IMPROVING THE ERROR FLOOR PERFORMANCE OF LDPC CODES

by

Abdullah Sarıduman

B.S., Electrical and Electronics Engineering, TOBB Economy and Technology University, 2010

M.S., Electrical and Electronics Enginnering, Boğaziçi University, 2013

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2020

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to my co-advisors, Assoc. Prof. Ali Emre Pusane and Prof. Zeki Caner Taşkın. Without their guidance and persistent help, this dissertation would not have been possible. One simply could not wish for better and friendlier supervisors.

Besides my co-advisors, I would like to thank the rest of my thesis committee: Prof. Ayşın Baytan Ertüzün, Assoc. Prof. Ertuğrul Başar, Prof. Emin Anarım, and Asst. Prof Tunçer Baykaş for their encouragement and insightful comments.

I am also thankful to my friends from the wireless communication laboratory. Their support and care helped me overcome setbacks and stay focused on my graduate study. I greatly value their friendship and I deeply appreciate their belief in me.

Most importantly, none of this would have been possible without the love and patience of my family: my mom Ayten Sarıduman, my father Mustafa Sarıduman, my brother Alper Sarıduman, and my sister Şeyma Sarıduman. I owe them everything and wish I could show them just how much I love and appreciate them.

ABSTRACT

IMPROVING THE ERROR FLOOR PERFORMANCE OF LDPC CODES

Low-density parity-check codes (LDPC) have become one of the most popular error-correcting code families in recent years due to their high error correction performance and easy implementation. They became one of the biggest candidates to become a standard in the next-generation wireless communication systems. In particular, quasi-cyclic (QC)-LDPC codes have been chosen as the standard codes for 5G mobile broadband. However, at high signal-to-noise ratio values, the error-correcting performance of LDPC codes decreases due to harmful structures called trapping sets. Designing an LDPC code with few or no harmful structures is one of the popular topics in recent years. In this thesis, firstly, an adaptive linear decoder is designed that can decode LDPC codes with high error-correcting performance. Then, simulated annealing algorithms are proposed to reduce the number of cycles and trapping sets. Finally, new QC-LDPC codes have been designed with the proposed algorithm. In terms of trapping sets distribution, the best short-length QC-LDPC code constructions in the literature have been achieved.

ÖZET

LDPC KODLARININ HATA ZEMİNİ PERFORMANSINI GELİŞTİRMEK

Düşük-yoğunluklu eşlik-denetim kodları (LDPC) yüksek hata düzeltme performansları ve uygulamasının kolay olması nedeniyle son yıllarda en çok kullanılan hata düzelten kod ailelerinden biri olmuştur. Yeni nesil kablosuz iletişim sistemlerinde standart olabilecek en büyük adaylardan biri haline geldi. Özellikle, QC-LDPC kodları 5G mobil geniş bant için standart kodlar olarak seçilmiştir. Bununla birlikte, yüksek sinyal-gürültü oranı değerlerinde, tuzak kümeleri adı verilen zararlı yapılardan dolayı hata düzeltme performansı azalmaktadır. Bu zararlı yapıların olmadığı veya az sayıda olduğu bir LDPC kod tasarlamak son yıllardaki popüler konulardan biridir. Bu tezde ilk önce LDPC kodlarını daha iyi çözebilecek bir adaptive linear decoder tasarlanmıştır. Daha sonra, çevrim ve tuzak kümesi sayısını azaltan benzetimli tavlama algoritmaları önerilmiştir. Son olarak ise QC-LDPC kodlar önerilen algoritma ile tasarlanmıştır. Trapping sets dağılımı bakımından literatürdeki en iyi kısa uzunluklu QC-LDPC kodları üretilmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i
ABSTRACT ii
ÖZET iii
LIST OF FIGURES vi
LIST OF TABLES ix
LIST OF SYMBOLS x
LIST OF ACRONYMS/ABBREVIATIONS xii
1. INTRODUCTION 1
1.1. Overview 1
1.2. LDPC Codes 2
1.3. Contribution of The Thesis 6
1.4. Thesis Outline 7
2. LDPC Codes 9
2.1. Introduction of the LDPC Codes 9
2.2. Decoding of the LDPC Codes 14
2.2.1. Message-passing Decoders 14
2.2.2. Linear Programming Decoders 17
2.3. Failures of LDPC Codes 21
2.4. TS Enumeration of LDPC Codes 26
2.5. Importance Sampling 29
3. Adaptive Linear Decoding Algorithms 36
3.1. Integer Adaptive Linear Decoder (IALP) 36
3.2. Heuristic Adaptive Linear Decoder (HALP) 39
3.2.1. Cycles generated with check equations only from 2F class 43
3.2.2. Cycles with one check equations from the 3F class 43
3.2.3. Cycles with two check node from the 3F class 45
4. Design of LDPC Codes 51
4.1. Progressive Edge Growth Algorithm 51

4.2. Randomized Progressive Edge Growth Algorithm 52
4.3. PEG Based Simulated Annealing Algorithm 54
4.4. RandPEG-noTS53 Algorithm 65
4.5. RandPEG-noTS53 Based Simulated Annealing Algorithm 67
5. On the Construction of QC-LDPC Codes 73
5.1. Simulated Annealing Algorithm 75
6. CONCLUSIONS 88
6.0.1. Future Works 89
REFERENCES 91
APPENDIX A: Maximum Likelihood Ratio 99
APPENDIX B: PROOF THEOREM 4.1 101
APPENDIX C: PROOF THEOREM 4.2 and 4.3 102

LIST OF FIGURES

Figure 1.1. Basic Elements of Channel Coding 2

Figure 2.1. The Tanner Graph of an (8,3) LDPC Code. 10

Figure 2.2. An Encoding Circuit for the Quasi-Cyclic Code. 13

Figure 2.3. A local Polytope 19

Figure 2.4. A Cycle Example 22

Figure 2.5. The Tanner Graph of a (3,1) TS 23

Figure 2.6. The Tanner Graph of a (5,3) ETS 24

Figure 2.7. The Tanner graph of an absorbing set. 25

Figure 2.8. The Tanner graph of a fully absorbing set. 25

Figure 2.9. ETSs Example 27

Figure 2.10. Expansion of an ETS S of Size a to ETSs of Size $a+1$. 28

Figure 2.11. Expansion of Input ETSs to ETSs of Size up to t. 28

Figure 2.12. Example of Error Floor Estimation 31

Figure 2.13. Example of Biasing Densities 33

Figure 2.14. Example of Importance Sampling 34

Figure 3.1. IALP Decoder. 39

Figure 3.2. Representation of Check Nodes with Auxiliary Variables 40

Figure 3.3. Example of a Cycle Contains Only Check Nodes From 2F Class . 44

Figure 3.4. Example of a Cycle Contains One Check Node From the 3F Class 44

Figure 3.5. HALP Decoder. 48

Figure 3.6. Time Performance 49

Figure 3.7. Frame Error Rate Performance 50

Figure 4.1. A Subgraph of a Variable Node v_j: Depth-l Tree 52

Figure 4.2. Progressive Edge Growth (PEG) Algorithm. 53

Figure 4.3. Flowchart of a Simulated Annealing Algorithm 56

Figure 4.4. PEG Based Simulated Annealing Algorithm 58

Figure 4.5. SA-Accept Everything 59

Figure 4.6. SA-Temperature Change 60

Figure 4.7. SA-Adaptive Step Size 61

Figure 4.8. SA-Objective Value 61

Figure 4.9. BER Performance of (155,64) LDPC Codes 64

Figure 4.10. The Tanner Graph of a (5,3) ETS 66

Figure 4.11. The Tanner Graph of a (6,4) ETS 67

Figure 4.12. RandPEG-noTS53 Based Simulated Annealing Algorithm 70

Figure 4.13. BER Performance of LDPC Codes 72

Figure 5.1. Objective Value Change 74

Figure 5.2. Hill-Climbing Algorithm 76

Figure 5.3. Simulated Annealing Algorithm for QC-LDPC Codes 78

Figure 5.4. Improvement of The Objective Value 79

Figure 5.5. FER Performance of the Constructed QC-LDPC Codes. 82

Figure 5.6. FER Performance of the Constructed QC-LDPC Codes. 84

Figure 5.7. BER Performance of the Constructed QC-LDPC Codes. 86

LIST OF TABLES

Table 2.1. The Code-words List 12

Table 2.2. Elementary Trapping Sets of Tanner (155,64) Code 29

Table 2.3. The Contribution of Elementary Trapping Sets to Error Floor . . . 35

Table 4.1. Girth Number 62

Table 4.2. Elementary Trapping Sets of (155,64) LDPC Code Obtained by Simulated Annealing Algorithm 63

Table 4.3. List of Elementary Trapping Sets 63

Table 4.4. ETS's Distribution of LDPC Codes 71

Table 5.1. ETS of LDPC Codes within the Range of a $\leq$ 10 and b $\leq$ 3 80

Table 5.2. ETS of LDPC Codes within the Range of a $\leq$ 12 and b $\leq$ 3 81

Table 5.3. Time Performance of the Simulated Annealing Algorithm 85

LIST OF SYMBOLS

a	Number of variable nodes in a trapping set
b	Number of satisfied check nodes in a trapping set
c_j	j-th check node
C	Check node set
$d(c_j)$	Degree of j-th check node
$d(v_i)$	Degree of i-th variable node
$E(x)$	Set of check nodes in N_x with even degree
f_{bias}	Biasing estimate
g	Girth size
g_i	Function value of check node i.
$\mathbf{G}$	Generator matrix
G	Bipartite graph
$G_(x)$	Induced graph of the set x
$\mathbf{H}$	Parity-check matrix
$\mathbf{I}$	Identity matrix
k	Number of redundant bits
l	Depth of a tree
L	Size of permutation matrix
L_{in}	List of input trapping sets
L_{out}	List of output trapping sets
m	Check node numbers
n	Variable node numbers
$N(c_j)$	Neighbors of j-th check node
$N(v_i)$	Neighbors of i-th variable node
$N_{v_i}^l$	Set of check nodes in the tree spreading form v_j with depth l
$O(x)$	Set of check nodes in N_x with odd degree
$p_{i,j}$	Shifting value of (i, j) block matrix
p_{MC}	Naive Monte Carlo estimate

$\mathbf{P}$	Permutation matrix
$\mathbf{r}$	Received vector
S	Odd set
T	Temperature
$\mathbf{u}$	Information vector
$\mathbf{v}$	Code-word vector
v_i	i-th variable node
V	Variable node set
w_{ij}	Weight variable of (i,j) ETS
w	Weighting function
x	Subset of variable nodes
$\mathbf{z}$	Index vector of variable nodes.
Z_i	Bernoulli random variable
α	Parameter that determines active or passive check nodes.
β	Number of 1's in the row.
δ	Number of 1's in the column
ϵ	Failure on a trapping set
η	Cooling factor
$\hat{u}_i$	Estimated information symbol
$\hat{v}_i$	Estimated code symbol
γ	Cost vector of the objective function
λ	Biased value
Λ	Coefficient of the code-word polytope
ψ	Points in the code-word polytope
ρ	Distance coefficient
$\Omega_{i,j}$	Outgoing message from the check nodes
$\varphi_{i,j}$	Incoming message to the check nodes
ς	Boltzmann's constant
ζ	LDPC code

LIST OF ACRONYMS/ABBREVIATIONS

ACE	Approximate Cycle Extrinsic
ALP	Adaptive Linear Programming
AS	Absorbing Sets
AWGN	Additive White Gauss Noise
BER	Bit Error Rate
BFS	Breadth-first Search
BP	Belief Propagation
BPSK	Binary Phase-Shift Keying
BSC	Binary Symmetrical Channel
DVP	Digital Video Broadcasting
ETS	Elementary Trapping Sets
FAS	Fully Absorbing Sets
FER	Frame Error Rate
HALP	Heuristic Adaptive Linear Programming
IALP	Integer Adaptive Linear Programming
IS	Importance sampling
LDPC	Low-density Parity-Check
LETS	Leafless Elementary Trapping Sets
LLR	Log Likelihood Ratio
LSS	Layered Superset
QC-LDPC	Quasi-cyclic LDPC Codes
ML	Maximum-likelihood
PEG	Progressive Edge Growth
RS	Reed-Solomon
LP	Linear Programming
SA	Simulated Annealing
SNR	Signal-to-noise Ratio
SPA	Sum-product Algorithm

TS	Trapping sets
VLSI	Very Large Scale Integration

1. INTRODUCTION

1.1. Overview

Nowadays, there is a great need for reliable and efficient digital data transfer and storage. This demand is increasing day by day with the establishment of high speed and large scale networks. To meet this requirement, it is necessary to check the errors that may occur while transferring data. In this way, data can be reliably recovered. Communication systems transmit data from one point to another over the air or wireline. The reliability of the data received by the destination largely depends on the channel and the surrounding noise. Shannon has shown that reliable transmission can be done with his proposed coding theory [1]. In this theorem, it is shown that if the data rate is less than the channel capacity and the code length goes to infinite, all errors in the received data can be corrected. These developments raised the issue of channel coding.

Channel coding is a process that corrects bit and frame errors in digital communication systems. Channel coding is performed both at the transmitter and at the receiver. In the transmitter part, channel artificial information, named as redundant bits, are added to actual information in order to ensure that no information was lost during the data transmission. Channel coding enables the detection and correction of errors in the received information. The basic elements of channel coding are given in Figure 1.1. In the encoder part, a code-word, $\mathbf{v}$, is created by adding redundant bits to the message vector, $\mathbf{u}$, to be sent. The code-word is distorted as it passes through the channel and reaches the receiver as a code-vector, $\mathbf{r}$. At the decoder, the information vector, $\mathbf{u}$, is estimated by using the code-vector and the information of the appropriate operations in the encoder.

To date, many error control coding applications have been developed to encode and decode the sent data and received data, respectively. The decoding time, error

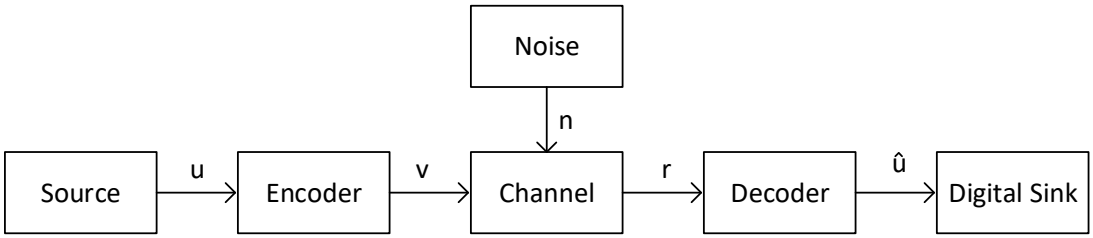

Figure 1.1. Basic Elements of Channel Coding

correction capability, and implementation complexity of each error control codes are different. Reed-Solomon (RS) codes, turbo codes, and low-density parity-check codes are some important channel coding works [2–4].

1.2. LDPC Codes

Low-density parity-check (LDPC) codes are one of the most popular channel codes used today due to their high error-correcting performance. It was shown that LDPC codes, when decoded with iterative decoding, achieve high error rate performance, which is very close to the Shannon limit [5]. LDPC codes were first discovered by Gallagher in 1962 [4]. Nobody paid much attention to LDPC codes for about 20 years. Then, in 1981, Tanner expounded the LDPC codes from a graphical point of view [6]. Subsequently, no significant studies were conducted on it until 1999. In 1999, it was rediscovered by [7]. In particular, it was demonstrated that long LDPC codes approached Shannon limits with iterative decoding based belief propagation decoders [8–10]. These discoveries have made LDPC codes a strong candidate for the standards of many communication and digital storage systems. Today, LDPC codes are used in many standards, especially in 5G, and are seen as an important research topic [11, 12]. Many different versions of LDPC codes are being employed nowadays. In particular, quasi-cyclic LDPC codes (QC-LDPC) have very simple encoder designs, and they are one of the strongest candidate for next-generation wireless communication system standards.

The design of the LDPC code differs for each system to be used. For example, storage systems require a higher code rate (8/9 and higher) [13]. In addition, QC-LDPC codes are preferred by storage systems, since their very large scale integration (VLSI) implementation cost is low [14, 15]. QC-LDPC codes with code lengths 16200 and 64800 are accepted as a standard for digital video broadcasting (DVB) version-2. Apart from these, LDPC codes are used for other communication systems such as optical communications, wireless broadband, Gigabit Ethernet [16, 17].

LDPC codes belong to the linear code family. In the encoder part of LDPC codes, the message vector to be sent is multiplied by the generator matrix, $\mathbf{G}$, as given in Equation (1.1) to create the code-word, $\mathbf{v}$.

$$\mathbf{uG} = \mathbf{v} \quad mod(\mathbf{2}) \tag{1.1}$$

The code-vector, $\mathbf{r}$, a corrupted version of $\mathbf{v}$, is decoded with the help of a parity-check matrix, $\mathbf{H}$, that satisfies the Equation (1.2).

$$\mathbf{vH^T} = 0 \quad mod(\mathbf{2}). \tag{1.2}$$

The decoding part of an LDPC code could be achieved in many ways. There are many different hard-decision and soft-decision methods, such as majority-logic decoding, bit-flipping decoding, a posteriori probability decoding, and iterative decoding based on belief propagation. Although decoders based on a posteriori probability decoding method have high error-correcting performance, they require high complexity. Decoders based on the iterative decoding method require less complexity but provide good error-correcting performance comparing to the posteriori probability decoding method. Therefore, iterative decoding based belief propagation decoders are the most preferred decoder algorithms today.

Maximum-likelihood (ML) decoding, which maximizes the likelihood function of received vector given the code symbols, provides the best error performance. The out-

put vector of ML decoders are named as ML code-word. However, its implementation for decoding long codes is not feasible. Therefore, alternatively, linear decoders have been proposed. Linear programming (LP) decoder was first developed in 2005 by Feldman, Wainwright, and Karger [18]. Their LP decoder has the ML certificate property; whenever it outputs a code-word, it is guaranteed to be the ML code-word. When the LP decoder finds a vector with fractional values, it is assumed that the decoding ends incorrectly. In the studies carried out in the following years to advance the LP decoder, they followed a similar path of the original LP decoder proposed by [18]. When the LP decoder outputs a vector with fractional values as a solution, the constraints that make this solution infeasible, called as valid constraints, is added to the LP model. With additive constraints, LP decoder finds another solution. This process continues until a vector with integer values is found as a feasible solution [19–23]. The error-correcting performance of these decoders that are named as adaptive linear programming (ALP) decoders, has begun to approach the error-correcting performance of ML decoders with added constraints. The principles of the ALP decoders introduced in the literature are basically similar. The parts in which these studies are different appear in the methods of finding valid constraints. Although adaptive linear decoders have better error-correcting performance, they require high complexity. This is the price to be paid for better error performance.

For the iterative decoding based on belief propagation algorithms, the selection of the $\mathbf{H}$ matrix directly affects the performance of the decoder. The performance of the decoder is determined by the structure of the bipartite graph representation of the parity-check matrix. Due to the presence of some harmful structures in the bipartite graph, LDPC codes encounter error floor problems at the high signal-to-noise ratio (SNR) region despite their high error-correcting performance capacity. These structures are named as trapping sets (TS) [24]. When the distribution of trapping sets is known, error floor performance could be estimated using importance sampling [24–26]. Designing a parity-check matrix with a bipartite graph that contains a small number of trapping sets is one of the most important research topics in recent times. It is noted that the cycles in the bipartite graph, also named as the Tanner graph,

consist of the trapping sets [24]. It is not feasible to get rid of the cycles completely for a finite graph. However, removing the small cycles of the Tanner graph allows the designer to get rid of some small trapping sets that cause error floor problem. Therefore, the size of the smallest cycle in the graph, also named as the girth, is an important parameter for the construction of LDPC codes. While constructing LDPC codes, many studies have aimed to reduce the number of cycles in the Tanner graph and tried to increase the size of the smallest cycle in the Tanner graph. One of the first algorithms proposed with this idea, named as the progressive edge growth algorithm (PEG), constructs the graph in an iterative way that maximizes the cycle size in each iteration [27]. In the following years, new algorithms similar to the PEG algorithm have been proposed [28–31]. An improved PEG algorithm given in [29] tries to reduce the number of cycles with the smallest cycle size. [28] and [30] select edges according to their connectivity of the cycle with the rest of the graph in each step. [31] avoids trapping sets in the graph by evaluating certain properties of the cycles introduced in each edge placement.

After QC-LDPC codes have become so popular, many works have been published to design QC-LDPC codes. In 2015, one of the first studies of them proposed a PEG-like algorithm that does not include (5,3) TSs and tries to minimize the number of (6,4) TSs [32]. However, because of its complexity, it generates only small-length QC-LDPC codes. The pioneers focus on the construction QC-LDPC codes with a certain girth value instead of the construction of QC-LPDC codes with an optimized trapping set distribution. The main reason that the trapping sets optimization increases the complexity of algorithms significantly. In 2016, [33] exhaustively searched for QC-LDPC codes with a given degree distribution and girth. To achieve this task in a feasible time, [33] prunes the search space by removing many codes that are isomorphic. Many of the pruning techniques to reduced the size of search space were proposed in the following years [34–36]. On the other hand, new techniques have been developed to find the TS distribution of QC-LDPC codes, [37–40]. Since these suggested algorithms have reduced the complexity of trapping sets optimization, it paves the way for constructing QC-LDPC codes with trapping sets optimized. In 2018, [41] suggested a PEG-like

algorithm that controls the cycle structures and constructs QC-LDPC codes that do not include some dominant trapping sets. In 2019, [42] has managed to generate QC-LDPC codes with the smallest base matrices that provide a certain trapping set distribution. In 2020, [43] eliminated some dominant trapping sets with the help of search algorithms.

1.3. Contribution of The Thesis

In this thesis, at the first step, adaptive linear programming decoders are designed to improve the error-correcting performance of LDPC codes. Then, LDPC codes with high error-correcting performance are designed using simulated annealing (SA) based algorithms.

In the first part, two new adaptive linear decoder algorithms have been proposed. Of these, the suggested heuristic adaptive linear programming (HALP) decoder finds the valid constraints in a shorter time compared to existing adaptive linear programming decoders. However, its error-correcting performance is not as good as the performance of the most recent decoders. Although the proposed adaptive linear decoder algorithms do not have a better code solution than the recent studies in the literature, they could be used with current decoders in a hybrid system, since their decoding time is shorter than other decoders.

In the second part, research was directed to find LDPC codes with the smallest cycle. LDPC codes with a much better cycle distribution are generated. Although our generated codes have the best cycle distributions among modern codes, the desired gain in the bit error rate (BER) and frame error rate (FER) could not be achieved, because these codes still have some of the dominant trapping sets. In particular, QC-LDPC codes that are constructed in recent works such as [40, 42] have better error-correcting performance.

In the last part, LDPC codes with better frame error rate performance are tried to be found. For this goal, the design of QC-LDPC codes, which are one of the most popular codes of the last period, has been examined. First of all, dominant trapping sets were determined by importance sampling, and QC-LDPC codes that did not contain these sets were found by a simulated annealing based algorithm. QC-LDPC codes with the best trapping set distribution and best error-correcting performance in modern codes were found and presented in this thesis. In addition to that, the best results in the literature have been accomplished for the problem of finding the smallest QC-LDPC codes with a certain TS distribution.

1.4. Thesis Outline

Fundamentals of LDPC codes are given in Chapter 2. The decoding methods to decode LDPC codes are explained as well. Moreover, errors encountered in decoder algorithms are demonstrated and the reason behind these errors is explained in detail. The effects of these errors on code performance are illustrated.

In Chapter 3, the proposed adaptive linear decoder algorithms are introduced. The decoding time and error-correcting performance of the proposed decoders are compared with other studies.

In Chapter 4, the best known LDPC code design algorithms are mentioned, and the PEG-based simulated annealing algorithm and RandPEG based simulated annealing algorithm developed in the thesis is explained. In the last part of this chapter, the performance of the proposed LDPC codes has been compared with the popular LDPC codes.

In Chapter 5, studies on the construction of QC-LDPC codes, one of the most popular codes of recent years, are given. Significant works in the literature are given in the first part of the chapter. Then, the proposed simulated annealing algorithm is explained in detail. QC-LDPC codes produced by the simulated annealing algorithm

are given in the last parts of the chapter. Their error-correcting performance and trapping set distribution is compared with the most recent works.

In Chapter 6, thesis studies are summarized and concluded. New ideas are suggested for future studies.

2. LDPC Codes

2.1. Introduction of the LDPC Codes

Definition 1. *A (β, δ)-regular LDPC code is the null space of a parity-check matrix* $\mathbf{H}$*, where rows consist of β 1's, columns consist of δ 1's, and most importantly both β and δ are very small in comparison with the number of the columns and rows in* $\mathbf{H}$*. If the columns or the rows in* $\mathbf{H}$ *have different weights, then the corresponding LDPC code is called irregular.*

An (n, k) LDPC code can be represented in terms of a bipartite graph, G (also called the Tanner graph), with two sets of nodes: n variable nodes, $V = \{v_1, \ldots, v_n\}$, and $m = n - k$ check nodes, $C = \{c_1, \ldots, c_m\}$ [6]. Variable nodes correspond to code symbols (columns in $\mathbf{H}$) and check nodes correspond to parity-check equations (rows in $\mathbf{H}$). An edge connects a variable node to a check node if and only if the corresponding code symbol participates in the corresponding parity check node. The nodes connected to the i-th variable (j-th check) node are referred to as its neighbors and denoted as $N(v_i)$ $(N(c_j))$. The degree of a node, therefore, is given as $|N(v_i)|$ or $|N(c_j)|$. d_c and d_v denote the degree of check nodes and variable nodes, respectively. The induced subgraph of subset $x \subseteq V$, G_x, is a bipartite graph containing x, $N(x)$ and their corresponding edges. For the induced graph G_x, the set of the check nodes in $N(x)$ with even degree are represented as $E(x)$, and the set of the check nodes in $N(x)$ with odd degree are represented as $O(x)$.

Example 2. *The Tanner graph of an (8,3) LDPC code with parity-check matrix* $\mathbf{H}$ *is represented in Figure 2.1. The filled circles represent the variable nodes and the empty*

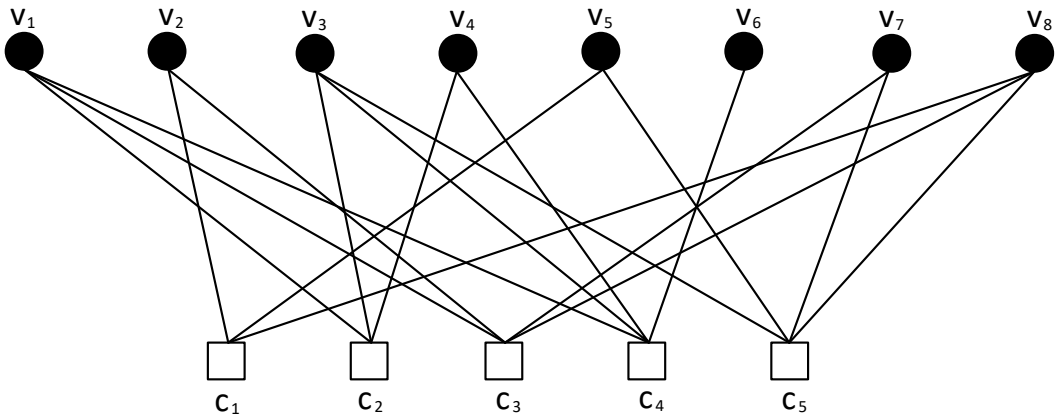

Figure 2.1. The Tanner Graph of an (8,3) LDPC Code.

squares represent the check nodes. The corresponding parity-check matrix is given by

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{2.1}$$

For example, the first row of $\mathbf{H}$ *shows the connection of first check node,* c_1*, to the variable nodes. When the* i^{th} *element of the first row is 1, then there is a connection between* c_1 *and* v_i*. Therefore, there are edges between* c_1 *and* v_2, v_5, v_8*.*

QC-LDPC codes belong to the family of LDPC codes. The distinguishing feature of QC-LDPC codes from other LDPC codes is that their pariy-check matrix consists of shifted identity matrices. Thanks to this, the code-words in the encoder part can be produced in a short time with shift registers that consume little energy. The parity

check matrix of QC-LDPC codes, H, is constructed from identity matrices as

$$H = \begin{bmatrix} I(p_{0,0}) & I(p_{0,1}) & ... & I(p_{0,d_c-1}) \\ I(p_{1,0}) & I(p_{0,1}) & ... & I(p_{1,d_c-1}) \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ I(p_{d_v-1,0}) & I(p_{d_v-1,1}) & ... & I(p_{d_v-1,d_c-1}) \end{bmatrix} \tag{2.2}$$

where $I(p_{i,j})$ is the $L \times L$ identity matrix whose elements in each row are cyclically shifted to the left by $p_{i,j}$. The parity check matrix of QC-LDPC codes can be also presented by a permutation shift matrix P and a lifting degree L,

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,d_c-1} \\ p_{1,0} & p_{0,1} & \cdots & p_{1,d_c-1} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ p_{d_v-1,0} & p_{d_v-1,1} & \cdots & p_{d_v-1,d_c-1} \end{bmatrix}. \tag{2.3}$$

A quasi-cyclic code is a linear code for which cyclically shifting a code-word a fixed number, p, results in another code-word. If p value is 1, the quasi-cyclic code is a cyclic code. This feature enables the encoding and decoding parts of cyclic codes to be designed with a simple shift register and logic circuits. Although it does not provide full cyclic symmetry, codes with partial cyclic structure have become popular nowadays. In particular, it will be one of the basics of the next-generation communication systems.

Table 2.1. The Code-words List

Index	**Code-word**
1	0 0 0 0 0 0 0 0 0
2	1 1 1 1 0 0 1 1 0
3	1 1 0 1 1 1 1 0 0
4	1 0 0 1 1 0 1 1 1
5	0 0 1 0 1 1 0 1 0
6	0 1 1 0 1 0 0 0 1
7	0 1 0 0 0 1 0 1 1
8	1 0 1 1 0 1 1 0 1

Example 3. *A (9,3) quasi-cyclic code which is generated by a generator matrix,* $\mathbf{G}$ *is given as*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{2.4}$$

The code-words that are generated by $\mathbf{G}$ given in Equation (2.4) are presented in Table 2.1. For instance, if the second code-word (111100110) is shifted three symbol positions to the right, another code-word (110111100) is generated. However, if the second code-word is shifted one or two to the right, another code-word isn't obtained. Therefore, the shifting constraint of this QC-LDPC code is 3. The encoder of the code is given in Figure 2.2.

Let $u = (u_0, u_1, u_2)$ be the information vector that will be encoded. In the first state, the content of the register will be (u_0, u_1, u_2) and the output terminal of shift

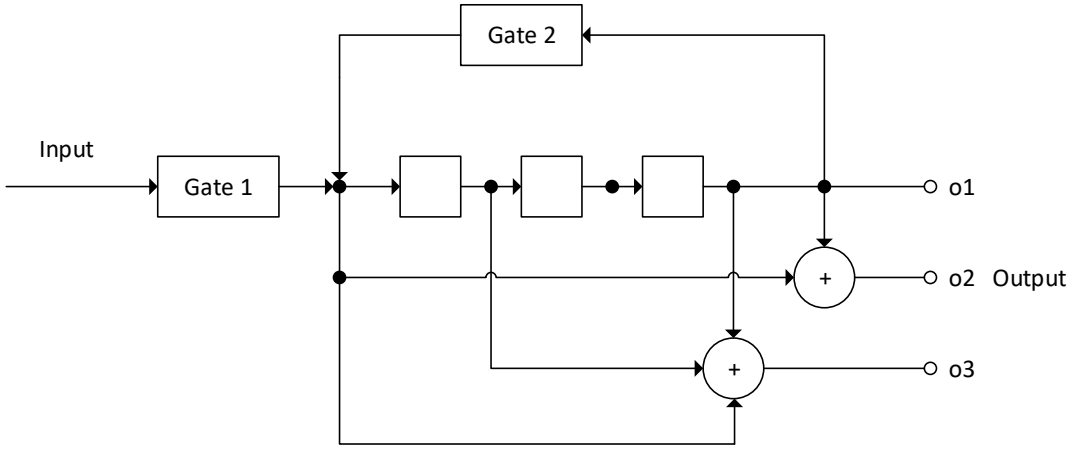

Figure 2.2. An Encoding Circuit for the Quasi-Cyclic Code.

register will be as

$$v_2^0 = u_2, \tag{2.5}$$

$$v_2^1 = u_0 + u_2, \tag{2.6}$$

$$v_2^2 = u_0 + u_1 + u_2. \tag{2.7}$$

After the register is shifted once, the content of the register will be (u_2, u_0, u_1). Therefore, the output terminal will be as

$$v_1^0 = u_1, \tag{2.8}$$

$$v_1^1 = u_1 + u_2, \tag{2.9}$$

$$v_1^2 = u_0 + u_1 + u_2. \tag{2.10}$$

After the register is shifted once again, the content of the register will be (u_1, u_2, u_0).

Therefore, the output terminal will be as

$$v_0^0 = u_0, \tag{2.11}$$

$$v_0^1 = u_0 + u_1, \tag{2.12}$$

$$v_0^2 = u_0 + u_1 + u_2. \tag{2.13}$$

The encoding will be completed after the second shift. The code-word generated by this shift register will be as

$$\mathbf{v} = (v_0^2, v_0^1, v_0^0, v_1^2, v_1^1, v_1^0, v_2^2, v_2^1, v_2^0). \tag{2.14}$$

Equation (2.14) can also be written as

$$\mathbf{v} = (v_0^2, u_0, v_0^0, v_1^2, u_1, v_1^0, v_2^2, v_2^1, u_2). \tag{2.15}$$

The code-word given in Equation (2.15) has systematic form. It consists of three blocks with one information bit and two redundant bits.

2.2. Decoding of the LDPC Codes

2.2.1. Message-passing Decoders

Thanks to the sparse structure of LDPC codes, it is possible to get close to the performance of optimal decoding by using iterative decoding algorithms based on message-passing algorithms. The main significant feature of a message-passing algorithm is that check and variable nodes exchange messages with their neighboring variable and check nodes. In the first iteration, the received vector is directly transmitted to the check nodes as an incoming message, $q_{i,j}$, since there are no messages from the check nodes yet. After that, check nodes compute outgoing messages, $o_{j,i}$, depending on the incoming messages and local code constraints, and send messages to their neighboring variable nodes.

For an LDPC code with a cycle free Tanner graph G, iterative decoding can have the same performance as maximum likelihood decoding. The estimated code symbol $\hat{u_i}$ with maximum-likelihood can be written as

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} P(\mathbf{r} \mid u_i), \qquad i = 1, 2, ..., n, \tag{2.16}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \frac{P(\mathbf{r} \mid u_i)P(u_i)}{P(u_i)}. \tag{2.17}$$

Applying Bayes' rule and using the fact that $P(u_i)$ is constant, we obtain

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} P(\mathbf{r}, u_i), \tag{2.18}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} P(\mathbf{r}, \mathbf{u}), \tag{2.19}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} P(\mathbf{r} \mid \mathbf{u})P(\mathbf{u}), \tag{2.20}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} P(\mathbf{r} \mid \mathbf{u}). \tag{2.21}$$

Equation (2.21) can be written as

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} \prod_{j=1}^{n} P(r_j \mid u_j), \tag{2.22}$$

since the channel is assumed to be memoryless. Evaluations from Equation (2.16) to Equation (2.22) show that the maximum likelihood symbol decision can be written as the sum-product equation. The most common message-passing algorithm, the belief propagation (BP), also known as the sum-product algorithm (SPA), is derived from Equation (2.22). When the Tanner graph of an LDPC code is cycle free, SPA has the same performance as that of the maximum likelihood decoding algorithm. However, all finite-length LDPC codes have cycles in their Tanner graph, which decrease the performance of SPA decoding. These cycles also lead to several problematic structures

that devastate the performance of SPA decoding.

Sum-Product Algorithm There are 4 steps in the sum-product (belief propagation) algorithm:

1.Initialization Each variable node is initialized with the received vector ($\mathbf{r}$). For each check node, their value is initialized to zero. The initial log likelihood ratio (LLR) is calculated for each variable node as

$$LLR(v_i) = \ln\left\{\frac{P(v_i = 1|r_i)}{P(v_i = 0|r_i)}\right\} \tag{2.23}$$

When the channel is additive white Gaussian noise (AWGN) channel, Equation (2.23) can be written as

$$LLR(v_i) = 2r_i/\sigma^2. \tag{2.24}$$

Messages are exchanged between connected check nodes and variable nodes in each iteration. For the first step, incoming and outgoing messages are initialized to zero.

2.Check node Update The log likelihood ratio and the outgoing message from the check nodes to variables are computed in this step. LLR for the each check node, c_j, is computed using the incoming message, $\varphi_{i,j}$, from variable nodes, v_i, as

The outgoing message ($\Omega_{i,j}$) from the check nodes to variable nodes is given by

$$\Omega_{i,j} = 2\tanh^{-1} e^{\ln LLR(c_i) - \ln\tanh\varphi_{i,j}/2} \tag{2.25}$$

3.Variable-node update The outgoing message from the variable nodes to check nodes and the log-likelihood ratio is computed in this part. LLR for each variable node,

v_i, is updated using the incoming message, $\Omega_{i,j}$, from check nodes, c_j, as

$$LLR(v_i) = LLR(v_i) + \sum_{\forall c_j \in N(v_i)} \Omega_{i,j}. \tag{2.26}$$

The outgoing message from the variable nodes to check nodes is computed simply as

$$\varphi_{i,j} = LLR(v_i) - \Omega_{i,j}. \tag{2.27}$$

4.Hard-Decision The hard-decision is done using the value of $LLR(v_i)$. If $LLR(v_i) < 0$, then the decoded bit $\hat{v_i} = 0$, and if $LLR(v_i) \geq 0$, then $\hat{v_i} = 1$.

$$\hat{\mathbf{v}}\mathbf{H}^{\mathbf{T}} = 0. \tag{2.28}$$

When the Equation (2.28) is satisfied, then the algorithm is halted. The $\hat{v}$ gives the estimation of the code-word. If the Equation (2.28) is not satisfied, then the process goes back to Step 2. If the algorithm doesn't find a code-word in a maximum iteration number that is defined by the user, then it outputs the failure.

2.2.2. Linear Programming Decoders

It is difficult to examine the behavior of message-passing decoders due to the cycles in the Tanner graph. They also do not have the maximum-likelihood certificate property. Jon Feldman proposed an alternative decoder with ML certificate property in 2005, [18]. When this decoder completes the decoding process, it generates an ML code-word or a non-integer pseudo code-word.

The ML decoder for an LDPC code can be expressed as

$$\text{minimize } \gamma^T \mathbf{v} \tag{2.29}$$

$$\text{constraints: } \mathbf{v} \in \zeta \tag{2.30}$$

γ presents the cost vector of the objective function. The convex hull of code-words belonging to ζ code creates the code-word polytope. The code-word polytope is defined as

$$poly(\zeta) = \left\{ \sum_{v \in \zeta} \Lambda_v v : \Lambda_v \geq 0, \sum_{v \in \zeta} \Lambda_v = 1 \right\}. \tag{2.31}$$

. The fact that the corner points of the code-word polytope generate all code-words belonging to the ζ code allows the linear programming to be used for the decoding. The best solution of the linear programming will be a corner point of the code-word polytope. Therefore, ML decoder can be defined by linear programming. If we denote each point in the code-word polytope with the vector ψ, the components of the vector can be expressed by $\psi_i = \sum_v \Lambda_v v_i$. Linear programming optimization method can find ML code-word on the code-word polytope as

$$\text{minimize } \sum_{i=1}^{n} \gamma_i \psi_i \tag{2.32}$$

$$\text{constraints: } \psi \in poly(\zeta) \tag{2.33}$$

The biggest problem with this linear programming model is that the number of code-words is usually too high for codes used in practice. Expressing the code-word polytope with linear constraints is not usually possible due to a large number of constraints. As an alternative solution, Feldman proposed a relaxed polytope, [18]. The local polytope, which is a convex envelope of local code-words, can be constructed for each check node. Due to the definition of LDPC codes, local code-words will be small in size and few in number. As a result, it is possible to show local polytopes with a few linear constraints. Feldman produced a polytope called the basic polytope at the intersection of local

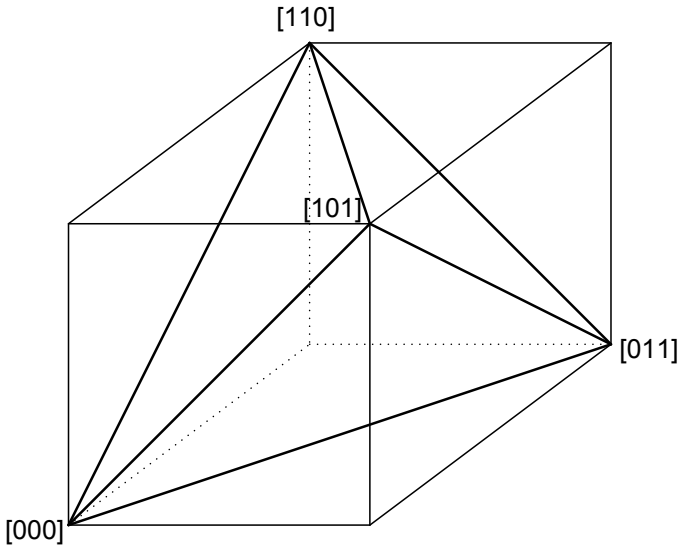

Figure 2.3. A local Polytope

polytopes. Since the code-words are located at the intersection of local code-words, the basic polytope will cover all of the code-words. The local polytope for a three-neighbor check node is given in Figure 2.3. The local polytope for a check node can be created by banning single-weight vectors that do not generate local code-words. These banned vectors are defined as

$$\sum_{v_i \in S} v_i - \sum_{v_i \in (N(c_j) \setminus S)} v_i \leq |S| - 1 \;\; \forall S \subseteq N(c_j), |S| \, odd. \tag{2.34}$$

S represents the set of all single-weighted vectors belonging to a check node, and the $N(.)$ operator shows the set containing neighbors of a given node. Constraint 2.34 can also be shown as

$$g_i = \sum_{v_j \in S} (1 - v_j) + \sum_{v_j \in (N(c_i) \setminus S)} v_j \geq 1 \tag{2.35}$$

for each odd vector. Let $poly(\zeta_i)$ will be the local polytope of the i. check node. Then,

the LP decoder is defined as

$$\text{minimize } \sum_{j=1}^{n} \gamma_j v_j \tag{2.36}$$

$$\text{constraints: } \mathbf{v} \in poly(\zeta_1) \cap poly(\zeta_2) ... \cap poly(\zeta_m) \tag{2.37}$$

The basic polytope is a relaxed polytope, including the convex envelope of code-words. The corners of the basic polytope contain code-words as well as pseudo code-words with fractional numbers. For this reason, a linear programming module, which is using the basic polytope, gives the same result as the ML decoder when it finds a solution that is containing only integer variables. On the other hand, if the result contains a fractional number, the LP decoder assumes that it has reached an incorrect result (named as pseudo-codeword). To sum up, the LP decoder using the basic polytope does not have the ML decoder performance but has the ML certificate property. The LP decoder that uses only the basic polytope will be named as the standard LP decoder for the rest of the thesis.

When the standard LP decoder finds a fractional solution, a constraint that makes this solution an infeasible solution can be searched. By eliminating fractional solutions, the error-correcting performance of the standard LP decoder can be increased and converged to the error-correcting performance of ML decoders. This type of LP decoder is called an adaptive linear programming decoder.

An ALP decoder, which requires more complexity than the standard LP decoder, but has better error-correcting performance than the standard LP decoder, was designed in [19]. In [19], it was aimed to obtain redundant parity check equations by summing fractional check nodes that can create a cycle in the Tanner graph. However, there was no guarantee that the redundant parity check node obtained at the end of this process could make the active solution to become an infeasible solution. It has been shown by the same authors that the parity check equations that can make the fractional outcome an infeasible solution must meet some conditions [20,21,44,45]. [20]

reduces the constraint numbers to define the polytope with a separation algorithm. A technique that improves the error-correcting performance of standard LP decoders by performing basic line-column operations on the parity control matrix is proposed in [21]. After [21], a similar approach applied with the generator matrix G in [45].

2.3. Failures of LDPC Codes

As mentioned early, cycles decrease the performance of the decoding algorithms. An example of a cycle is given in Figure 2.4. In this figure, v_2, v_5, and v_7 consist of a cycle with size 6. When these three variable nodes in the error state, the decoding algorithm will most probably fail. Every check node connected to these variable nodes (v_2, v_5, and v_7) has exactly two variable nodes from this set, and when there is an error in these three variables, the two erroneous messages coming to the check nodes can hide the harmful effect of each other.

Since all finite-length LDPC codes have cycles in their Tanner graph, one way to decrease this kind of decoding failure is to increase the size of cycles in the Tanner graph. Therefore, when constructing an LDPC code, it is necessary to create it so as to maximize the smallest cycle size in the Tanner graph. In graph theory, the minimum size of cycles is named as *girth*. Therefore, a good LDPC code should have a high girth value. However, at some point, it is sometimes impossible to increase girth value in the graph. In this circumstance, it is also desired to have few cycles that have girth size.

Another important property of the Tanner graph of an LDPC code is its cycle's approximate cycle extrinsic message degree (ACE) [28, 30]. ACE is the measure of connectivity for a cycle to the rest of the graph through its symbol nodes. For example, in Figure 2.4, given cycle has no connectivity to the rest of the graph, since v_2, v_5, and v_7 have only two edges which all of them are belong to the cycle. Therefore, its ACE value is zero. If the ACE value is so small, then the iterative decoding algorithm couldn't change the values of variable nodes belongs to the cycle. As a result of that,

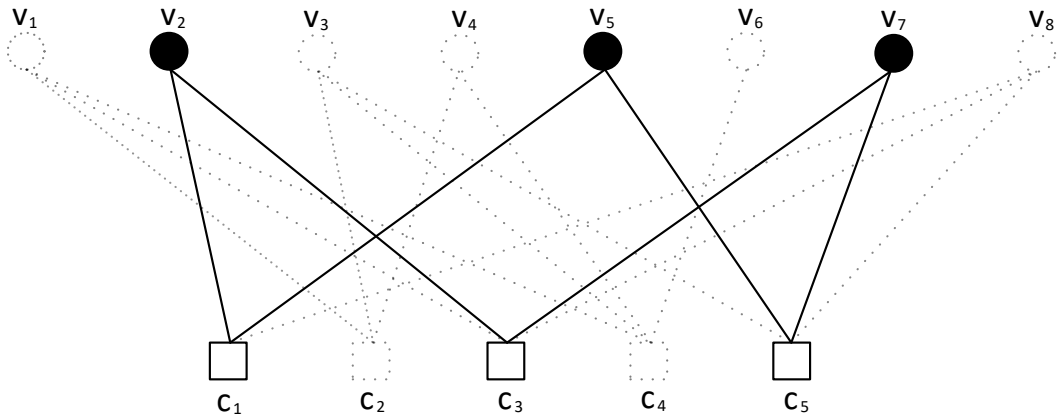

Figure 2.4. A Cycle Example

the decoding algorithm will fail. For regular LDPC codes, the cycles of the same size will have the same ACE values.

It has been shown that trapping sets created by the cycles are the main reason for the decrease of the error correction performance caused by the cycles and their ACE values [24].

Definition 4. *$x \subseteq V$ is an (a, b) trapping set if $|x| = a$ and $|O(x)| = b$.*

An (a, b) trapping set with small values for both of the parameters has the potential to harm the decoding performance, since a small value of a makes it more likely to observe such an error structure at the channel output and a small value of b makes it more likely for a decoder to get stuck during the decoding iterations. A special class of trapping sets are called elementary trapping sets and are defined as

Definition 5. *An (a, b) elementary trapping set x is an (a, b) trapping set such that $|N(c_j)| = 1$, $\forall c_i \in O(x)$ and $|N(c_j)| = 2$, $\forall c_j \in E(x)$.*

Among the trapping sets, it was proven that elementary trapping sets (ETS) cause the main negative contribution to the error floor performance, [24, 26, 46–48]. These are the trapping sets whose induced subgraph only contains check nodes of degree one

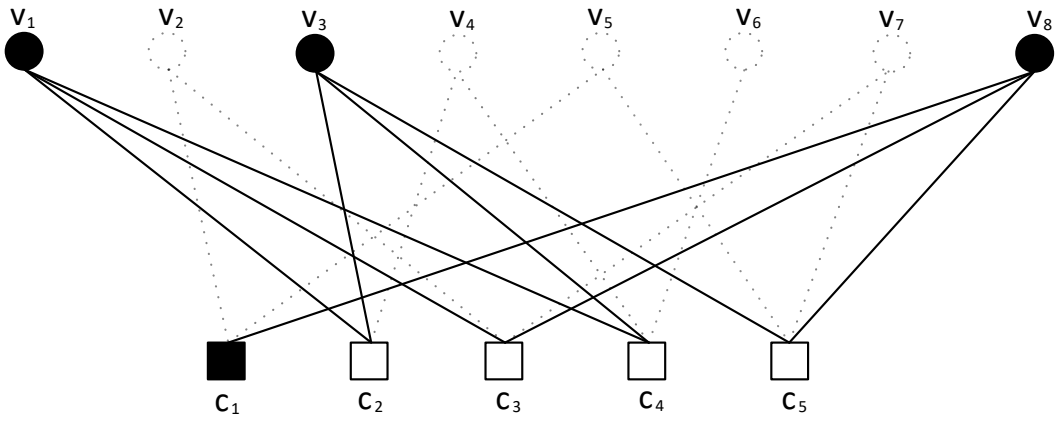

Figure 2.5. The Tanner Graph of a (3,1) TS

or two. It was also shown that smaller size of error-prone structures are more dominant than bigger ones in the error floor region because of the fact that the probability of occurrence for small error structures is higher [24]. Therefore, it is fair to say that the most harmful trapping sets are the elementary trapping sets with a small number of a and b. A (3,1) ETS example is given in Figure 2.5. The subgraph induced by the set of v_1, v_3, and v_8 creates a (3,1) TS, since all check nodes except for c_1 has even number of variable nodes from these set. Moreover, this set is also an elementary TS since the degree of the check nodes in the subgraph is at most two. Another example of a TS is given in Figure 2.6. The variable nodes v_1, v_2, v_3, v_4 and v_5 consist of a (5,3) ETS.

Definition 6. *An elementary trapping sets is a leafless elementary trapping set (LETS) such that each the variable node is connected to at least two satisfied check nodes.*

Although trapping sets and elementary trapping sets vaguely describe the potential damage they could cause to the decoding process, they are mostly conceptual structures. A subclass of them, called absorbing sets, provide us with a more realistic definition [49].

Definition 7. *An absorbing set (AS) is a trapping set such that all the variable nodes that take part in the AS are connected to more satisfied check nodes than unsatisfied ones.*

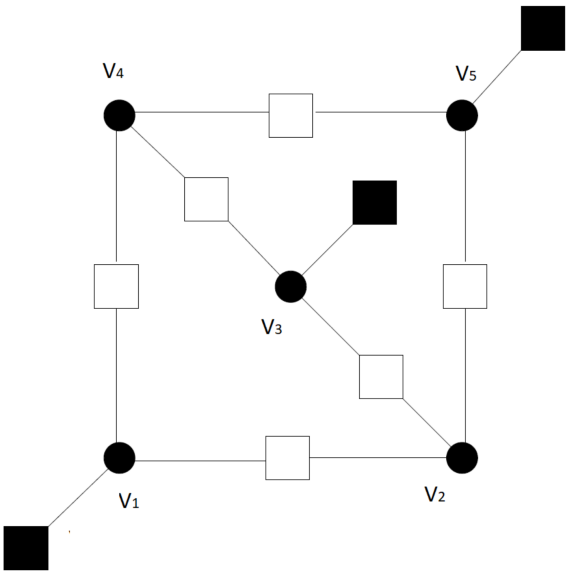

Figure 2.6. The Tanner Graph of a (5,3) ETS

An AS is called fully absorbing, when all variable nodes in the graph, both within the induced graph and elsewhere, have more satisfied neighboring check nodes than unsatisfied ones. Finally, similar to the trapping set definitions, an AS is called elementary if the induced graph only has check nodes of weights 1 and 2. The following two examples demonstrate the conditions for the decoder to stop processing when an AS is encountered.

Example 8. *$x = \{v_1, v_3\}$ is an AS for the Tanner graph in Figure 2.7, since both v_1 and v_3 have less odd degree neighbors in their induced subgraph, c_3 or c_5, than their remaining neighbors, c_2 and c_4. On the one hand, it is not a fully absorbing set (FAS), because v_8 has two odd neighbors, c_3 and c_5, from G_x and only one neighbor, c_1, from G'_x. On the other hand, it is an elementary absorbing set (EAS) owning to fact that every check node in G_x has degree two.*

Example 9. *$x = \{v_1, v_3, v_8\}$ in Figure 2.8 is an FAS, since it is an AS and every variable node in G'_x has less neighbors from $O(x)$ than their remaining neighbors.*

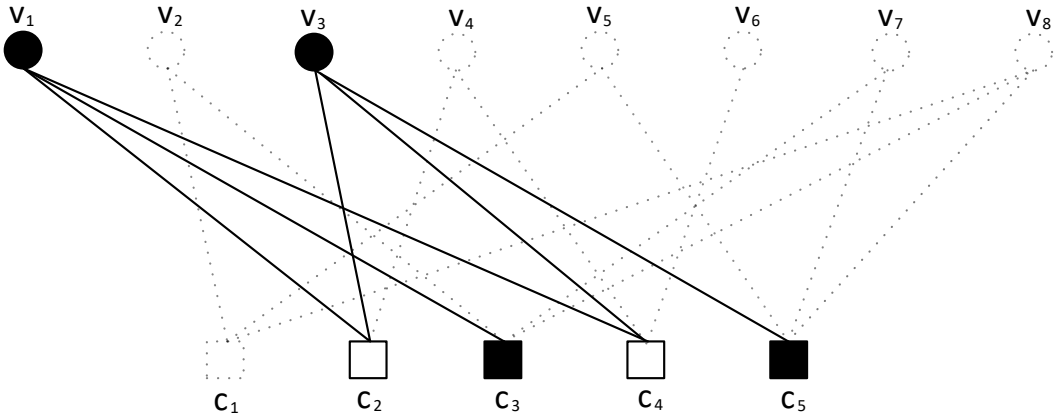

Figure 2.7. The Tanner graph of an absorbing set.

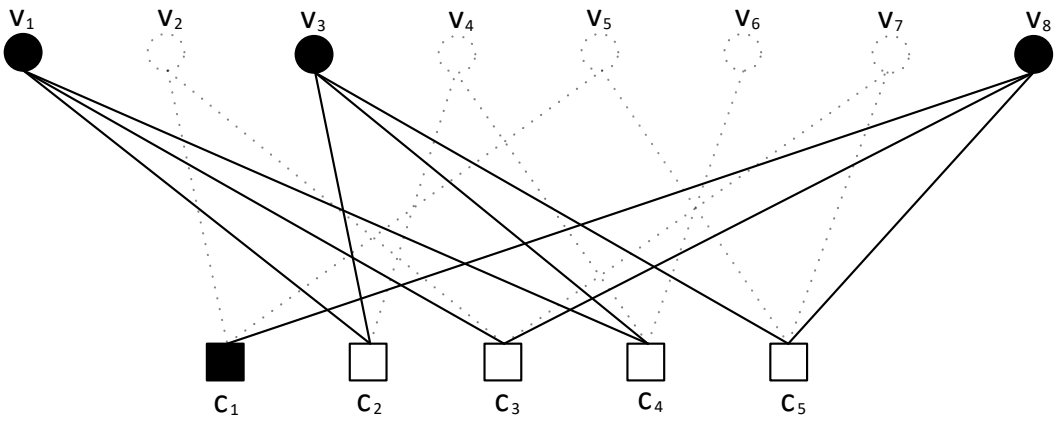

Figure 2.8. The Tanner graph of a fully absorbing set.

2.4. TS Enumeration of LDPC Codes

Knowing the positions and numbers of the dominant trapping sets in the Tanner graph of an LDPC code provides important benefits in many ways. First, the error floor performance of an LDPC code can be found using these dominant trapping sets [24–26]. Also, the information on the dominant TSs can be used in the decoder part. It has been observed that some changes made in the decoder according to the TS distribution increase the error floor performance [50]. In addition, knowing the TS distribution can enable the design of LDPC codes with high error floor performance. In [51], it is shown that some known dominant TS of the base matrix can be removed by replacing the edges of the two base matrices.

Although knowing trapping sets distribution has all these benefits, it is not easy to find dominant TSs. Finding trapping sets and stopping sets and determining the TS with the smallest size has been shown to be NP-hard [52], [53]. Efficient algorithms were developed in early studies to find small TSs [37,54–56]. However, they failed to find TS of large-scale LDPC codes. In 2014, Mehdi Karimi and Amir H. Banihashemi showed that TSs can be found more efficiently by inputting small TSs in their algorithms, [37]. By defining a relationship between TSs, which they call layered superset (LSS) property, they quickly and accurately provided the presence of TS's LDPC codes.

The layered superset relationship between an ETS and one of its subsets is defined in [57] as

Definition 10. *Consider an (a,b) elementary trapping set S in G. Let $Y \subset S$ be an ETS in G of size $x < a$. We can say that S is a layered superset of Y if there exists a nested sequence of ETSs: $Y = S^{(0)} \subset S^{(1)} \subset ... \subset S^{(a-x)} = S$, such that $S^{(i)} \in G$ has size $a + i$ for i=0,..., x-a.*

Definition 10 implies that if S is a layered superset of Y, then we can reach S from Y by recursively expanding trapping sets. In [37], it was shown that if S is an elementary trapping set of size a in G, then for each elementary trapping set $S' \in T_S^{a+1}$

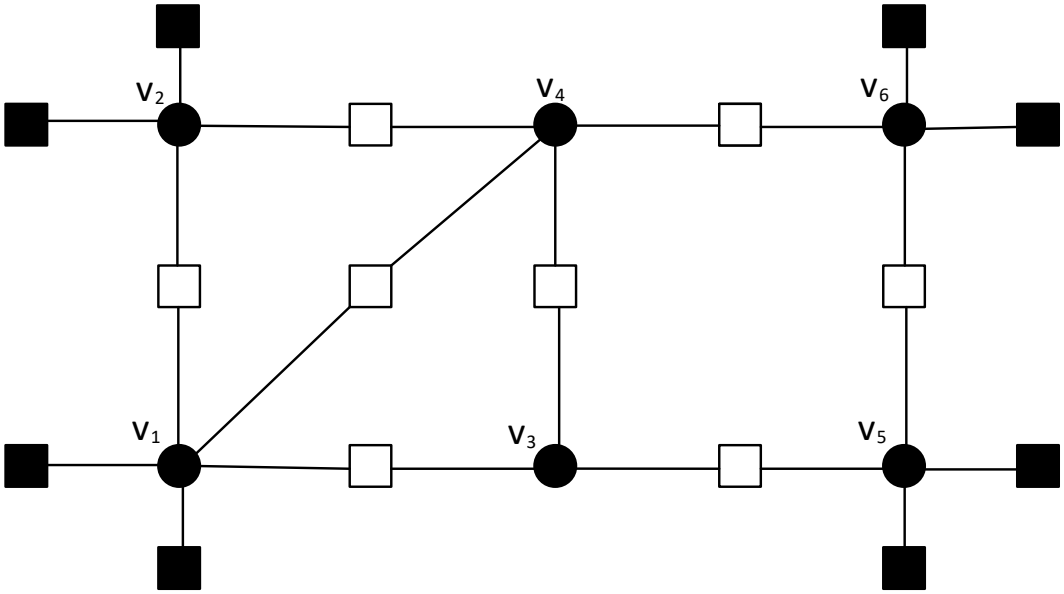

Figure 2.9. ETSs Example

the variable node in $S' \setminus S$ is only connected to unsatisfied check nodes of S. Therefore, expansion of elementary trapping sets are done over only unsatisfied check nodes of S.

Example 11. *An example of a Tanner graph of an LDPC code is given in Figure 2.9. Let subsets* $S = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ *and* $V_1 = \{v_1, v_2, v_3\}$. *S and* V_1 *are (6,6) and (3,6)ETSs, respectively. However, we can not say that S is an LSS of* V_1*, since it is not possible add* v_5 *or* v_6 *while creating a new trapping set. However, S is an LSS of* $V_2 = \{v_3, v_4, v_5, v_6\}$ *with the following nested sequence of ETSs:* $V_2 \subset \{v_1, v_3, v_4, v_5, v_6\} \subset S$.

The pseudocode of expansion of an ETS S is given in Figure 2.10. The complexity of this algorithm to expand an (a, b) ETS is $O(bd_c)$. Any cycle in a Tanner graph is an ETS, and enumeration of short cycles is not a hard task [58]. Therefore, for the expansion of ETSs, it could start with small cycles. With the algorithm given in Figure 2.11, starting from a short cycle, we can enumerate all ETSs up to size t. In [37] and [57], it was shown that most of the ETS can be enumerated with Figure 2.11. Before this algorithm is adapted to the suggested algorithms in this work, it has been applied to the Tanner code (155,64) given in [59] to show that the algorithm

Require elementary trapping set, S.

$\emptyset \rightarrow$ NEWSET;

Generate $O(S)$ and $E(S)$ as the set of unsatisfied check nodes and satisfied check nodes of S, respectively;

Generate $V_2(S)$ as the set of variable nodes which have at least two connections with the check nodes in $O(S)$ and have no connection with the check nodes in $E(S)$;

for each element v in $V_2(S)$ **do**

$S \cup v \rightarrow S'$;

$NEWSET \cup S' \rightarrow NEWSET$;

end for

Output NEWSET.

Figure 2.10. Expansion of an ETS S of Size a to ETSs of Size $a+1$.

Require G, L_{in}, k.

$L_{out}^i = S \in L_{in}, |S| = i$ for $i = 2, ..., t.$;

for $i = 2$ to k **do**

for all $S \in L_{out}^i$ **do**

NEWSET=OneExpansion(S);

$L_{out}^i \cup NEWSET \rightarrow L_{out}^{i+1}$;

end for

end for

Output $L_{out} = L_{out}^2 \cup ... \cup \mathrm{L}_{out}^k$.

Figure 2.11. Expansion of Input ETSs to ETSs of Size up to t.

Table 2.2. Elementary Trapping Sets of Tanner (155,64) Code

ETS Class	**LSS_g**	**LSS_g+2**	**LSS_g+4**	**LSS_g+6**	**Total Multiplicity**
(4,4)	465				465
(5,3)	155				155
(5,5)		3720			3720
(6,4)		930			930
(6,6)			22630		22630
(7,3)		930			930
(7,5)			8835	7440	16275
(7,7)				140430	140430
(8,2)		465			465
(8,4)			5115		5580
(9,3)			1860		1860
(10,2)			465	930	1395

works correctly. Since the girth of Tanner code, g, is 8, cycles with size 8, 10, 12, and 14 $(g, g+2, g+4, g+6)$ are used as input the algorithm. The total multiplicity of the ETS of Tanner Code is listed in Table 2.4. These numbers are the same as given in [57].

2.5. Importance Sampling

Density evolution is used to estimate the performance of block codes [60]. However, due to the error floor problem that LDPC codes have, they make mistakes in estimating the performance of LDPC codes. Different importance sampling (IS) approaches have been developed to determine the error floor performance since Monte Carlo simulation at high SNR values is also not achieved in a feasible time [24–26]. In addition to that, with the help of IS methods, the contribution of each TS class to the error floor can be estimated. Thus, the dominant TS can be determined with IS methods. To determine dominant trapping sets, the slightly modified version of the algorithms given in [37] and [57] are applied in this thesis.

In general, a two-pronged attack for determining the error floor is used. In the first step, the dominant trapping sets are searched via different methods such as biased cycles, exhaustive search, expanding trapping sets method. In the previous section, expanding trapping sets methods were explained in detail. In the second step, IS is applied to only one trapping set from each class and the total FER of the code is estimated by adding the FER contribution of each trapping class.

A classic Monte Carlo simulation generates a lot of independent channel realizations, and records the output of each trial. Let's assume that there are N independent channel realizations, then the output of each trial can be shown as Bernoulli random variable,

$$Z_i = \begin{cases} 1, & \text{if the decoder fails at trial i,} \\ 0, & \text{otherwise.} \end{cases} \tag{2.38}$$

These Bernoulli indicator variables then yield the naive Monte Carlo estimate as

$$p_{MC} = \frac{1}{N} \sum_{i=1}^{N} Z_i. \tag{2.39}$$

The goal of importance sampling is to modify the original density such that infrequent errors become more likely. For a Gaussian density, it is sufficient to shift the mean as shown in Figure 2.12. The shift value of the mean is presented as λ for the rest of the paper.

If we draw samples according to biasing density f_{bias} instead of the original density, the importance sampling estimator is computed as

$$p_{IS} = \frac{1}{N} \sum_{i=1}^{N} Z_i w_i(y, \lambda), \tag{2.40}$$

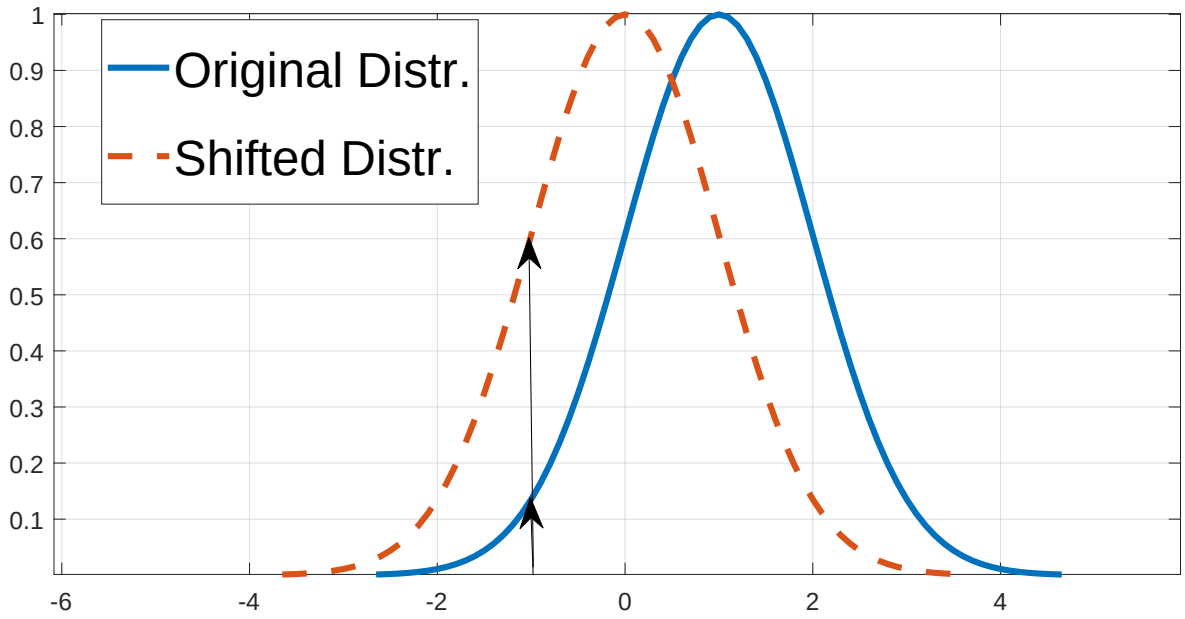

Figure 2.12. Example of Error Floor Estimation

where w_i, reweighting function, is defined as

$$w_i(y,\lambda) = \frac{f_{bias}(y)}{f_{original}(y)}. \tag{2.41}$$

When f_{bias} is employed, the decoder converges the trapping sets. When we assume that an the additive white Gaussian noise (AWGN) channel using binary phase-shift keying (BPSK) modulation is used, the importance sampling re-weighting function for trapping set with K variable nodes is computed as

$$w(y,\lambda) = \frac{e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{K}(y_j-1+\lambda_j)^2}}{e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{K}(y_j-1)^2}}. \tag{2.42}$$

If we assume that all one's code-word is sent to the decoder, and noise with distribution f_{bias} is added to only variable nodes of that trapping sets, Equation (2.42) can be

written as

$$w(\lambda) = \frac{e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{K}(1-1+\lambda_j)^2}}{e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{K}(1-1)^2}}, \tag{2.43}$$

$$w(\lambda) = e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{K}(\lambda_j)^2}, \tag{2.44}$$

$$w(\lambda) = e^{-\frac{K\lambda^2}{2\sigma^2}}. \tag{2.45}$$

Let's assume that ϵ_T denote the set of inputs that give rise to a failure on a trapping set T. Then,

$$FER = \sum_{T} Pr\{\epsilon_T\} \tag{2.46}$$

When we use a biasing input, Equation (2.46) can be written as

$$FER = \int_{-\infty}^{\infty} Pr\{\epsilon_T \mid \lambda\} w(\lambda) d\lambda, \tag{2.47}$$

$$FER = \int_{-\infty}^{\infty} Pr\{\epsilon_T \mid \lambda\} e^{-\frac{K\lambda^2}{2\sigma^2}} d\lambda. \tag{2.48}$$

To compute FER, we need to determine the function $Pr\{\epsilon_T \mid \lambda\}e^{-\frac{K\lambda^2}{2\sigma^2}}$. The function is expected to decay quickly as λ moves away from the maximum. Therefore, the integral is largely depend on λ's that are near the maximum. The first part of integral,$Pr\{\epsilon_T \mid \lambda\}$ is estimated using a Monte Carlo simulation. The second part is analytically computed easily.

Figure 2.13 shows the summary of the error floor evaluation of an (8,2) trapping set in the Tanner graph at 6.5dB. The curve labeled "Simulated Frame Error Rate" indicates the probability of an otherwise random input failing on the given trapping set as a function of s. The curve labeled "Input Prob Rate" shows the probability of biased input. The curve named "Contributed Fail Rate" gives the product of these two curves. The total failure rate on (8,2) trapping sets is given as the integral of "Contributed Fail Rate" with respect to biased value (λ). For each TS given in Table

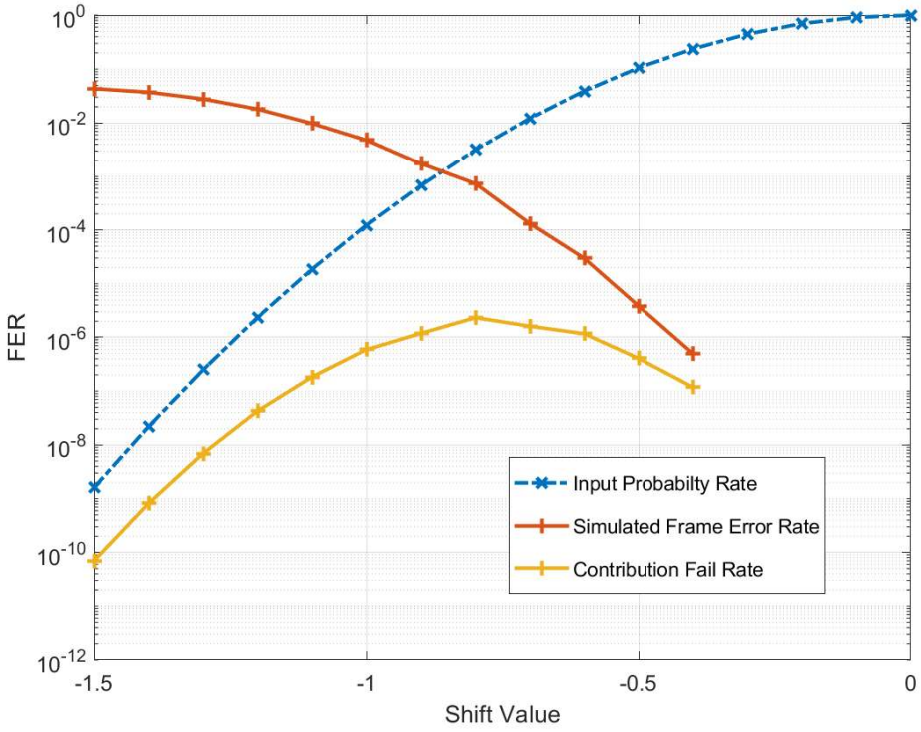

Figure 2.13. Example of Biasing Densities

2.2, importance sampling is applied. For the 5.5 dB SNR point, the contribution of each TS is presented in Table 2.3. The failure rate shows the average contribution of one TS to the error floor. Its multiplication with the total number gives the total contribution for that TS class. As it can be seen in Table 2.3, (7,3) TS and (8,2) TS are the most dominant trapping sets for this LDPC code.

For each dB level, the total contribution of each dominant trapping sets as given in Table 2.3 can be estimated with importance sampling. Thanks to this, we can estimate the error-correcting capacity of LDPC codes based on its trapping sets distribution. Bit error rate (BER) performance estimation for (155,64) Tanner code is given in Figure 2.14. Importance sampling and Monte Carlo simulation give similar results in the region where trapping sets are dominant.

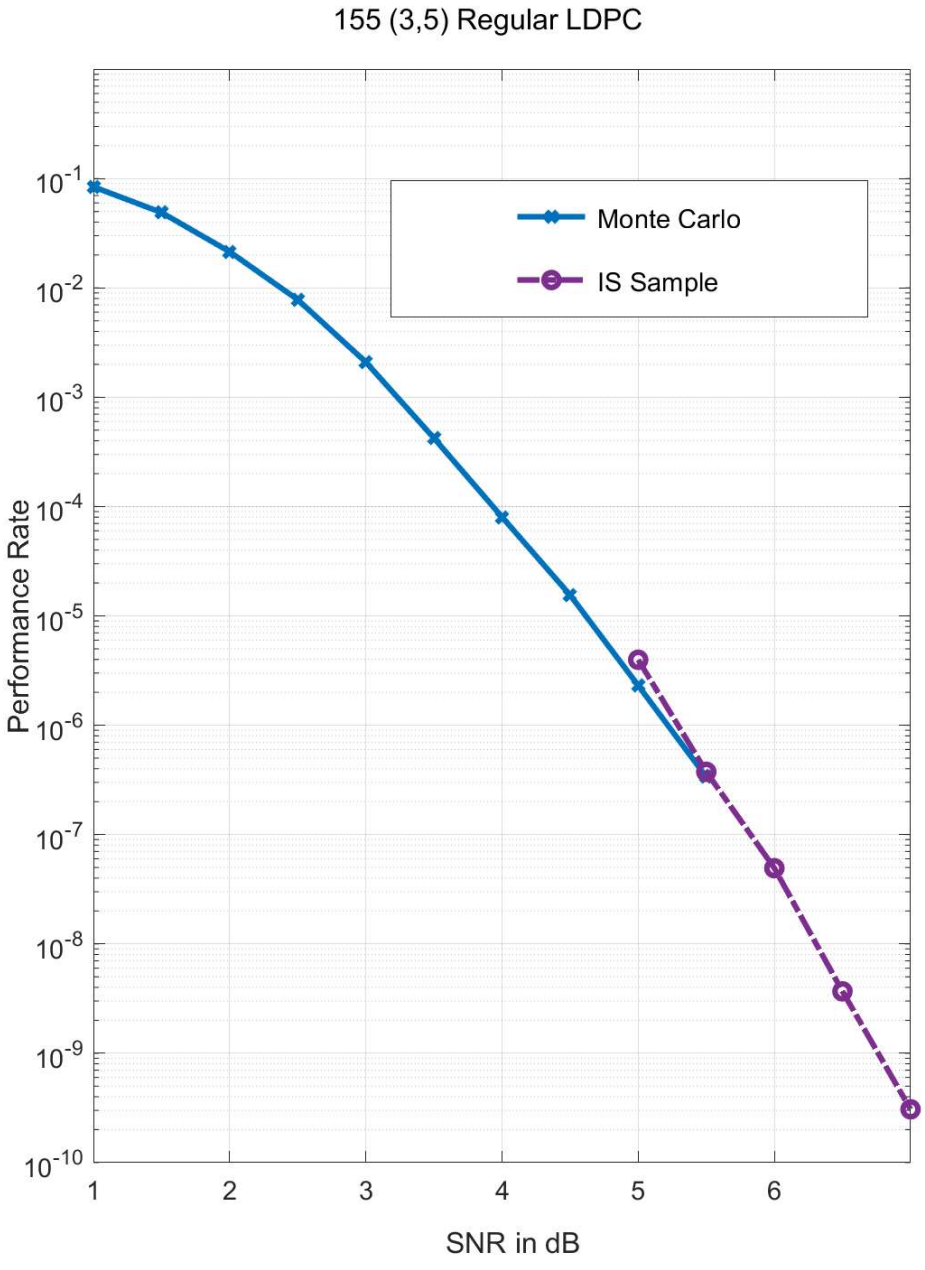

Figure 2.14. Example of Importance Sampling

Table 2.3. The Contribution of Elementary Trapping Sets to Error Floor

TS Class	**Number of Trapping Sets**	**Failure Rate**	**Total Contribution**
TS(4,4)	465	1.01e-12	4.7e-10
TS(5,3)	155	6.59e-11	1.02e-08
TS(5,5)	3720	-	
TS(6,4)	930	6.62e-13	6.16e-10
TS(6,6)	22630	-	-
TS(7,3)	930	1.35-10	1.25e-07
TS(7,5)	7440	-	-
TS(7,7)	140430	-	-
TS(8,2)	465	3.28e-10	1.52e-07
TS(8,4)	5580	2.54e-13	1.42e-09
TS(9,3)	1860	2.83e-12	5.27e-09
TS(10,2)	1395	1.41e-11	1.96e-08
		Total Rate	**3.16e-07**

3. Adaptive Linear Decoding Algorithms

In this thesis, the LP decoder is defined in a similar way to the LP decoder proposed in the [18] study so that the nodes that create cycles can be found more efficiently. Redundant parity check equations, which are proven to produce valid constraints, were efficiently found with the proposed integer programming model and a heuristic method in this work.

3.1. Integer Adaptive Linear Decoder (IALP)

When a redundant parity check node is produced, one of the odd-weighted vectors belonging to the check node should not satisfy the constraint in Equation (3.1) to create a valid constraint.

$$g_i = \sum_{x_j \in S} (1 - x_j) + \sum_{x_j \in (N(c_i) \setminus S)} x_j \geq 1 \ . \tag{3.1}$$

Lemma 3.1. *A lower bound of the smallest value that g_i can take for any check node can be found in a simple way. When all variable nodes greater than 0.5 belong to the S set and all variable nodes less than 0.5 do not belong to the S set, a lower bound is found. If the S set is odd, the result is the smallest value of g_i*

Proof: When the value of the x_i is greater than 0.5, $1 - x_i$ will be less than x_i. In this case, x_i must belong to the S set according to Equation (3.1) in order to minimize the value of g_i. When the value of x_i is less than 0.5, x_i will be less than $1 - x_i$. Under these conditions, the x_i must belong to the $N(c_j) \setminus S$ set to minimize g_i. □

Lemma 3.1 indicates that the lower bound of any check node can be found by only knowing the values of the variable nodes adjacent to the check node. Furthermore, if the number of variable nodes that have a value greater than 0.5 is odd, then this lower limit is equal to the smallest value $g(x)$ can take. With these in mind, an optimization

model has been developed that looks for a c_j check node with the smallest $g_j(x)$ value that is less than 1.

The proposed integer optimization model can be explained using a small example. Let's assume that the standard LP decoder finds a vector with fractional values as

$$\mathbf{x} = [0.9 \quad 1 \quad 0.12 \quad \dots \quad 0 \quad 0.60]. \tag{3.2}$$

In the light of Lemma 3.1, we can find the contribution of each variable node, $\hat{x}_i$, to the lower bound of the function $g_j(x)$ as

$$\widehat{x_i} = 0.5 - |(x_i - 0.5)|, \qquad i = 1, 2, ..., n. \tag{3.3}$$

When x_i is greater than 0.5, then $\widehat{x_i} = 1 - x_i$ will be active, and x_i will belong to odd set. In this example, the contribution vector will be

$$\widehat{\mathbf{x}} = [0.1 \quad 0 \quad 0.12 \quad \dots \quad 0 \quad 0.40]. \tag{3.4}$$

A $\mathbf{z}$ vector indicating the position of variables nodes that are greater than 0.5 is defined as

$$\mathbf{z} = [1 \quad 1 \quad 0 \quad \dots \quad 0 \quad 1]. \tag{3.5}$$

When x_i belongs to the odd set, then z_i takes 1; otherwise, z_i is equal to zero. In the light of these parameters, an integer programming model has been developed as

$$\text{minimize} \sum_{i=1}^{n} \omega_i \widehat{x_i} \tag{3.6}$$

$$\text{constraints:} \sum_{j=1}^{m} \alpha_j H_{j,i} + \omega_i = 2k_i \qquad i = 1, 2, ..., n. \tag{3.7}$$

$$\sum_{i=1}^{n} \omega_i z_i + 1 = 2\eta, \tag{3.8}$$

$$\sum_{j=1}^{m} \alpha_j \geq 2, \tag{3.9}$$

$$k_i, \eta \in \mathbb{Z}, \tag{3.10}$$

$$\omega_i, \alpha_j \in \{0, 1\}. \tag{3.11}$$

To produce redundant check nodes, some rows of the **H** are summed. The parameter α_j determines that the j^{th} row will be active or passive for the summation. If α_j becomes 1, it indicates that the j^{th} check node will be one of the check nodes that will be summed to generated new redundant check node. The ω_i parameter is used to show that whether the i^{th} variable node is in the neighbor of the generated redundant check node or not. With the help of constraint in Equation (3.7), the rows of **H** matrix are summed, and the variable nodes that are in the neighbor of the generated redundant check nodes are determined. With Equation (3.8), it is ensured that the number of variable nodes belonging to the set S is an odd number. Constraint in Equation (3.9) forces that at least two check nodes are summed to generate the redundant check node.

When the IP model is employed, it is not necessarily required to find the optimum solution. When a feasible solution with the objective function value that is less than 1, we know that it can generate a valid constraint for the current LP model. Therefore, the algorithm is terminated when the objective value is less than 1. If the valid constraint is added to the current LP, a better solution could be found with the LP model. According to that, the whole algorithm is given in Figure 3.1.

Require Parity-check matrix (**H**), received vector (**r**), time threshold (t_{max}).
Use the standard LP decoder, and obtain a solution vector, **x**;
Produce the contribution vector, $\hat{\mathbf{x}}$, as $\widehat{\mathbf{x}} = 0.5 - |(\mathbf{x} - 0.5)|$;
while The solution isn't integer **do**
  Generate a valid constraint using the IALP model;
  Add the valid constraint to the last LP model;
  Solve the problem, and obtain new solution, **x**;
  Produce the contribution vector, $\hat{\mathbf{x}}$;
  if Time limit is exceeded **then**
    break while
  end if
end while
Output the maximum-likelihood code-word is found or a lower bound was found due to the time limit.

Figure 3.1. IALP Decoder.

3.2. Heuristic Adaptive Linear Decoder (HALP)

For the redundant check node to be a valid constraint, the collected check nodes must contain at least one cycle [19]. However, it is a tricky problem to find all of these cycles. In addition to that, there is no guarantee that the redundant check node, which will be obtained by summing the check nodes from the cycle, generates a valid constraint. Therefore, the right cycles should be found to produce a valid constraint. In the heuristic adaptive linear decoder (HALP) proposed in this thesis, check nodes are redefined using auxiliary variables so as not to change the structure of the basic polytope. Cycles were found quickly and efficiently with the new definition. It has also been proven that the resulting redundant node can create a valid constraint under certain conditions. For a check node, c_1, that has degree 6, let's assume that its representation is given as

$$[v_1 + v_2 + v_3 + v_4 + v_5 + v_6]_2 = 0. \tag{3.12}$$

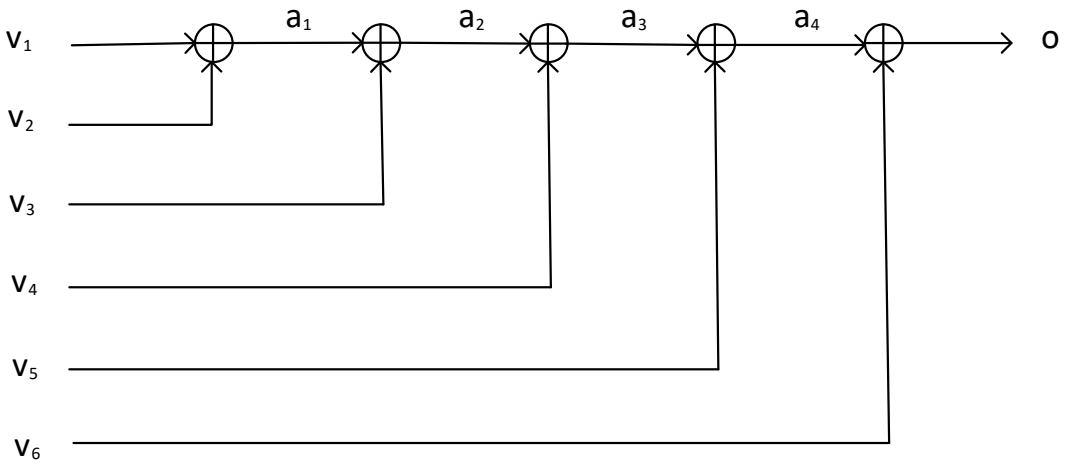

Figure 3.2. Representation of Check Nodes with Auxiliary Variables

The linear sum of the c_1 check node can be represented as in Figure 3.2 using auxiliary variables (a_1, a_2, a_3, a_4). Since all computation is done in binary, (3.12) can be represented by four equations as given in

$$[v_1 + v_2 + a_1]_2 = 0, \tag{3.13}$$

$$[a_1 + v_3 + a_2]_2 = 0, \tag{3.14}$$

$$[a_2 + v_4 + a_3]_2 = 0, \tag{3.15}$$

$$[a_3 + v_5 + v_6]_2 = 0. \tag{3.16}$$

As a result of these operations, it is shown that instead of a check node with degree of 6, 4 different check nodes with degree of 3 can be used.

Lemma 3.2. *While the number of constraints forming the basic polytopes can be expressed exponentially proportional to the check node degree, d_c, for the standard LP decoder, the basic polytope produced using auxiliary variables can be expressed with the number of constraints that is linearly proportional to d_c value.*

Proof: It is necessary to prohibit all odd vectors to define the local polytope of a check node. Therefore, exactly $2^{(d_c-1)}$ constraints are required. On the other hand,

instead of using one check node, $d_c - 2$ check nodes with degree 3 can be used. In this case, $2^{3-1} = 4$ constraints are required for each check node. Since $(d_c - 2)$ check nodes are created in total, the local polytope of the check node can be displayed with $(4 \times (d_c - 2))$ constraints using this method. $\square$

Defining the basic polytope by auxiliary variables will not only reduce the number of constraints but will also make it easier to find the redundant check nodes that produce valid inequalities. The local polytope of Equation (3.13) is defined by constraints given as

$$v_1 + v_2 \geq a_1, \tag{3.17}$$

$$v_1 + a_1 \geq v_2, \tag{3.18}$$

$$v_2 + a_1 \geq v_1, \tag{3.19}$$

$$v_1 + v_2 + a_1 \leq 2. \tag{3.20}$$

Lemma 3.3. *When the standard LP decoder decodes the basic polytope identified using auxiliary variables, the result vector may contain fractional values. In this case, all check-equations will have three variable nodes in their neighbor, and they will belong to one of three different classes*

(1) The class of 0F (0-fractional)
The set of the check nodes that don't contain fractional variable nodes, but they contain only integer variable nodes.

(2) The class of 2F (2-fractional)
The set of the check nodes containing two fractional variables nodes and one integer variable node.

(3) The class of 3F (3-fractional)
The set of the check nodes containing only three fractional variables nodes.

Proof: It is sufficient to demonstrate that any check node does not contain one fractional variable node and 2 integer variables node. The optimum solution of the linear programming method should be a corner point of the basic polytope. In this case, the optimum point corresponds to the intersection of at least two surfaces. Without loss of generality, we can use the local polytope of Equation (3.13). So at least 2 of the constraints indicated by Equations (3.17), (3.18), (3.19) and (3.20), needs to be provided with the equality. Therefore, the sum of the value of two variable nodes belonging to a check node must equal the value of the third variable node of this check node. The sum of two integers will be an integer, and the sum of the one integer and one fractional number will be fractional. Therefore, it is not possible for only one of the variable nodes to be fractional.□

Constructing the polytope with the auxiliary variables not only reduces the number of constraints but also ensures that the value of the variable nodes in the solution founded by LP decoder is formed in accordance with some certain rules. To show these rules, the local polytope of Equation (3.13) will be used.

Lemma 3.4. *If the value of the integer variable of a check node from the class 2F is 0, the values of the other two fractional variables are equal.*

Proof: If a_1 is accepted as 0 without losing generality for Equation (3.13), Equation (3.18) and Equation (3.19) ensures that values of v_1 and v_2 are equal. □

Lemma 3.5. *If the value of the integer variable of a check node belonging to class 2F is 1, the sum of the other two fractional variables is 1.*

Proof: If a_1 is accepted as 1 without losing generality for Equation (3.13), Equation (3.17) and Equation (3.20) ensures that the sum of v_1 and v_2 are one. □

Lemma 3.6. *The sum of the values of two fractional variables of a check node of the 3F class is equal to the value of the other variable.*

Proof: At least 2 of the constraints indicated by Equations (3.17), (3.18), (3.19) and (3.20), needs to be provided with the equality. Therefore, the sum of the value of two variables belonging to a check node must equal the value of the third variable of this check node.

After adding check nodes to create a redundant check node, the number of fractional variables connected to the redundant check node can be controlled. This authority over the number of fractional variables gives some significant advantages to find valid constraints.

3.2.1. Cycles generated with check equations only from 2F class

The redundant check node obtained from the sum of the two check nodes of the 2F class will belong to the 2F class if they have one common fractional variable. The check node of class 2F adjacent to each other with fractional variables can easily be found on the Tanner graph ($\mathcal{O}(1)$). If these check nodes consist of a cycle, the redundant check node, $c_{redundant}$, that is created by summing these check nodes in binary won't consist of any variable node with fractional value as given in Figure 3.3. The filled circles present the variable node with a fractional value. These variable nodes will be removed when all check nodes from the cycle are added in binary. If the number of the variable nodes from $N(c_{redundant})$ with value 1 is odd, then a S set is created such that the value g_i will be zero. Therefore, a strong valid constraint is generated with the local polytope of $c_{redundant}$.

3.2.2. Cycles with one check equations from the 3F class

Cycles consisting of check nodes from 2F and containing one check node from the 3F class will have only one fractional variable if the check nodes are added in binary. Figure 3.4 shows that every fractional variable will be removed except for the one variable node. Therefore, a redundant check node,$c_{redudant}$, will be created such that it consists of only one fractional variable node

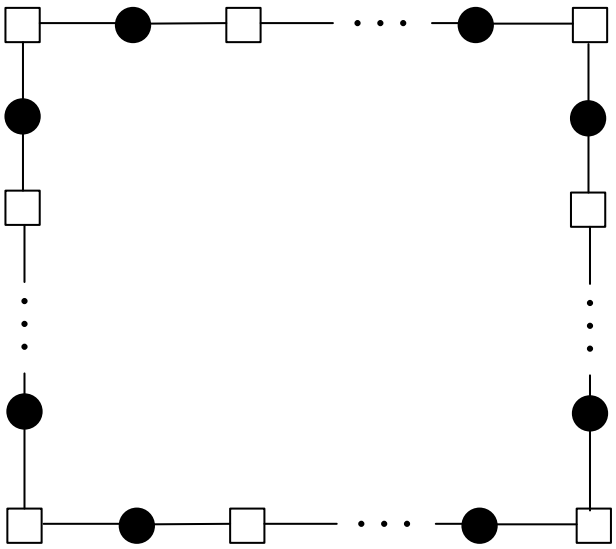

Figure 3.3. Example of a Cycle Contains Only Check Nodes From 2F Class

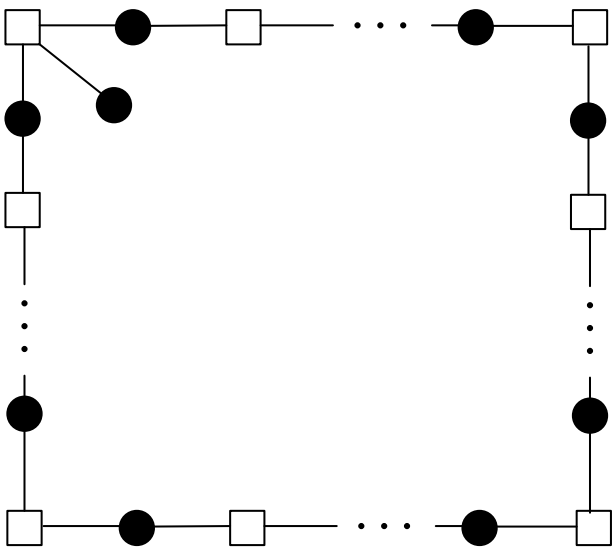

Figure 3.4. Example of a Cycle Contains One Check Node From the 3F Class

Lemma 3.7. *If a redundant check node, $c_{redundant}$, contains only one fractional variable, $\hat{x}$, then its local polytope generates a valid constraint.*

Proof: There are two case:

(1) Let's assume that there are odd number of integral neighbors of the redundant check node with value 1. Then if the set S is constructed with only the variables with value 1,

$$g_i = \sum_{x_j \in S} (1 - x_j) + \sum_{x_j \in (N(c_i) \setminus S)} x_j = \hat{x} \tag{3.21}$$

will be satisfied.

(2) Let's assume that there are even number of integral neighbors of the redundant check node with value 1. Then if the set S is constructed with only the variables with value 1 and $\hat{x}$,

$$g_i = \sum_{x_j \in S} (1 - x_j) + \sum_{x_j \in (N(c_i) \setminus S)} x_j = 1 - \hat{x} \tag{3.22}$$

will be satisfied.

For both cases, the value of g_i less than 1. Therefore, there must be a valid constraint.

3.2.3. Cycles with two check node from the 3F class

The check node, which is the sum of the cycles that contain two check nodes from the 3F class, will contain only two fractional decision variables.

Lemma 3.8. *If a redundant check node contains two variable fractional nodes and their sum is different from 1, or the values of the variables are not equal, the local*

polytope of the check node does not contain the last solution that the LP decoder has found. Therefore, it generates a valid constraint.

Proof: In order to minimize the g_i value given in Constraint 2.35, it can be assumed that decision variables with a value of 1 belong to the set S, and the decision variables with a value of 0 belong to the set $(N(c_i) \backslash S$. In this case, the integer variables do not contribute to g_i. Therefore, except for fractional variables, the value of g_i will be zero. The remaining fractional variables should be added to sets such that the odd-weight set must be odd. The remaining two fractional decision variables (assuming v_1 and v_2) can be in 3 different states:

1) $v_1 \leq 0.5$ and $v_2 \leq 0.5$. If set S has an odd number of elements, these two variables will be not included in the odd-weight set (S) and g_i would be less than 1. If the S set has an even number of elements, one of the variables must belong to the S set. Without lost of generality, suppose $v_1 \geq v_2$. If v_1 belongs to the S set, $g_i = 1 - v_1 + v_2$ will be satisfied. Then, the only condition for g_i to be equal or greater than 1 will be $v_1 = v_2$.

2) $v_1 \geq 0.5$ and $v_2 \leq 0.5$. If the S set has an odd number of elements, these two decision variables must be included in either the S set or the other set. If the closest one to the value of 0.5 is added to the set to which it will contribute more (let's assume that v_2 is the closest one), the constraint will be

$$g_i = (1 - v_1) + (1 - v_2) = 1 + (0.5 - v_2) - (v_1 - 0.5). \tag{3.23}$$

g_i will be less than 1 because v_2 is closer to 0.5 than v_1. If the values of two fractional decision variable are at the same distance from 0.5, g_i takes the value of 1. $v_1 - a = v_2 + a = 0.5$, then $v_1 + v_2 = 1$.

3) $v_1 \geq 0.5$ and $v_2 \geq 0.5$. If set S has an odd number of elements, these two variables will be included in the odd weight set (S),

$$g_i = (1 - v_1) + (1 - v_2) = 2 - v_1 - v_2 \tag{3.24}$$

will be satisfied. Since the sum of v_1 and v_2 is greater than 1, g_i will be less than 1. If the S set has an even number of elements, one of the variables must belong to the S set. Without lost of generality, suppose $v_1 \geq v_2$. If v_1 belongs to the S set, $g_i = 1 - v_1 + v_2$ will be satisfied. Then, the only condition for g_i to be equal or greater than 1 will be $v_1 = v_2$.

As shown in the examples above, it appears that the number of check nodes of the 3F class in the cycles directly affects the difficulty of finding valid inequality. The heuristic adaptive linear programmer decoder intuitively searches the cycle with the minimum number of nodes of the 3F class and finds valid inequalities in a short time with this method.

The proposed HALP decoder is given in Figure 3.5. In the IALP decoder, finding the valid constraints could take a long time due to the integer programming optimization method. With HALP decoder, valid constraints were found very quickly. To observe the time performance of the HALP decoder, the time performance of the ACG-ALP decoder given in [21], and IALP decoder were compared. Tanner code (155.64), based on Tanner's pseudo-matrices, commonly used for simulations, was used [61]. To compare the time performance of the decoders, different vectors were produced on the AWGN channel with 1.8 dB signal to noise ratio (SNR), and the change of the mean objective function values of LP models over time is given in Figure 3.6. As can be observed from Figure 3.6, the HALP decoder has increased the objective function value of the LP model faster than other decoders. However, after some point, the HALP decoder cannot increase the objective function value of the LP model. Therefore, error performance is expected to be lower, especially compared to the ACG-ALP decoder. To compare the frame error rate performance of decoders, (155,64) Tanner code is

Require Parity-check matrix (**H**), received vector (**r**), time threshold (t_{max}).
Redefine the parity-check matrix using auxiliary variables;
Use the standard LP decoder, and obtain a solution;
while The solution isn't integer **do**
  Search cycles that there are exactly y check nodes from 3F class;
  if Cycles generates a redundant check node that has valid constraint **then**
    Add the valid constraint to the base polytope;
    Solve the problem, and obtain new solution;
  else
    Increase y value and go to STEP 4;
  end if
  if Time limit is exceeded **then**
    break while
  end if
end while
Output The maximum-likelihood code-word is found or a lower bound was found due to the time limit.

Figure 3.5. HALP Decoder.

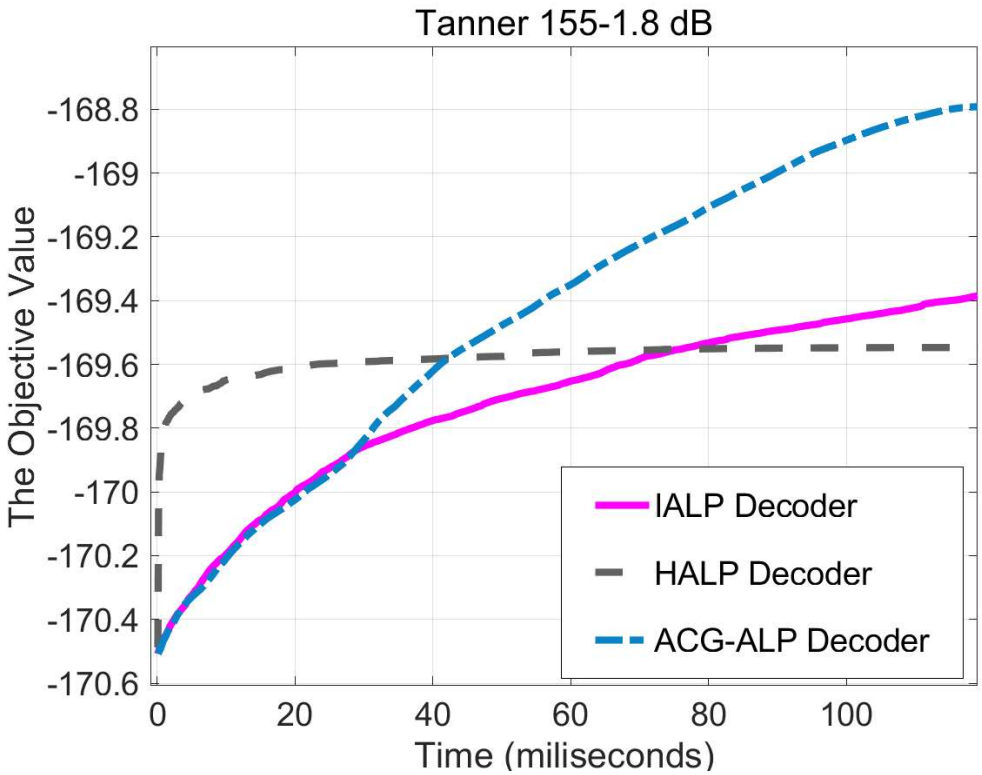

Figure 3.6. Time Performance

used. Figure 3.7 gives error-correcting performance graphs for different SNR values of decoders. Although HALP decoder has worse error performance than ACG-ALP decoder, it has been shown that it finds the valid constraints much faster than other algorithms in early steps. However, it becomes difficult to find valid constraints after some steps. Therefore, it is aimed that the HALP decoder should work with other decoders to find valid constraints instead of working as a single decoder. The proposed ALP decoders were published in [22] and [23].

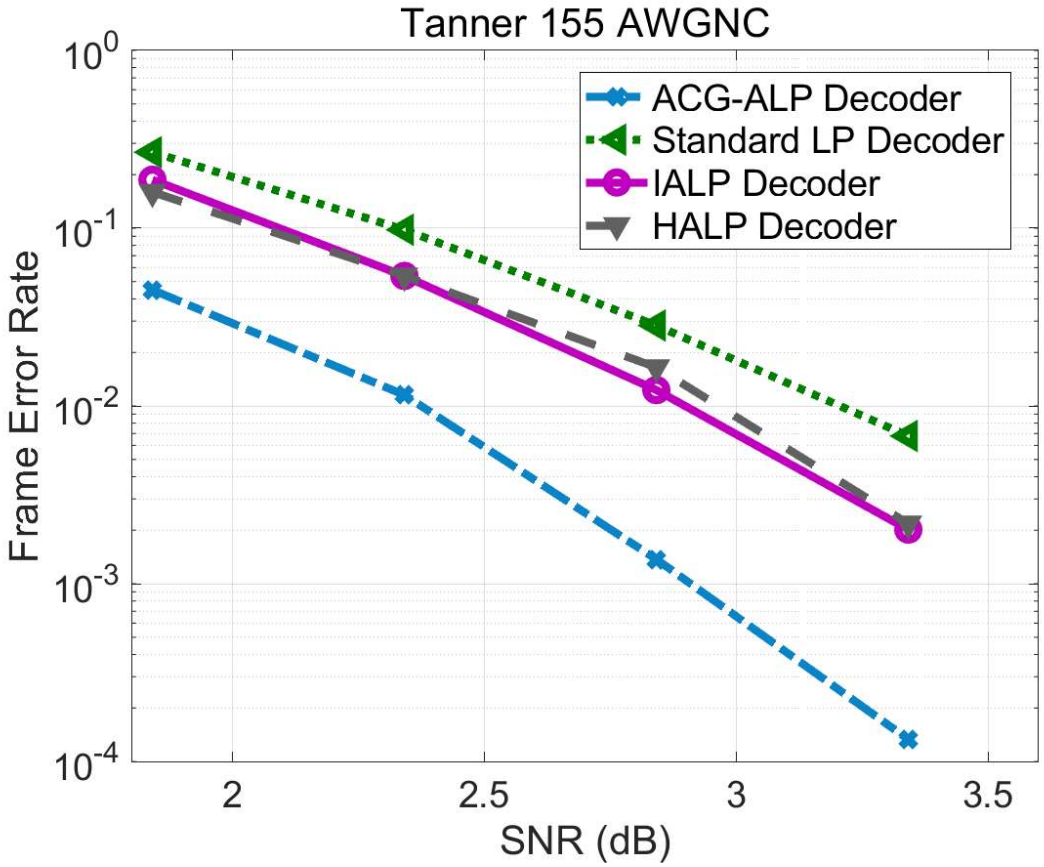

Figure 3.7. Frame Error Rate Performance

4. Design of LDPC Codes

In this section, two different simulated annealing algorithms are proposed to construct regular LDPC codes. Proposed progressive edge growth based simulated annealing algorithm and RandPEG-noTS53 based simulated annealing algorithm are explained in detail. Before explaining the developed algorithm, the PEG algorithm and its advanced version, Rand-PEG algorithm, are briefly mentioned. After that, the RandPEG-noTS53 algorithm which focuses directly on reducing trapping sets numbers is described.

4.1. Progressive Edge Growth Algorithm

Over the past few decades, many algorithms have been employed to design LDPC codes. Progressive edge growth algorithm is one of the first proposed algorithms [27]. PEG algorithm constructs the Tanner graph step by step. When determining the first edge of a variable node, the PEG algorithm selects the check node with the lowest degree in the graph so far. When determining the second and subsequent candidate edges, the cycles that can be created by the candidate edges are considered. The set of check nodes that are the farthest from the variable node is determined by using a breadth-first search (BFS) algorithm. A sub-graph spreading from variable node v_j is given in Figure 4.1. One of the check nodes from the set that consists of the farthest check nodes is selected and connected to the variable node. After the Tanner graph is updated by establishing the best-choice edge, the placement procedure is repeated for the remaining edges. Selecting the farthest check nodes provides the largest possible cycle.

For a given variable node v_j, its neighbor within depth l, $N_{v_j}^l$, is defined as the set of consisting of all check nodes reached by a tree spreading from variable node v_j within depth l. The complementary set of $N_{v_j}^l$, $\overline{N}_{v_j}^l$, is defined as $C \setminus N_{v_j}^l$. If v_j is connected to check nodes belong to $N_{v_j}^l$, the shortest cycle passing through this new

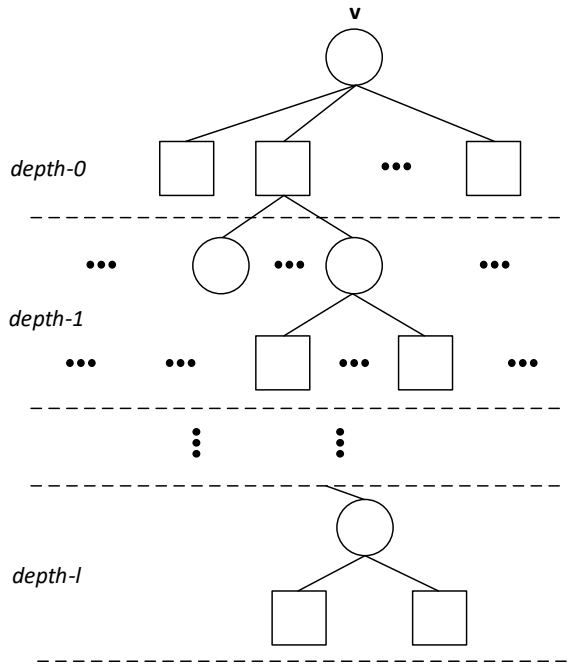

Figure 4.1. A Subgraph of a Variable Node v_j: Depth-l Tree

edge is guaranteed to be no shorter than 2(l+2). PEG algorithm uses these results, and tries to build LDPC codes that have girth $2(l+2)$. PEG algorithm suggested in [27] is presented in Figure 4.2. The check node selection procedure from $\overline{N}_{v_j}^{l}$ causes different results. Although girth value doesn't change with various selection procedure, the number of cycles with girth size changes significantly. In the following years, many PEG-bases algorithms have been suggested [28, 29, 62, 63].

4.2. Randomized Progressive Edge Growth Algorithm

Randomized progressive edge growth algorithm is slightly updated version of PEG algorithm, [64]. Instead of spanning to the maximal possible depth, it spans the tree only up to a certain depth, l_{max}. This depth is determined by the user at the beginning of the algorithm.

When the construction tree is spanned up to the depth l_{max}, the candidate check node that will be connected to the active variable node is determined as

Require Size of **H** matrix (m × n), degree distribution

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

  for $k = 0$ to $d_{v,j} - 1$ **do**

    if $k = 0$ **then**

      Create edge (c_i, V_j) and assign to $E^0_{v,j}$; $E^0_{v,j}$ is the first edge connected to v_j and c_i is the check node that has the lowest check node degree under the current graph setting.

    else

      Expand a subgraph from v_j up to depth l under the current graph setting such that the cardinality of $N^l_{v,j}$ stops increasing but is less less than m, or $N^l_{v,j}$ is not empty set but $N^{l+1}_{v,j}$ is empty, then edge(c_i, v_j) assign to $E^k_{v,j}$, where $E^k_{v,j}$ is the k.th edge incident to v_j, and c_i is a check node picked from the set $N^l_{v,j}$ having the lowest check node degree;

    end if

  end for

end for

Output H matrix

Figure 4.2. Progressive Edge Growth (PEG) Algorithm.

(i)- If there are candidates at depth l_{max}, then remove all the candidates that are not exactly at the depth l. If there is no candidate, decrease l_{max}.

(ii)- The number of cycles that would be created for each candidate is computed. The candidates that creates the minimum number of cycles are found.

(iii)- The lowest degree of the candidates that creates the minimum number of cycles is computed. Then, the check node with the lowest degree that creates the minimum number of cycles is selected.

With this method, [64] decreased the multiplicity of the girth, i.e the number of cycles with the length equal to the girth.

4.3. PEG Based Simulated Annealing Algorithm

PEG algorithms generate a completed graph, which is locally optimized in terms of cycle size. If the d edges are removed from the completed graph, and then it is filled with the PEG algorithm, a solution which is only d edges different from the initial solution is found. If the removed edges are selected from the cycles with the girth size, the new solution could have less number of the multiplicity of the girth. However, this method cannot be guaranteed to give better results. However, with a probabilistic technique, this method can approach the global optimum.

Simulated Annealing algorithm is a probabilistic technique that seeks the global optimum solution of a problem. Instead of techniques such as hill-climbing, gradient descent, it is preferred for problems where the global optimum results are much more valuable than the local optimum results. The simulated annealing algorithm was used for the first time in 1983 by Kirkpatrick, Gelatt Jr. and Vecchi for the traveling salesman problem [65]. The name of annealing comes from a technique involving heating and controlled cooling of a material to increase the size of the crystals and reduce their imperfections. As the solution set is discovered, the decrease in the possibility

of accepting worse solutions is interpreted as cooling. Accepting worse solutions is the basic idea of the simulated annealing algorithm. With this method, the algorithm can be extracted from the local optimum point.

The general working principle of simulated annealing algorithms is given in Figure 4.3. In each step, the algorithm randomly selects a solution close to the previous solution and measures its quality. It accepts the new solution as the current solution or does not change the current solution. It decides these states according to the objection function value and the current temperature value. If the new solution is better than the current solution, it is accepted as the current solution. On the other hand, a worse solution is accepted or rejected according to the acceptance probability function calculated based on the current temperature value. By reducing the temperature value in each iteration, the probability of acceptance of worse solutions decreases over time. For the acceptance probability function, Metropolis' criterion based on Boltzman's probability is applied. The acceptance probability is determined with

$$P(A) = e^{-\Delta E/(\varsigma T)}, \tag{4.1}$$

where ΔE is the difference between objective function value of the current solution and candidate solution, ς is Boltzmann's constant, and T is the temperature, respectively.

For the rate of convergence of the simulated annealing algorithm, the generation mechanisms of candidate solutions and the distance between the candidate solution and current solution should be determined properly. Otherwise, the algorithm could get stuck at a local optimum point. The distance is defined as the number of edges that will be replaced and is represented as d. To generate the neighbors, firstly d edges are removed from the bipartite graph. After that, the PEG algorithm fulfills the bipartite graph. The edges that will be removed are chosen according to their contributions to the objective value. For the PEG based simulated annealing algorithm, the objection function is determined as the total number of cycles with girth size.

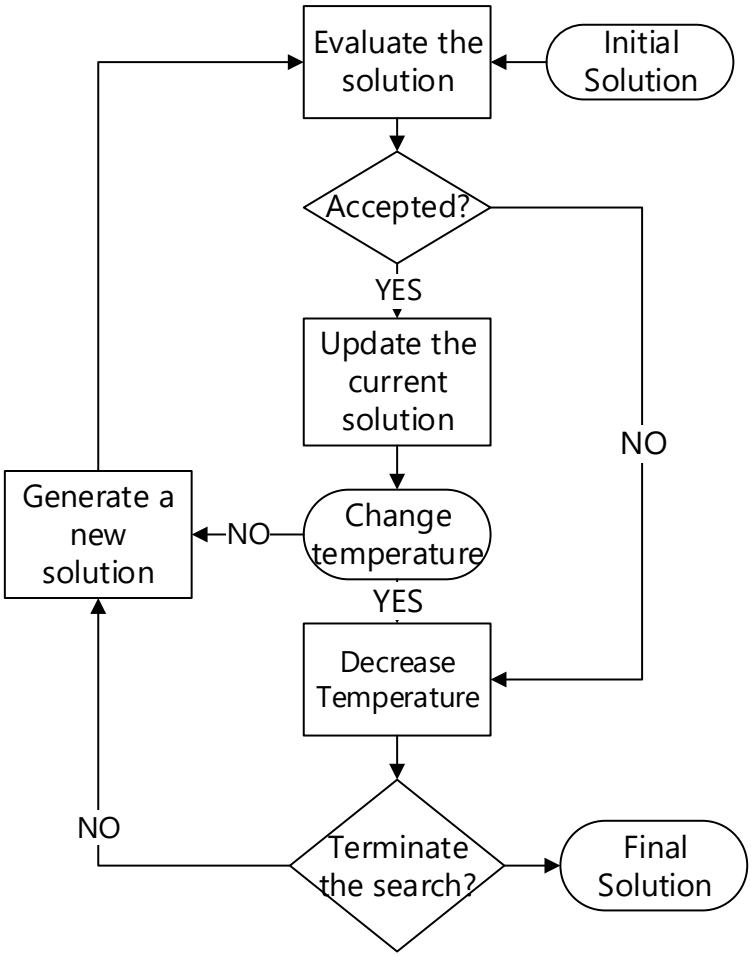

Figure 4.3. Flowchart of a Simulated Annealing Algorithm

To avoid getting stuck at a local optimum point, the distance should be large enough that the algorithm should jump to another region. However, if the distance is too large, convergence to a global optimum becomes impossible. Therefore, the proper distance between neighbors should be defined.

The proposed simulated annealing PEG algorithm to construct LDPC codes is given in Figure 4.4. Firstly, a parity-check matrix is produced with the PEG algorithm. In step 2, this generated matrix is assigned to the current active matrix as $H_1^{cand.}$. It is also registered as the best matrix H^{best}. In step 3, the cost function value of the generated matrix is calculated. This calculation is taken as the number of cycles that have the size of girth value in this algorithm. In step 5, a certain amount of edges that cause cycles in the size of girth value are removed. The number of edges to be removed is controlled with D and ρ variable. While ρ is taken very close to 1, D variable is taken as approximately ten percent of total edges. After removing the cycle-causing edges from the active matrix, new edges are again constructed using the PEG algorithm, and the cost value of the new matrix is calculated. In step 8, the cost values of the previous matrix and the newly produced matrix are compared. If there is an improvement of the objective value, the generated matrix is assigned as the active matrix for the next iteration. If a matrix with a worse objection function value has been produced, the new solution is either accepted or rejected at a certain probability using Equation (4.1). The current temperature T determines the acceptance of the worse solutions. As this temperature decreases over time, the algorithm will reject worse solutions more often. Therefore, while every solution is accepted with a higher probability in the first steps of the algorithm, only solutions that have better objective value are accepted in the last steps. In step 20, the active matrix is compared to the best solution so far, and if it has better results, it is assigned as the best matrix.

Different versions of the algorithm given in Figure 4.4 have been tried to get the best results from the proposed algorithm. For this goal, a regular LDPC code with (155,64) code length is designed. In one version of the algorithm, only candidates that showed improvement in their objective value were accepted as new candidates. In the

Require degree distribution d_c, d_v, objective function $g(x)$, max iteration number M, initial temperature T, and code size n, neighbor distance D, cooling factor η, distance coefficient ρ.

Generate $H^{initial}$ matrix by using PEG algorithm;

$H^{initial} \rightarrow$, $H_1^{cand.}$, $H^{best.}$;

$g(H_1^{initial}) \rightarrow min.cost$;

for $i = 1$ to M **do**

  Remove exactly $D \times 0.1 \times n$ edges from $H_i^{cand.}$ matrix;

  Fill $H_i^{cand.}$ with PEG algorithm, and assign it to $H^{temp.}$;

  Compute the cost $g(H^{temp})$;

  if $g(H^{temp}) \leq g(H_i^{cand})$ **then**

    $H^{temp} \rightarrow H_{i+1}^{cand.}$;

  else

    Compute $P(A)$ using Equation (4.1);

    Generate an output r from a uniform R.V. over [0,1];

    if $r \leq P(A)$ **then**

      $H^{temp} \rightarrow H_{i+1}^{cand.}$;

    else

      $H_i^{cand} \rightarrow H_{i+1}^{cand.}$;

      Go to STEP 4;

    end if

  end if

  T_{i+1}=η T_i;

  $D = D \times \rho$;

  if $g(H_i^{cand.}) < min.cost$ **then**

    $H_i^{cand.} \rightarrow H^{best}$;

    $g(H_i^{cand}) \rightarrow min.cost$;

  end if

end for

Output $H^{best.}$

Figure 4.4. PEG Based Simulated Annealing Algorithm

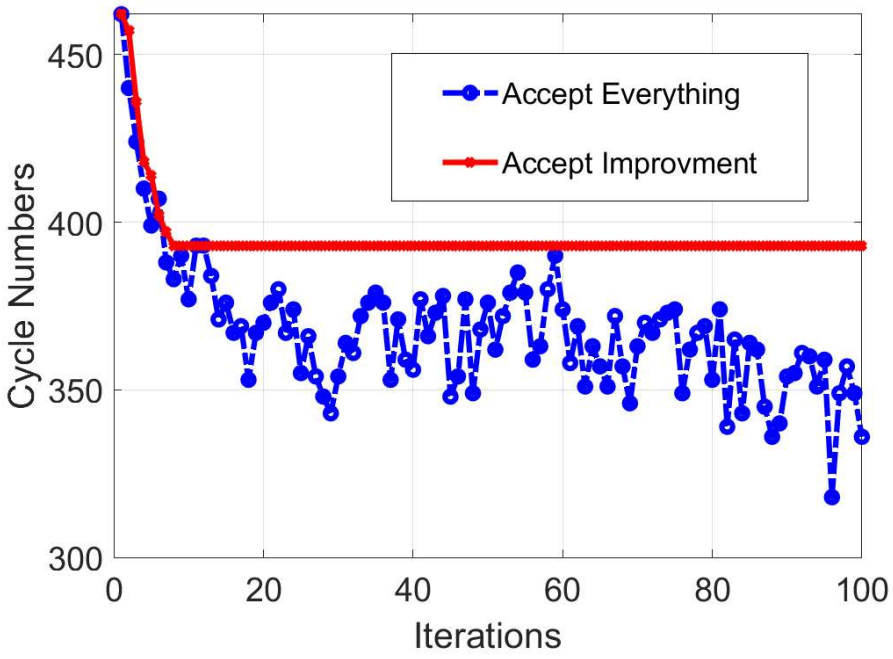

Figure 4.5. SA-Accept Everything

other version, all candidates were accepted as new candidates. A comparison of these two approaches is given in Figure 4.5. When only candidates that improve objective value are selected, the algorithm is blocked at a local best point as expected. When any candidate is accepted, the cycle number with the girth size decreases dramatically.

In the Figure 4.6, the effect of the cooling factor is given. It is observed that the algorithm gives better results without cooling. The reason for that PEG algorithm gives the best local optimum result. When harmful edges are removed, PEG fills the graph such that it generally improves the previous cycle distribution. Figure 4.7 shows the benefit of changing the number of edges to be removed in an adaptive way. It appears that choosing neighbors with decreasing distance rather than a fixed distance to the active solution gives better results. The small distance boosts the benefit of the PEG algorithm. In addition, after the algorithm is completed, resetting the settings and restarting the algorithm improves the objective value, since the PEG algorithm has some randomness. In the Figure 4.8, it is observed that the objective value is

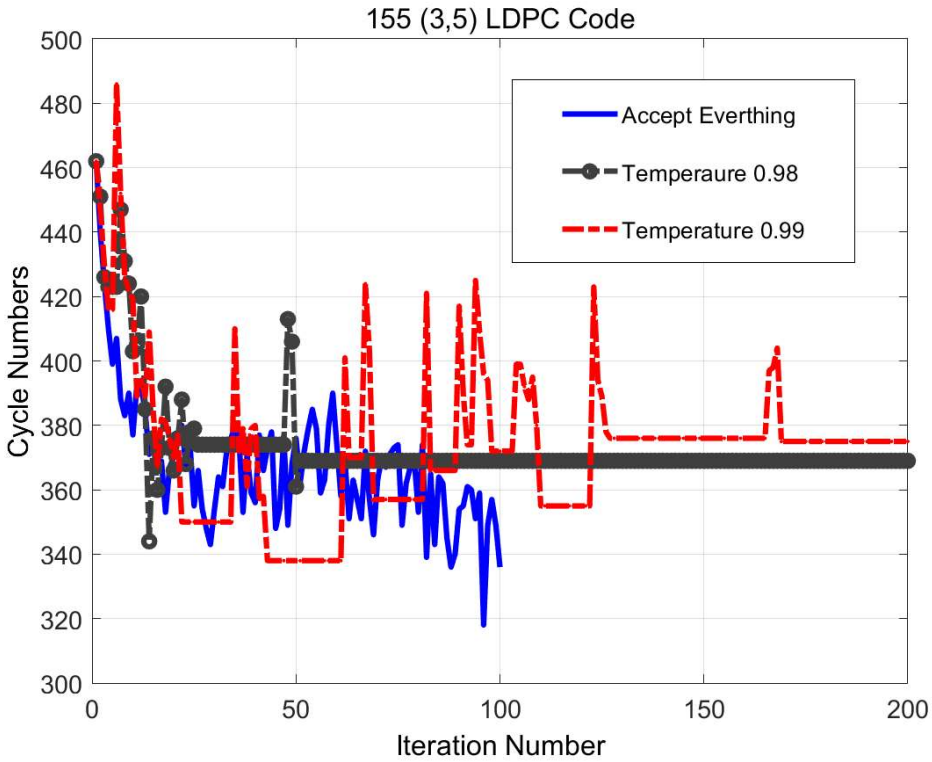

Figure 4.6. SA-Temperature Change

further improved when algorithm is restarted every 80 iterations.

Rand-PEG algorithm can also be used instead of the PEG algorithm that is employed in Figure 4.4. The Rand-PEG based simulated annealing algorithm gives better results because the Rand-PEG algorithm can build better matrices than the PEG algorithm in terms of the cycle distribution. The cycle numbers of Tanner graphs, which are generated by the PEG algorithm, the RandPEG algorithm, the Tanner code, the PEG based simulated annealing, and RandPEG based simulated annealing algorithms are given in Table 4.1. A Family of circulant-based quasi-cyclic LDPC codes, namely Tanner code [61] is one of the best codes with length 155. Its girth value is 8 and the multiplicity of cycles with length equal the girth is only 465. As seen in Table 4.1, the Rand-PEG based simulated annealing algorithm, and PEG based simulated annealing algorithm have constructed (155,64) LDPC code with the same girth value, but with a fewer multiplicity of cycles with girth size. For (504,252) LDPC codes, improved PEG algorithm in [64] has a girth value 8, and has a girth multiplicity of 452, whereas our

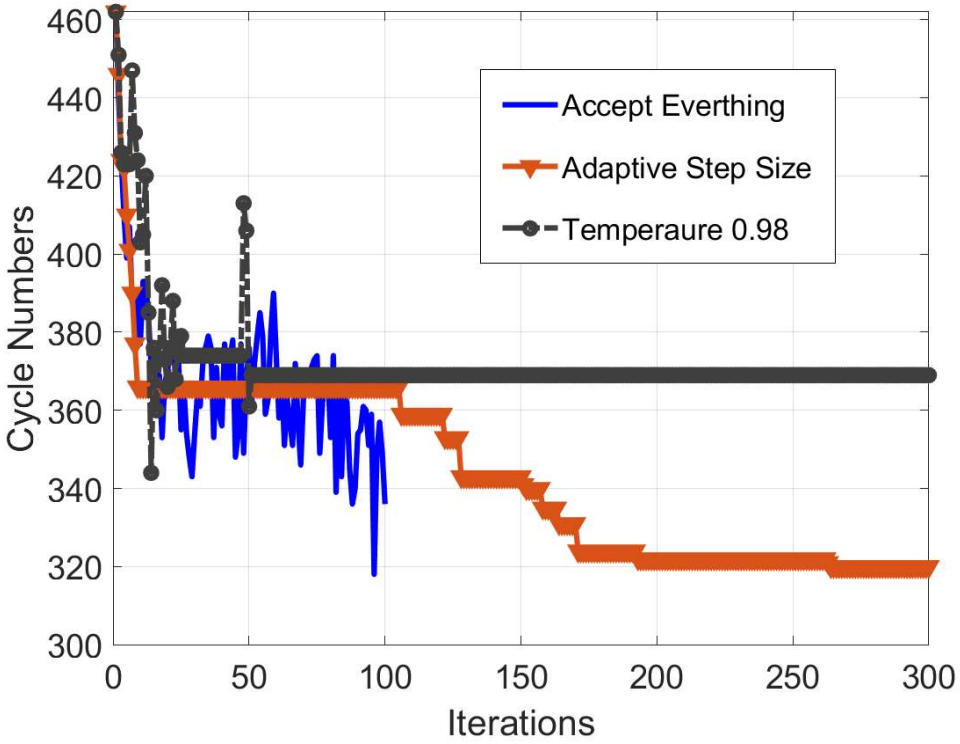

Figure 4.7. SA-Adaptive Step Size

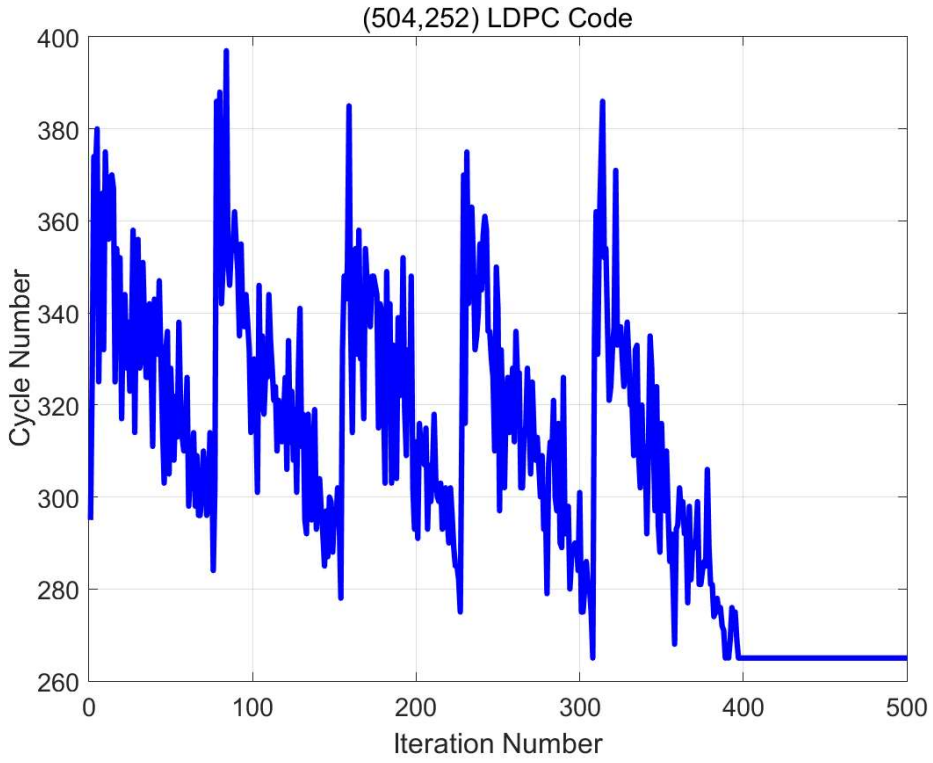

Figure 4.8. SA-Objective Value

Table 4.1. Girth Number

Code Structure	**Size and Degree Distribution**		
	155(3,5)	**504(3,6)**	**1008(3,6)**
PEG	501	808	167
Rand-PEG	464	452	31
Tanner Code	465	-	-
PEG Based Simulated Annealing Algorithm	402	368	42
Rand-PEG Based Simulated Annealing Algorithm	325	238	0

suggested algorithms have girth multiplicity of 238 and 368. The superior performance of the simulated annealing algorithms increases when LDPC code with length 1008 is employed. While other algorithms have a girth value 8, Rand-PEG based simulated annealing algorithm removes all cycles with size 8, and the generated LDPC code has a girth value 10.

Cycle performance does not mean that the performance in the error floor directly improves. For the improvement, it is necessary to look at the distribution of the trapping set. To analyze the performance of the simulated annealing algorithms, the ETS list of designed LDPC codes is generated by using the ETS expansion algorithm given in Figure 2.11. Since the girth of Tanner code, g, is 8, cycles with size 8, 10, 12, and 14 $(g, g+2, g+4, g+6)$ are used as input the algorithm. The total multiplicity of ETS of the constructed LDPC code is listed in Table 4.2.

ETSs distribution of (155,64) Tanner code and LDPC codes generated by PEG algorithm, Rand-PEG algorithm, PEG based SA algorithm, and Rand-PEG based SA algorithm is given in Table 4.3. When those codes are compared, it is clear to see that the simulated annealing algorithms not only improve the cycle distribution but also trapping sets distribution of the given algorithm. However, the error-correcting performance of the generated LDPC code should reflect this success. In the Table 2.3, the contribution of each ETS to the error floor in 5.5. dB SNR region is given for the (155,64) Tanner code. The Table 2.3 shows that the dominant trapping sets for Tanner

Table 4.2. Elementary Trapping Sets of (155,64) LDPC Code Obtained by Simulated Annealing Algorithm

ETS Class	**LSS_g**	**LSS_g+2**	**LSS_g+4**	**LSS_g+6**	**Total Multiplicity**
(4,4)	325				325
(5,3)	4				4
(5,5)		4518			4518
(6,4)		584			584
(6,6)			22995		22995
(7,3)		89			89
(7,5)			12030	7611	19631
(7,7)				146274	146274
(8,2)		16			16
(8,4)			3453		3453
(9,3)			704		704
(10,2)			80	1	81

Table 4.3. List of Elementary Trapping Sets

ETS Class	**Tanner Code**	**PEG Algorithm**	**Rand-PEG Algorithm**	**PEG Based SA Algorithm**	**Rand-PEG Based SA Algorithm**
(4,4)	465	576	464	402	325
(5,3)	155	31	18	11	4
(6,4)	930	1456	1160	1014	584
(7,3)	930	208	154	123	89
(8,2)	465	22	16	16	16
(8,4)	5580	4252	3811	1736	3453
(9,3)	1860	899	817	749	704
(10,2)	1395	89	85	83	81

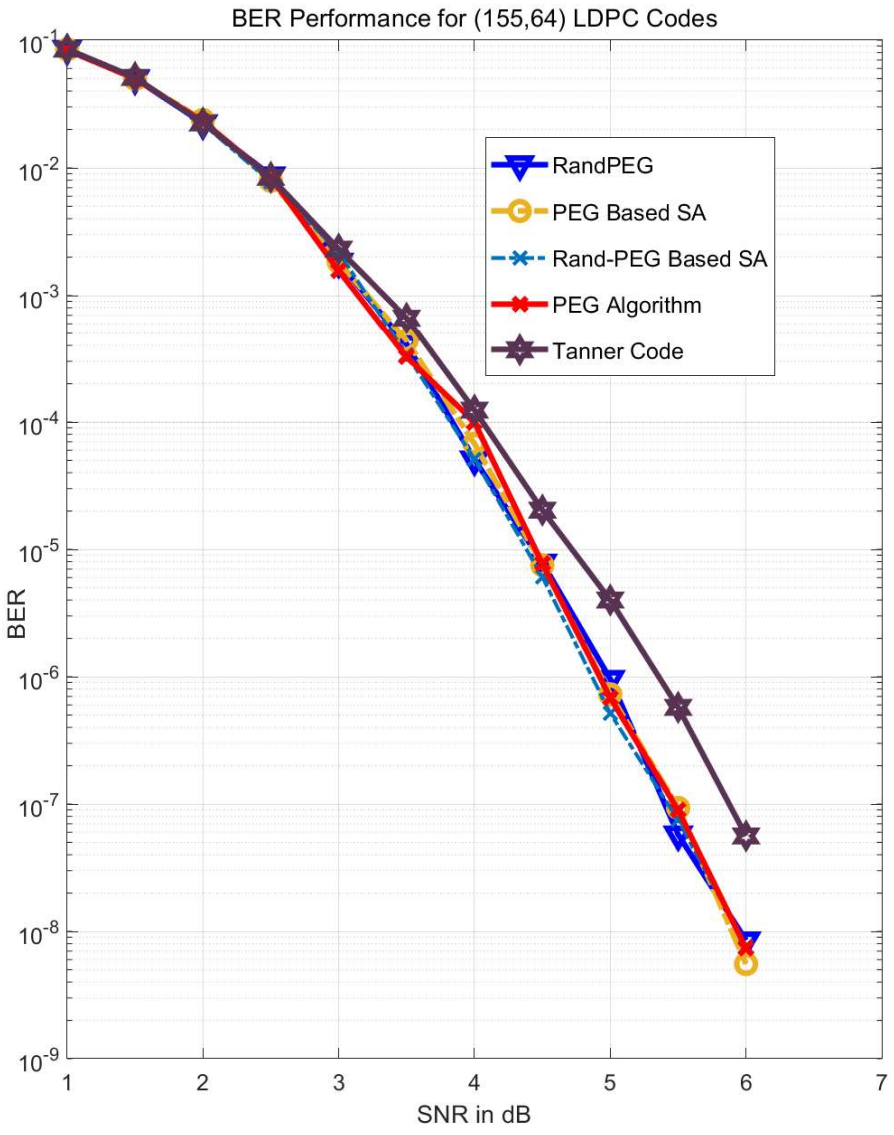

Figure 4.9. BER Performance of (155,64) LDPC Codes

Code are (7,3) ETS, (8,2) ETS, and (10,2) ETS. For the (8,2) ETS's distribution, there is no improvement. Also, the improvement of (10,2)ETS's distribution is bare. Therefore, the gain for error-correcting performance isn't expected so large. The bit error rate performance of LDPC codes given in the Table 4.3 is presented in Figure 4.9. Decreasing the dominant trapping sets number doesn't change the performance overall. For a significant gain, the dominant trapping sets should be removed totally.

4.4. RandPEG-noTS53 Algorithm

RandPEG-noTS53 Algorithm has been proposed by [32]. Using the RandPEG Algorithm given in Section 4.2, they try to detect and avoid the trapping sets. Their main focus on the (5,3) ETSs, and (6,4) ETSs. Firstly, they explain how trapping sets can be detected in the computation tree that is used in the PEG algorithm. After that, they include new criteria when selecting the edges in the RandPEG algorithm, so that (5,3) ETSs are completely removed and the number of (6,4) ETSs is minimized.

The cycles are easily estimated in-depth l when constructing the LDPC codes with the PEG algorithm. On the other hand, detecting the trapping sets is a tedious process. Therefore, [32] provides a characterization to detect (5,3) ETSs and (6,4) ETSs when constructing the subgraph (see Section 4.1). [32] shows that there is only one type of topology for (5,3) ETSs. This topology is given in Figure 4.10. [32] provides that there are exactly two different topologies for (6,4) ETSs. The topologies of (6,4) ETSs are given in Figure 4.11. When expanding the depth-l tree, different copies of the same check can appear at different levels of the subgraph. This check node is named as c_j^i, where i is the level of the j. copy of the check node. The topologies of (5,3) ETSs and (6,4)ETSs provides the following theorems:

Theorem 4.1. *When expanding the depth-l tree of a variable node v_i, a (5,3) ETS is detected if and only if there is a check node such that it has two copies as c_1^7 and c_2^7 with the same parent, and and the path $v_i, ..., c_1^7$ and the path $v_i, ..., c_2^7$ have three common variable nodes.*

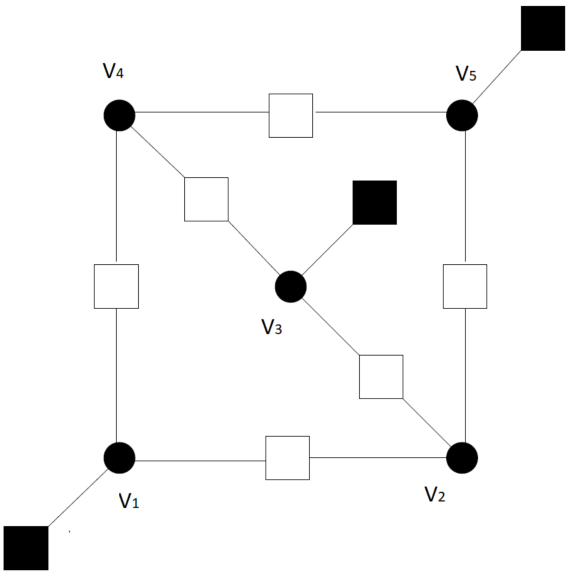

Figure 4.10. The Tanner Graph of a (5,3) ETS

Proof: See Appendix B

Theorem 4.2. *When expanding the depth-l tree of a variable node v_i, a (6,4) ETS with the topology given in Figure 4.11(a) is detected if and only if one of two case occurs: 1) There is a check node which has two copies as c_1^7 and c_2^{11} with the same parent, and the path $v_i, ..., c_1^7$ and the path $v_i, ..., c_2^{11}$ have four common variable nodes.*
2) There is a check node which has two different copies at the level 7, and their paths have two common variable nodes.

Proof: See Appendix C.

Theorem 4.3. *When expanding the depth-l tree of a variable node v_i, a (6,4) ETS with the topology given in Figure 4.11(b) is detected if and only if one of two case occurs: 1) There is a check node which has two copies as c_1^7 and c_2^9 with the same parent, and the path $v_i, ..., c_1^7$ and the path $v_i, ..., c_2^9$ have three common variable node.*
2) There is a check node which has two different copies at the level 9, and their paths have four common variable nodes.

Proof: See Appendix C.

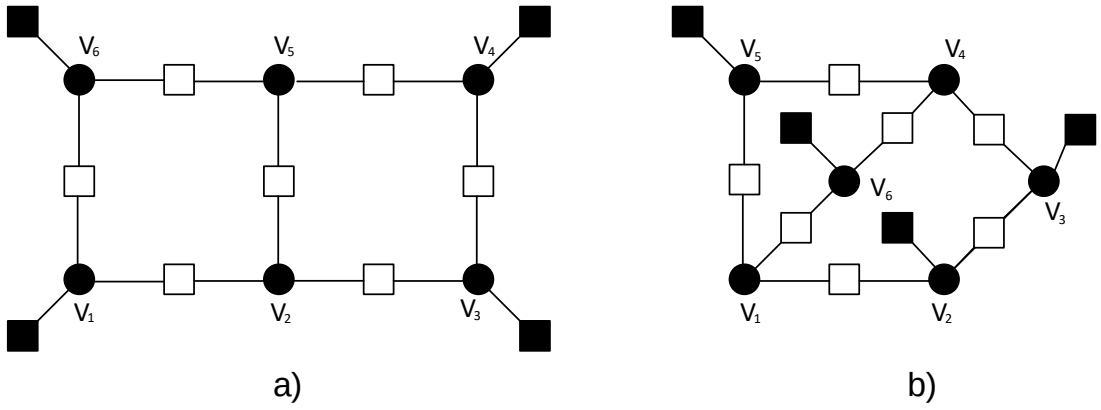

Figure 4.11. The Tanner Graph of a (6,4) ETS

When the construction tree is spanned up to the depth-l_{max}, the candidate check node that will be connected to the active variable node is determined with rules of Randomized PEG algorithm [64], and the additional rules of RandPEG-noTS53 algorithm as

(i) Remove all check nodes that appear at least once in the depth-3 tree. It will prevent the algorithm from creating cycles with size less than 8.

(ii) Remove all check nodes that create (5,3) ETSs. Then, select the check node that creates the minimum number of (6,4) ETSs.

The RandPEG-noTS53 algorithm doesn't decrease the number of (6,4) ETSs only, it also decreases the number of (7,3)ETSs, and (8,2)ETSs, since (6,4) ETS is a layered superset of (7,3) ETSs, and (8,2) ETSs. (see Definition 10)

4.5. RandPEG-noTS53 Based Simulated Annealing Algorithm

RandPEG-noTS53 based simulated annealing algorithm is similar to the PEG based simulated annealing algorithm. The main differences come from the definition of the objective function. Instead of minimizing the number of cycles with girth size,

the number of (6,4)ETSs and their successors' sets ((7,3) ETSs and (8,2) ETSs) is minimized. The objective function is defined as

$$\text{minimize } f(x) = w_{64}n_{64} + w_{73}n_{73} + w_{82}n_{82} \tag{4.2}$$

where w_{ij} is the weight variable of (i,j) ETS and n_{ij} is the total number of (i,j) ETSs. The values of weight variables are chosen according to their error floor contribution. Their contribution is calculated under importance sampling, [24], [26]. In this manner, their value is defined as

$$w_{64} << w_{73} << w_{82}, \tag{4.3}$$

The edges that will be removed are chosen according to their contributions to the objective value. For instance, if an edge exists in k different (8,2) ETSs, then its contribution will be $w_{82} \times k$. The most harmful d number of edges is removed from G. To avoid getting stuck at a local optimum point, the distance should be large enough that the algorithm should jump to another region. However, if the distance is too large, convergence to a global optimum becomes impossible. Similar to the PEG based simulated annealing algorithm, an adaptive distance schedule is used for the algorithm. The generation of the neighbor candidates should be executed in a short time such that the total iteration number of simulated annealing algorithms will be high enough to generate better results. Therefore, it needs to enumerate cycles and trapping sets in a short time. Therefore, the enumerating algorithm presented in Figure 2.11 is used. Table 2.2 shows that to find the (5,3) ETS, (6,4) ETS, (7,3) ETS, and (8,2) ETS, cycles with size g and $g+2$ will be sufficient. Moreover, RandPEG-noTS53 enumerates (6,4) ETSs to for each candidate check node in each step of construction, and it takes so long time. Therefore, applying a simulated annealing algorithm directly to RandPEG-noTS53 isn't efficient. For this reason, the RandPEG-noTS53 algorithm is slightly modified. In the edge construction process, (6,4) ETSs aren't enumerated. Only (5,3) ETSs are taken into account. As a result of that, when the original RandPEG-noTS53 completed one iteration, the simulated RandPEG-noTS53 algorithm had already completed 100

iterations.

The RandPEG-noTS53 based simulated annealing algorithm is presented in Figure 4.12. In the first step, the Tanner graph of the initial LDPC code is generated with the RandPEG-noTS53 algorithm. In step 2, the cycles with size of g and $g+2$ in the graph are detected. After that, (6,4) ETSs, (7,3) ETSs, and (8,2) ETSs are enumerated with the algorithm given in Figure 2.11. Then, using the Equation (4.2), the cost of each edge of the graph is computed. In step 10, the edges that have the highest cost is removed from the graph. Then, in step 9, the graph is reconstructed with the RandPEG-noTS53 algorithm again. Then, the total cost of the current graph is computed in Step 10. If the generated active graph has less dominant trapping sets than the previous one has, it is assigned as the next candidate graph. Otherwise, the acceptance probability is computed with Equation (4.1). If the solution is accepted, it is assigned as the next candidate graph. Otherwise, the algorithm tries to find another solution. In Step 21 and 22, the parameter of cooling and neighbor distance is updated. Whenever the candidate graph has the best objective value until that time, it is registered as the best graph.

Enumerating cycles for long-LDPC codes are not possible due to high complexity. Therefore, the RandPEG-noTS53 based simulated annealing algorithm is only applied for (155,64) regular LDPC codes. The contribution of each ETS class in the 6 dB region is computed with importance sampling, [24]. The distribution of ETSs of LPDC codes and the contribution of ETSs to the error floor is presented in Table 4.4. As given in Table 4.4, the most dominant ETSs are (7,3)ETSs and (8,2) ETSs. RandPEG-NoTS53 based simulated annealing algorithm creates cycles, (4,4) ETSs, more than the Rand-PEG based simulated annealing algorithm and RandPEG-noTS53 algorithm. However, it decreases the dominant trapping sets number. In particular, the number of (8,2) ETSs decreases to only 2. Therefore, it is expected that RandPEG-NoTS53 based simulated annealing algorithm show some improvements in the error floor region.

Require d_c, d_v, D, T, $g(x)$, M, ρ, η

Generate $G^{initial}$ with RandPEG-noTS53 Algorithm;

$G^{initial} \rightarrow$, $G_1^{cand.}$, $G^{best.}$;

$g(G_1^{initial}) \rightarrow min.cost$;

for $i = 1$ to M **do**

Find cycles with size g and $g+2$ in $G_i^{cand.}$, and assign to L_{in};

Find ETSs using algorithm given in Figure 2.11;

Evaluate the cost of each edge in $G_i^{cand.}$ with Equation (4.2);

Remove the most harmful d number of edges from $G_i^{cand.}$;

Fill $G_i^{cand.}$ with RandPEG-noTS53 algorithm, and assign it to $G^{temp.}$;

Compute the cost $g(G^{temp})$;

if $g(G^{temp}) \leq g(G_i^{cand})$ **then**

$g(G^{temp}) \leq g(G_i^{cand})$

else

Compute $P(A)$ using Equation (4.1);

Generate an output r from a uniform R.V. over [0,1];

if $r \leq P(A)$ **then**

$G^{temp} \rightarrow G_{i+1}^{cand.}$;

else

$G_i^{cand} \rightarrow G_{i+1}^{cand.}$;

Go to STEP 4;

end if

end if

T_{i+1}=η T_i

$D = D \times \rho$

if $g(H_i^{cand.}) < min.cost$ **then**

$G_i^{cand.} \rightarrow G^{best}$

$g(G_i^{cand}) \rightarrow min.cost$

end if

end for

Figure 4.12. RandPEG-noTS53 Based Simulated Annealing Algorithm

Table 4.4. ETS's Distribution of LDPC Codes

ETS Class	**Error Floor Contribution in 6 dB SNR**	**Rand-PEG Based SA Algorithm**	**RandPEG NoTS53 Algorithm**	**RandPEG NoTS53 Based SA Algorithm**
(4,4)	2.1e-13	325	453	497
(5,3)	4.8e-12	4	0	0
(6,4)	3.6e-13	584	1183	1248
(7,3)	1.7e-11	89	135	96
(8,2)	5.8e-11	16	12	2
(8,4)	6.4e-14	3453	4948	5139
(9,3)	4.3e-13	704	965	901
(10,2)	2.51e-12	81	112	99

To compare the BER performance of LDPC codes given in Table 4.4, BPSK over AWGN channel with belief propagation for 50 iterative number have been employed. The BER performance for (155,64)LDPC codes is given in Figure 4.13. When Figure 4.9 is analyzed, the impact of simulated annealing doesn't recognize. On the other hand, its contribution to the original RandPEG-noTS53 algorithm is clear in Figure 4.13. In the error floor region (6dB for this size), the RandPEG-noTS53 algorithm based simulated annealing algorithm shows its improvements. The reason for that, RandPEG-noTS53 algorithm directly tries to decrease the dominant trapping sets. On the other hand, enumerating the ETSs increases complexity. Therefore, because of its complexity issue, the RandPEG-noTS53 algorithm based simulated annealing algorithm couldn't be employed for longer LDPC codes.

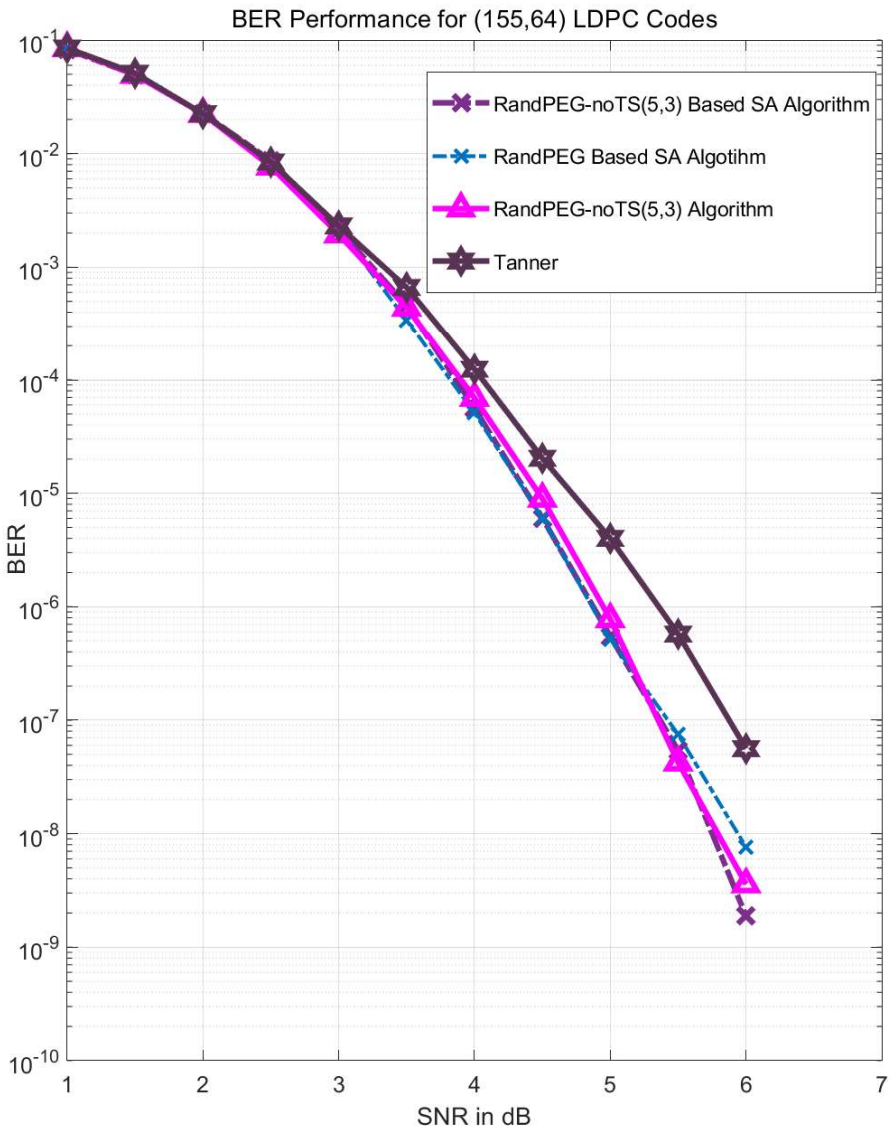

Figure 4.13. BER Performance of LDPC Codes

5. On the Construction of QC-LDPC Codes

As mentioned early in the thesis, QC-LDPC codes are the code family of LDPC codes that its parity check matrix has a special form. Its parity check matrix is constructed from the shifted identity matrices with size L as given in Equation (5.1).

$$H = \begin{bmatrix} I(p_{0,0}) & I(p_{0,1}) & ... & I(p_{0,d_c-1}) \\ I(p_{1,0}) & I(p_{0,1}) & ... & I(p_{1,d_c-1}) \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ I(p_{d_v-1,0}) & I(p_{d_v-1,1}) & ... & I(p_{d_v-1,d_c-1}) \end{bmatrix}. \tag{5.1}$$

PEG algorithm and its advanced versions can also be used to construct QC-LDPC codes. When the location of an edge from the identity matrix is established, other edges are automatically connected according to the identity matrix. Let's assume that the first variable is connected to the j^{th} check node, then i^{th} variable is connected to the $[j+i]_L^{th}$ check node. Because of the symmetrical structure, all of the other variables in the identity matrix will have the same local girth value. When the PEG algorithm and its advanced versions are applied to QC-LDPC codes, only the RandPEG-noTS53 algorithm generates better results. The reason for that can be that finding trapping sets are much easier for the QC-LDPC codes.

Firstly, the simulated annealing algorithms given in Figure 4.4 and Figure 4.12 is applied to generate QC-LDPC codes. However, they cannot improve the objective value and doesn't converge to global optimum solution as Figure 2.4. The example of an objective value change is given in Figure 5.1. For any two solutions, there is a distance at least the block size L. Therefore, the closest neighbor is L distance away

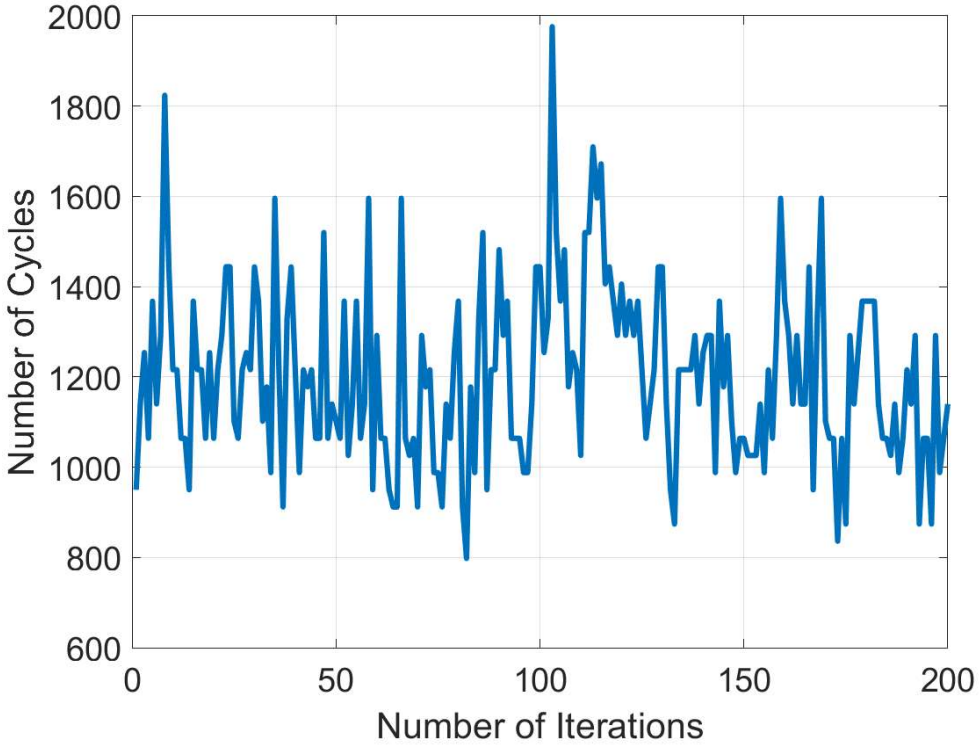

Figure 5.1. Objective Value Change

from the active solution. Therefore, the direction of the simulated annealing algorithm doesn't change, since the value of L is not a small number in general. As a result of that different approaches should be taken.

The structure of QC-LDPC codes provides some tools which makes the construction of QC-DLPC codes easier. As a result of that, a lot of works focus on these tools such as [41, 42, 66]. The most important facilitation of QC-LDPC codes is finding its cycles. [67] proved that a cycle of length k exists in the Tanner graph if and only if

$$\sum_{l=0}^{k-1}(p_{i_l,j_l} - p_{i_{l+1},j_l}) = 0 \quad mod \;\; L, \tag{5.2}$$

where $i_k = i_0$, $i_l \neq i_{l+1}$, $j_l \neq j_{l+1}$ is satisfied.

5.1. Simulated Annealing Algorithm

Definition 12. *A permutation shift matrix P^S is a neighbor of the permutation shift matrix P^T if and only if the girth of the QC-LDPC code generated by P^S is greater than or equal to the girth of the QC-LDPC code generated by P^T, and there is exactly one $p^S_{i,j} \in P^S$ such that*

$$p^S_{i,j} \neq p^T_{i,j} \mod L, \tag{5.3}$$

where i=1, 2, ..., d_s, j=1, 2, ..., d_c and $p^T_{i,j} \in P^T$.

Definition 13. *A set that consists of permutation shift matrices is the neighbor set of the permutation shift matrix P^T if and only if every permutation shift matrix in the set is a neighbor of the permutation shift matrix P^T. This set will be represented as $N(P^T)$ in the rest of the thesis.*

It is trivial to find the neighbor set of a permutation shift matrix by using Equation (5.2) and Definition 12. There are exactly $L \times d_c \times d_s$ candidate permutation shift matrices that could be neighbors of a permutation shift matrix, and they can be checked whether they provide the equality in Equation (5.2) or not in polynomial time.

For short-length QC-LDPC codes, [37] and [38] show that all (a, b) ETS can be found with a simple search algorithm. In this work, Algorithm 1 given in [37] is adopted to find the ETSs. Instead of counting all ETSs, only one variable node is selected from each block, and the cycles and ETSs to which it is connected to are counted. Then, the results are multiplied with L to take into account the multiplicity that is caused by the quasi-cyclic structure. Therefore, it is sufficient to examine exactly d_c variable nodes. The disadvantage of this approach is that it causes the ETS distribution to be found roughly instead of exactly, because some of ETSs with the same isomorphic structures are counted multiple times. The advantage is that iteration time is considerably shortened. In the last iteration, however, the ETS distribution of the resulting matrix is exhaustively determined by Algorithm 1 given in [37].

Require Permutation shift matrix $P^{initial}$, objective function $g(x)$, max. iteration number M;

$P^{initial} \rightarrow P^{cand}$;

for $i = 0$ to M **do**

Generate $N(P^{cand})$;

Computes the cost $g(P^i)$ for $\forall\ P^i \in N(P^{cand})$;

Find the $P^j \in N(P^{cand})$ with the minimum cost;

$P^j \rightarrow P^{cand}$;

end for

Output $P^{cand.}$.

Figure 5.2. Hill-Climbing Algorithm

Using a hill-climbing algorithm given in Figure 5.2, it is possible to improve the ETS distribution of the initially given permutation shift matrix. For line 5 of the algorithm, we can select the candidate permutation shift matrix in three different ways:

(1) Best gain: selecting the one with minimum cost,

(2) Uniform distribution: selecting with equal probability among the ones whose cost value is less than the cost value of the current candidate,

(3) Gain distribution: selecting with a normalized probability according to their gains in the objective function.

These algorithms are likely to get stuck in a locally optimal point regardless of the selection policy. On the other hand, a simulated annealing algorithm can help converge to a globally optimal solution. According to the gain in the objective value, a new solution is accepted or not in the simulated annealing algorithm. If there is a positive contribution, the new solution is accepted. Otherwise, Metropolis' criterion based on Boltzman's probability is applied. The acceptance probability is determined

with

$$P(A) = e^{-\Delta E/(kT)}, \tag{5.4}$$

where ΔE is the difference between objective function value of the current solution and candidate solution, k is Boltzmann's constant, and T is the temperature, respectively.

A geometric cooling rule is employed in this work. The update rule is defined as

$$T_{i+1} = \eta T_i, \tag{5.5}$$

where η is the temperature constant that is smaller than 1 but close to 1. The initial temperature is chosen as a big number to accept every solution in the early iterations.

The proposed simulated annealing algorithm is given in Figure 5.3. Decision variables and decision weights of the objective function are critical to the error-correcting performance of the QC-LDPC code generated by the simulated annealing algorithm. The number of the dominant ETS in the Tanner graph is used as the main decision variables in this work. For an (a, b) ETS class, the number of ETS that are present in the Tanner graph is represented as $n_{a,b}$. Objective weights are chosen in proportion to the contribution of the ETS class to the error floor. The weight vector with three different decision weights are determined, $\mathbf{w}$=[w_1 w_2 w_3], according to following rules:

- The coefficients of the decision variables are determined as w_1 for the ETS classes (S_1) that do not exist in the Tanner graph of the permutation shift matrix ($P^{initial}$) taken as input for the simulated annealing algorithm. By selecting w_1 value too large, the creation of non-existent ETS in the Tanner graph of $P^{initial}$ is prevented.

- The coefficient of w_2 is used for the ETS classes (S_2) which are required to be completely removed in Tanner graph of $P^{initial}$. The number of w_2 is chosen to be

Require Permutation shift matrix $P^{initial}$, objective function, $g(x)$, max. iteration number M, initial temperature T.

$P^{initial} \rightarrow P^{cand.}$, $P^{best.}$

$g(P^{initial}) \rightarrow min.cost$

for $i = 0$ to M **do**

  Generate $N(P^{cand.})$

  Randomly select a permutation shift matrix P^j from $N(P^{cand.})$

  Compute the cost $g(P^j)$

  if $g(P^j) \leq g(P^{cand.})$ **then**

    $P^j \rightarrow P^{cand.}$

  else

    Compute $P(A)$ using Equation (4.1);

    Generate an output r from a uniform R.V. over [0,1];

    if $r \leq P(A)$ **then**

      $P^j \rightarrow P^{cand.}$

    else

      Go to STEP 5

    end if

  end if

  T_{i+1}=η T_i

  if $g(P^{cand.}) < min.cost$ **then**

    $P^{cand.} \rightarrow P^{best}$

    $g(P^{cand}) \rightarrow min.cost$

  end if

end for

Output $P^{best.}$

Figure 5.3. Simulated Annealing Algorithm for QC-LDPC Codes

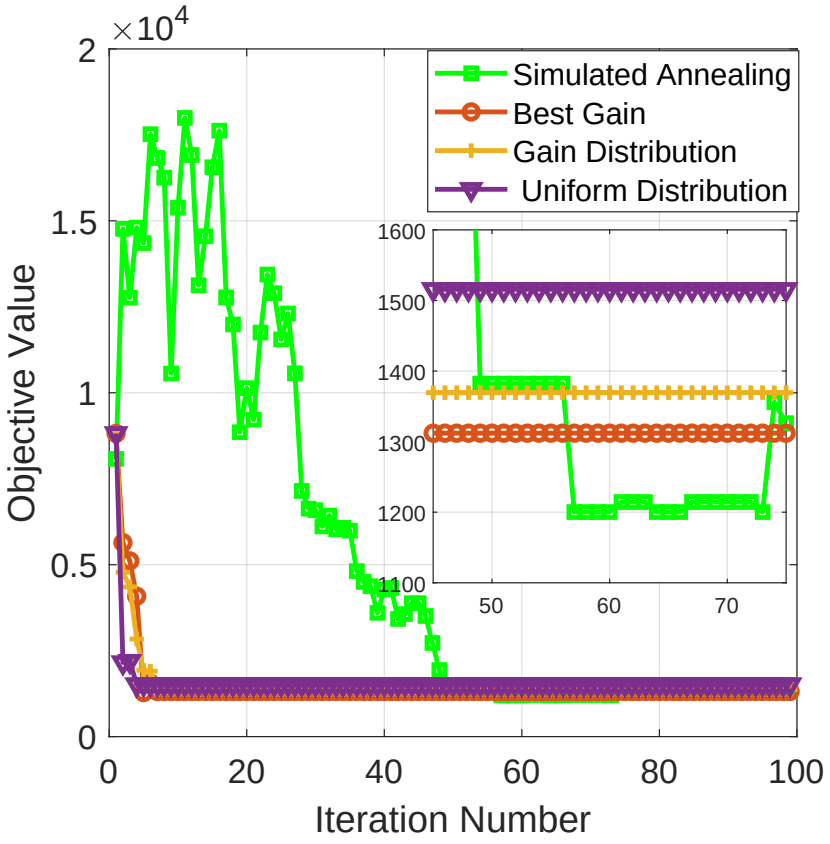

Figure 5.4. Improvement of The Objective Value

much smaller than w_1 and greater than w_3. During the iterations of the simulated annealing algorithm, the ETS in this class are allowed to occur sometimes, but has been targeted to be destroyed at the end of the algorithm.

- For the ETSs classes (S_3), whose numbers in the Tanner graph are desired to be reduced, w_3 coefficients are used.

In this manner, the objective function of the simulated annealing algorithm is given as

$$g(x) = \sum_{(a,b)\in S_1} w_1 n_{a,b} + \sum_{(a,b)\in S_2} w_2 n_{a,b} + \sum_{(a,b)\in S_3} w_3 n_{a,b}. \tag{5.6}$$

In Figure 5.4, the performance of hill-climbing algorithms and simulated annealing are compared. The average change of objective functions for 20 different examples

Table 5.1. ETS of LDPC Codes within the Range of a ≤ 10 and b ≤ 3

ETS	**Tanner**	**RandPEG-noTS(5,3)**	$\mathbf{P_1}$
(5,3)	155	-	-
(7,3)	930	62	-
(8,2)	465	-	-
(9,3)	1860	496	155
(10,2)	1395	31	-

are illustrated in this example. The hill-climbing algorithms improve the objective value quickly in early iterations. However, they get stuck in local optimal points after a few iterations. On the other hand, the simulated annealing algorithm can escape out of local minimal points. As a result, the simulated annealing algorithm can converge to better solutions compared to hill-climbing algorithms as expected.

In this section, four examples will be presented to analyze the performance of the proposed method. All simulation results are based on the sum-product message-passing decoding algorithm with the maximum number of iterations equal to 100. For each SNR point, fifty frame errors are counted.

Example 14. *A regular QC-LDPC code with d_s=3, d_c=5 and L=31 is constructed. Parameters of the simulated annealing algorithm are taken as T=1000, λ=0.99, S_1=$\{(6,2)\}$, S_2=$\{(5,3),(8,2),(10,2)\}$, S_3=$\{(7,3),(9,3)\}$, $\mathbf{w}$=$[10^6\ 10^3\ 10^0]$. To generate the input permutation shift matrix, progressive edge algorithm (PEG) given in [27] is used. According to these settings, the permutation shift matrix, P_1, is generated as*

$$P_1 = \begin{bmatrix} 15 & 29 & 17 & 18 & 12 \\ 6 & 21 & 14 & 20 & 2 \\ 23 & 23 & 4 & 13 & 22 \end{bmatrix}. \tag{5.7}$$

The ETS distributions of QC-LDPC codes constructed by Tanner Codes in [61], RandPEG no(5,3) Algorithm in [32], and P_1 are given in Table 5.1. The results in this table show that the QC-LDPC code generated by the permutation shift matrix P_1 avoids

Table 5.2. ETS of LDPC Codes within the Range of a $\leq$ 12 and b $\leq$ 3

ETS	$\mathbf{C_1}$	$\mathbf{P_2}$	$\mathbf{C_2}$	$\mathbf{P_3}$
(9,3)	246	-	252	126
(11,3)	1230	656	2142	1197
(12,2)	123	-	63	-

almost every (a, b) *elementary trapping set that satisfies* $a \leq 10$ *and* $b \leq 3$*, except for a few (9,3) ETS.*

Figure 5.5 shows the FER performance of several regular LDPC codes listed in Table 5.1 for BSPK modulation over the additive white Gaussian noise (AWGN) channel. It can be seen clearly that the suggested QC-LDPC codes have better FER performance compared to the existing QC-LDPC codes in the high SNR region, since P_1 permutation shift matrix doesn't have dominant trapping sets.

Example 15. *A regular QC-LDPC code with* d_s*=3,* d_c*=5 and L=41 is constructed. Parameters of the simulated annealing algorithm are taken as T=10000,* λ*=0.8,* S_1*=* $\{(5,3),(6,2),(7,3),(8,2),(10,2)\}$*,* S_2*=*$\{(12,2)\}$*,* S_3*=*$\{(9,3),(11,3)\}$*,* $\mathbf{w}$*=[*10^6 10^4 10^0*]. For the input permutation shift matrix, the matrix* C_1 *given in [41] is used. According to these settings, the permutation shift matrix,* P_2*, is constructed as*

$$P_2 = \begin{bmatrix} 0 & 27 & 0 & 16 & 6 \\ 28 & 26 & 5 & 22 & 20 \\ 11 & 11 & 4 & 9 & 16 \end{bmatrix}. \tag{5.8}$$

The ETS distributions of QC-LDPC codes constructed by the permutation shift matrix C_1*, given in [41], and permutation shift matrix* P_2 *are presented in Table 5.2. The statistics in this table show that the QC-LDPC code generated by* P_2 *avoids every* (a, b) *trapping set that satisfies* $a \leq 12$ *and* $b \leq 2$*.*

Example 16. *A regular QC-LDPC code with* d_s*=3,* d_c*=6 and L=63 is constructed. Parameters of the simulated annealing algorithm are taken as T=10000,* λ*=0.95,* S_1*=*

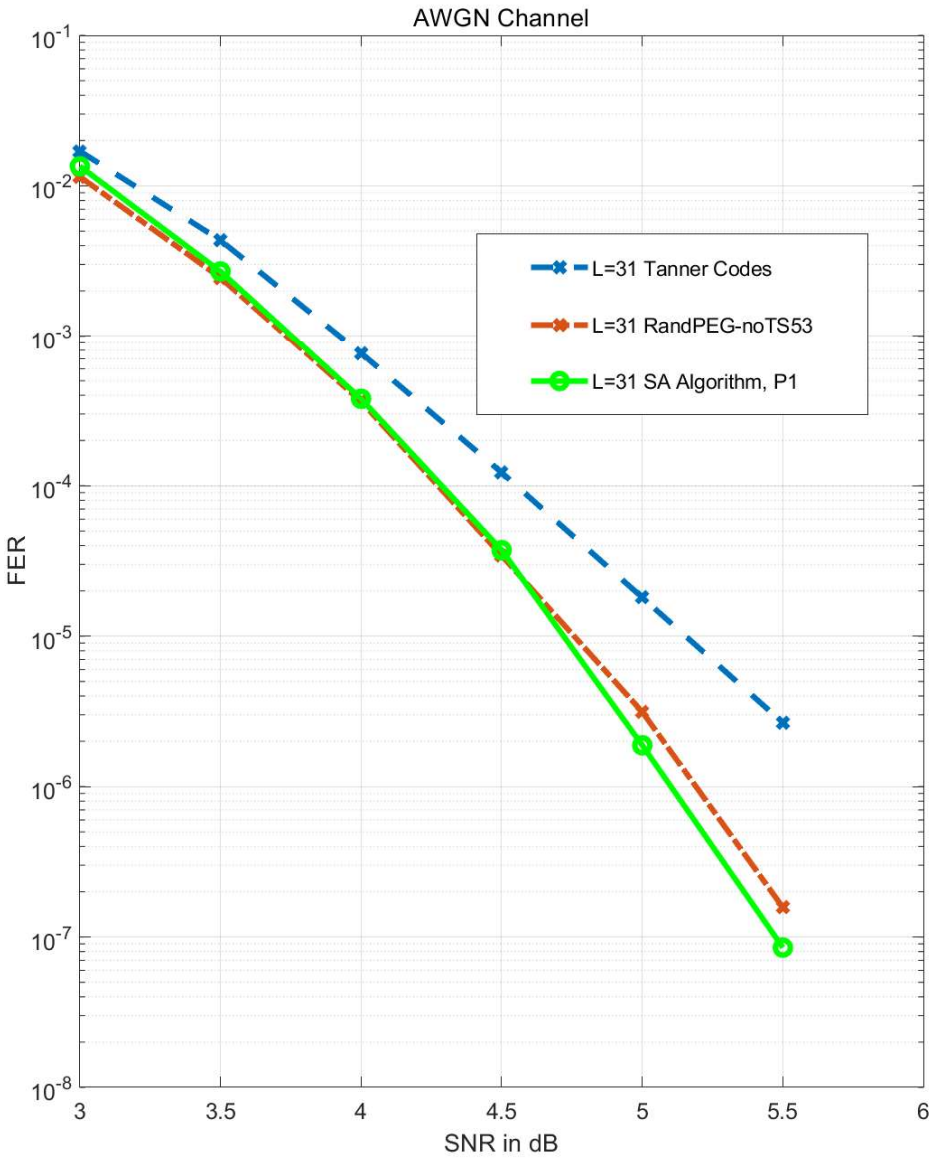

Figure 5.5. FER Performance of the Constructed QC-LDPC Codes.

$\{(5,3),(6,2),(7,3),(8,2),(10,2)\}$, S_2=$\{(12,2)\}$, S_3=$\{(9,3),(11,3)\}$, $\mathbf{w}$=$[10^6\ 10^4\ 10^0]$. *For the input permutation shift matrix, the matrix* C_2 *given in [41] is used. According to these settings, the permutation shift matrix,* P_3*, is constructed as*

$$P_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 20 & 8 & 33 & 29 & 59 \\ 0 & 9 & 26 & 24 & 32 & 58 \end{bmatrix}. \tag{5.9}$$

As in Example 15, ETSs that satisfy $a \leq 12$ *and* $b \leq 2$ *are completely removed from the Tanner graph of the proposed permutation shift matrix* P_3*. The ETS list of the suggested QC-LDPC codes is given in the Table 5.2.*

Figure 5.6 shows the FER performance of several regular LDPC codes listed in Table 5.2 BPSK modulation over AWGN channel. It can be seen clearly that the suggested QC-LDPC codes have better FER performance compared to the QC-LDPC codes generated in [41] in the high SNR region.

Example 17. *In this example, we improve the lifting degree of the codes with the fully-connected 3×5, 3×6, and 3×10 base graphs given in [41] and [42]. In these studies, the goal is to construct QC-LDPC codes that do not contain leafless ETSs (LETSs) satisfying the conditions* $a \leq 8$, $b \leq 3$ *with the smallest possible lifting degree. [41] found the codes with lifting degrees 41, 61, and 181, for the three base graphs, respectively. [42] improved these values as 27, 41, and 165, respectively. In this work, QC-LDPC codes with minimum lifting values 24, 39, and 152 are found by the proposed algorithm. Permutation matrices with the minimum lifting degrees of these codes are obtained as*

$$P_4 = \begin{bmatrix} 11 & 16 & 3 & 13 & 20 \\ 11 & 2 & 19 & 12 & 17 \\ 3 & 16 & 0 & 16 & 13 \end{bmatrix}, \tag{5.10}$$

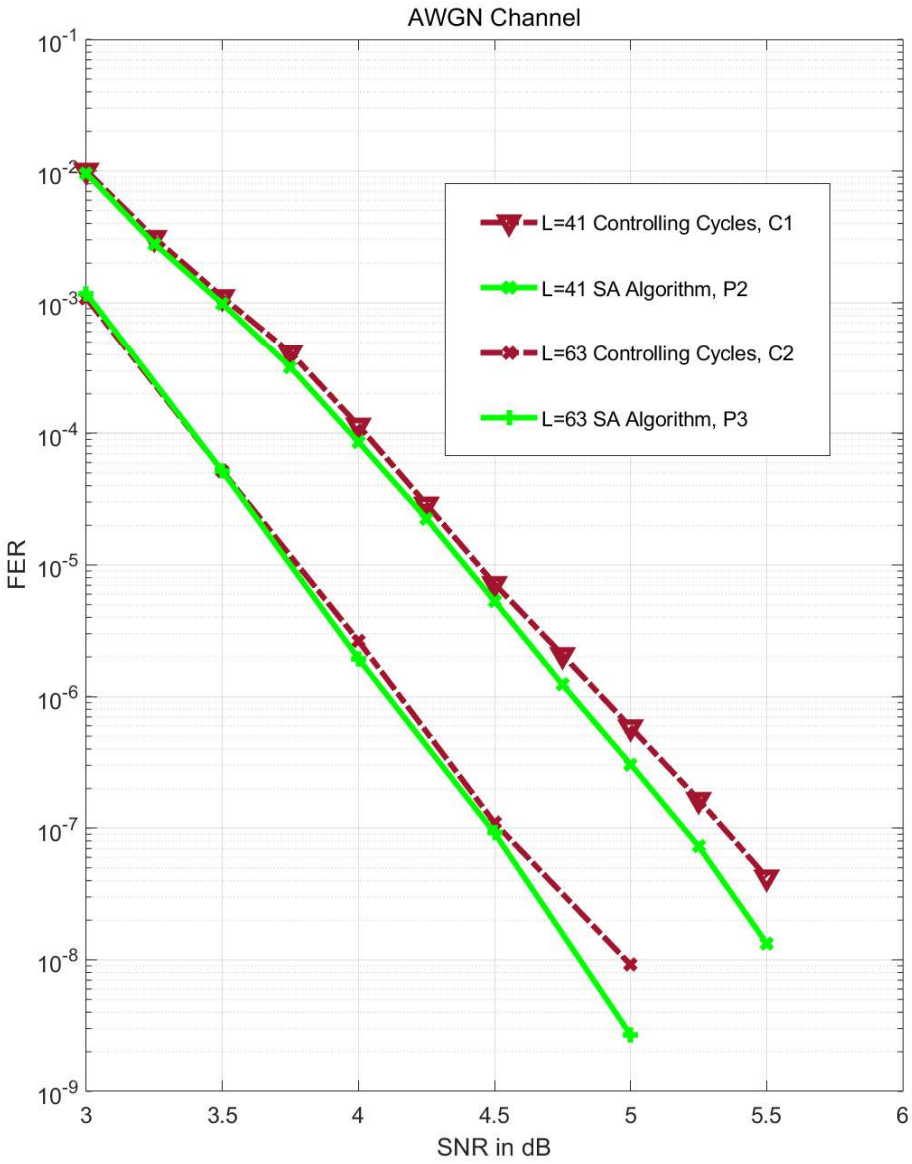

Figure 5.6. FER Performance of the Constructed QC-LDPC Codes.

Table 5.3. Time Performance of the Simulated Annealing Algorithm

	P1	**P2**	**P3**	**P4**	**P5**	**P6**
Max. Size of Counted Cycles	14	16	16	10	12	12
Iteration Number	1000	200	50	147	262	47
Time	3h	23h	75h	0.1h	9h	25h

$$P_5 = \begin{bmatrix} 33 & 5 & 32 & 30 & 29 & 22 \\ 15 & 14 & 1 & 33 & 10 & 34 \\ 23 & 4 & 38 & 0 & 15 & 6 \end{bmatrix}, \tag{5.11}$$

$$P_6 = \begin{bmatrix} 131 & 2 & 70 & 65 & 80 & 101 & 116 & 14 & 116 & 10 \\ 41 & 87 & 119 & 119 & 68 & 73 & 54 & 48 & 4 & 133 \\ 3 & 12 & 150 & 147 & 11 & 5 & 33 & 114 & 27 & 133 \end{bmatrix}. \tag{5.12}$$

Except for the cooling factor λ, the parameters of the simulated annealing algorithm are taken for these three base graphs as $T = 100$, S_1=$\{(6,2)\}$, S_2=$\{(5,3),(8,2)\}$, S_3=$\{(7,3)\}$, $\mathbf{w}$=$[10^3\ 10^1\ 10^0]$. λ values are taken as 0.9, 0.9, and 0.8, respectively. To generate the input permutation shift matrices, PEG algorithm is used for each base graph.

Although trapping sets are defined by [24] for the binary symmetrical channel (BSC) and AWGN channels, the produced QC-LDPC has also been tried on different channels. Tests were carried out on the Rician fading channel, which is suitable for urban areas. The ratio between the power in the direct path and the power in the other scattered paths, K, is accepted as 1 for the simulations. Figure 5.7 presents the FER performance comparison of QC-LDPC given in Example 14 with other QC-LDPC codes. As Figure 5.7 shows there is a significant dB gain. However, this gain decreases

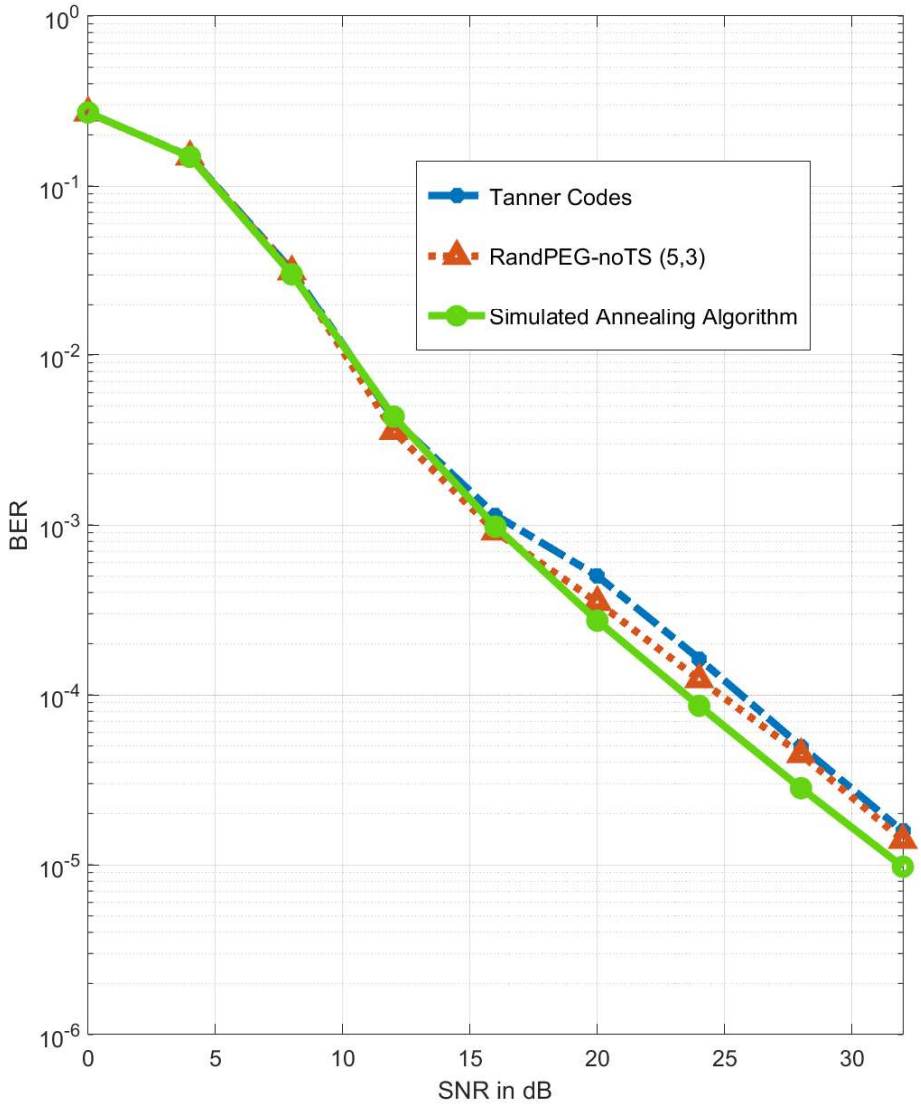

Figure 5.7. BER Performance of the Constructed QC-LDPC Codes.

in smaller values of K.

Although it is an offline task to produce a permutation matrix, the construction time of the permutation matrices given in the examples is presented in Table 5.3 to demonstrate the complexity of the proposed algorithm. The algorithm ran on a desktop computer with 2GHz CPU and 46 GB RAM. The selected parameters of the simulated annealing algorithm determine the construction times of the P_1, P_2, and P_3 matrices. The algorithm continues even if it achieves the best result in a certain iteration. On the other hand, for the construction of the P_4, P_5, and P_6 matrices, the algorithm is terminated when the objective function value is 0, i.e., when the desired target is reached.

In Table 5.3, it is observed that although P_6 matrix code length is 1520, it is constructed faster than P_3 matrix. The reason for this is that cycles with the maximum length of 12 are sufficient to find TSs satisfying $a \leq 8$, $b \leq 3$, and cycles with the maximum length of 16 are required to find (12,2)TSs.

Cycle counting is the most time and memory consuming part of the algorithm. However, in practice it is quite manageable for small-size cycles. Therefore, QC-LDPC codes without small-size TSs can be generated with the suggested algorithm even for large d_c and L values, whereas the generation of large QC-LDPC codes with relatively large-sized TSs optimized is not possible. However, since small TSs are more dominant, the proposed algorithm can be used for the construction of large QC-LDPC codes as well. Moreover, the recent works have been providing news cycle and trapping sets enumeration algorithms, which make the proposed simulated annealing algorithm to work in a shorter time. The proposed QC-LDPC codes in this section were published in [68].

6. CONCLUSIONS

LDPC codes are one of the most popular codes used for error correction in recent years. In particular, QC-LDPC codes are widely used in the next-generation communication systems, since hardware implementation of QC-LDPC codes is easy. Therefore, in this thesis, the construction and decoding of LDPC codes are mainly studied.

The belief propagation algorithm is commonly used in practice for decoding LDPC codes. However, studies on linear programming decoders were also conducted in the literature, since it has ML certificate. In this thesis, two different adaptive linear programming decoders are developed; integer adaptive linear decoder and heuristic adaptive linear decoder. IALP decoder achieves high performance when sufficient time is given. However, it does not achieve the desired performance during a realistic decoding period. The HALP decoder works very fast, but it does not exceed certain decoding performance. On the other hand, HALP decoder can work with other ALP decoders because of its fast process. The solution HALP decoder found can be given as an input for the other ALP decoders.

The main subjects of the thesis are the construction of LDPC codes. In the first stage of the thesis, it was studied on the setup of cycle-optimized LDPC codes. The proposed simulated annealing algorithm in the thesis tries to destroy the small cycles of the LDPC code given as an input. However, despite the improvement of the cycle distribution, the desired improvement in error correction could not be demonstrated. Therefore, a simulated annealing algorithm has been developed using the RandPEG-noTS53 algorithm, which tries to reduce the number of some trapping sets types. Although this algorithm improves error-correcting performance, the desired level could not be reached because it requires high complexity.

In the last stage of the thesis, the design of the QC-LDPC codes, which became very popular in recent years, was studied. Cycle and TS search is much easier in

QC-LDPC codes compared to other codes due to its structure. For this reason, a slightly different algorithm has been proposed than simulated annealing algorithms that produce standard LPDC codes. In the proposed algorithm, it tries to create a Tanner graph by attacking trappings sets that are dominant for the QC-LDPC codes in that desired size. For this purpose, trapping sets enumeration and importance sampling algorithms in the literature are also employed. The complexity of algorithms with minor changes is reduced significantly. Thanks to all this, a suitable simulated annealing algorithm has been developed to generate QC-LDPC codes that avoid small trapping sets. Among the modern algorithms, the simulated annealing algorithm generates the best QC-LDPC codes in terms of the distribution of the trapping set. Some dominant trappings are completely removed from the Tanner graph by the suggested algorithm. Moreover, Monte Carlo simulations clearly show that proposed QC-LDPC codes are superior to existing designs for QC-LDPC codes.

Since ETSs are the most dominant TSs, code optimizations are made according to ETSs distribution. Elementary sets of AS, FAS, LETS are also optimized, since they are the sub-class of ETSs. On the one hand, proposed algorithms could be extended easily for all trapping sets. On the other hand, the gain won't be so significant due to the fact that ETSs are the main cause of the error floor, and the complexity of the algorithms will be so high because of the enumeration of trapping sets.

6.0.1. Future Works

Intensive works on QC-LDPC codes still continues. Notably, finding the cycle and trapping sets of QC-LPDC codes in the Tanner graph becomes easier thanks to the latest proposed algorithms. This facilitation will make it easier for the simulated annealing algorithm to achieve better results even for slong-length QC-LDPC codes. If the simulated annealing algorithm, which has already achieved one of the best results for short-length, can also construct large-length QC-LDPC codes, it will be one of the standard-setting algorithms.

Besides the simulated annealing algorithms, there are other search algorithm that do not get stuck at a local point. The tabu search algorithm is a strong candidate algorithm to replace the simulated annealing algorithm for the construction of LPDC codes. The functionality of tabu search algorithm has been proven for the travelling salesman problem. A tabu search based algorithm can be used to construct LPDC codes if the tabu list, the list of the forbidden neighbors, is well-defined.

The message-passing algorithm is the standard decoding algorithm for most of the wireless communication systems. A small change of message passing algorithm can contribute the tackle with trapping sets for QC-LDPC codes. When the code-word is right-shifted with value p, another code-word is generated for QC-LDPC codes. Therefore, the received vector can be shifted in a way that the resulting trapping sets of the received vector can be removed. After decoding, the estimated vector can be left-shifted to reach the original estimated vector. For the efficiency of this algorithm, the Tanner graph of the QC-LDPC code should be designed properly.

The adaptive linear programming decoders are given in Chapter 3 have not the best error-correcting performance comparing to other adaptive linear programming decoders. The suggested HALP decoder works fine at the first steps of decoding. However, after some point, it couldn't find valid constraints to improve the decoding. If the reason for that can be enlightened, the HALP decoder can be the best linear decoder in the literature. Therefore, future work should be done to find the reason for getting stuck.

Obtained codes in this thesis have been tested using two-channel modes and one modulation (AWGN and Rician channel, and BPSK). These codes could be applied to other channels and modulations.

REFERENCES

1. Shannon, C. E., "A Mathematical Theory of Communication", *Bell System Technical Journal*, Vol. 27, No. 3, pp. 379–423, 1948.

2. Wicker, S. B. and S. Kim, *Fundamentals of Codes, Graphs, and Iterative Decoding*, Kluwer Academic Publishers, USA, 2002.

3. Lin, S. and D. Costello, *Error Control Coding: Fundamentals and Applications*, Pearson-Prentice Hall, London, 2004.

4. Gallager, R., "Low-Density Parity-Check Codes", *IRE Transactions on Information Theory*, Vol. 8, No. 1, pp. 21–28, 1962.

5. Chung, S.-Y., J. Forney, G.D., T. Richardson and R. Urbanke, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit", *IEEE Communications Letters*, Vol. 5, No. 2, pp. 58–60, 2001.

6. Tanner, R., "A Recursive Approach to Low Complexity Codes", *IEEE Transactions on Information Theory*, Vol. 27, No. 5, pp. 533–547, 1981.

7. MacKay, D. J. C., "Good Error-Correcting Codes Based on Very Sparse Matrices", *IEEE Transactions on Information Theory*, Vol. 45, No. 2, pp. 399–431, 1999.

8. Shu Lin, D. J. Costello and M. J. Miller, "Automatic-repeat-request Error-control Schemes", *IEEE Communications Magazine*, Vol. 22, No. 12, pp. 5–17, 1984.

9. Chandran, S. R. and S. Lin, "Selective-repeat-ARQ Schemes for Broadcast Links", *IEEE Transactions on Communications*, Vol. 40, No. 1, pp. 12–19, 1992.

10. Benelli, G., "Some ARQ Protocols with Finite Receiver Buffer", *IEEE Transactions on Communications*, Vol. 41, No. 4, pp. 513–523, 1993.

11. Eizmendi, I., M. Velez, D. Gómez-Barquero, J. Morgade, V. Baena-Lecuyer, M. Slimani and J. Zoellner, "DVB-T2: The Second Generation of Terrestrial Digital Video Broadcasting System", *IEEE Transactions on Broadcasting*, Vol. 60, No. 2, pp. 258–271, 2014.

12. Richardson, T. and S. Kudekar, "Design of Low-Density Parity Check Codes for 5G New Radio", *IEEE Communications Magazine*, Vol. 56, No. 3, pp. 28–34, 2018.

13. Hongwei Song, Jingfeng Liu and B. V. K. V. Kumar, "Low complexity LDPC codes for Partial Response Channels", *IEEE Global Telecommunications Conference*, Vol. 2, pp. 1294–1299, 2002.

14. Hongwei Song and B. V. K. Vijaya Kumar, "Low-density Parity Check Codes for Partial Response Channels", *IEEE Signal Processing Magazine*, Vol. 21, No. 1, pp. 56–66, 2004.

15. Hongwei Song, Jingfeng Liu and B. V. K. V. Kumar, "Large Girth Cycle Codes for Partial Response Channels", *IEEE Transactions on Magnetics*, Vol. 40, No. 4, pp. 3084–3086, 2004.

16. Djordjevic, I. B. and B. Vasic, "High Code Rate Low-density Parity-check Codes for Optical Communication Systems", *IEEE Photonics Technology Letters*, Vol. 16, No. 6, pp. 1600–1602, 2004.

17. Vasic, B., I. B. Djordjevic and R. K. Kostuk, "Low-density parity Check Codes and Iterative Decoding for Long-haul Optical Communication Systems", *Journal of Lightwave Technology*, Vol. 21, No. 2, pp. 438–446, 2003.

18. Feldman, J., M. Wainwright and D. Karger, "Using Linear Programming to Decode Binary Linear Codes", *IEEE Transactions on Information Theory*, Vol. 51, No. 3, pp. 954–972, 2005.

19. Taghavi, M.-H. and P. Siegel, "Adaptive Methods for Linear Programming Decod-

ing", *IEEE Transactions on Information Theory*, Vol. 54, No. 12, pp. 5396–5410, 2008.

20. Tanatmis, A., S. Ruzika, H. Hamacher, M. Punekar, F. Kienle and N. Wehn, "A Separation Algorithm for Improved LP-Decoding of Linear Block Codes", *IEEE Transactions on Information Theory*, Vol. 56, No. 7, pp. 3277–3289, 2010.

21. Zhang, X. and P. Siegel, "Adaptive Cut Generation Algorithm for Improved Linear Programming Decoding of Binary Linear Codes", *IEEE Transactions on Information Theory*, Vol. 58, No. 10, pp. 6581–6594, 2012.

22. Sarıduman, A., A. E. Pusane and Z. C. Taşkın, "An Integer Programming Based Trapping Set Search Technique", *20th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, 2012.

23. Sarıduman, A., A. E. Pusane and Z. C. Taşkın, "Adaptive Linear Programming for Decoding LDPC Codes", *22nd Signal Processing and Communications Applications Conference (SIU)*, pp. 706–709, 2014.

24. Richardson, T., "Error floors of LDPC Codes", *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, Vol. 41, pp. 1426–1435, 2003.

25. Cole, C. A., S. G. Wilson, E. K. Hall and T. R. Giallorenzi, "A General Method for Finding Low Error Rates of LDPC Codes", *CoRR*, Vol. abs/cs/0605051, 2006, `http://arxiv.org/abs/cs/0605051`.

26. Dolecek, L., Z. Zhang, M. Wainwright, V. Anantharam and B. Nikolic, "Evaluation of the Low Frame Error Rate Performance of LDPC Codes Using Importance Sampling", *IEEE Information Theory Workshop*, pp. 202–207, 2007.

27. Hu, X.-Y., E. Eleftheriou and D. Arnold, "Regular and Irregular Progressive Edge-Growth Tanner Graphs", *IEEE Transactions on Information Theory*, Vol. 51,

No. 1, pp. 386–398, 2005.

28. Xiao, H. and A. H. Banihashemi, "Improved Progressive-edge-growth (PEG) Construction of Irregular LDPC Codes", *IEEE Communications Letters*, Vol. 8, No. 12, pp. 715–717, 2004.

29. Khazraie, S., R. Asvadi and A. H. Banihashemi, "A PEG Construction of Finite-Length LDPC Codes with Low Error Floor", *IEEE Communications Letters*, Vol. 16, No. 8, pp. 1288–1291, August 2012.

30. Vukobratovic, D. and V. Senk, "Generalized ACE Constrained Progressive Edge-Growth LDPC Code Design", *IEEE Communications Letters*, Vol. 12, No. 1, pp. 32–34, 2008.

31. Healy, C. T. and R. C. de Lamare, "Design of LDPC Codes Based on Multipath EMD Strategies for Progressive Edge Growth", *IEEE Transactions on Communications*, Vol. 64, No. 8, pp. 3208–3219, 2016.

32. Diouf, M., D. Declercq, S. Ouya and B. Vasic, "A PEG-like LDPC Code Design Avoiding Short Trapping Sets", *2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 1079–1083, 2015.

33. Tasdighi, A., A. H. Banihashemi and M. Sadeghi, "Efficient Search of Girth-Optimal QC-LDPC Codes", *IEEE Transactions on Information Theory*, Vol. 62, No. 4, pp. 1552–1564, 2016.

34. Xu, H., D. Feng, R. Luo and B. Bai, "Construction of Quasi-Cyclic LDPC Codes via Masking With Successive Cycle Elimination", *IEEE Communications Letters*, Vol. 20, No. 12, pp. 2370–2373, 2016.

35. Wang, D., L. Wang, X. Chen, A. Fei, C. Ju and Z. Wang, "Construction of QC-LDPC Codes Based on Pre-Masking and Local Optimal Searching", *IEEE Communications Letters*, Vol. 22, No. 6, pp. 1148–1151, 2018.

36. Sadeghi, M., "Optimal Search for Girth-8 Quasi Cyclic and Spatially Coupled Multiple-Edge LDPC Codes", *IEEE Communications Letters*, Vol. 23, No. 9, pp. 1466–1469, 2019.

37. Karimi, M. and A. H. Banihashemi, "On Characterization of Elementary Trapping Sets of Variable-Regular LDPC Codes", *IEEE Transactions on Information Theory*, Vol. 60, No. 9, pp. 5188–5203, 2014.

38. Hashemi, Y. and A. H. Banihashemi, "On Characterization and Efficient Exhaustive Search of Elementary Trapping Sets of Variable-Regular LDPC Codes", *IEEE Communications Letters*, Vol. 19, No. 3, pp. 323–326, March 2015.

39. Dehghan, A. and A. H. Banihashemi, "Counting Short Cycles in Bipartite Graphs: A Fast Technique/Algorithm and a Hardness Result", *IEEE Transactions on Communications*, Vol. 68, No. 3, pp. 1378–1390, 2020.

40. Karimi, B. and A. H. Banihashemi, "Construction of QC LDPC Codes With Low Error Floor by Efficient Systematic Search and Elimination of Trapping Sets", *IEEE Transactions on Communications*, Vol. 68, No. 2, pp. 697–712, 2020.

41. Tao, X., Y. Li, Y. Liu and Z. Hu, "On the Construction of LDPC Codes Free of Small Trapping Sets by Controlling Cycles", *IEEE Communications Letters*, Vol. 22, No. 1, pp. 9–12, 2018.

42. Naseri, S. and A. H. Banihashemi, "Construction of Girth-8 QC-LDPC Codes Free of Small Trapping Sets", *IEEE Communications Letters*, Vol. 23, No. 11, pp. 1904–1908, 2019.

43. Naseri, S. and A. H. Banihashemi, "Spatially Coupled LDPC Codes With Small Constraint Length and Low Error Floor", *IEEE Communications Letters*, Vol. 24, No. 2, pp. 254–258, 2020.

44. Miwa, M., T. Wadayama and I. Takumi, "A Cutting-plane Method Based on

Redundant Rows for Improving Fractional Distance", *IEEE Journal on Selected Areas in Communications*, Vol. 27, No. 6, pp. 1005–1012, 2009.

45. Falsafain, H. and S. R. Mousavi, "A Generator-Matrix-Based Approach for Adaptively Generating Cut-Inducing Redundant Parity Checks", *IEEE Communications Letters*, Vol. 20, No. 4, pp. 640–643, 2016.

46. Zhang, Z., L. Dolecek, B. Nikolic, V. Anantharam and M. Wainwright, "GEN03-6: Investigation of Error Floors of Structured Low-Density Parity-Check Codes by Hardware Emulation", *IEEE Global Telecommunications Conference*, pp. 1–6, 2006.

47. Laendner, S., T. Hehn, O. Milenkovic and J. B. Huber, "The Trapping Redundancy of Linear Block Codes", *IEEE Transactions on Information Theory*, Vol. 55, No. 1, pp. 53–63, 2009.

48. Diao, Q., Y. Y. Tai, S. Lin and K. Abdel-Ghaffar, "Trapping set structure of LDPC codes on finite geometries", *Information Theory and Applications Workshop (ITA)*, pp. 1–8, 2013.

49. Hashemi, Y. and A. H. Banihashemi, "New Characterization and Efficient Exhaustive Search Algorithm for Leafless Elementary Trapping Sets of Variable-Regular LDPC Codes", *IEEE Transactions on Information Theory*, Vol. 62, No. 12, pp. 6713–6736, 2016.

50. Yang Han and W. E. Ryan, "LDPC Decoder Strategies for Achieving Low Error Floors", *Information Theory and Applications Workshop*, pp. 277–286, 2008.

51. Ivkovic, M., S. Chilappagari and B. Vasic, "Eliminating Trapping Sets in Low-Density Parity-Check Codes by Using Tanner Graph Covers", *IEEE Transactions on Information Theory*, Vol. 54, No. 8, pp. 3763–3768, 2008.

52. Krishnan, K. and P. Shankar, "Computing the Stopping Distance of a Tanner

Graph is NP-Hard", *IEEE Transactions on Information Theory*, Vol. 53, No. 6, pp. 2278–2280, 2007.

53. McGregor, A. and O. Milenkovic, "On the Hardness of Approximating Stopping and Trapping Sets", *IEEE Transactions on Information Theory*, Vol. 56, No. 4, pp. 1640–1650, 2010.

54. Abu-Surra, S., D. DeClercq, D. Divsalar and W. Ryan, "Trapping Set Enumerators for Specific LDPC Codes", *Process to Information Theory and Applications Workshop*, pp. 1–5, 2010.

55. Kyung, G. B. and C.-C. Wang, "Finding the Exhaustive List of Small Fully Absorbing Sets and Designing the Corresponding Low Error-Floor Decoder", *IEEE Transactions on Communications*, Vol. 60, No. 6, pp. 1487 –1498, 2012.

56. Sarıduman, A., A. E. Pusane and Z. C. Taşkın, "An Integer Programming-based Search Technique for Error-prone Structures of LDPC Codes", *AEU - International Journal of Electronics and Communications*, Vol. 68, No. 11, pp. 1097 – 1105, 2014.

57. Karimi, M. and A. H. Banihashemi, "Efficient Algorithm for Finding Dominant Trapping Sets of LDPC Codes", *IEEE Transactions on Information Theory*, Vol. 58, No. 11, pp. 6942–6958, 2012.

58. Xiao, H. and A. H. Banihashemi, "Error Rate Estimation of Low-density Parity-check codes on Binary Symmetric Channels Using Cycle Enumeration", *IEEE Transactions on Communications*, Vol. 57, No. 6, pp. 1550–1555, 2009.

59. Tanner, R. M., "Spectral Graphs for Quasi-cyclic LDPC Codes", *IEEE International Symposium on Information Theory*, p. 226, 2001.

60. Richardson, T. J. and R. L. Urbanke, "The Capacity of Low-density Parity-check Codes Under Message-passing Decoding", *IEEE Transactions on Information Theory*, Vol. 47, No. 2, pp. 599–618, 2001.

61. Tanner, R., D. Sridhara, A. Sridharan, T. Fuja and D. Costello Jr, "LDPC Block and Convolutional Codes Based on Circulant Matrices", *IEEE Transactions on Information Theory*, Vol. 50, No. 12, pp. 2966–2984, 2004.

62. Healy, C. T. and R. C. de Lamare, "Decoder-Optimised Progressive Edge Growth Algorithms for the Design of LDPC Codes with Low Error Floors", *IEEE Communications Letters*, Vol. 16, No. 6, pp. 889–892, 2012.

63. Jiang, X., X. Xia and M. H. Lee, "Efficient Progressive Edge-Growth Algorithm Based on Chinese Remainder Theorem", *IEEE Transactions on Communications*, Vol. 62, No. 2, pp. 442–451, 2014.

64. Venkiah, A., D. Declercq and C. Poulliat, "Design of Cages with a Randomized Progressive Edge-growth Algorithm", *IEEE Communications Letters*, Vol. 12, No. 4, pp. 301–303, 2008.

65. Kirkpatrick, S., C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671–680, 1983.

66. Tasdighi, A., A. H. Banihashemi and M. Sadeghi, "Symmetrical Constructions for Regular Girth-8 QC-LDPC Codes", *IEEE Transactions on Communications*, Vol. 65, No. 1, pp. 14–22, 2017.

67. Fossorier, M. P. C., "Quasicyclic low-density parity-check codes from circulant permutation matrices", *IEEE Transactions on Information Theory*, Vol. 50, No. 8, pp. 1788–1793, 2004.

68. Sarıduman, A., A. E. Pusane and Z. C. Taşkın, "On the Construction of Regular QC-LDPC Codes With Low Error Floor", *IEEE Communications Letters*, Vol. 24, No. 1, pp. 25–28, 2020.

APPENDIX A: Maximum Likelihood Ratio

For a transmitted code-word $\mathbf{x} \in \zeta$ and received vector $\mathbf{r}$, the maximum-likelihood code-word is

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\zeta}{argmax} P(\mathbf{r} \mid \mathbf{x}). \tag{A.1}$$

If channel is memoryless, (A.1) becomes

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\zeta}{argmax} \prod_{i=1}^{n} P(r_i \mid x_i). \tag{A.2}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\zeta}{argmin} \left(-ln \prod_{i=1}^{n} P(r_i \mid x_i) \right). \tag{A.3}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\zeta}{argmin} \left(-\sum_{i=1}^{n} lnP(r_i \mid x_i) \right). \tag{A.4}$$

If a constant variable, independent of $\mathbf{x}$, is added to equation, the result does not change.

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\zeta}{argmin} \left(\sum_{i=1}^{n} lnP(r_i \mid 0) - \sum_{i=1}^{n} lnP(r_i \mid x_i) \right). \tag{A.5}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\zeta}{argmin} \left(\sum_{i=1}^{n} ln \frac{P(r_i \mid 0)}{P(r_i \mid x_i)} \right). \tag{A.6}$$

The trick is that the summation of

$$\sum_{i=1}^{n} ln \frac{P(r_i \mid 0)}{P(r_i \mid x_i)} \tag{A.7}$$

is 0 if $x_i = 0$ and it is equal to

$$\sum_{i=1}^{n} ln \frac{P(r_i \mid x_i = 0)}{P(r_i \mid x_i = 1)} \tag{A.8}$$

if $x_i = 1$. Therefore, Equation (A.6) can be written as

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \in \zeta}{argmin} \left(\sum_{i=1}^{n} x_i \left(ln \frac{P(r_i \mid x_i = 0)}{P(r_i \mid x_i = 1)} \right) \right). \tag{A.9}$$

The coefficients of the maximum-likelihood objective function is then

$$\mu = ln \frac{P(r_i \mid x_i = 0)}{P(r_i \mid x_i = 1)}. \tag{A.10}$$

APPENDIX B: PROOF THEOREM 4.1

As given in Figure 4.10, the (5,3) ETS has only one topology structure. It is composed by three cycles with size 8. Let's assume that there is a candidate check node c_i in depth l_1 and l_2 when constructing the spreading tree with PEG algorithm from a variable,v_{root}. The copies of the check node,c_i, will be presented as c_i^{l1} and c_i^{l2}. When the root variable, v_{root}, is connected to c_i, at least two cycles with size g_1 and g_2, where $g_1 = l_1 + 1$, and $g_2 = l_2 + 1$ is created. The set of variables existing on the path from c^{l1} and c^{l2} to the root is presented as $Pv(c^1)$ and $Pv(C^2)$, respectively. Moreover, Definition 5 emphasizes that all check nodes must have two neighbours if it consists of a elementary trapping sets. Therefore, c^{l1} and c^{l2} must have the same parent in the tree. As a result of that, an extra cycle will be constructed when v_{root}, is connected to c_i.

(1) Let's assume that $l_1 \neq 7$, then $g_1 \neq 8$. Therefore, if the root is connected to c_i, there will be no cycle with size 8. It is same for the c_i^{l2}. Since (5,3) ETS is composed by three cycle, $l_1 = 7$ and $l_2 = 7$ must be satisfied.

(2) When $l_1 = 7$ and $l_2 = 7$, there are two different paths as $v_{root}, ..., c_i^{l1}$, and $v_{root}, ..., c_i^{l2}$. Two cycle with size 8 will be created in any circumstance. The third cycle with size 8 is created only if these two path has three common variable nodes. If n presents the common variable node between $Pv(c^1)$ and $Pv(C^2)$, the size of third cycle will be $g = 7 + 7 + 6 - 4n = 20 - 4n$. g will be 8, if only if $n = 3$.

APPENDIX C: PROOF THEOREM 4.2 and 4.3

The proof of Theorem 4.2 and 4.2 are almost identical. Therefore, only the proof of Theorem 4.2 will be given.

As given in Figure 4.11, the (6,4) ETS has two topology structure. For the topology given Figure 4.11(a), it is composed by two cycles with size 8 and a cycle with size 12. Let's assume that there is a candidate check node c_i in depth l_1 and l_2 when constructing the spreading tree with PEG algorithm from a variable,v_{root}. The copies of the check node,c_i, will be presented as c_i^{l1} and c_i^{l2}. When the root variable, v_{root}, is connected to c_i, at least two cycles with size g_1 and g_2, where $g_1 = l_1 + 1$, and $g_2 = l_2 + 1$ is created. The set of variables existing on the path from c^{l1} and c^{l2} to the root is presented as $Pv(c^1)$ and $Pv(C^2)$, respectively. Moreover, Definition 5 emphasizes that all check nodes must have two neighbours if it consists of a elementary trapping sets. Therefore, c^{l1} and c^{l2} must have the same parent in the tree. As a result of that, an extra cycle will be constructed when v_{root}, is connected to c_i.

(1) Let's assume that $l_1 = 7$ and $l_2 = 11$, Since $g_1 = 8$ and $g_2 = 12$, a cycle with size 8 should be created as an extra. If n presents the common variable node between $Pv(c^1)$ and $Pv(C^2)$, the size of third cycle will be $g = 7 + 11 + 6 - 4n = 24 - 4n$. Therefore, g will be 8, if only if $n = 4$.

(2) Let's assume that $l_1 = 7$ and $l_2 = 7$. Since $g_1 = 8$ and $g_2 = 8$, a cycle with size 12 should be created as an extra. If n presents the common variable node between $Pv(c^1)$ and $Pv(C^2)$, the size of third cycle will be $g = 7 + 7 + 6 - 4n = 20 - 4n$. Therefore, g will be 12, if only if $n = 2$.