260 vectors, we can have a very accurate estimate for the FER of the 4-bit decoder. Instead of using Algorithm 3, if we were to obtain the FER estimate by the modified cycle enumeration algorithm, the complexity would be significantly higher. To see this, note that $|C_8| = 13289$ and the number of patterns with distance $\sqrt{198}$ in the list PL_k is 264. Thus, even if Algorithm 2 only enumerates patterns of distance $\sqrt{198}$, it needs to perform $264 \times 13289 = 3508296$ iterative decoding instances, which is over 3000 times larger than the first approach. Note that in this calculation, we have not even considered the enumeration and test of the patterns in PL_1 with distance smaller than $\sqrt{198}$. □

5.2.5 Calculation of Probabilities of Different Sets

In Equation (5.2.7), we need to calculate the probabilities of different input vector sets, namely, sets $\underline{U}(\widetilde{E}_{D,l})$, $l \in L$ and the set $S_{d>d_{th}}$. By definition, we have $P(\underline{U}(\widetilde{E}_{D,l})) = \sum_{\vec{r} \in \underline{U}(\widetilde{E}_{D,l})} P(\vec{r})$ and $P(S_{d>d_{th}}) = \sum_{\vec{r} \in S_{d>d_{th}}} P(\vec{r})$. Enumerating the vectors in these sets and calculating their probabilities is however too complex due to the large number of vectors in these sets. In this subsection, we propose efficient ways of calculating these probabilities.

To calculate $P(\underline{U}(\widetilde{E}_{D,l}))$ for a given $l \in L$, we consider two cases.

In the first case, if there is only one vector $\vec{r}$ in $\widetilde{E}_{D,l}$, then $P(\underline{U}(\widetilde{E}_{D,l})) = P(\underline{\mathcal{V}}(\vec{r}))$. We thus have:

$$P(\underline{\mathcal{V}}(\vec{r})) \;=\; P(\vec{t} \leq \vec{r}) = \prod_{i=1}^{n} P(t_i \leq r_i) = \prod_{i \in l} \left(\sum_{t_i \leq r_i} p(t_i) \right) = \prod_{i \in l} \left(Q\left(\frac{1 - x_i}{\sigma} \right) \right),$$

where x_i is the largest real value corresponding to the quantized value r_i.

For the case where $\widetilde{E}_{D,l}$ has multiple vectors, let $\mathcal{W} = \mathcal{A}_q^{|l|}$ and let $\vec{r}(l)$ denote the sub-pattern of $\vec{r} \in \mathcal{V}$ corresponding to location l. Clearly, $\vec{r}(l) \in \mathcal{W}$. For $\vec{x} = \{x_1, \cdots, x_{|l|}\} \in \mathcal{W}$, $\underline{\mathcal{W}}(\vec{x}) \triangleq \{\vec{y} \in \mathcal{W} : \vec{y} \leq \vec{x}\}$, i.e., $\underline{\mathcal{W}}(\vec{x})$ contains all the $|l|$-tuples in the set $\mathcal{W}$ which are below $\vec{x}$. Let $\underline{V}(\widetilde{E}_{D,l}) \triangleq \bigcup_{\vec{r} \in \widetilde{E}_{D,l}} \underline{\mathcal{W}}(\vec{r}(l))$, i.e., $\underline{V}(\widetilde{E}_{D,l})$ contains all the $|l|$-tuples, each of which is below the sub-pattern $\vec{r}(l)$ for at least one $\vec{r} \in \widetilde{E}_{D,l}$. In practice, $\underline{V}(\widetilde{E}_{D,l})$ can be enumerated efficiently due to the small value of $|l|$. The probability of the set $\underline{U}(\widetilde{E}_{D,l})$ is then calculated using the following lemma.

Lemma 5.2.1 For a given location l, $P(\underline{U}(\widetilde{E}_{D,l})) = \sum_{\vec{x} \in \underline{V}(\widetilde{E}_{D,l})} \left(\prod_{i=1}^{|l|} p(x_i) \right)$, where $p(x)$ is given by Equation (5.2.1).

Proof: Let $\xi(l, \vec{x}) \triangleq \{\vec{t} \in \mathcal{V} : \vec{t}(l) = \vec{x}\}$, $\vec{x} \in \mathcal{W}$. The set $\xi(l, \vec{x})$ contains all the vectors in $\mathcal{V}$ whose sub-patterns corresponding to location l are the same and equal to $\vec{x}$. We first show that sets $\underline{U}(\widetilde{E}_{D,l})$ and $\bigcup_{\vec{x} \in \underline{V}(\widetilde{E}_{D,l})} \xi(l, \vec{x})$ are equal.

For any vector $\vec{a} \in \underline{U}(\widetilde{E}_{D,l})$, by definition, there exists at least one $\vec{r} \in \widetilde{E}_{D,l}$, such that $\vec{a} \leq \vec{r}$. By the definition of "below", we have $\vec{a}(l) \leq \vec{r}(l)$. Thus, $\vec{a}(l) \in \underline{V}(\widetilde{E}_{D,l})$ which implies $\vec{a} \in \bigcup_{\vec{x} \in \underline{V}(\widetilde{E}_{D,l})} \xi(l, \vec{x})$. On the other hand, for any vector $\vec{a} \in \bigcup_{\vec{x} \in \underline{V}(\widetilde{E}_{D,l})} \xi(l, \vec{x})$, by definition, $\vec{a}(l) \in \underline{V}(\widetilde{E}_{D,l})$. Thus, there is at least one $\vec{r} \in \widetilde{E}_{D,l}$ such that $\vec{a}(l) \leq \vec{r}(l)$. Since all positions of $\vec{r}$ except those in l have value

$2^{q-1}-1$, we have $\vec{a} \leq \vec{r}$. Therefore, $\vec{a} \in \underline{U}(\widetilde{E}_{D,l})$. So, sets $\underline{U}(\widetilde{E}_{D,l})$ and $\bigcup_{\vec{x}\in\underline{V}(\widetilde{E}_{D,l})} \xi(l,\vec{x})$ are the same.

It is clear that sets $\xi(l,\vec{x})$ are disjoint for different $\vec{x} \in \underline{V}(\widetilde{E}_{D,l})$. Thus, we have:

$$\begin{aligned} P(\underline{U}(\widetilde{E}_{D,l})) &= \sum_{\vec{x}\in\underline{V}(\widetilde{E}_{D,l})} P(\xi(l,\vec{x})) \\ &= \sum_{\vec{x}\in\underline{V}(\widetilde{E}_{D,l})} P(\vec{t}\in\mathcal{V} : \vec{t}(l) = \vec{x}) \\ &= \sum_{\vec{x}\in\underline{V}(\widetilde{E}_{D,l})} P(t_i \in \mathcal{A}_q : i \notin l, 1 \leq i \leq n) P(\vec{t}(l) = \vec{x}) \\ &= \sum_{\vec{x}\in\underline{V}(\widetilde{E}_{D,l})} \left(\prod_{i=1}^{|l|} p(x_i) \right) \end{aligned}$$

■

To obtain $P(S_{d>d_{th}})$, we propose an algorithm which computes $P(S_d)$ for all possible distances $d \in \{0, \cdots, d_M\}$. $P(S_{d>d_{th}})$ is then calculated as $\sum_{d>d_{th}} P(S_d)$. For a code with block length n and a q-bit decoder, although the number of input vectors is an exponential function of n, i.e., $(2^q-1)^n$, the number of possible Euclidean distances between an input vector and $\vec{R}$ increases only with $\sqrt{n}$ and is upper bounded by $2(2^{q-1}-1)\sqrt{n}$. The algorithm creates and updates two lists: the distance list DL and the corresponding probability list $P(S_d)$, $d \in$ DL. The algorithm starts from the cases where the input vector length is one and continues to the cases where the length is increasingly larger and finally reaches n. Let $\vec{R}(1:i)$,$i = 1, \cdots, n$, denote the vector of length i consisting of the first i elements of $\vec{R}$ in the same order as they appear in $\vec{R}$. Consider the case where the algorithm has already updated lists DL and $P(S_d)$ for cases where the input vector length is k. When the new position $k+1$ is added,

Algorithm 4 Calculation of $P(S_d)$, $d = 0, \cdots, d_M$

```
for i=1 to n
  if i==1
    for each r ∈ A_q
      d²_new = [r − (2^{q−1} − 1)]² and P_{d_new} = p(r);
      Add d_new and P_{d_new} to lists DL and P(S_d), respectively;
    end
  else
    for each r ∈ A_q
      if r==2^{q−1} − 1
        for each d ∈DL
          P(S_d) = P(S_d) · p(r);
        end
      else
        for each d ∈DL
          d²_new = d² + [r − (2^{q−1} − 1)]²;
          if d_new ∈DL: P(S_{d_new}) = P(S_{d_new}) + P(S_d) · p(r);
          else: Add d_new to list DL, P(S_{d_new}) = P(S_d) · p(r);
        end
    end
end
```

for each value $r \in \mathcal{A}_q$ in this position and for each distance $d \in$DL, the algorithm first calculates the new distance d_{new} of the input vector to $\vec{R}(1:k+1)$. If d_{new} is already in the list DL, the algorithm just updates the value of $P(S_{d_{new}})$. If d_{new} is not in the list, the algorithm adds d_{new} to DL and updates the value of $P(S_{d_{new}})$. At the end of this round of the algorithm, $P(S_d)$ is the probability that an input vector of length $k+1$ has distance d to $\vec{R}(1:k+1)$. The pseudo code of the algorithm is given in Algorithm 4. In the algorithm, for each $r \in \mathcal{A}_q$, $p(r)$ is given by Equation (5.2.1). The functionality of Algorithm 4 can be easily verified by induction on the length of the input vectors.

Algorithm 4 is also used to generate the list PL_k and the set D_k' needed in Algorithm 2. For a given number of error positions (e.g., $g/2+k$), PL_k stores all possible

patterns for each possible distance corresponding to the $g/2+k$ positions. To obtain PL_k, Algorithm 4 runs from $i=1$ to $g/2+k$ and records the corresponding patterns for different values of d.

5.2.6 Computational Complexity

Based on the above discussions, by combining Algorithms 2 and 3, we can obtain estimates for q-bit decoders for different values of q. In both algorithms, the computational complexity is mainly due to the instances of iterative decoding. In Algorithm 2, the number of iterative decoding instances depends on the size of the set T_k, the number of patterns in the list PL_k corresponding to different distances and the number of rounds the algorithm needs to go through before stopping. To generate estimates for larger q, we use Algorithm 3 based on the results of Algorithm 2 for a small value of q. As we have shown in Example 5.2.7, the computational complexity of Algorithm 3 is usually much less than that of the direct application of Algorithm 2. In general, the overall computational complexity of the proposed approach is much less than that of the Monte Carlo simulation, especially in the high SNR (error floor) region.

Note that while the proposed approach can easily generate estimates of the error rate for different SNR values when the sets $\widetilde{E}_d$ for small values of d are found, in the Monte Carlo simulation, a new set of input vectors has to be generated and decoded for each value of SNR. Moreover, by combining Algorithms 2 and 3, we can obtain estimates easily for decoders with different values of q. This is while Monte Carlo simulations have to be performed for each q-bit decoder separately.

5.3 Experimental Results

Our estimation approach can be, in principle, applied to any symmetric decoder. We first consider the quantized MS described in [81]. We report our results for three LDPC codes taken from [58]. The first code is the (273, 82) code used previously in Example 5.2.7. The second code is a (1008, 504) regular LDPC code with variable node degree 3 and check node degree 6. The third code is a (504, 252) LDPC code constructed by the progressive edge-growth (PEG) algorithm of [15] and has variable node degree 3 and check node degrees 5, 6 and 7. The first two codes have girth 6 while the girth of the PEG code is 8. For all the three codes, we use $c_{th} = 2$ and $I_{max} = 100$.

Figure 5.7 shows the simulation and estimation results for the (273, 82) code for 3-bit, 4-bit and 5-bit MS decoders. Algorithm 2 is used to obtain the estimate for the 3-bit decoder. By using the cycle set C_8, we estimate the minimum distance d_m to be $\sqrt{38}$ and find 24 vectors in the set $\widetilde{E}^3_{\sqrt{38}}$. The estimates of 4-bit and 5-bit decoders are obtained by using Algorithm 3 based on the results of 3-bit and 4-bit decoders, respectively. The estimated value of d_m for the two decoders is respectively $\sqrt{198}$ and $\sqrt{902}$. As can be seen in Figure 5.7, for all the three decoders, based on Equation (5.2.7), we are able to obtain accurate estimates of the FER which closely match the Monte Carlo simulation results for the whole range of SNR values of interest.

Figures 5.8 and 5.9 show the simulation and estimation results for the (1008, 504) and (504, 252) codes for different q-bit decoders, $q = 3, 4, 5$, respectively. For these codes, the computational complexity of Algorithm 2 is quite high even to obtain the

estimate for the 3-bit decoder. We have thus applied Algorithm 2 with the cycle set T_3 to a 2-bit decoder first (note that, for the (504, 252) code, since the girth is 8, $T_3 = \{C_8, \cdots, C_{14}\}$). As a result, we obtain $|\widetilde{E}^2_{\sqrt{9}}| = 10$, $|\widetilde{E}^2_{\sqrt{10}}| = 100$, for the (1008, 504) code, and $|\widetilde{E}^2_{\sqrt{11}}| = 670$, $|\widetilde{E}^2_{\sqrt{12}}| = 1407$, for the (504, 252) code. We then use Algorithm 3 to obtain the estimates for the 3-bit, 4-bit and 5-bit decoders based on the results of 2, 3, and 4-bit decoders, respectively. It can be seen from Figures 5.8 and 5.9 that the estimates are accurate for all three decoders of both cases over the whole SNR range of interest. The distance spectrums estimated by Algorithm 3 for different decoders of each code are given in Table 5.1. In Table 5.1, for each case, we list the first three and the last distances in increasing order and the corresponding multiplicities. For example, for the case where the (273, 82) code is decoded by the 4-bit decoder, the entry $\frac{\sqrt{198}}{24}$ in the table means that 24 input vectors of distance $\sqrt{198}$ to $\vec{R}$ were found by Algorithm 3; $\frac{\sqrt{216}}{48}$ means that the maximum distance obtained by Algorithm 3 is $\sqrt{216}$ and there are 48 input vectors of this distance. For all codes, the values of d_{th} used in the estimates are given in Table 5.2.

Table 5.3 shows the number of iterative decoding instances that need to be performed to generate the estimates for each case. In the table, a number with a plus sign means the extra number of iterative decoding instances that Algorithm 3 needs to perform to generate the estimate if the results of the previous q-bit decoder is available. For example, after Algorithm 2 generates the estimate for the 3-bit decoder of the (273, 82) code (by performing 7.36×10^6 iterative decodings instances), Algorithm 3 needs to perform 1124 and 11688 instances of iterative decoding for the cases of 4-bit and 5-bit decoders, respectively. From Table 5.3, one can see that the

proposed approach is significantly more efficient than the Monte Carlo simulation, especially at the high SNR region.

We have also applied the estimation technique to the (1008, 504) code decoded by a 3-bit BP decoder with non-uniform quantization proposed in [84] ($c_{th} = 3.3$ and $I_{max} = 100$). The proposed quantization scheme is optimized for regular LDPC codes with variable node degree 3 and check node degree 6. For this case, Algorithm 2 is directly applied to the 3-bit decoder with set T_3. As a result, d_m is estimated as $\sqrt{73}$ and $|\widetilde{E}^3_{\sqrt{73}}| = 24$. We also find vectors of distances $\sqrt{75}$, and $\sqrt{76}$ when the algorithm stops. The estimation and simulation results are given in Figure 5.10. The value of d_{th} is found to be $\sqrt{3600}$. It can be seen that the estimate is rather accurate for the whole SNR range of interest. However, the number of iterative decoding instances to generate the estimate is relatively high (about 3×10^9) due to the lack of results for the 2-bit decoder. More accurate results can be obtained by using the cycle set T_3. We however did not attempt this due to the large complexity. Comparing the results of Figure 5.10 with those of Figure 5.8 shows that although the 3-bit BP decoder has a superior performance at lower SNR values compared to the 3-bit MS decoder, the trend reverses at higher SNR values.

Table 5.1: Distance Spectrums Estimated by Algorithm 3 for Different Decoders of Each Code. (Format: $\frac{d}{|\widetilde{E}_d|}$)

	3-bit	4-bit	5-bit
(273,82)	-	$\frac{\sqrt{198}}{24}, \frac{\sqrt{200}}{12}, \frac{\sqrt{202}}{24}, \ldots, \frac{\sqrt{216}}{48}$	$\frac{\sqrt{902}}{24}, \frac{\sqrt{904}}{12}, \frac{\sqrt{906}}{24}, \ldots, \frac{\sqrt{1017}}{24}$
(1008,504)	$\frac{\sqrt{64}}{16}, \frac{\sqrt{65}}{4}, \frac{\sqrt{67}}{54}, \ldots, \frac{\sqrt{96}}{6}$	$\frac{\sqrt{336}}{28}, \frac{\sqrt{343}}{36}, \frac{\sqrt{345}}{6}, \ldots, \frac{\sqrt{544}}{6}$	$\frac{\sqrt{1350}}{24}, \frac{\sqrt{1386}}{12}, \frac{\sqrt{1397}}{12}, \ldots, \frac{\sqrt{2544}}{2}$
(504,252)	$\frac{\sqrt{80}}{67}, \frac{\sqrt{82}}{402}, \frac{\sqrt{84}}{134}, \ldots, \frac{\sqrt{108}}{67}$	$\frac{\sqrt{428}}{67}, \frac{\sqrt{434}}{202}, \frac{\sqrt{435}}{67}, \ldots, \frac{\sqrt{480}}{201}$	$\frac{\sqrt{1964}}{67}, \frac{\sqrt{1971}}{67}, \frac{\sqrt{1978}}{402}, \ldots, \frac{\sqrt{2118}}{402}$

Table 5.2: Values of d_{th} for Different Cases.

	3-bit	4-bit	5-bit
(273, 82)	$\sqrt{790}$	$\sqrt{4700}$	$\sqrt{22300}$
(1008, 504)	$\sqrt{3450}$	$\sqrt{20060}$	$\sqrt{94000}$
(504, 252)	$\sqrt{1750}$	$\sqrt{10100}$	$\sqrt{47500}$

Table 5.3: Number of Iterative Decodings in Different Cases.

	2-bit	3-bit	4-bit	5-bit
(273, 82)	-	7.3×10^6	+1124	+11688
(1008, 504)	4.3×10^7	+45272	$+8.2 \times 10^5$	$+3.1 \times 10^6$
(504, 252)	3.5×10^8	$+9.6 \times 10^5$	$+1.1 \times 10^6$	$+3.9 \times 10^6$

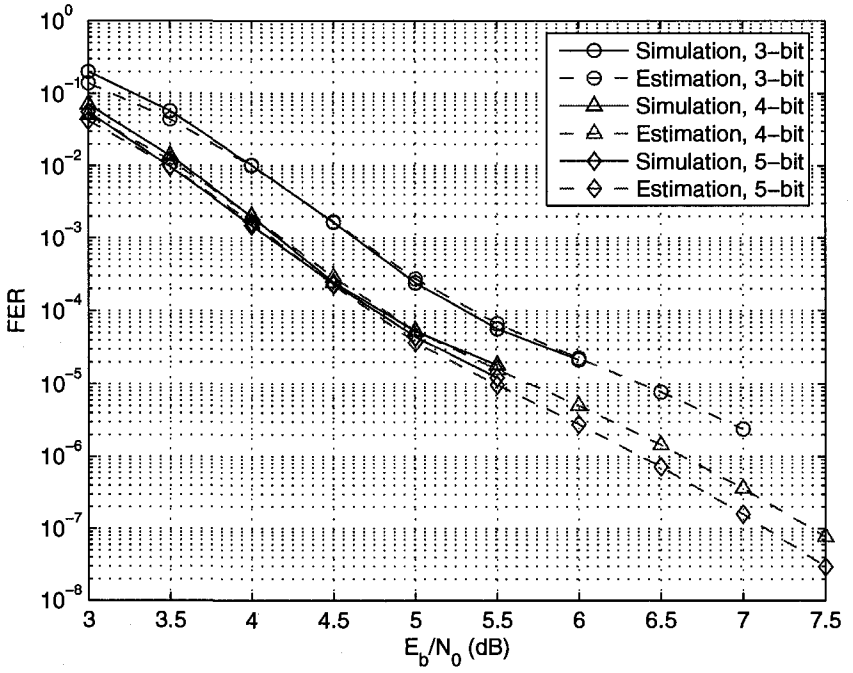

Figure 5.7: Simulation and estimation results of the (273, 82) code decoded by 3-bit, 4-bit and 5-bit MS decoders ($c_{th} = 2$ and $I_{max} = 100$).

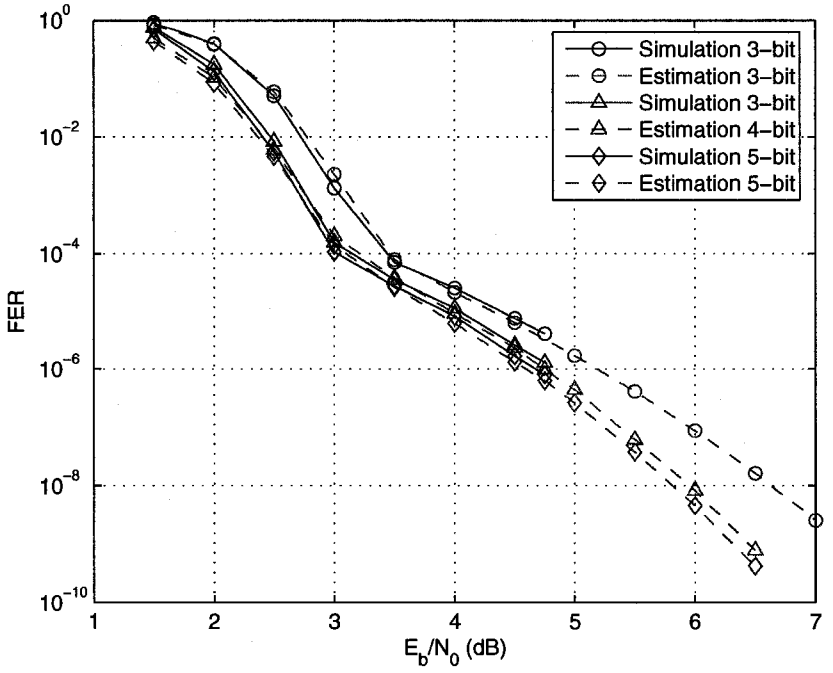

Figure 5.8: Simulation and estimation results of the (1008,504) code decoded by 3-bit, 4-bit and 5-bit MS decoders ($c_{th} = 2$ and $I_{max} = 100$).

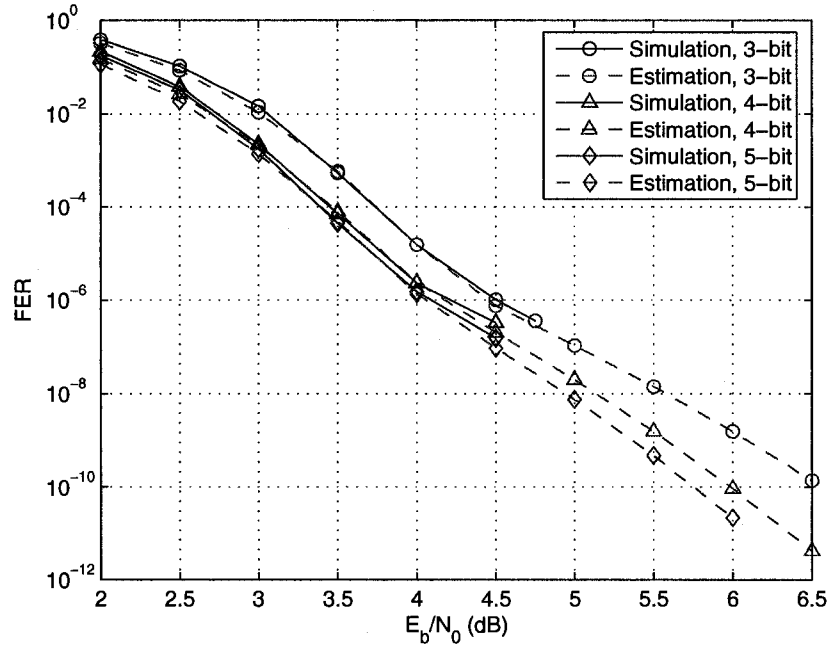

Figure 5.9: Simulation and estimation results of the (504, 252) code decoded by 3-bit, 4-bit and 5-bit MS decoders ($c_{th} = 2$ and $I_{max} = 100$).

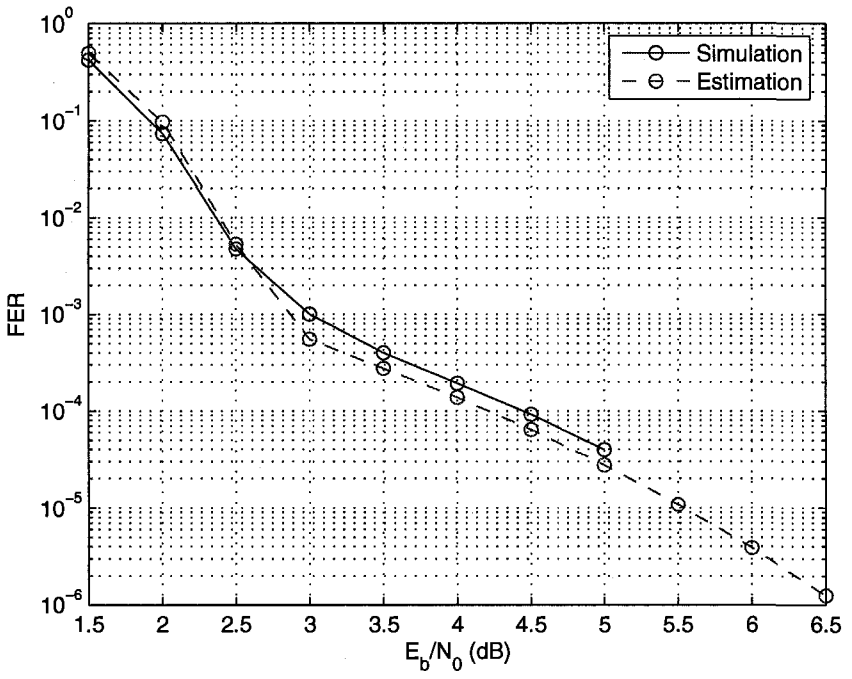

Figure 5.10: Simulation and estimation results of the (1008, 504) code decoded by 3-bit BP decoder ($c_{th} = 3.3$ and $I_{max} = 100$).

5.4 Summary

In this chapter, we proposed an algorithm to estimate the performance of a given finite-length LDPC code decoded by a q-bit (soft-decision) iterative decoder. The proposed method can be applied in principle to any symmetric quantized version of different soft-decision decoding algorithm. We showed by examples that the proposed method can provide accurate FER estimates for the whole SNR region of practical interest. The computational complexity of the proposed method can be much less than that of the Monte Carlo simulation, especially at the low FER region (error floor region). Several techniques including modified cycle enumeration and Algorithm 3

were proposed to reduce the computational complexity. For codes with block lengths larger than about 1000, the computational complexity of the proposed technique is still relatively high. Thus, it is of great practical interest to investigate more efficient techniques to estimate the sets E_d for small values of d.

Chapter 6

Modified PEG Construction Algorithm

In previous chapters, trapping sets (or more precisely, the initial error patterns which cause the decoder to get trapped into trapping sets) are used to obtain estimates of the FER and BER of LDPC codes for iterative decoding algorithms. One can also use the information on the trapping sets in the Tanner graph to construct good codes (graphs). Since trapping sets are the problematic structures in the Tanner graph for iterative decoding algorithms, it is desired to improve the trapping sets of the graph. Improving trapping sets of a Tanner graph means that one would try to reduce the number of trapping sets with small number of variable nodes and unsatisfied check constraints. In this chapter, by increasing the connectivities of cycles to the rest of the graph, we propose a method to improve the trapping sets in the code's graph under belief-propagation (BP) algorithm. Our method is based on the well-known PEG construction which is known to construct LDPC codes at finite block lengths with very good performance. We propose a very simple modification to PEG construction for irregular codes, which considerably improves the performance at high SNR with

no sacrifice in the low SNR performance.

6.1 Introduction

Construction of good LDPC codes at short and intermediate block length is of great practical importance. Among the existing methods such as those in [15,32,57,86,87], one of the most successful approaches for the construction of finite block lengths is the progressive edge-growth (PEG) algorithm proposed by Hu *et al* [15]. PEG construction builds up a Tanner graph, or equivalently a parity-check matrix, for an LDPC code on an edge-by-edge basis and by maximizing the local girth at variable nodes in a greedy fashion. It is simple and also flexible in that it can be applied to construct codes of arbitrary length and rate. In addition, the PEG algorithm can be used to construct linear-time encodable LDPC codes which also perform very well. Moreover, the construction can be used for both regular and irregular LDPC codes. For irregular codes, in particular, the results of [15] show that PEG construction with variable-node degree distributions optimized by density evolution results in very good performance, especially in the waterfall region. It is, however, well-known that the good performance of highly optimized irregular codes in the waterfall region is usually counter-balanced by a relatively poor performance in the error-floor region [32]. In this chapter, we propose a very simple modification to the PEG algorithm which considerably improves the performance of constructed irregular codes in the high SNR region.

In the PEG algorithm it often happens that one has a few candidate check nodes to

connect to a given variable node (all such candidates results in the same local girth under the current graph structure). In the standard PEG algorithm, among such check-node candidates, one chooses either the one with the smallest index or just one at random (both choices result in more or less the same performance [15]).[1] In the proposed modification, by borrowing an idea from [86,87], among the candidate check nodes, we select the one which maintains the highest degree of connectivity for the newly created cycles to the rest of the graph. This implies that these cycles will receive a large amount of extrinsic information from the rest of the graph and thus would not impose as much harm as they would if they were more isolated. The idea borrowed from [86, 87] is to quantify the connectivity of a cycle to the rest of the graph by a simple parameter called "approximate cycle extrinsic message degree (ACE)." This parameter, which will be defined in the next section, has been used in [86,87] to design LDPC codes with low error floors. The codes constructed in [86, 87] however perform worse than the codes constructed by standard PEG algorithm in the waterfall region.[2] This is while for our construction the performance is either similar (at low SNR) or better (at high SNR) than that of standard PEG. We analyze the performance of PEG and modified PEG codes in the high-SNR region and show that the better performance of modified PEG in the error floor region is due to larger minimum distance as well as improved trapping sets.

[1]Other versions of the PEG algorithm (non-greedy and look-ahead-enhanced) have also been discussed in [15], but none of them provides any non-negligible performance improvement over the standard PEG.

[2]For an irregular LDPC code of length 4000 and rate 0.5, with maximum variable node degree 8, PEG code outperforms the construction of [86, 87] by about 0.4dB [87] in the waterfall region.

6.2 Improved PEG Algorithm

Using the same notations and definitions as in [15], we describe the standard PEG algorithm for constructing a Tanner graph with n variable nodes and m check nodes in the following:

Algorithm 5 PEG algorithm [15]

for $j = 0$ to $n-1$ do{
  for $L = 0$ to $d_{v_j} - 1$do {
    if $L = 0$ {
      $E_{v_j}^0 \leftarrow$edge (c_i, v_j), where $E_{v_j}^0$ is the first edge incident to variable node v_j, and c_i is one check node such that it has the lowest check-node degree under the current graph setting $E_{v_0} \cup E_{v_1} \cup \cdots \cup E_{v_j-1}$.
    }
    else {
      expand a subgraph from variable node v_j up to depth l under the current graph setting such that the cardinality of $N_{v_j}^l$ stops increasing but is less than m, or $\overline{N_{v_j}^l} \neq \emptyset$ but $\overline{N_{v_j}^{l+1}} = \emptyset$, then $E_{v_j}^L \leftarrow$edge (c_i, v_j), where $E_{v_j}^L$ is the L-th edge incident to v_j, and c_i is one check node picked from the set $\overline{N_{v_j}^l}$ having the lowest check-node degree.
    }
  }
}

In the PEG algorithm (Algorithm 5), both variable nodes and check nodes are ordered according to their degrees in non-decreasing order, d_{v_j} is the degree of variable node v_j, and $N_{v_j}^l$ and $\overline{N_{v_j}^l}$ denote the set of all check nodes reached by a tree spreading from variable node v_j within depth l, and its complement, respectively.

In PEG algorithm, when a new edge is to be connected to a variable node, we may have multiple candidate check nodes available.[3] In our proposed modification, we focus on cases where $L \geq 1$ and $\overline{N_{v_j}^l} \neq \emptyset$ but $\overline{N_{v_j}^{l+1}} = \emptyset$. In such cases, for each

[3]As mentioned before, although several approaches have been proposed in [15] to deal with this issue, they all result in codes that perform more or less the same.

candidate check node in $\overline{N_{v_j}^l}$ (all with the lowest degree), the addition of the new edge $E_{v_j}^L$ to the graph creates new cycles in the graph, all with length $2(l+2)$. We denote the set of candidate check nodes corresponding to the L-th edge of variable node v_j by $\mathbf{\Omega}_{v_j}^L$. In the modified algorithm, from set $\mathbf{\Omega}_{v_j}^L$, we select the check node whose associated cycles have the highest degree of connectivity to the rest of the graph under the current graph setting. As a measure of connectivity for a cycle, we count the number of edges by which the cycle is connected to the rest of the graph through its variable nodes. This is equal to $\sum_i (d_i - 2)$, where the summation is taken over all the variable nodes in the cycle and d_i is the degree of the ith variable node. This measure was first used in [86,87], and was referred to as ACE. To ensure a high degree of connectivity for the new cycles, from set $\mathbf{\Omega}_{v_j}^L$, we select the check node that maximizes the minimum ACE for the new cycles. In the case that there is more than one node in $\mathbf{\Omega}_{v_j}^L$ with this property, we select one at random. As we will see in the next section, the increase in connectivity of the graph improves both the minimum distance and the trapping sets of the code and thus results in a better error-floor performance. Clearly, the modification proposed here is only effective for the construction of irregular codes. For regular codes, the ACE of all the newly created cycles is the same regardless of the selected check node.

The success of the proposed modification depends on the frequency in which the cardinality of $\mathbf{\Omega}_{v_j}^L$ is greater than one throughout the PEG algorithm. Based on our simulations, this happens quite often. In fact, for both of the codes simulated in the next section, more than half of the edges faced the situation in which the cardinality of $\mathbf{\Omega}_{v_j}^L$ is greater than one. This has resulted in considerable performance improvement

over standard PEG as demonstrated in the next section.

6.3 Simulation Results

In our simulations, we construct four irregular LDPC codes with the same block lengths, rate and variable node degree distribution as those in [15]. All four codes have rate 0.5, and are constructed based on the optimized variable node degree distribution given in Table II of [13] with maximum variable node degree 15. Two codes have parameters $(n, k) = (504, 252)$, while the other two codes are $(1008, 504)$ codes. For each set of parameters, one code is constructed by standard PEG algorithm, and the other one by the modified PEG algorithm proposed here. To provide a comparison between the complexity of PEG and modified PEG algorithms, we measured the running time of both algorithms on a 1GHZ Pentium III processor. For the (504, 252) codes, PEG and modified PEG take about 0.35 and 0.39 seconds, respectively. These numbers for the (1008, 504) codes are 1.43 and 1.55, respectively. It can be see that the proposed modification only results in a small increase in the running time.

Following [15], we use binary phase shift keying (BPSK) modulation over AWGN channel with BP for iterative decoding. Similar to [15], the maximum number of iterations for the decoder is set to 80. For each SNR, the simulation continues until we get at least 100 codewords in error. To have a fair comparison between PEG and modified PEG algorithms, the same noise vectors are used for the codes with the same block lengths.

Figure 6.1 gives the BER and the FER curves for the four codes. It can be

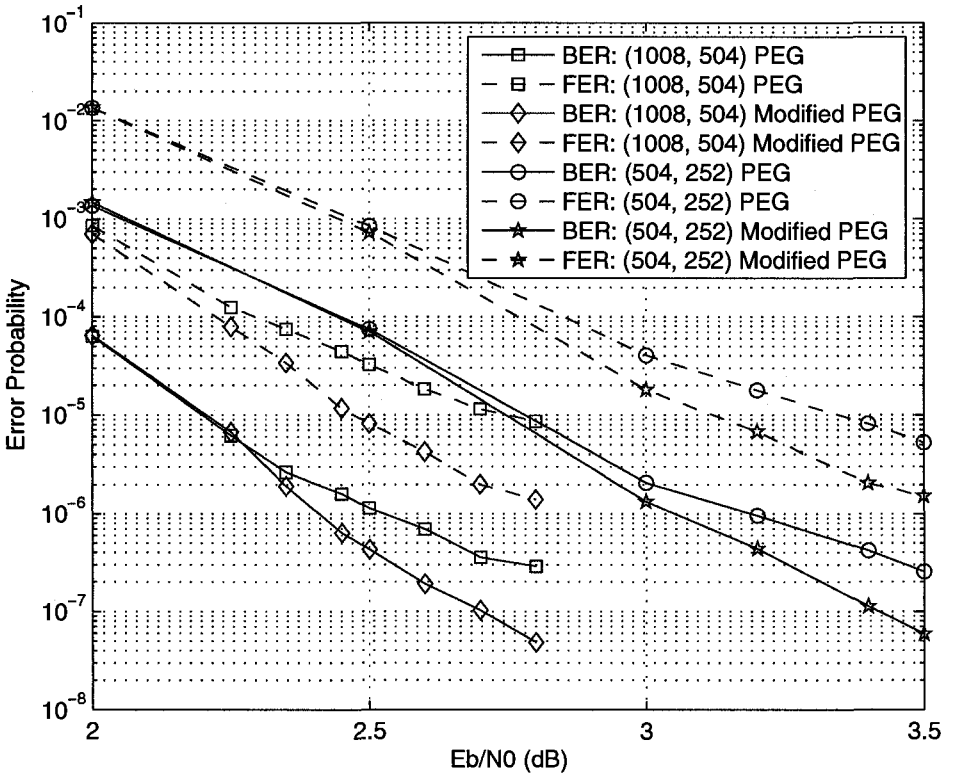

Figure 6.1: BER and FER curves for the PEG and modified PEG codes decoded by sum-product algorithm.

seen that, for both block lengths, at high SNR values, the codes constructed by the modified PEG algorithm outperform the corresponding PEG codes considerably. At lower SNR values the codes perform more or less the same, with modified PEG codes never performing worse.

To analyze the improvement in the error floor region by modified PEG construction, we examined the errors for both PEG and modified PEG codes in this region. The results for (1008, 504) codes are reported here and similar trends exist for (504, 252) codes. For (1008, 504) codes, we focus on the errors at the highest simulated SNR point which is $E_b/N_0 = 2.8$dB. It is known that for the AWGN channel, errors in

the error floor region are mainly due to low-weight codewords as well as trapping sets (also referred to as near-codewords [64]). The former contribute to undetected errors while the latter are responsible for detected errors. Following [14], we use notation (a, b) to denote a trapping set with a variable nodes and b unsatisfied parity-check equations. Trapping sets with both small values of a and b are main contributors to the detected part of errors in the error floor region [14, 64].

For (1008, 504) PEG and modified PEG codes, the percentage of undetected errors at 2.8 dB are 36% and 10%, respectively. We have also observed that for the PEG code, the lowest codeword weight (lowest weight of undetected error) is 12, while for the modified PEG code, this value is increased to 15. For the PEG code, as a relatively large percentage of the errors in the error floor region are attributed to undetected errors, this increase in the weight of lowest weight codewords for modified PEG compared to standard PEG has made some contribution to the better performance of improved PEG in the error floor region.

Moreover, for the PEG code, the values of a and b for trapping sets are on average smaller than those of the modified PEG code. For example, for the PEG code, the trapping set with the smallest a and b is (6,1). There are also several other trapping sets with $b = 1$ and $a \leq 13$.This is while for the modified PEG, only one trapping set with $b = 1$ is observed, and even for that trapping set, $a = 15$, which is larger than all the corresponding values for the PEG code. The larger values of a and b for the trapping sets of modified PEG codes compared to those of their PEG counterparts is another reason for the better performance of improved PEG in the error floor region.

It should be noted that parity-check matrices for irregular PEG codes with pa-

rameters identical to those used in this chapter are given in [58]. These codes, which perform better than the standard PEG codes reported here, are constructed by incorporating an extra search step in each stage of PEG algorithm to maximize the girth of the left-hand subgraph [15] of symbol nodes [88]. Similar steps can also be incorporated in our modified PEG algorithm at a cost of larger complexity. Even in their current form, our modified PEG codes outperforms the codes given in [58]. For example, we have tested the (1008, 504) code given in [58] at 2.8 dB. The BER and FER for this code are 8.2×10^{-8} and 2.4×10^{-6}, respectively, compared to 4.8×10^{-8} and 1.4×10^{-6}, for our codes. Also for the code in [58], the lowest weight of undetected errors is 13 (compared to 15 for our code). The code in [58] also has many trapping sets with small values of a and b. These include (1, 2), (2, 4), (3, 3) sets, and several trapping sets with $b = 1$ and $a \leq 14$. For our (1008, 504) code, the only observed trapping set with $b = 1$ is (15, 1). Other trapping sets, partially ordered based on b and a, respectively, are (10, 2), (13, 2), (4, 3), (7, 4), ...

6.4 Summary

In this chapter, we propose a very simple modification to the PEG algorithm which considerably enhances the performance at high SNR region without any degradation in the low SNR performance. The modification is based on creating a higher degree of connectivity in the Tanner graph of the code without sacrificing the girth distribution of the graph. This appears to improve both the minimum distance and the trapping sets of the code.

Chapter 7

Successive Relaxation for Decoding of LDPC Codes

The application of successive relaxation (SR) to the fixed-point problem associated with the iterative decoding of LDPC codes is studied in this chapter. We consider finite-length codes decoded by belief propagation (BP) and a well-known approximation of it, referred to as the min-sum (MS) algorithm, over a binary input AWGN channel. For both algorithms, we show that the application of SR in different domains results in different error correcting performance. In particular, the performances of SR in LLR and LR domains for BP and MS are compared, and it is shown that SR-MS-LLR has the best performance. Our results show that for both BP and MS, SR algorithms perform better than their successive substitution (SS) counterparts. The improvement in performance increases with the maximum number of iterations. For the tested codes, SR-MS-LLR outperforms standard BP by up to about 0.5 dB, offering an attractive solution in terms of performance/complexity tradeoff.

7.1 Introduction and Motivation

The difficulty in analyzing LDPC codes with finite block length under iterative message-passing algorithms originates from the fact that the Tanner graph representation of these codes includes many short cycles. These cycles, even for memoryless channels where the initial messages are independent, create dependencies among the messages after the first few iterations have passed.

The existence of cycles and dependencies among messages has some interesting consequences. One example is that while belief propagation (BP) (sum-product) converges to the optimal APP for bits on a cycle-free graph [48], on graphs with cycles, it only provides a suboptimal solution. There has been some research to improve the performance of BP for short codes [89–92]. In another direction, researchers have been busy devising algorithms that are less complex than BP [80, 81, 93]. The most well-known algorithm in this category for soft-decision decoding are the min-sum (MS) algorithm and its modifications. As mentioned in Chapter 2, min-sum is also important as it converges to the ML codeword on cycle-free graphs [48]. Only slightly inferior to sum-product in performance, min-sum has a much simpler check node operation compared to that of BP (refer to Chapter 2 for details). The variable node operation is the same for both algorithms. Considering that the check node operation is much more complex than the variable node operation in BP, the complexity of min-sum is substantially less than that of BP.

In their conventional form, iterative message-passing algorithms pass messages between the check nodes and the variable nodes and subsequently between the variable

nodes and the check nodes in each iteration. The message passing is performed in parallel or according to the flooding schedule in each direction, i.e., all the nodes on one side of the Tanner graph send their messages to the nodes on the other side simultaneously in discrete-time and in a synchronized fashion. Making a correspondence between one iteration and a time unit starting from time zero, the output messages of a variable node (check node) at a discrete-time t are only a function of the input message to that node at time t $(t-1)$ and also the initial message in the case of variable nodes. The output message vector $\vec{V}$ of variable nodes at time $t+1$ is then related to $\vec{V}$ at time t by the following equation

$$\vec{V}_{t+1} = h(\vec{V}_t, \vec{V}_0) \tag{7.1.1}$$

where $h(\cdot)$ represents the combination of variable and check node operations in one iteration, and $\vec{V}_0$ is the vector of initial messages. The number of elements of vectors $\vec{V}_0$, $\vec{V}_t$ and $\vec{V}_{t+1}$ is equal to the number of edges in the graph, which is the same as the number of nonzero elements in the parity-check matrix of the code. This framework is referred to as *successive substitution* (SS), pointing out the fact that it in fact represents the application of the well-known iterative numerical method of successive substitution to the following fixed-point problem of iterative decoding

$$\vec{V} = h(\vec{V}, \vec{V}_0). \tag{7.1.2}$$

Inspired by the dynamics of analog decoders, a different iterative message-passing

algorithm was introduced in [94]. This algorithm, which is based on the application of the well-known *successive relaxation* (SR) method to the fixed-point problem of (7.1.2), has memory and is represented by

$$\vec{V}_{t+1} = \vec{V}_t + \beta \left(h(\vec{V}_t, \vec{V}_0) - \vec{V}_t \right), \tag{7.1.3}$$

where β is called the "relaxation factor", and can be optimized for the best performance. For $\beta = 1$, SR reduces to SS. The optimal value of β, which is usually less than one [94], results in performance improvement of a few tenths of a dB over SS ($\beta = 1$) [94].

While it is known that SR outperforms SS in general, no comparative analysis of the performance of SR for different algorithms and in different domains is available. In this chapter, we study the application of SR to BP and MS in both LR and LLR domains. We use notations SR-BP-LR, SR-BP-LLR, SR-MS-LR and SR-MS-LLR to identify the four algorithms. The comparison among the four algorithms leaves us with three main conclusions:

1. For BP, SR in both LR and LLR domains performs similarly;

2. For MS, SR in the LLR domain outperforms SR in the LR domain;

3. SR-BP-LR and SR-BP-LLR perform in between SR-MS-LR and SR-MS-LLR.

Combining 1, 2 and 3, we thus conclude that among the four SR algorithms, SR-MS-LLR has the best performance. Moreover, the performance of SR-MS-LLR is superior to that of the conventional SS-BP by up to about 0.5 dB for the tested codes. This

makes SR-MS-LLR one of the best choices for iterative decoding of LDPC codes in terms of performance/complexity tradeoff.

The rest of this chapter is organized as follows: In the next section, we explain the system model, provide the variable and check node operations for BP and MS in LR and LLR domains, and describe successive relaxation in LR and LLR domains. Section 7.3 is used for simulation results, and Section 7.4 concludes this chapter.

7.2 Successive Relaxation in LR and LLR Domains

7.2.1 BP and MS in LR and LLR domains

We consider the application of a binary (n, k) LDPC code over a binary input additive white Gaussian noise (BIAWGN) channel. Consider a BIAWGN channel with input X that can take values ± 1, and has output $Y = X + N$, where N is a Gaussian with mean zero and variance $\sigma^2 = N_0/2$, independent of X. In the LLR domain, as mentioned in Chapter 2, messages are in the form of $\Lambda(p_{-1},p_1) = \ln(p_1/p_{-1})$, where p_{-1} and p_1 are the probabilities of a bit being -1 or 1, respectively. The initial messages in the LLR domain for BP and MS algorithms for each bit are $2Y/\sigma^2$ and Y, respectively. The following variable and check node operations describe BP and MS in the LLR domain for two input messages. For a larger number of inputs, a cascade of two-input modules can produce the output.

BP in LLR domain (Λ_0 is the initial message):

$$\text{VAR: } f(\Lambda_0, \Lambda_1, \Lambda_2) = \Lambda_0 + \Lambda_1 + \Lambda_2$$

$$\text{CHK: } g(\Lambda_1, \Lambda_2) = 2\tanh^{-1}\left(\tanh\left(\frac{\Lambda_1}{2}\right)\tanh\left(\frac{\Lambda_2}{2}\right)\right).$$

MS in LLR domain (Λ_0 is the initial message):

$$\text{VAR: } f(\Lambda_0, \Lambda_1, \Lambda_2) = \Lambda_0 + \Lambda_1 + \Lambda_2$$

$$\text{CHK: } g(\Lambda_1, \Lambda_2) = \text{sgn}(\Lambda_1\Lambda_2)\min(|\Lambda_1|, |\Lambda_2|),$$

where $\text{sgn}(x) = 1$, for $x > 0$, and $= -1$, otherwise.

In the LR domain, messages are in the form of $\lambda(p_{-1}, p_1) = p_1/p_{-1}$, and the following operations are performed in the variable nodes and check nodes for BP and MS.

BP in LR domain (λ_0 is the initial message):

$$\text{VAR: } f(\lambda_0, \lambda_1, \lambda_2) = \lambda_0\lambda_1\lambda_2$$

$$\text{CHK: } g(\lambda_1, \lambda_2) = \frac{1 + \lambda_1\lambda_2}{\lambda_1 + \lambda_2}.$$

MS in LR domain (λ_0 is the initial message):

$$\text{VAR: } f(\lambda_0, \lambda_1, \lambda_2) = \lambda_0\lambda_1\lambda_2$$

$$\text{CHK: } g(\lambda_1, \lambda_2) = \left(\min\left(\lambda_1^{\text{sgn}(\lambda_1-1)}, \lambda_2^{\text{sgn}(\lambda_2-1)}\right)\right)^{\text{sgn}(\lambda_1-1)\text{sgn}(\lambda_2-1)}.$$

It is worth noting that although the operations for MS in LR domain are min(.) and product, we still use the term min-sum for this algorithm to emphasize the fact

that this set of operations is in fact equivalent to that of MS in the LLR domain; it is just performed in a different domain (LR instead of LLR).

7.2.2 SR in LR and LLR domains

On cycle-free graphs, the conventional (successive substitution) version of BP (SS-BP) and MS (SS-MS) converges to the optimal maximum APP solution for bits and codewords, respectively. This is regardless of the domain in which the algorithms are implemented. For graphs with cycles, which always appear in the representation of good codes [17], however, SS-BP and SS-MS are suboptimal. The sub-optimality is attributed to the dependencies created among the messages passed along the edges of the graph throughout the iterative process, and can be interpreted as lower reliability of messages compared to the optimal scenario, i.e., the true reliability of messages is overestimated in the suboptimal algorithm [92].

The main idea behind SR is to weigh down the reliability of newly created messages by only partially moving in their direction. In SR, this is performed by moving slightly (a fraction of β) along the line connecting the current set of messages $\vec{V}_t$ and the newly created messages $h(\vec{V}_t, \vec{V}_0)$. This is shown in Figure 7.1(a) for two messages, i.e., where the message vectors have only two elements, in the LLR domain by full line. Unlike the case for SS, the implementation of SR in different domains however, results in different trajectories for iterative decoding. This is explained in Figures 7.1(a) and 7.1(b), where one iteration of SR is shown in LLR and LR domains, respectively. For each case, the corresponding trajectory in the other domain has also been shown.

The trajectories for SR in the LLR domain and the LR domain are shown by full and dashed lines, respectively. In the figures, we have assumed $\vec{V}_t' = \exp(\vec{V}_t)$ and thus $h'(\vec{V}_t', \vec{V}_0') = \exp\left[h(\vec{V}_t, \vec{V}_0)\right]$, where the exponential operation is defined component wise. The figures demonstrate that the application of SR in the LR domain (SR-LR) is equivalent to moving along a logarithmic curve in the LLR domain, while SR-LLR is equivalent to following an exponential trajectory in the LR domain. So, for the two SR algorithms, although the start and the end points of the two trajectories are the same, their directions are different. This results in different messages at the end of the SR iteration if $\beta < 1$. For $\beta = 1$, SR reduces to SS and the messages at the end of the iteration in both cases are the same and are equal to the end point of the trajectory.

While in this chapter, we make no attempt in finding the best domain in implementing the SR algorithm, we observe that the performance of SR for MS in the LLR domain is in general superior to its performance for MS-LR, BP-LR and BP-LLR.

7.3 Simulation Results

The application of asymptotic analysis tools such as density evolution to SR algorithms is not straight forward due to the existence of memory. We have thus performed extensive simulations on a number of LDPC codes to study these algorithms. We perform simulations on an optimized irregular (1268, 456) LDPC code [57] and a regular (504, 252) code [58] over a binary input AWGN channel. These are the same codes as those used in [94]. In our simulations, we stop iterations as soon as a code-

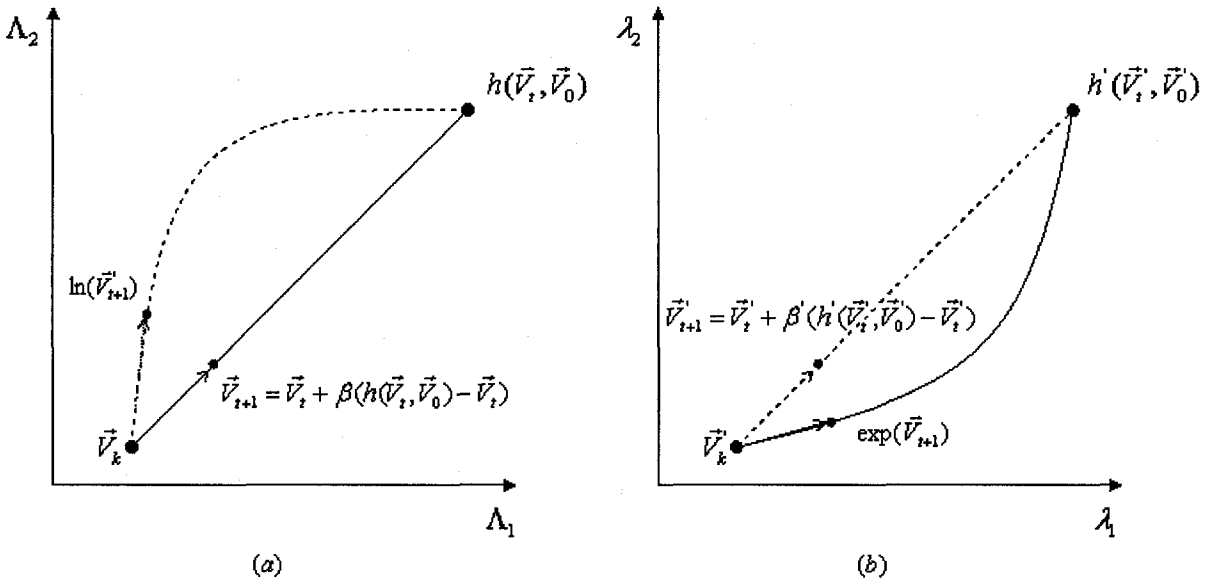

Figure 7.1: Illustration of trajectories of SR in the LLR and the LR domains.

word is detected or when a maximum number of iterations I_{max} is reached. We have investigated the performance of each code with SR for both BP and MS algorithms in both LR and LLR domains, i.e., under SR-BP-LR, SR-BP-LLR, SR-MS-LR and SR-MS-LLR. For all the reported results, we have 100 codeword errors per simulation point. Note that, although the SR-LLR is still symmetric [12], the SR-LR is not. Thus, in the simulations, instead of transmitting the all-zero codeword, we need to transmit randomly generated codewords. Two values of $I_{max} = 200$ and $I_{max} = 10000$ are considered here.

For a given maximum number of iterations I_{max}, and a given SNR, there exists a value of β that minimizes the error rate of the SR method. Our experiments show that the optimal β is rather independent of SNR and is only a function of the code, the decoding algorithm (BP or MS in LR or LLR domain), and I_{max}. To find the optimal β, we perform simulations for different codes and algorithms. Our results

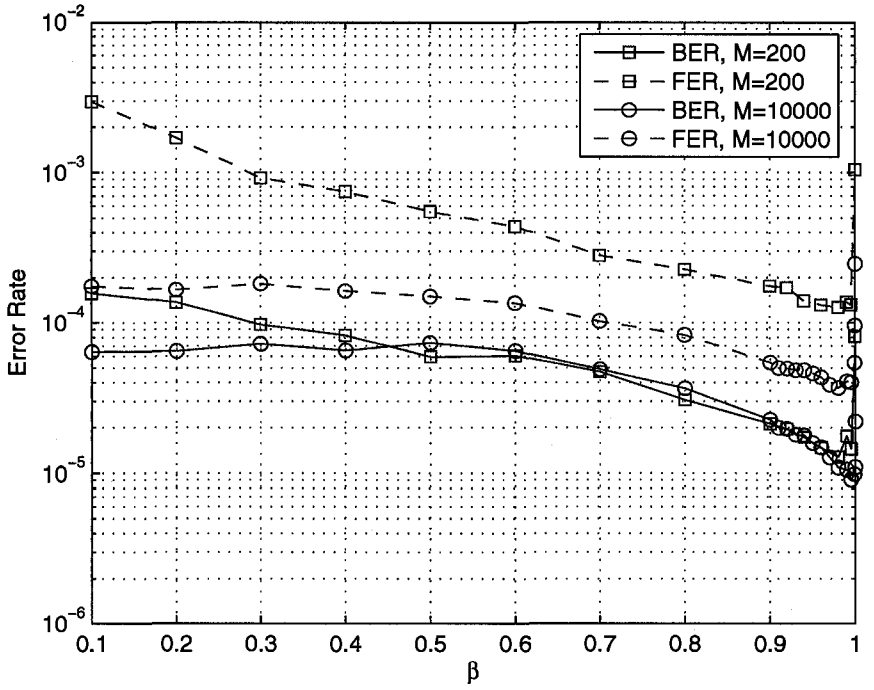

Figure 7.2: BER and FER for different values of β for the (504, 252) code decoded by SR-MS-LR.

show that while the performance of SR-MS-LR is highly sensitive to the choice of β, the performance of the other three algorithms is more robust against changes in β. Figures (7.2) and (7.3) show the BER and FER for different values of β for the (504, 252) code at 3 dB decoded by SR-MS-LR and SR-MS-LLR, respectively. In each figure, results for $I_{max} = 200$ and 10000 are shown. From the figures, the optimal β for this code for SR-MS-LR and SR-MS-LLR are found to be 0.98 and 0.5 for both $I_{max} = 200$ and $I_{max} = 10000$, respectively. The optimal values of β for other SR algorithms and for the (1008, 504) code are obtained in a similar way. Figure (7.2) also shows the sensitivity of the performance of SR-MS-LR to the choice of β.

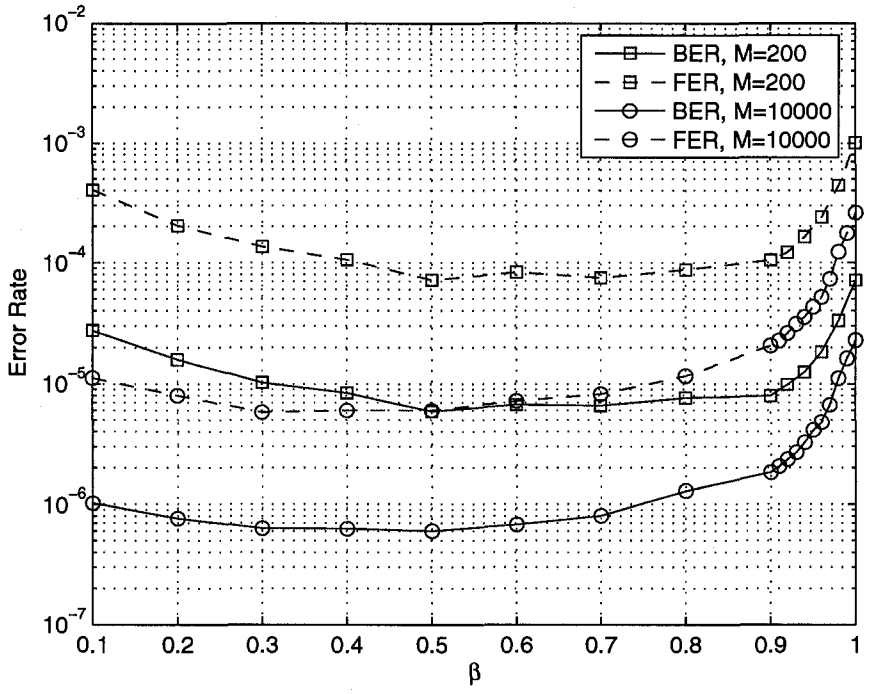

Figure 7.3: BER and FER for different values of β for the (504, 252) code decoded by SR-MS-LLR.

In Figures 7.4 and 7.5, we have reported the BER curves of (1268, 456) and (504, 252) codes, respectively, for SR-BP and SR-MS in LR and LLR domains with $I_{max} = 10000$. All the curves correspond to the optimal values of β. In each figure, we have also reported the BER of the conventional SS-BP and SS-MS as reference. As can be seen, all the SR algorithms outperform their SS counterparts. While SR-BP-LLR performs more or less the same as SR-BP-LR and results in better performance than SS-BP, SR-MS-LLR results in the best BER performance among all the algorithms. The performance of SR-MS-LR is superior to SS-MS and almost identical to SS-BP. This is the worst performance among SR algorithms. The fact that SR-MS-LLR has the best performance and the simplest variable and check node operations among the

four SR algorithms, and that it also performs considerably better than SS-MS and SS-BP, make SR-MS-LLR a prime candidate for the iterative decoding of LDPC codes. In general, the improvement in performance of SR over SS improves by increasing I_{max}. To see the effect of the maximum number of iterations on the performance improvement, in Figures 7.6 and 7.7, we have repeated the experiments with $I_{max} = 200$. As can be seen from the figures, although SR still performs better than SS, the improvement in performance is less compared to the case where $I_{max} = 10000$.

To analyze the difference between the performance of SR-MS-LLR and those of SS-MS and SS-BP, we examine and categorize the failures of all algorithms and track the changes with the increase in the maximum number of iteration I_{max}. In particular, for the (504, 252) code and the (1268, 456) code, we present the results at E_b/N_0=(3.0 dB, 3.5 dB), and (2.25 dB, 2.5 dB), in Tables 7.1 and 7.2, respectively. At each SNR, we first categorize the 100 decoding failures of SS-MS for $I_{max} = 200$. We then apply the 100 input vectors, corresponding to the 100 failures, to the other decoders and categorize the results. As can be seen, for $I_{max} = 200$, all the 100 failure instances of SS-MS for both codes and at all the SNR values are Random-like errors studied in Chapter 3. The results also show that as I_{max} increases to 1000 and 10,000, many of these Random-like errors converge to the right codeword, indicating the transient nature of those errors. For example, for the (504, 252) code at 3 dB, by increasing I_{max} from 200 to 1000, 51 of the cases that demonstrated Random-like behavior for $I_{max} = 200$, will converge to the right codeword. This number increases to 77 as I_{max} is further increased to 10,000.

The results of Tables 7.1 and 7.2 also indicate that the improvement in perfor-

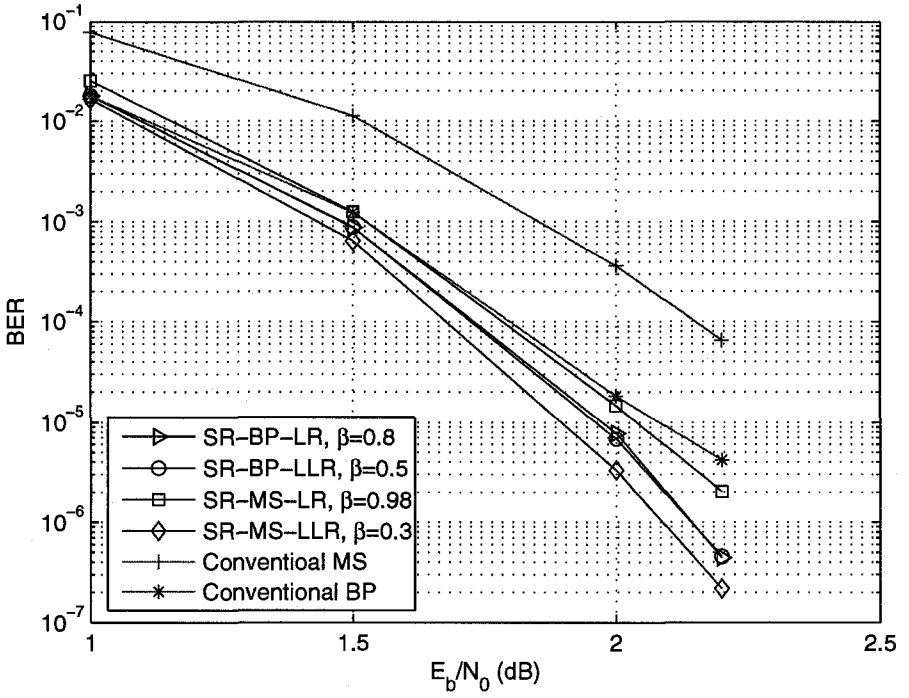

Figure 7.4: BER curves of (1268, 456) code with maximum number of iterations $I_{max} = 10000$.

mance of SR-MS-LLR in comparison with SS-MS and SS-BP is mainly due to the reduction in the Random-like behavior. This effect is in general more prevalent for larger values of I_{max}. For example, Table 7.1 shows that for the (504, 252) code, by applying the 100 input vectors, which caused Random-like errors for SS-MS with $I_{max} = 200$ at SNR=3.5 dB, to the SS-BP decoder with $I_{max} = 200$, only 30 cases remain in Random-like errors and there is also one case of Fixed-pattern failure. By applying the same 100 input vectors to SR-MS-LLR with $I_{max} = 200$, 95 of them converge to the right codeword and only 5 remain as Random-like errors (a reduction of 95% compared to SS-MS). By increasing I_{max} to 10,000 for SR-MS-LLR, all the 100 input vectors can be corrected (a reduction of 100% compared to SS-MS).

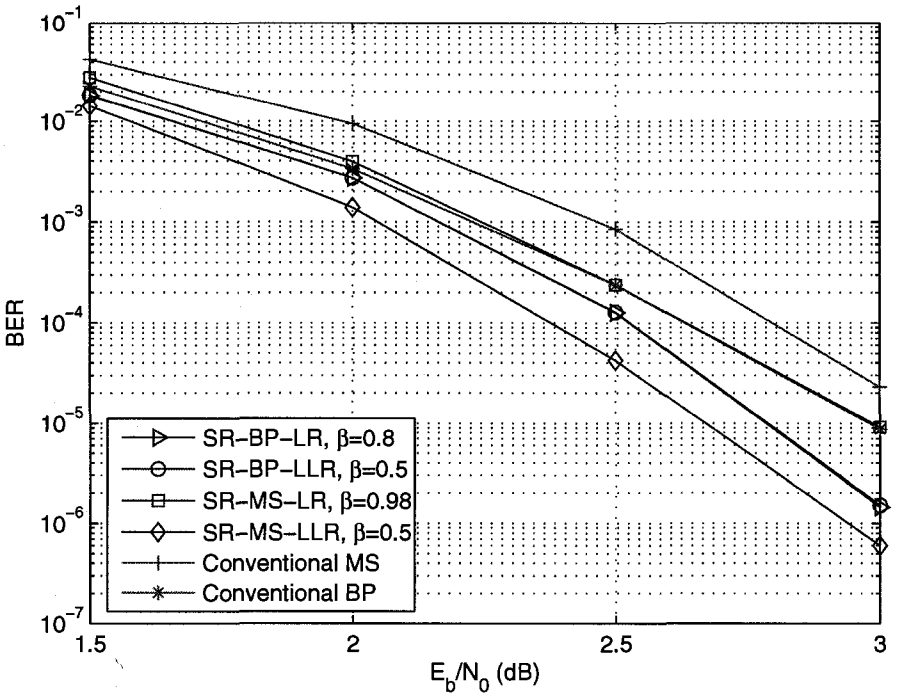

Figure 7.5: BER curves of (504, 252) code with maximum number of iterations $I_{max} = 10000$.

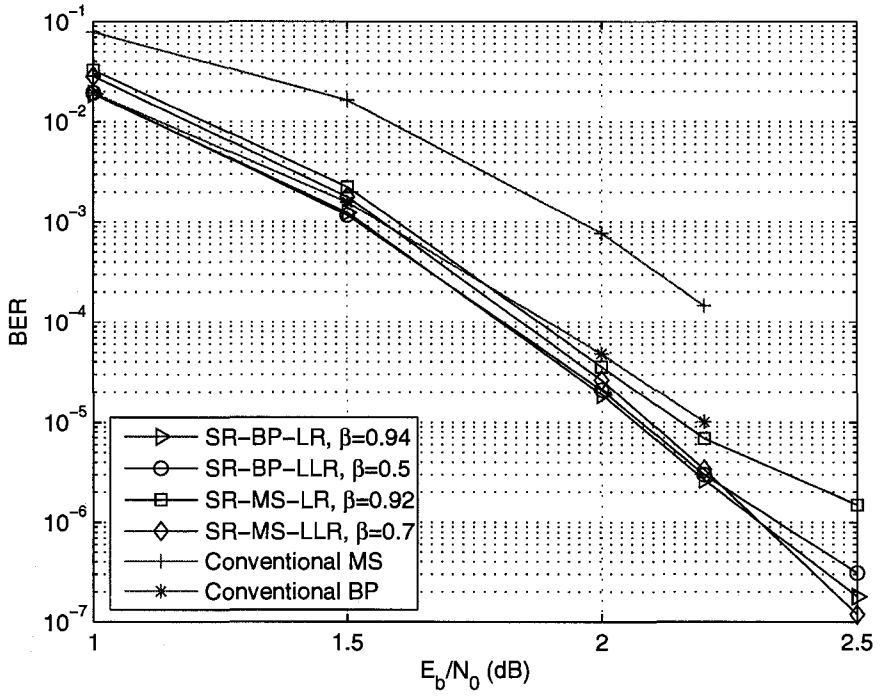

Figure 7.6: BER curves of (1268, 456) code with maximum number of iterations $I_{max} = 200$.

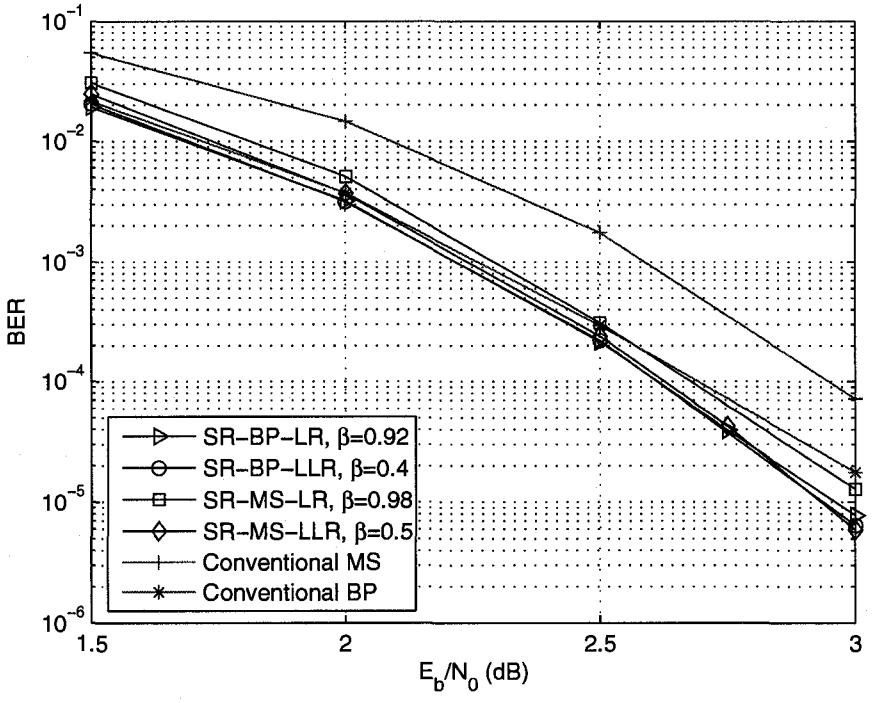

Figure 7.7: BER curves of (504, 252) code with maximum number of iterations $I_{max} = 200$.

Table 7.1: Breakdown of Decoder Failures Based on Their Type for the (504, 252) Code, Decoded by SS-MS, SS-BP and SR-MS-LLR, as a Function of the Maximum Number of Iteration I_{max} at Different SNR Values; "R", "O", and "F" Stand for "Random-like", "Oscillatory-pattern" and "Fixed-pattern", Respectively.

SNR	3dB									3.5dB								
I_{max}	200			1000			10000			200			1000			10000		
Error Type	R	O	F	R	O	F	R	O	F	R	O	F	R	O	F	R	O	F
SS-MS	100	0	0	49	0	0	23	0	0	100	0	0	30	0	0	9	0	0
SS-BP	14	0	0	13	0	0	7	0	0	30	0	1	17	0	1	9	0	1
SR-MS-LLR	8	0	0	1	0	0	0	0	0	5	0	0	2	0	0	0	0	0

Table 7.2: Breakdown of Decoder Failures Based on Their Type for the (1268, 456) Code, Decoded by SS-MS, SS-BP and SR-MS-LLR, as a Function of the Maximum Number of Iteration I_{max} at Different SNR Values; "R", "O", and "F" Stand for "Random-like", "Oscillatory-pattern" and "Fixed-pattern", Respectively.

SNR	2.25dB									2.5dB								
I_{max}	200			1000			10000			200			1000			10000		
Error Type	R	O	F	R	O	F	R	O	F	R	O	F	R	O	F	R	O	F
SS-MS	100	0	0	49	0	0	30	0	0	100	0	0	42	0	0	19	0	0
SS-BP	14	0	0	13	0	0	7	0	0	30	0	1	17	0	1	9	0	1
SR-MS-LLR	2	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0

7.4 Summary

The application of successive relaxation (SR) to belief propagation (BP) and min-sum (MS) algorithms in LR and LLR domains is studied in this chapter. Our results show that among the four variants of SR, SR-MS-LLR performs the best (especially for larger maximum number of iterations and high SNR values) and has the lowest complexity. With a large maximum number of iterations, this algorithm also provides substantial improvement over standard BP and MS algorithms for codes with short block lengths. It is therefore an attractive candidate for implementation. Our study shows that the superior performance of SR-MS-LLR compared to standard BP and MS algorithms is mainly due to the reduction of Random-like failures of the decoder.

While the performance of SR-MS-LLR is impressive, we have no reason to believe that the application of SR to the MS algorithm in the LLR domain provides the best performance in the category of SR message-passing algorithms. Investigating the performance of SR with other algorithms and in other domains is an interesting topic for future research.

Chapter 8

Conclusions and Future work

8.1 Summary of the Thesis

In this thesis, we addressed several issues of finite-length LDPC codes: performance analysis, code construction and decoding.

- In Chapter 3, we studied the structure of trapping sets and the relationship between the initial error patterns and the trapping sets for hard-decision iterative decoding algorithms. We categorized the trapping sets for hard-decision iterative decoding algorithms. We derived simple lower and upper bounds on the FER for hard-decision iterative decoding algorithms, which can provide good estimates of the FER for sufficiently small crossover probabilities. We proposed a general estimation method based only on the information of the smallest error patterns the decoder fails to correct (set E_J). We estimated the contributions of error patterns with larger weights to the total error rates based on the set E_J and obtained very accurate estimates for both the FER and the BER for

the whole range of crossover probabilities of practical interest. This is while the computational complexity of our estimates can be much less than that of the conventional Monte Carlo simulation. We showed that this method can be applied to both regular and irregular codes and to a variety of hard-decision iterative decoding algorithms.

- To obtain the values of J and $|E_J|$, direct enumeration was used in Chapter 3. The computational complexity of the direct enumeration increases as n^J, which limits its application to codes with small block lengths and J. In Chapter 4, we proposed an algorithm to approximate the values of J and $|E_J|$ using cycle enumeration. This reduced the computational complexity by several orders of magnitude without any degradation in the accuracy of the error rate estimates from a practical point of view. We also modified the estimation equations in Chapter 3 to include further steps for improving the proposed estimates and for identifying their convergence. With this proposed algorithm, we showed that the estimation method proposed in Chapter 3 can be applied to almost any LDPC code of practical interest.

- In Chapter 5, we proposed a combinatorial approach to estimate the error rate performance of LDPC codes decoded by (quantized) soft-decision iterative decoding algorithms. This method can be seen as a non-trivial extension of the method proposed in Chapters 3 and 4. The method is based on efficient enumeration of input vectors with small distances to a reference vector whose elements are selected to be the most reliable values from the input alphabet. Several

techniques, including modified cycle enumeration, were employed to reduce the complexity of the enumeration. The error rate estimate is derived by testing the input vectors of small distances and estimating contribution of larger distance vectors. We demonstrated by a number of examples that the proposed method provides accurate estimates of error rate with computational complexity much lower than that of Monte Carlo simulations, especially at the error floor region.

- In Chapter 6, a simple modification to the well-known PEG construction was proposed. In this modification, we improved both the minimum distance and the trapping sets of the code's graph by increasing the connectivities of cycles to the rest of the graph. This resulted in considerable improvement in the performance in high SNR region without sacrificing low SNR performance.

- Chapter 7 was devoted to the application of successive relaxation (SR) to the iterative decoding of LDPC codes. We applied SR to both BP and MS in different domains (LR and LLR). We showed by simulations that SR algorithms outperform their SS counterparts and the best performing and the least complex algorithm is SR-MS-LLR. The improvement in performance is shown to be an increasing function of the maximum number of iterations.

8.2 Possible Topics for Future Research

- In Chapter 3, we proposed a technique to estimate the performance of LDPC codes decoded by hard-decision iterative decoding algorithms. The key point

of this technique is finding values of J and $|E_J|$. Cycle enumeration algorithm was proposed in Chapter 4 to estimate the values of J and $|E_J|$ with low computational complexity for many cases. There are, however, cases where the computational complexity of the cycle enumeration algorithm is still very high. One such example was shown in Chapter 4 (Example 4.4.1). To obtain the real values of J and $|E_J|$, the algorithm has to perform more rounds. Also, for high rates codes, the computational complexity of the algorithm becomes very high if we go beyond two rounds due to the high density of the graphs. Thus, it would be interesting to improve the proposed algorithm to handle these cases with lower computational complexity. For cases where the structure of trapping sets are known, one could incorporate this information into the search algorithm to further reduce the computational complexity. It would be also nice to obtain J and $|E_J|$ using analytical approach.

- In Chapter 5, we proposed a technique to estimate the performance of LDPC codes decoded by (quantized) soft-decision iterative decoding algorithms. This technique is limited to codes with block length upto a thousand bits. For codes with larger block lengths, the computational complexity of this technique would be too high. It is of great practical interest to further reduce the computational complexity such that this technique can be applied to a wide range of practical codes. It would be nice to find the BER estimate as well.

- In Chapter 6, we showed that by increasing the connectivity of cycles to the rest of the graph, we can improve the trapping sets. It's also interesting to

find other factors related to the trapping sets such that more constraints can be incorporated into the PEG algorithm to further improve the performance of the code. Following our work, there have been some works on this, see, e.g. [95]. The proposed modified PEG algorithm works for irregular graphs. How to improve trapping sets for regular graphs is also of practical interest.

- In Chapter 7, we showed that applying SR to different domains results in different improvement in performance. While the performance of SR-MS-LLR is impressive, we have no reason to believe that the application of SR to the MS algorithm in the LLR domain provides the best performance in the category of SR message-passing algorithms. We made no attempt to find the optimal domain in this thesis. Investigating the dynamics of SR in other domains and finding the optimal domain is very interesting. The improvement obtained by SR over SS can be related to the improvement in the trapping sets, especially in the error floor region. It would be interesting to investigate this relationship and obtain more insights into the dynamics of SR algorithms.

Bibliography

[1] R. G. Gallager, *Low density parity check codes.* Sc.D. thesis, Massachusetts Institute of Technology, 1960.

[2] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21-28, Jan. 1962.

[3] R. G. Gallager, *Low-Density Parity-Check Codes.* No. 21 in Research Monographs Series, Cambridge, MA: M.I.T. Press, 1963.

[4] R. M. Tanner, "A recursive approach to low-complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533-547, Sept. 1981.

[5] G. A. Margulis, "Explicit construction of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71-78, 1982.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes (1)," in *Proc. IEEE ICC* 1993, vol. 2, (Geneva, Switzerland), pp. 1064-1070, May 1993.

[7] D. J. C. MacKay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics Letter*, vol. 45, no. 2, pp. 399-431, March 1997.

[8] "Special Issue on Codes on Graphs and Iterative Algorithms," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, Feb. 2001.

[9] "The Turbo Principle: From Theory to Practice," *IEEE J. Select. Areas Comm.*, vol. 19 no. 5, May 2001.

[10] "The Turbo Principle: From Theory to Practice II," *IEEE J. Select. Areas Comm.*, vol. 19 no. 9, Sept 2001.

[11] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 599-618, Feb. 2001.

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

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

[14] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. on Communications, Control and Computing*, 2003.

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

[16] A. Ashikhmin, G. Kramer, and S. Ten Berink, "Extrinsic Information Transfer Functions: model and erasure channel properties," *IEEE Trans. Inform. Theory*, vol. 50, no. 11, pp. 2657-2673, Nov. 2004

[17] T. Etzion, A. Trachtenbeg, and A. Vardy, "Which codes have cycle-free Tanner graphs?" *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 2173-2181, Sept. 1999.

[18] H. Xiao and A. H. Banihashemi, "Performance of LDPC codes on binary symmetric channels with small crossover probabilities," in *Proc. 23rd Biennial Symposium on Communications*, pp. 1-4, Queen's University, Kingston, Ontario, Canada, May 29-June 1, 2006.

[19] H. Xiao and A. H. Banihashemi, "Estimation of bit and frame error rates of LDPC codes on binary symmetric channels," in *Proc. 10th Canadian Workshop on Inform. Theory (CWIT)*, Edmonton, Alberta, June 6 - 8, 2007.

[20] H. Xiao and A. H. Banihashemi, "Estimation of bit and frame error rates of low-density parity-check codes on binary symmetric channels," *IEEE Trans. Comm.*, vol. 55, pp. 2234-2239, Dec. 2007.

[21] H. Xiao and A. H. Banihashemi, "Error rate estimation of finite-length low-density parity-check codes on binary symmetric channels using cycle enumeration," in *Proc. IEEE International Symp. on Inform. Theory (ISIT)*, Nice, France, 2007.

[22] H. Xiao and A. H. Banihashemi, "Error rates estimation of low-density parity-check codes on binary symmetric channels using cycle enumeration," submitted to *IEEE Trans. Comm.*, Feb. 2007, revised, Jan. 2008.

[23] H. Xiao and A. H. Banihashemi, "Error rate estimation of LDPC codes decoded by soft-decision iterative algorithms," accepted for presentation in *IEEE International Symp. on Inform. Theory (ISIT)*, Toronto, Canada, 2008.

[24] H. Xiao and A. H. Banihashemi, "Error rate estimation of low-density parity-check codes decoded by soft-decision iterative algorithms," submitted to *IEEE JSAC*. April 2008.

[25] H. Xiao and A. H. Banihashemi, "Improved Progressive-Edge-Growth (PEG) Construction of Irregular LDPC Codes," in *Proc. IEEE Globecom'04*, pp. 489-492, Dallas, Texas, Nov. 29 - Dec. 3, 2004.

[26] H. Xiao and A. H. Banihashemi, "Improved Progressive-Edge-Growth (PEG) Construction of Irregular LDPC Codes," *IEEE Comm. Lett.*, vol. 8, no. 12, pp. 715 - 717, Dec. 2004.

[27] H. Xiao, S. Tolouei, and A. H. Banihashemi, "Successive Relaxation for Decoding of LDPC Codes", accepted for presentation in *24rd Biennial Symposium on Communications*, Queen's University, Kingston, Ontario, Canada, 2008.

[28] H. Xiao, S. Tolouei, and A. H. Banihashemi, "Successive Relaxation for Decoding of LDPC Codes", submitted to *IEEE Trans. Comm.*, April 2008.

[29] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proc. 29th Annu. ACM Symp. Theory of Computing*, 1997, pp. 150-159.

[30] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proc. 30th Annu. ACM Symp. Theory of Computing*, 1998, pp. 249-258.

[31] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, Feb. 2001.

[32] D. J. C. MacKay, S. Wilson, and M. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 1449-1454, Oct. 1999.

[33] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, Feb. 2001.

[34] M. Ardakani and F. R. Kschischang, "Gear-shift decoding," in *Proc. 21st Biennial Symp. Comm.*, pp. 86-90, Queen's University, Kingston, Ontario, Canada, June 2002.

[35] M. Ardakani and F. R. Kschischang, "On the optimality of Gallager's decoding algorithm B," in *Proc. Int'l Symp. Turbo Codes and Related Topics*, (Brest, France), pp. 165-168, Sept. 2003.

[36] P. Zarrinkhat, A. H. Banihashemi, and H. Xiao, "Time-invariant and switch-type hybrid iterative decoding of low-density parity-check codes," *Ann. Telecommun.*, vol. 60, no. 1-2, pp. 103-131, Jan.-Feb. 2005

[37] P. Zarrinkhat and A. H. Banihashemi, "Hybrid hard-decision iterative decoding of regular low-density parity-check codes," *IEEE Comm. Lett.*, vol. 8, pp. 250-252, Apr. 2004

[38] P. Zarrinkhat and A. H. Banihashemi, "Threshold values and convergence properties of majority-based algorithms for decoding regular low-density parity-check codes," *IEEE Trans. Comm.*, vol. 52, pp. 2087-2097, Dec. 2004.

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

[40] R. J. McEliece, D. J. C. MacKay and J. -F. Cheng, "Turbo decoding as an instance of Pearl's 'belief propagation' algorithm," *IEEE J. Select. Areas Comm.*, vol. 16, pp. 140-152, Feb. 1998.

[41] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Comm.*, vol. 16, pp. 219-230, Feb. 1998.

[42] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, 2rd ed. San Francisco, CA: Kaufmann, 1988.

[43] R. E. Neapolitan, *Probabilistic Reasoning in Expert Systems*, New York: Wiley, 1990.

[44] R. W. Chang and J. C. Hancock, "On receiver structures for channels having memory," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 463-468, Oct. 1966.

[45] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Ann. Math. Stat.*, vol. 37, no. 6, pp. 1554-1563, Dec. 1966.

[46] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284-287, Mar. 1974.

[47] N. Wiberg, H. -A. Loeliger and R. Kotter, "Codes and iterative decoding on general graphs," *Euro. Trans. Telecommun.*, vol. 6, no. 5, pp. 513-525, Sept. - Oct. 1995.

[48] N. Wiberg, *Codes and decoding on general graphs.* PhD thesis, Linkoping University, Sweden, 1996.

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

[50] Y. Weiss and W. T. Freeman, "On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 736-744, Feb. 2001.

[51] C. A. Cole, S. G. Wilson, E. K. Hall, and T. R. Giallorenzi, "A general method for finding low error rates of LDPC codes," submitted to *IEEE Trans. Inform. Theory*, May 2006.

[52] S. -Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 657-670, Feb. 2001.

[53] F. Lehmann and G. M. Maggio, "An approximation analytical model of the message passing decoder of LDPC codes," in *Proc. Int'l Symp. Inform. Theory* '02, p. 31, Lausanne, Switzerland, June-July 2002.

[54] S. Ten Brink, "Iterative decoding trajectories of parallel concatenated code," In *Proc. 3rd IEEE/ITG Conf. Source and Channel Coding*, pp. 75-80, Munich, Germany, Jan. 2000.

[55] H. E. Gamal, *On the theory and applications of space-time and graph based codes.* PhD thesis, University of Maryland, College Park, 2000.

[56] S. Ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Comm.*, vol. 49, pp. 1727-1737, Oct. 2001

[57] Y. Mao and A. H. Banihashemi, "A heuristic Search for good low-density parity-check codes using girth distribution," in *Proc. IEEE ICC '01*, Helsinki, Finland, vol. 1, pp. 41-44, June 2001.

[58] D. J. C. MacKay, http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[59] S. Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Comm. Lett.,* vol 5, pp. 58-60, Feb. 2001.

[60] M. Chiani and A. Ventura, "Design of efficiently-encodable moderate-length high-rate irregular LDPC codes," in *Proc. IEEE Globecom Conf. '01*, pp. 990-994, Nov. 2001.

[61] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate LDPC codes," *IEEE Trans. Comm.*, vol. 52, pp. 564-571, Apr. 2004.

[62] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol 47, no. 2, Feb. 2001.

[63] J. Rosenthal and P. O. Vontobel, "Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis," in *Proc. 38th Annual Allerton conference on Communication, Control and Computing*, pp. 248-257.

[64] D. J. C. MacKay and M. Postol, "Weaknesses of margulis and ramanujan-margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.

[65] J. Ha, J. Kim, D. Klinc, and S. W. McLaughlin, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Trans. Inform. Theory*, pp. 728-738, vol. 52, no. 2, Feb. 2006.

[66] T. Morita, M. Ohta, and T. Sugawara, "Efficiency of short LDPC codes combined with long Reed-Solomon codes for magnetic recording channels," *IEEE Trans. Magnetics*, pp. 3078-3080, vol. 40, no. 4, July 2004.

[67] S. Sankaranarayanan, I. B. Djordjevic, and B. Vasic, "Iterative decodable codes on m flats for WDM high-speed long-haul transmission," *J. Lightw. Technol.*, vol. 23, no. 11, pp. 3696-3701, Nov. 2005.

[68] Q. H. Spencer, "Short-block LDPC codes for a low-frequency power-line communications system," in *Proc. 2005 Int. Symp. Power Line Comm.and Its Appl.*, April 2005, pp. 95-99.

[69] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes," *IEEE Trans. Inform. Theory*, pp. 1570-1579, June 2002.

[70] B. Xia and W. E. Ryan, "Importance Sampling for Tanner Trees", *IEEE Trans. Inform. Theory*, Vol. 51, No. 6, pp. 2183-2189, June 2005.

[71] B. Xia and W. E. Ryan, "Estimating LDPC codeword error rates via importance sampling," in *Proc. IEEE International Symp. on Inform. Theory (ISIT)*, 2004.

[72] Y. Zhang and W. E. Ryan, "Structured eIRA codes: performance analysis and construction" *IEEE Trans. Comm.*, vol. 55, pp. 837-844, May, 2007.

[73] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," in *Proc. IEEE Inter. Conf. on Commun. (ICC)*, Istanbul, Turkey, June 11-15, 2006.

[74] S. Sankaranarayanan, S. K. Chilappagari, R. Radhakrishnan, and B. Vasic, "Failures of the Gallager B decoder: analysis and applications," *Inform. Theory and Applications Workshop*, University of California, San Diego, Feb. 6-10, 2006.

[75] L. Bazzi, T. J. Richardson, and R. L. Urbanke, "Exact thresholds and optimal codes for the binary-symmetric channel and Gallager's decoding algorithm A," *IEEE Trans. Inform. Theory*, vol. 50, pp. 2010-2021, Sept. 2004.

[76] M. Stepanov and M. Chertkov, "The error-floor of LDPC codes in the Laplacian channel," in *Proc. 43rd Allerton Conference on Commun., Control and Computing*, Sept. 2005, Monticello, IL, USA.

[77] M. Stephanov, V. Chertkov, M. Chertkov and B. Vasic, "Diagnosis of weakness in modern error correction codes: a physics approach," *Phy. Rev. Lett.* 95, 228701 (2005).

[78] M. Stepanov and M. Chertkov, "Instanton analysis of low-density parity-check codes in the error-floor regime," in *Proc. IEEE International Symp. on Inform. Theory (ISIT)*, Seattle, Washington, USA, July, 2006

[79] M. Chertkov and M. Stepanov, "Pseudo-codeword landscape," in *Proc. IEEE International Symp. on Inform. Theory (ISIT)*, 2007

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

[81] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding LDPC codes," *IEEE Trans. Comm.*, vol. 53, no. 4, pp. 549-554, April 2005.

[82] L. Ping and W. K. leung, "Decoding low density parity check codes with finite quantization bits," *IEEE Comm. letters*, vol. 4, no. 2, pp. 62-64, Feb. 2000.

[83] J. Thorpe, Low-complexity approximation to belief propagation for LDPC codes. [Online]: http://www.ee.caltech.edu/jeremy/research/papers/research.html

[84] J. K-S. Lee and J. Thorpe, "Memory-efficient decoding of LDPC codes," in *Proc. IEEE International Symp. on Inform. Theory (ISIT)*, Adelaide, Australia, 2005.

[85] Z. Zhang, L. Dolecek, M. Wainwright, V. Anantharam, and B. Nikolic, "Quantization effects in low-density parity-check decoders," in *Proc. IEEE international Conf. on Comm. (ICC)*, Glasgow, UK, June 2007.

[86] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," in *Proc. IEEE ICC '03*, Anchorage, Alaska, USA, vol. 5, pp. 3125-3129, May 2003.

[87] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Comm.*, vol. 52, pp. 1242-1247, Aug. 2004.

[88] X. -Y. Hu, private communication, March 2004.

[89] M. Isaka, M. P. C. Fossorier, and H. Imai, "On the suboptimality of iterative decoding for turbo-like and LDPC codes with cycles in their graph representation," *IEEE Trans. Comm.*, vol. 52, pp. 845-854, May 2004.

[90] Y. Mao and A. H. Banihashemi, "Decoding low-density parity-check codes with probabilistic schedule," *IEEE Comm. Letters*, vol. 5, no. 10, pp. 414-416, Oct. 2001.

[91] H. Xiao and A. H. Banihashemi, "Graph-based message-passing schedules for decoding LDPC codes," *IEEE Trans. Comm.*, vol. 42, no. 12, pp. 2098-2105, Dec. 2004.

[92] M. R. Yazdani, S. Hemati, and A. H. Banihashemi, "Improving belief propagation on graphs with cycles," *IEEE Comm. Letters*, vol. 8, no. 1, pp. 57-59, Jan. 2004.

[93] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity-check codes," *IEEE Trans. Comm.*, vol. 50, no. 3, pp. 406-414, March 2002.

[94] S. Hemati and A. H. Banihashemi, "Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes," *IEEE Trans. Comm.*, vol. 54, no. 1, pp. 61-70, Jan. 2006.

[95] D. Vukobratovis and V. Senk, "Generalized ACE constrained progressive edge-growth LDPC code design", *IEEE Comm. letters*, vol. 12, no. 1, pp. 32-34, Jan. 2008.