INTEGER PROGRAMMING BASED ANALYSIS OF DECODING FAILURES FOR LDPC CODES

by

Abdullah Sarıduman

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

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

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2013

INTEGER PROGRAMMING BASED ANALYSIS OF DECODING FAILURES FOR LDPC CODES

APPROVED BY:

Assist. Prof. Ali Emre Pusane .....
(Thesis Supervisor)

Assoc. Prof. Z. Caner Taşkın .....
(Thesis Co-supervisor)

Prof. Hakan Deliç .....

Assist. Prof. Tınaz Ekim Aşıcı .....

Assoc. Prof. Mutlu Koca .....

DATE OF APPROVAL: 31.07.2013

ACKNOWLEDGEMENTS

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

Besides my co-advisors, I would like to thank the rest of my thesis committee: Prof. Hakan Deliç, Assist. Prof. Tınaz Ekim Aşıcı, and Assoc. Prof. Mutlu Koca for their encouragement and insightful comments.

I am also thankful to my close friends İsmail Terkeşli and Kenan Türksoy. 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 old 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.

Lastly, I would also like to thank to the Turkish Scientific and Technological Research Council (TÜBİTAK) for supporting me through BİDEB scholarship program. This thesis has been supported in part by TÜBİTAK under project number 111E276 and by the European Union under FP7 Marie Curie International Reintegration Grant number 268264.

ABSTRACT

INTEGER PROGRAMMING BASED ANALYSIS OF DECODING FAILURES FOR LDPC CODES

Low-density parity-check (LDPC) codes are one of the most efficient and common error correcting codes thanks to their high error performance. Using iterative message-passing decoding or linear programming (LP) decoding with large block sizes, LDPC codes can achieve near-capacity performance while maintaining almost linear encoding and linear decoding complexity (in block length). However, LDPC codes experience the error floor phenomenon in high signal-to-noise ratio (SNR) region due to the presence of small error prone structures in the Tanner graph representation of LDPC codes. The error floor is observed as the flattening of the error performance curve at high SNR values. In this thesis, an efficient, general framework is presented for finding common, devastating error prone structures of any finite length LDPC code. The smallest stopping set for the binary erasure channel (BEC), the smallest fully absorbing set, the smallest absorbing set, and the smallest elementary trapping set for the binary symmetric channel (BSC) are found and then an algorithm for enumerating small error prone structures is proposed. With the knowledge of error prone structures, it is possible to estimate the error floor performance of any LDPC code and increase its performance in the error floor region via carefully modifying its design.

ÖZET

LDPC KODLARIN KOD ÇÖZÜMÜ HATALARININ TAM SAYI PROGRAMLAMA TABANLI ANALİZİ

Düşük-yoğunluklu eşlik-denetim kodları (LDPC), yüksek hata performansları ve yaygın kullanım alanları sayesinde en verimli ve popüler hata düzelten kod ailelerindendir. Yinelemeli mesaj geçirme kod çözücüleri veya doğrusal programlama kod çözücüleri kullanarak çok büyük kod uzunlukları için LDPC kodlarla kanal kapasitesine yakın hata performansı elde edilebilir. Ancak, küçük boyutlu hataya-neden-olan-yapıların varlığı yüzünden LDPC kodları yüksek işaret gürültü oranları için hata tabanı sorununa sahiptir. Buna göre, yüksek işaret gürültü oranlarında koda ait hata başarım eğrisinin iyileşmesi durmakta ve işaret gürültü oranından bağımsız bir miktar hata sabit kalmaktadır. Bu tezde sonlu-uzunluklu LDPC kodlarında yaygın ve yıkıcı küçük hata yapılarının bulunması için genel bir en iyileme tabanlı yaklaşım sunulmuştur. İkili silme kanallar (BEC) için en küçük tıkayan küme, ikili simetrik kanallar (BSC) için en küçük yutan küme, tam-yutan küme ve basit tuzak küme bulunmuştur. Ayrıca bu küçük boyutlu sorunlu kümelerin tamamını bulabilen ve listeleyebilen bir algoritma geliştirilmiştir. Küçük özel hata yapıları bilgileri ile herhangi bir LDPC kodun hata tabanı performansını tahmin etmek ve kod tasarımında uygun değişiklikler yapılarak bu bölgedeki performansını artırmak mümkündür.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	Number of bit nodes in the Tanner graph of an error prone structure
b	Threshold for Gallager-B bit-flipping algorithm
C	Check node sets
c_i	i-th check node
$d(c_i)$	The degree of a check node c_i
$E(x)$	The set of check nodes in $N(x)$ with even degree in G_x
e_i	i-th excess variable
f_i	A binary representation of the situation of a check node c_i
G	The Tanner graph
$\mathbf{H}$	The parity check matrix
$h_{i,j}$	i-th row and j-th column of the parity check matrix
$\mathbf{I}$	Unit matrix
i	Identity of a check node
j	Identity of a variable node
k	Number of information bits in a codeword
m	Total check node numbers of a parity-check matrix
$N(c_j)$	The neighbors of j-th check node
n	Total numbers of bits in a codeword
$O(x)$	The set of check nodes in $N(x)$ with odd degree in G_x
p	Message from check node to variable node
q	Message from variable node to check node
R	Code rate
R_d	Design rate
$\mathbf{r}$	Binary received vector
S	Odd Sets
s_i	i-th slack variable
t	Number of check nodes with odd degree in the Tanner graph of an error prone structure

$\mathbf{u}$	Transmitted codeword vector
$\hat{\mathbf{u}}$	Estimated codeword vector
V	Variable node set
v_j	j-th variable node
$\mathbf{x}$	A subset of variable nodes
z	Objective value
β	Check node degree of a regular LDPC code
γ	Bit node degree of a regular LDPC code
μ	Cost vector of a maximum likelihood objective function
ζ	The codeword list of an LDPC code

LIST OF ACRONYMS/ABBREVIATIONS

AS	Absorbing Sets
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BP	Belief Propagation
BSC	Binary Symmetric Channel
EAS	Elementary Absorbing Sets
FAS	Fully Absorbing Sets
IP	Integer Programming
KKT	Karush-Kuhn-Tucker
LDPC	Low-Density Parity-Check
LP	Linear Programming
ML	Maximum Likelihood
PEG	Progressive Edge Growth
RPC	Redundant Parity Check
SNR	Signal-to-Noise Ratio
SPA	Sum-Product Algorithm
SS	Stopping Sets

1. INTRODUCTION

In digital communication systems, channel coding is used to transmit digital data and reduce the adverse effects of external factors such as noise, fading, and interference. Low-density parity-check (LDPC) codes, first proposed by Gallager [1] and later reintroduced by MacKay [2], have become one of the most focused areas in channel coding. The interest is particularly due to their near Shannon limit capacity under message-passing decoding algorithms. For instance, 10^6 bit error rate (BER) was achieved on the additive white Gaussian noise (AWGN) channel at 0.04 dB signal-to-noise ratio (SNR) away from the channel capacity by utilizing an LDPC code with block length 10^7 [3].

LDPC codes are known to experience the error floor phenomenon under iterative decoding. In the high SNR region, their frame error-rate performance abruptly changes and flattens out. Small error prone structures that exist in the bipartite graph representation of LDPC codes and low weight codewords are the main reasons of error floor phone phenomenon. For many LDPC codes, error prone structures are more dominant than the low weight codewords in the error floor region, as in the example of [4], since there are error prone structures with smaller sizes than the minimum codeword weight. All error prone structures come from cycles in the bipartite graph representation of LDPC codes, called the Tanner graph. However, the considered channel type determines the effects of the error prone structures on the decoding performance. For communication over the binary erasure channel (BEC), stopping sets are the main problematic structures, whereas trapping sets are the main problematic structures for communication over the binary symmetric channel (BSC), as well as the AWGN channel [5, 6]. After Richardson discovered the trapping sets, trapping sets received much attention. Among the trapping sets, it was proven that elementary trapping sets, absorbing sets, and fully absorbing sets causes the main negative contribution to the error floor performance, [6–8]. It was also shown that smaller size of error prone structures are more dominant than bigger ones in the error floor region due to fact that the probability of occurrence for small error structures is higher [6].

The knowledge of the small dominant problematic structures of an LDPC code allows us to approximate and/or bound its performance for maximum-likelihood decoding and iterative decoding. For example, In [6], Richardson uses trapping sets to estimate the error floor performance of LDPC codes. Moreover, this knowledge provides the code designer with a metric to further tweak the code design to achieve better error performance. For instance, [9] proposed an efficient method to improve LDPC code designs in terms of the error floor performance when small dominant problematic structures are known at the decoder. Therefore, a general framework to find dominant problematic structures is very useful in many ways.

As well as small-error prone structures, minimum distance is a vitally important code property of LDPC codes. Minimum distance is the smallest Hamming distance between any two distinct codewords of a code. There are several studies present in the literature that calculate the minimum distance of an LDPC code with different computational complexities. Although the calculation of the minimum distance is an NP-hard problem, a very efficient method is proposed in [10] to calculate the minimum distance of an LDPC code using the branch and cut algorithm. Its idea mainly inspired our work. In [10], integer programming (IP) is employed to model the code constraints and calculate the minimum distance. Similar to the minimum distance problem, finding the most dominant problematic structures, minimum trapping sets and minimum stopping sets, are not easy tasks; in fact, these problems are also shown to be NP-hard in [11, 12]. A search algorithm to find trapping sets was proposed in [13], but the proposed algorithm assumes that the complete cycle spectrum of the LDPC code is known; however, determining the complete cycle spectrum of a code is not an easy task, either. Another relevant work, [14], proposed an algorithm to find small error prone structures by using an IP algorithm to find lower bound, but it finds optimum solution for only short LDPC codes.

Utilization of IP for finding important parameters of an LDPC code was first introduced by [10], where the authors developed an IP model to perform the minimum distance computation of LDPC codes. In [15], IP was again utilized to compute lower bounds on the minimum absorbing set size and these bounds were later utilized in a

search algorithm to find absorbing sets. However, IP can already achieve the task of finding small absorbing sets without the need of an additional search algorithm.

1.1. Contribution of Thesis

In this thesis, we propose an efficient, general framework to find the smallest dominant problematic structures and enumerate small dominant problematic structures of an LDPC code by means of the IP optimization technique. In [6] and [7], it was shown that the problematic structures for communication over the BSC also cause problems for communication over the AWGN channel. Therefore, determining these small structures can be beneficial for the AWGN channel as well as the BSC. Our proposed method takes the parity-check matrix representation of an LDPC code as an input and finds the important parameters of an LDPC code, including the smallest absorbing set size, the smallest fully absorbing set size, the smallest elementary absorbing set size, and the smallest stopping set size. An iterative IP algorithm is then proposed to enumerate all error prone structures. The solution is provably optimal due to the use of IP, i.e., if the proposed algorithm succeeds, the output is guaranteed to be optimal. The proposed algorithm is tested for LDPC codes with lengths suitable for practical implementations and it is demonstrated that the algorithm executes in a reasonable amount of time using modern computers.

1.2. Thesis Outline

We begin the thesis in Chapter 2 with the basic relevant definition and notations about LDPC codes. Some popular decoding algorithms for LDPC codes are given in this chapter. Chapter 3 describes error prone structures where decoding algorithms fail and error floor problem occurs. Chapter 4 gives some background on integer programming optimization techniques. The proposed integer programming models for finding error prone structures is given in Chapter 5. Moreover, an algorithm to enumerate all error prone structures of a given code are described in this chapter. Chapter 5 also provides some numerical results. Finally, the thesis is concluded in Chapter 6.

2. LDPC CODES AND DECODING ALGORITHMS

Low-density parity-check codes were first developed by Gallager in 1962 [1]. For almost 40 years, LDPC codes unfortunately did not receive much attention and were almost forgotten. In 1995, MacKay and Neal rediscovered LDPC codes and attracted a great amount of interest to the high performance of LDPC codes [2]. It was shown that LDPC codes, when decoded with iterative decoding, achieve high error rate performance (very close to the Shannon limit [3]).

2.1. LDPC Codes

Definition 2.1. *A (β, γ)-regular LDPC code is the null space of a parity-check matrix* $\mathbf{H}$*, where rows consist of β 1's, columns consist of γ 1's, and most importantly both β and γ are very small in comparison with 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, $C = \{c_1, \ldots, c_m\}$ [16]. Variable nodes correspond to code symbols and check nodes correspond to parity-check equations. An edge connects a variable node to a check node if and only if the corresponding code symbol participates in the corresponding parity-check equation. 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 $|N(c_j)|$. The subgraph of a subset of V induced by these vertices, i.e., having both endpoints in this set of vertices is a bipartite graph, G, defined as the union of the neighborhoods of vertices in x. $E(x)$ represents the set of the check nodes in $N(x)$ with even degree in G_x and $O(x)$ represents the set of check nodes in $N(x)$ with odd degree in G_x.

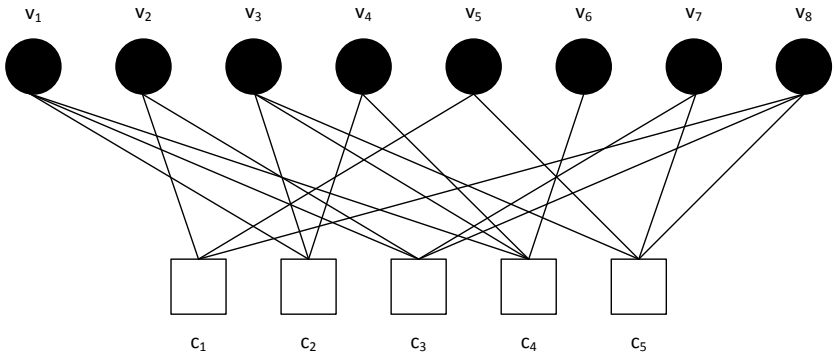

Figure 2.1. The Tanner graph representation of an (8,3) LDPC code.

The value of a check node is determined by its neighboring variable nodes. If summation of the value of the neighboring variable nodes is even, the corresponding check node is said to be satisfied. Otherwise, the check node is said to be unsatisfied. A vector $\mathbf{v} = \{v_1, \ldots, v_n\}$ is then a codeword, if and only if

$$\mathbf{vH^T} = 0 \quad mod(2). \tag{2.1}$$

All codewords of an LDPC code consist of codeword list, ζ, of the LDPC code.

Example 2.2. *The Tanner graph of an (8,3) LDPC code with parity-check matrix* $\mathbf{H}$ *is presented in Figure 1. The filled circles represent the variable nodes and the empty squares represent the check nodes. The corresponding parity-check matrix is given as*

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{2.2}$$

There are two different code rate descriptions of code rates for regular LDPC

codes. The first one, *the true rate*, is the classic rate:

$$R = \frac{k}{n}. \tag{2.3}$$

The second one is *the design rate*, defined as

$$R_d = 1 - \frac{\gamma}{\beta}. \tag{2.4}$$

It can be shown that

$$R \geq R_d. \tag{2.5}$$

The proof of Equation 2.5 can be given as:

$$R = \frac{k}{n} = \frac{n-(n-k)}{n} = 1 - \frac{n-k}{n} \geq 1 - \frac{m}{n} = 1 - \frac{\gamma}{\beta}. \tag{2.6}$$

In general, an (n, k) LDPC code has $m = n - k$ check equations. However, when some check equations are redundant, the number of check equations, m, is greater than $n-k$. $\frac{m}{n}$ is equal to $\frac{\gamma}{\beta}$, since the number of ones is $\mathbf{H}$ matrix is

$$m\beta = n\gamma. \tag{2.7}$$

In this regard, the code rate of the LDPC code given in Example 2.2 is

$$R = \frac{k}{n} = \frac{m-n}{n} = \frac{8-5}{8} = \frac{3}{8}. \tag{2.8}$$

In this example, k=$m - n$, since no check equation is redundant.

2.2. Decoding of LDPC Codes

When a codeword $\mathbf{u}$ is transmitted over a channel, the decoding algorithm tries to estimate $\hat{\mathbf{u}} \in \zeta$ by using the received vector $\mathbf{r}$.

2.2.1. Optimal Decoding

One way for decoding is simply to choose a codeword that has the highest probability to have been transmitted given a received vector. For optimal decoding, the estimated vector is

$$\hat{\mathbf{u}} = arg \max_{\mathbf{u}} P(\mathbf{u} \mid \mathbf{r}). \tag{2.9}$$

This kind of decoder is called *maximum a-posteriori* (MAP) decoder [17]. If all information codewords and the corresponding codewords $\mathbf{u} \in \zeta$ are equally likely, MAP can be rewritten as

$$\hat{\mathbf{u}} = arg \max_{\mathbf{u}} P(\mathbf{r} \mid \mathbf{u}). \tag{2.10}$$

Such a decoder is called *maximum likelihood* (ML) decoder [17].

2.2.2. Iterative Decoding

When an LDPC code with $\mid \zeta \mid = 2^k$ is considered, an optimal decoder must consider all possible codewords belonging to ζ. Since k is generally a big number, the size of ζ is considerably large. Therefore, computing the maximum likelihood codeword is an extremely complex process. 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 iteratively exchange messages with their neighboring variable and check

nodes. In the first iteration, the received vector is directly transmitted to the check nodes as incoming message, $q_{i,j}$, since there are no messages from the check nodes yet. After that, check nodes compute outgoing messages, $p_{j,i}$, 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 at

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} P(\mathbf{r} \mid u_i) \qquad i = 1, 2, ..., n \tag{2.11}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \frac{P(\mathbf{r} \mid u_i)P(u_i)}{P(u_i)} \tag{2.12}$$

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

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} P(\mathbf{r}, u_i) \tag{2.13}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} P(\mathbf{r}, \mathbf{u}) \tag{2.14}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} P(\mathbf{r} \mid \mathbf{u})P(\mathbf{u}) \tag{2.15}$$

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} P(\mathbf{r} \mid \mathbf{u}) \tag{2.16}$$

Equation 2.16 can be written as

$$\hat{u_i} = arg \max_{u_i \in \{0,1\}} \sum_{\mathbf{u} \in \zeta} \prod_{j=1}^{n} P(r_j \mid u_j), \tag{2.17}$$

since the channel is assumed to be memoryless. From Equation 2.11 to Equation 2.17 shows that the maximum likelihood symbol decision can be written as the sum-product equation. The most common message-passing algorithm, the sum-product algorithm

(SPA) (also know as belief propagation (BP)), is derived from Equation 2.17. 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. In SPA, exchanging messages are calculated as

$$q_{i,j} = log\frac{P(v_i = 0)}{P(v_i = 1)} + \sum_{j' \in N(i) \backslash j} p_{j'i}. \tag{2.18}$$

$$p_{j,i} = log\frac{1 + \prod_{i' \in N(j) \backslash j} tanh(q_{i'j})}{1 - \prod_{i' \in N(j) \backslash j} tanh(q_{i'j})}. \tag{2.19}$$

When hard-decision messages are received over BEC or BSC, outgoing messages from check nodes are calculated as summation of incoming messages in modulo-2 and incoming messages to check nodes are determined by the variable nodes in the previous iteration. One of the most common message passing algorithms based on hard-decision message exchange is Gallager's bit-flipping algorithm [1]. The bit-flipping algorithm, as its name suggests, flips the value of a variable node when at least b of its neighboring check nodes are unsatisfied. Such a flip would then reduce the number of unsatisfied check nodes in the Tanner graph. For the Gallager-A variant of the bit-flipping algorithm, b is exactly $|N(v_i)| - 1$ and it does not change with the iteration number. For the Gallager-B variant of the bit-flipping algorithm, b can be any number greater than $(|N(v_i)| - 1)/2$ and can also depend on the iteration number. The decoder continues flipping the variable nodes until a maximum number of iterations is reached or there are no remaining unsatisfied check nodes.

2.2.3. LP Decoding

Feldman in [18] introduced a linear programming (LP) based decoding algorithm for LDPC codes that has performance near the SPA algorithm and also possesses the maximum likelihood certificate, i.e., whenever it outputs a feasible solution, it is the same as the output of an ML decoder. This property of LP decoding is the reason of its growing popularity in the recent years.

To review Feldman's LP decoding for LDPC codes, the LP formulation with ML performance will be presented first. For a given code, ζ, the *codeword polytope* of ζ can be described as the convex hull of all possible codewords, given as

$$poly(\zeta) = \left\{ \sum_{u \in \zeta} \lambda_u u : \lambda_u \geq 0, \sum_{u \in \zeta} \lambda_u = 1 \right\}. \tag{2.20}$$

The vertices of $Poly(\zeta)$ are exactly codewords. The key point is that any linear programming attains its optimum at a vertex of its feasible polytope. Therefore, LP with the maximum likelihood objective function will have optimum performance. Coefficients of maximum likelihood objective function, $\mu = \{\mu_1, \mu_2, ..., \mu_n\}$, assuming that the channel is memoryless, can be written as

$$\mu_i = log \frac{P(r_i \mid v_i = 0)}{P(r_i \mid v_i = 1)}, \qquad i = 1, 2, ..., n. \tag{2.21}$$

The proof of maximum likelihood objective coefficients are given in Appendix A. To sum up, the LP model with maximum likelihood performance is

$$\text{minimize } \mu^T \mathbf{x} \tag{2.22}$$

$$\text{such that } \mathbf{x} \in poly(\zeta) \tag{2.23}$$

Solving a linear programming problem can be achieved in a practical way using the Simplex method, which requires a clear representation of linear constraints. $Poly(\zeta)$ can indeed be defined by a finite number of linear constraints. However, the number of these constraints increase exponentially in the codeword length. For this reason, it is impractical to design a maximum likelihood LP decoder. Therefore, Feldman characterized a new polytope, $Poly(A)$, that has vertices which consist of all of the vertices of $Poly(\zeta)$ and some additional fractional vertices.

In this approach, every check node, c_j, can be considered as a separate code and characterized with a polytope, which is the convex hull of all possible vectors that consist of $N(c_j)$ and satisfy the check node. In other words, we can simply call these vectors as local codewords.

Example 2.3. *For the LDPC code given in Example 2.2, the polytope of first check node is presented as Figure 2.2. The vertices of this polytope consist of local codewords of the first check node.*

For all of check nodes, the corresponding polytopes, $Poly(A_1), Poly(A_2), ..., Poly(A_m)$ can be generated as the convex hull of local codewords. Forbidding odd sets $S \in N(c_j)$ for the j. check node defines $Poly(A_j)$. For every check node c_j and all $S \subset N(c_j), \mid S \mid$ odd, the constraint sets

$$\sum_{x_i \in S} x_i + \sum_{x_i \in (N(c_j) \setminus S)} (1 - x_i) \leq \mid N(c_j) \mid -1 \tag{2.24}$$

constitute $Poly(A_j)$. The relaxed LP polytope is the intersection of local polytopes, obtained as

$$Poly(A) = Poly(A_1) \cap Poly(A_2) ... \cap Poly(A_m). \tag{2.25}$$

Thus, the relaxed LP decoding model will be given as

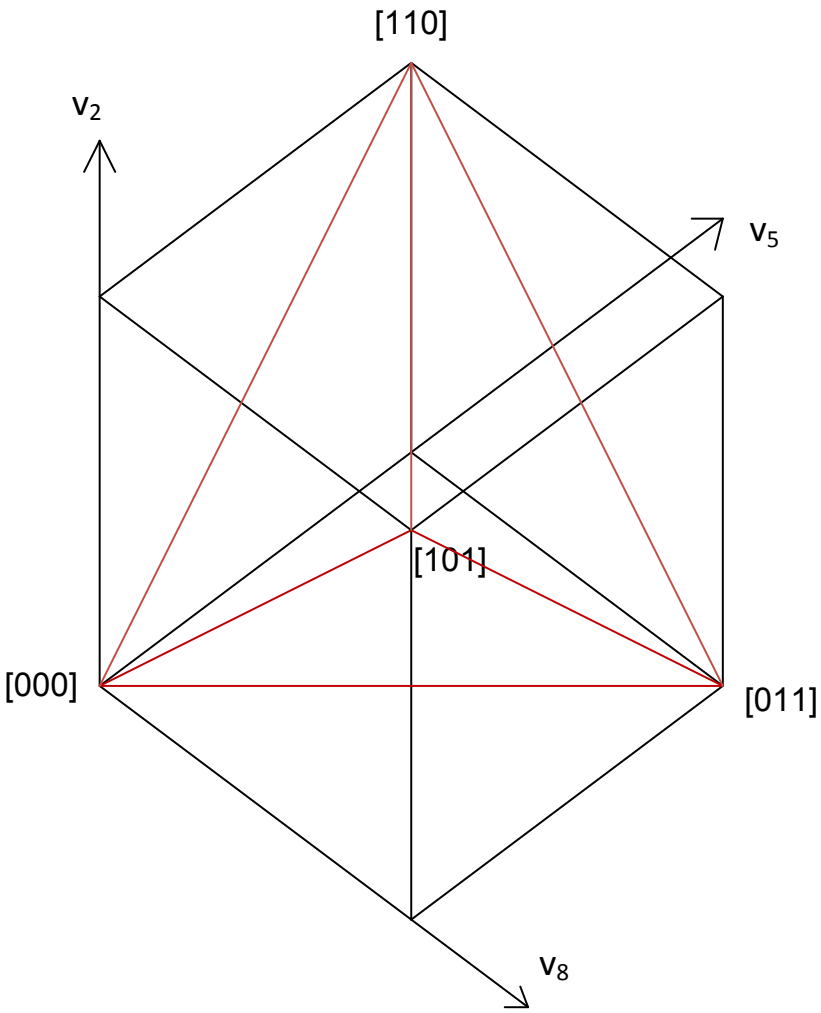

Figure 2.2. The convex hull of local codewords.

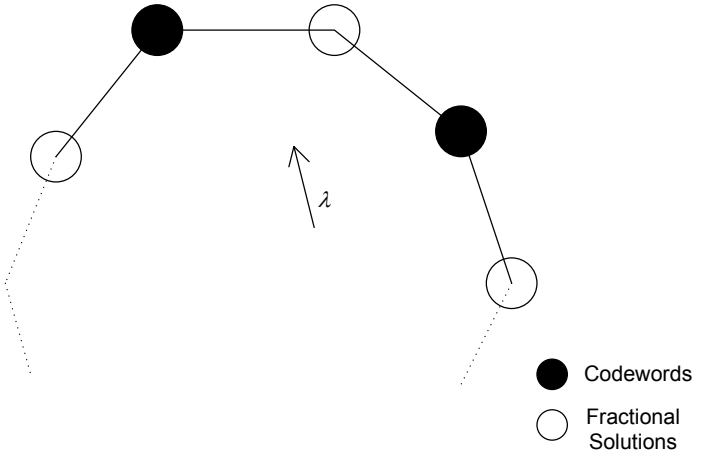

Figure 2.3. Description of $Poly(A)$.

$$\text{minimize } \mu^T \mathbf{x} \tag{2.26}$$

$$\text{such that: } \mathbf{x} \in poly(A) \tag{2.27}$$

Vertices having integer values in the $Poly(A)$ are the codewords of the LDPC code and the rest of vertices have fractional values. The two dimensional description of $Poly(A)$ is shown in Figure 2.3. When LP finds an integer solution, it is guaranteed that the solution is ML. When it outputs a fractional solution, it is accepted that decoding fails. For fractional solutions, a cut can be generated to increase performance of LP decoding. For example, Siegel added redundant parity check nodes (RPC) to eliminate fractional solutions and increase the performance of LP decoding [19].

3. ERROR PRONE STRUCTURES OF LDPC CODES

Error prone structures are considered to be the main reason of error floor problems. Properties of error prone structures are established by the channel type.

3.1. BEC Channel

A binary erasure channel is a communication channel model used frequently in information theory due to its simplicity. The variable nodes in the received vector in this model can be either transmitted bit or erased bit. The main goal of decoding for communication over BEC is to get rid of these erased bits. In order to achieve this task, the erasure decoder relies on a simple observation: if a single erased symbol participates in a parity-check equation, the original value of that code symbol is simply the modulo 2 summation of the other code symbols. Therefore, the erasure decoder processes all check nodes in the Tanner graph iteratively and inspects their neighborhoods. If there is only one erased symbol in the neighborhood, then its value can be restored and decoding continues. The decoder continues processing check nodes until a maximum number of iterations is reached or there are no remaining erased symbols.

The iterative decoder sometimes gets stuck, since, although there are still erased symbolS, there is not any one erased symbol in the neighborhood of any check nodes. In this situation, all check nodes have either zero or at least two erased symbols in their neighborhoods. Therefore, it is vital to identify these configurations, namely stopping sets (SS), and study their structures.

Definition 3.1. *$x \subseteq V$ is a SS if $|N(c_j) \cap G_x| \geqslant 2$, $\forall c_j \in N(x)$.*

When all variable nodes in any stopping sets are erased, then the iterative decoding will get stuck.

Example 3.2. *For the (8,3) LDPC code, whose Tanner graph is given in Figure 2.1, $x = \{v_2, v_5, v_7\}$ is a SS, since the degrees of check nodes c_1, c_3, and c_5 are all equal to*

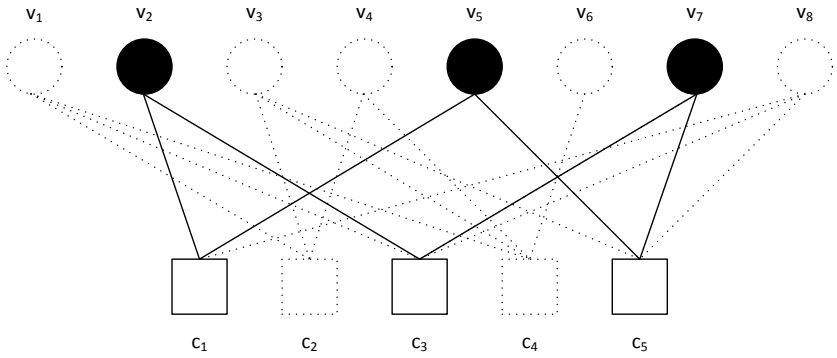

Figure 3.1. The Tanner graph of a stopping set.

two in G_x, given in Figure 3.3.

3.2. BSC and AWGN Channel

For communication over the BSC, the received vector consists of correctly and erroneously received symbols and it is the decoder's task to determine which symbols are in error. Since we deal with binary symbols, determining the error location is equivalent to restoring the original value of the symbol.

A decoding error occurs under Gallager's bit flipping algorithms when there are still unsatisfied check nodes in the graph, yet the decoder fails to flip any more variable nodes. This occurs when each variable node in the graph has more satisfied neighboring check nodes than unsatisfied ones. Error configurations that give rise to such a result are therefore of interest to us.

We begin by defining the structure of trapping sets, the general framework of error prone structures in the Tanner graph.

Definition 3.3. *$x \subseteq V$ is an (a, t) trapping set if $|x| = s$ and $|O(x)| = t$.*

An (a, t) 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 t 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 3.4. *An (a, t) elementary trapping set x is an (a, t) trapping set such that $|N(c_j)| = 1$, $\forall c_i \in O(x)$ and $|N(c_j)| = 2$, $\forall c_j \in E(x)$.*

This definition focuses on structures whose induced graphs consist of check nodes with minimal degrees. This makes it much harder to feed independent information to a check node from other correctly received symbols and help the decoding process.

Although trapping sets and elementary trapping sets vaguely describe the potential damage they could cause to the decoding process, they are mostly conceptual structures and their damage is heavily dependent on the chosen value of the (a, t) pair. A subclass of them, called absorbing sets, provide us with a more realistic definition. 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. This means that an AS, on itself, would be able to stop the decoding process. In fact, for the Gallager-B variant of the bit-flipping algorithm with invariant threshold b=$(|N(v_i)| - 1)/2$, all of the trapping sets are exactly AS.

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 3.5. *$x = \{v_1, v_3\}$ is an AS for the Tanner graph in Figure 2.1, 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*

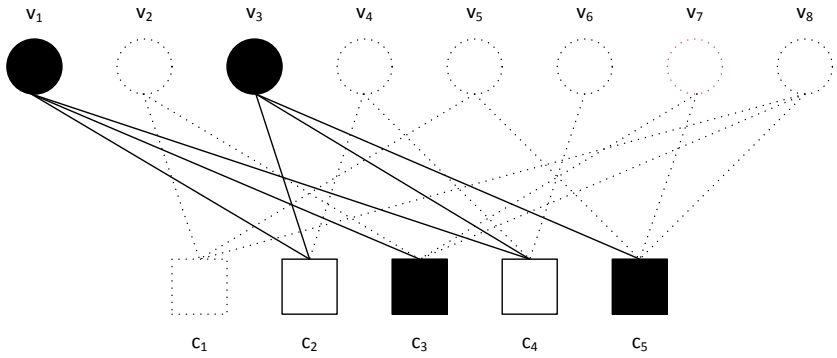

Figure 3.2. The Tanner graph of an absorbing set.

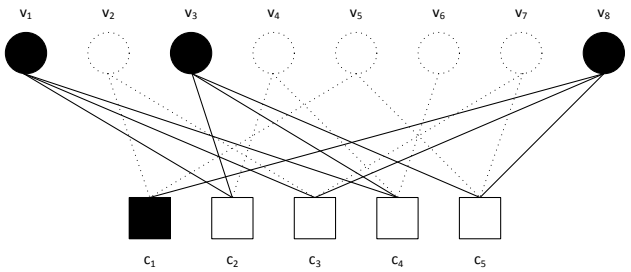

Figure 3.3. The Tanner graph of a fully absorbing set.

every check node in G_x has degree two.

Example 3.6. *$x = \{v_1, v_3, v_8\}$ 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.*

For communication over the BSC, it is clearly seen that these error prone structures cause problems under Gallager's bit flipping algorithm. More interestingly, Richardson in [6] showed that these problematic structures also lead to problems for communication over the AWGN channel under soft decision algorithms. Therefore, finding these structures will be helpful for not only communication over BSC but also for communication over AWGN channel.

4. INTEGER PROGRAMMING

Integer programming is one of the most popular linear optimization techniques and is widely used for modeling and solving optimization problems encountered in many different areas, such as traveling salesman problems, assignment problems and Knapsack problems. In this method, every decision point is defined as a *decision variable*. Linear functions of decision variables constitute *constraints* that are satisfied by feasible solutions. The quality of feasible solutions are determined by *the objective function*, which is also a linear function of decision variables. A general form of an integer programming problem is given as

$$\text{minimize } \mathbf{c}^T\mathbf{x} \tag{4.1}$$

$$\text{such that } \mathbf{Ax} = \mathbf{b} \tag{4.2}$$

$$\mathbf{Dx} \geq \mathbf{e} \tag{4.3}$$

$$\mathbf{Fx} \leq \mathbf{g} \tag{4.4}$$

$$\mathbf{x} \in Z. \tag{4.5}$$

It is NP-hard to solve IP problems [20]. However, several efficient methods have been proposed to solve IP problems in a reasonable time. IP problems could be efficiently solved for quite large problems by using these methods. The most popular method first converts the problem from an integer programming problems to a linear programming problem. Then, the optimum solution of LP relaxation is found with the Simplex method. If all the decision variables are integers, then the solution of LP relaxation is also optimum for the integer programming problem. The issue is that the optimum solution of LP relaxation could be a fractional point, which is not feasible in the original IP problem. Wolsey in [20] showed that branch-bound and branch-cut algorithms can deal with this problem by cutting fractional solutions. LP relaxation

with these algorithms will always find the optimum solution of IP problem.

4.1. Linear Programming Relaxations

The LP relaxation of an IP problem can be simply obtained by relaxing the integrality restrictions on the decision variables. A general form of an LP model is given as

$$\text{minimize } \mathbf{c}^T\mathbf{x} \tag{4.6}$$

$$\text{such that } \mathbf{Ax} = \mathbf{b} \tag{4.7}$$

$$\mathbf{Dx} \geq \mathbf{e} \tag{4.8}$$

$$\mathbf{Fx} \leq \mathbf{g} \tag{4.9}$$

$$\mathbf{x} \geq 0. \tag{4.10}$$

The feasible region of the LP relaxation is the extended version of the feasible region of the IP problem. If we define the feasible region of the LP relaxation as P, the feasible region of the original IP problem is $P \cap \mathbb{Z}^n$. As $(P \cap \mathbb{Z}^n) \subset P$ and the objective function is still same, this is clearly a relaxation. The solution of the LP relaxation gives a lower bound for the IP problem.

4.2. The Simplex Algorithm

Since the LP problem is convex programming problem, several optimization methods can be used to solve an LP problem, such as Karush-Kuhn-Tucker (KKT) method, Newton's method. Most of these methods do not take advantage of simplicity of linear problems. Therefore, the Simplex algorithm is mostly preferred. It can find the optimum solution of LP problems in polynomial time. The Simplex algorithm can be used, if and only if an LP problem has *standard form*. In this standard form, all constrains

are equations and all variables are non-negative. Any LP can be converted to a equivalent problem that has standard form. For each $\leq$ constraint, a *slack variable*, s_i, is defined, and for each $\geq$ constraint an *excess variable*, e_i, is defined. Adding slack variables to less than or equal to constraints and subtracting excess variable from greater than or equal to constraints can convert any LP problem to a problem in standard form. The standard form of an LP problem is, therefore, given as

$$\text{maximize } \mathbf{c}^T\mathbf{x} \tag{4.11}$$

$$\text{such that: } \mathbf{Ax} = \mathbf{b} \tag{4.12}$$

$$x_i \geq 0 \qquad (i = 1, 2, ..., n). \tag{4.13}$$

If it is assumed that $\mathbf{Ax} = \mathbf{b}$ has m linear equations and n decision variables, where $m \leq n$, a basic solution could be obtained by setting $n - m$ variables to 0 and solving problems for the remaining m variables. These $n - m$ variables are called *nonbasic variable* and these m variables called as *basic variables*. If the solution doesn't include any negative variables, it will be a basic feasible solution and a start point for the Simplex algorithm.

Theorem 4.1. *A point in the feasible region of an LP is an extreme point if and only if it is a basic feasible solution to the LP [21].*

Theorem 4.1 implies that the Simplex algorithm will start with an initial point at a corner of feasible region. After that, a nonbasic variable is exchanged with a basic variable (a nonbasic variable becomes a basic variable, and vice versa) to increase the value of objective function. When entering a nonbasic variable into basis, all nonbasic variables are searched and the one which increases objective function mostly is picked up as *entering variable*. If the solution of new problem is feasible, another extreme point is reached with better objective value. Iteratively, objective function value is increased until any candidate of entering variable can not increase objective function

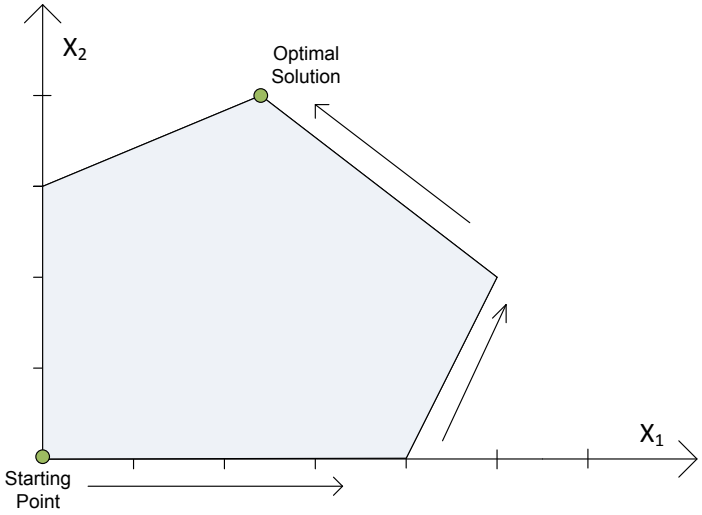

Figure 4.1. The Simplex algorithm.

value. In this point, the optimum solution of LP is found. Figure 4.1 shows how the Simplex algorithm gets through to the optimal point.

4.3. Branch-Bound and Branch-Cut Algorithms

The solution of LP relaxation gives a lower bound or upper bound (depends on either maximization problem or minimization problem) for optimum solution and therefore, branch-bound algorithms could be used. Branch-bound methods implicitly enumerate feasible points and find an optimal solution of IP problems. In branch-bound algorithms, LP relaxation of an IP problem is firstly solved. If all of the decision variables are integers, then the solution is also an optimum solution for the IP problem. If the optimal solution of LP is fractional, then two or more child nodes are generated by branching on fractional decision variables. If a decision variable, x_i, is a fractional number between l and $l + 1$ where $l \in Z$, then child nodes could be generated with adding new constraints as $x_i \geq l + 1$ and $x_i \leq l$. In this situation, the optimum fractional solution of parent node is not feasible in new child nodes and the optimum integer solution of parent node is still feasible for child nodes. A node is fathomed when it is not necessary to branch anymore. A node is fathomed in these situations:

- The node gives an integer feasible solution. If the objective value of the solution is better than current best solution, it becomes the new *candidate* solution.

- The node gives a fractional solution, but the fractional solution of the node is not better than the current *candidate* solution. In this case, it is certain that it can not yield a better solution than the current best solution, since the branching method doesn't increase the objective function value of child nodes.
- There is no feasible point for the node.

Improvements on branch-bound algorithms could decrease enumeration complexity by means of the reduction in required number of nodes for the proof of optimality. Introducing cuts into the branch-bound algorithm is one of these improvements.

Branch-cut algorithms produce new inequalities, called as *cuts*, which reduce the feasible region of relaxed LP. If the current optimal solution of LP is outside of this smaller feasible region, cuts are called *violated cuts*. It should be emphasized that new feasible regions of nodes must still cover the feasible region of the original IP problem. After solving a node and observing that it cannot be fathomed, a violated cut is searched. A violated cut sometimes can not be found. In this case, the node is branched. Finding strong cuts is not an easy task in optimization theory. However, their contribution can significantly decrease the overall computation time.

Example 4.2. *An integer program is given as*

$$\textit{maximize } 3x_1 + 2x_2 \tag{4.14}$$

$$\textit{s.t.: } x_1 + 2x_2 \leq 6 \tag{4.15}$$

$$3x_1 - x_2 \leq 3 \tag{4.16}$$

$$x_1, x_2 \in \mathbb{N}. \tag{4.17}$$

To solve this integer program problem, the LP relaxation of the original problem obtained by dropping the integrality restriction of decision variables, x_1 and x_2, is

Table 4.1. Canonical Form 1.

Row		Basic Variable
0	z - $3x_1$ - $2x_2$ = 0	z=0
1	x_1 + $2x_2$ + s_1 =6	s_1=6
2	$3x_1$ - x_2 + s_2 =3	s_2=3

solved with the Simplex algorithm. The relaxation of the IP problem is not in the standard form. Therefore, slack variables are used to make all constraints equations. The standard form of LP relaxation is given as

$$\text{maximize } z = 3x_1 + 2x_2 \tag{4.18}$$

$$\text{s.t.: } x_1 + 2x_2 + s_1 = 6 \tag{4.19}$$

$$3x_1 - x_2 + s_2 = 3 \tag{4.20}$$

$$x_1, x_2, s_1, s_2 \geq 0. \tag{4.21}$$

A system of linear equations in which each equation has a variable with a coefficient of 1 in that equation is in *canonical form*. The standard form of LP could be presented in the canonical form and a basic solution could be obtained by setting variables with a coefficient of 1 in each equation as a basic variable. The canonical form 1 presented in Table 4.1 has basic variables, s_1 and s_2, and a basic solution with objective value 0. The initial feasible solution is $x_1 = x_2 = 0$.

After obtaining a feasible solution, the Simplex algorithm will search for a better feasible solution. In the second iteration, x_1 is chosen as entering variable, since a unit increase in x_1 will cause the largest rate of increase in objective value, z. Therefore, the Simplex algorithm tries to increase x_1 as large as possible. However, increasing x_1 will change the values of current basic variables and it may cause a basic variable to become negative. The Simplex algorithm increases x_1 as long as current basic variables

Table 4.2. Canonical Form 2.

Row										Basic Variable
0	z	-	$3x_2$			+	s_2	$= 0$		z=3
1			$\frac{7}{3}x_2$	+	s_1	-	$\frac{1}{3}s_2$	=5		s_1=5
2	x_1	-	$\frac{1}{3}x_2$			+	$\frac{1}{3}s_2$	=1		x_1=1

Table 4.3. Canonical Form 3.

Row								Basic Variable
0	z		+	$\frac{9}{7}s_1$	+	$\frac{4}{7}s_2$	$= 0$	z=$\frac{66}{7}$
1		x_2	+	$\frac{3}{7}s_1$	-	$\frac{1}{7}s_2$	$= \frac{15}{7}$	$x_2 = \frac{15}{7}$
2	x_1		+	$\frac{1}{7}s_1$	+	$\frac{2}{21}s_2$	$= \frac{12}{7}$	$x_1 = \frac{12}{7}$

are non-negative. In row 1, x_1 could be at most 6, and x_1 could be at most 1 in row 2. Since row 2 requires smaller x_1 than row 1, row 2 is chosen as *pivot row*. Therefore, x_1 will be a basic variable in row 2. Using elementary row operations, x_1 can become a basic variable.

$$row0 + row2 \rightarrow row0 \tag{4.22}$$

$$row1 - \frac{row2}{3} \rightarrow row1 \tag{4.23}$$

$$\frac{row2}{3} \rightarrow row2 \tag{4.24}$$

The canonical form 2 in Table 4.2 yields the basic feasible solution. It is still possible to increase the objective value. The only non-basic variable that can increase the objective value is x_2. x_2 is chosen as the entering variable. Row 1 limit on $x_2 \leq \frac{15}{7}$ and row 2 does not limit on x_2. In this regard, row 1 is chosen as the pivot row. Using some elementary row operations, the canonical form 3 can be generated as in Table 4.3. The canonical form 3 yields the basic feasible solution with $z = \frac{66}{7}, x_1 = \frac{15}{7}, x_2 = \frac{12}{7}$. Now it is not possible to increase objective value, since row 0 doesn't have any non-basic variables with negative coefficient. Changing s_1 or s_2 will decrease objective value. Therefore, the point the Simplex algorithm reached is the optimum solution of the LP relaxation problem.

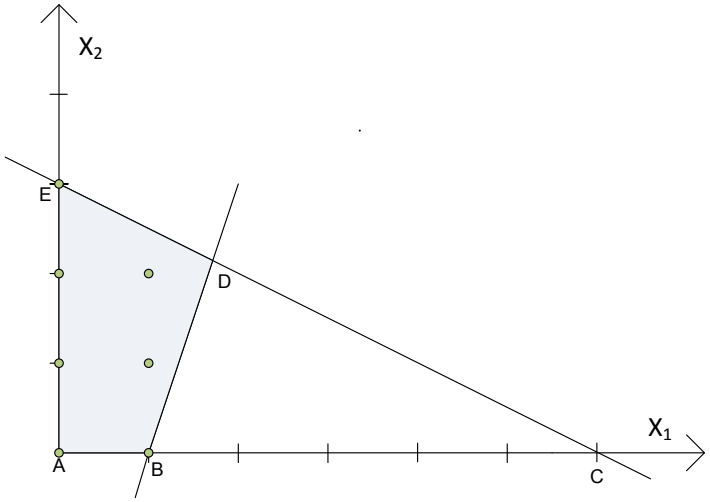

Figure 4.2. Example of an integer program.

The feasible region of LP relaxation and integer solutions are shown in Figure 4.2. The filled circles represent integer feasible solutions and shaded region represents the feasible region of the relaxed LP. We show that the Simplex algorithm first finds the extreme point A, then it reaches B and finally it finds the optimal point D. However, the optimum solution of the Simplex algorithm is fractional. For this reason, a branch or a cut is necessary. For instance, adding $x_1+x_2 \leq 3$ as a constraint will cut the current fractional optimum solution and all integer feasible solution will satisfy this constraint. However, in most cases, finding a strong cut is very hard, and therefore branching is more preferable. In this problem, we can branch on x_1 by using the constraints $x_1 \leq 1$ and $x_2 \geq 2$. Clearly, these constraints will cut the current fractional solution. Two linear programs will be generated with adding these constraints as seen in Figure 4.3. The feasible region of linear relaxations is shown in Figure 4.4

Whereas the first child node has a feasible region, the second child node doesn't have any feasible regions. As a result of this, the second child is fathomed. The Simplex algorithm will, therefore, search for an optimum solution of first node. If the optimum solution is integer, it will also be an optimum solution for IP problem. If the solution has fractional variables, then another execution of branching method could be used to reach integer solutions.

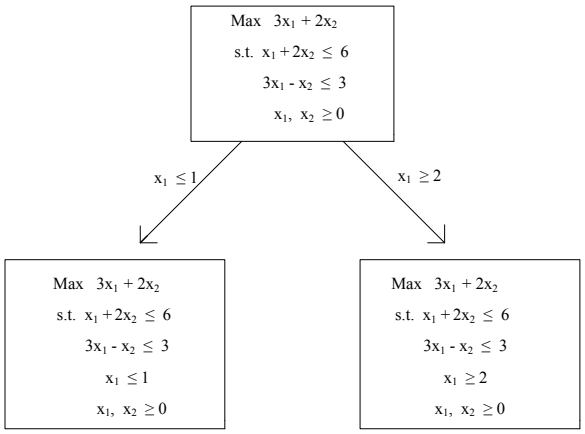

Figure 4.3. Example of branching.

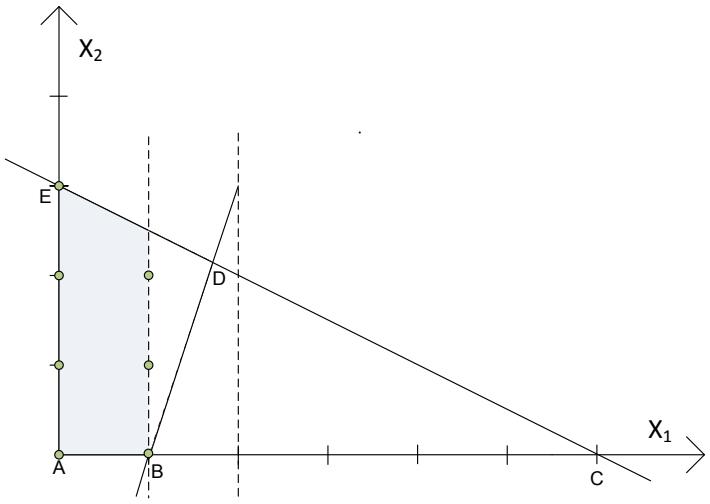

Figure 4.4. Feasible regions after branching.

5. OPTIMIZATION BASED ERROR PRONE STRUCTURES SEARCH

5.1. Finding Minimum Error Prone Structures

In this chapter, the proposed IP models to search for dominant problematic error sets for (n, k) LDPC codes with associated parity-check matrices $\mathbf{H}_{m\times n}$ over the BEC, BSC are developed and their performance is demonstrated via computer simulations.

In simulations, in order to demonstrate the applicability of the proposed optimization algorithms, both a family of circulant-based quasi-cyclic LDPC codes, namely Tanner codes [22], and families of randomly constructed LDPC codes, both fully random permutation matrix-based codes and structurally random codes obtained by the progressive edge growth (PEG) algorithm [23] are considered. LDPC codes obtained by using the PEG algorithm have the randomness property of the underlying code as well as maximal girth values, a property that manifests itself in increased values for the smallest error prone substructure sizes. The PEG construction relies on establishing edges in the Tanner graph of the LDPC codes in a way that avoids small cycles. The selection of an edge is determined by the potential impact on the girth. After the Tanner graph is updated by establishing the best-choice edge, the placement procedure is repeated for the remaining edges. Randomly constructed LDPC codes, because of the lack of the algebraic structure, have some problems such as encoding and implementation complexities. Algebraically constructed LDPC codes can deal with these issues, however, the girth, minimum distance, and error performance of algebraically constructed LDPC codes are not good as those of random codes. In the recent literature, the Tanner code family, comprised of blocks of circulant matrices, is proposed as one of best algebraically constructed LDPC code families with high minimum distance values [22]. In all of our IP models, the decision variables represent the variable nodes and it is assumed that a decision variable takes on the value 1 if the corresponding variable node is in error (in the BSC case) or is erased (in the BEC case).

5.1.1. Stopping Sets

The definition of a stopping set, given in Definition 3.1, indicates that a subset x of the variable nodes (used to represent the erased nodes) requires that the connected check nodes (in the induced graph) have multiple connections to x. Using this notation, the IP model is to find the minimum SS size over the BEC can be given as

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{5.1}$$

$$\text{s.t.: } 2y_i \leq \sum_{j=1}^{n} h_{i,j}x_j \quad i = 1, 2, \ldots, m \tag{5.2}$$

$$\sum_{j=1}^{n} h_{i,j}x_j \leq \sum_{j=1}^{n} h_{i,j}y_i, \quad i = 1, 2, \ldots, m \tag{5.3}$$

$$\sum_{i=1}^{n} x_i \geq 1 \tag{5.4}$$

$$y_i \in \{0, 1\}, \quad i = 1, 2, \ldots, m \tag{5.5}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n. \tag{5.6}$$

A decision variable, x_j, is equal to 1 if and only if the corresponding variable node is a member of the stopping set. Therefore, in order to find the minimum SS, the objective function given in (5.1) is simply defined as the summation of the decision variables. It is known that a check node in the existence of a stopping set has degree either zero (check node outside the induced subgraph) or at least two (check node inside the induced subgraph). To be able to model these two conditions, we first calculate the degree of a check node c_i, $i = 1, 2, \ldots, m$, as $d(c_i) = \sum_{j=1}^{n} h_{i,j}x_j$. In (5.2), we define a binary variable y_i that determines whether check node c_i is in the induced subgraph: y_i takes on the value 1 if c_i is in the induced subgraph and the value 0, otherwise. If the check node is in the induced subgraph ($y_i = 1$), the constraint (5.2) guarantees

that the degree $d(c_i)$ is at least two and the constraint (5.3) becomes inactive, since

$$\sum_{j=1}^{n} h_{i,j}x_j \leq \sum_{j=1}^{n} h_{i,j} \tag{5.7}$$

is satisfied for every possible binary $\mathbf{x}$ vector. If the check node is not in the induced subgraph ($y_i = 0$), the constraint (5.3) guarantees that $\sum_{j=1}^{n} h_{i,j}x_j$ is zero. Hence, (5.2) and (5.3) guarantee that the degree $d(c_i)$ is either zero or at least two for all $i = 1, 2, \ldots, m$. Finally, the constraint (5.4) is necessary to make the solution $x_j = 0$, $j = 1, 2, \ldots, n$, an infeasible choice.

The IP optimizations are performed on a computer with 2.27 GHz Intel Xeon CPU and 12 GB RAM using the CPLEX 12.3 software. Table 5.1 shows the smallest SS sizes (the variable nodes numbers in the smallest SS) of LDPC codes of different lengths as well as the time (in seconds) it takes to obtain these results. The optimizations are terminated when the program runs for more than 3 hours working time or more than 3 GB of memory is utilized. Although these computations do not require real-time processing, a 3GB memory / 3 hours time limit is imposed to terminate the optimization. For complete calculations, the smallest stopping sets sizes are presented as optimal. For incomplete calculations, we also present the lower and upper bounds (LB and UB) on the optimal value based on the state of the solver at the time of termination.

The result shows that IP is not efficient in finding the minimum SS size. Although optimal solutions for the IP problems can be obtained easily for block lengths up to 1,000, for larger block lengths, the defined termination criteria are met and the solver provides us with bounds on the optimal solution.

5.1.2. Fully Absorbing Set

Although fully absorbing sets are subsets of absorbing sets, they are easier to define using an IP model, and therefore, the IP model of finding the minimum fully

Table 5.1. Minimum SS of LDPC codes.

		Minimum Stopping Sets			
Code Structure	Size	Optimal	Lower Bound	Upper Bound	Time
Algebraic (Tanner Codes)	55	6			0
	155	18			247
	310	18			620
	620	18			761
	775	24			3763
	1085	time	21	24	10800
	2170	time	17	24	10800
	4340	time	16	24	10800
	8680	time	4	24	10800
Random (PEG)	54	4			0
	156	8			2
	312	11			30
	504	11			69
	756	memory	12	34	9096
	1008	time	5	44	10800
	2400	time	2	214	10800
	4200	time	2	414	10800
	8010	memory	1	743	1448
Random (Permutation)	54	2			2
	156	4			5
	312	2			5
	504	2			40
	756	8			952
	1008	memory	5	336	5961
	2400	time	3	800	10800
	4200	time	3	1400	10800
	8010	memory	1	2670	1370

absorbing set is given first.

The vector x in this definition determines the positions of errors and can be observed at either at the channel output or at some point during the iterations. An IP model to find the minimum FAS size over the BSC can be given as

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{5.8}$$

$$\text{s.t.:} \sum_{j=1}^{n} h_{i,j} x_j + f_i = 2d_i,\ i = 1, 2, \ldots, m \tag{5.9}$$

$$\sum_{i=1}^{m} h_{i,j} f_i \leq \sum_{i=1}^{m} h_{i,j}/2,\ \ j = 1, 2, \ldots, n \tag{5.10}$$

$$\sum_{i=1}^{m} f_i \geq 1 \tag{5.11}$$

$$f_i \in \{0, 1\},\ \ d_i \in \mathbb{Z},\ \ i = 1, 2, \ldots, m \tag{5.12}$$

$$x_j \in \{0, 1\},\ \ j = 1, 2, \ldots, n. \tag{5.13}$$

By definition, an error set is a FAS if and only if every variable node in the graph has more even-degree neighbors than odd-degree neighbors. This simple condition is sufficient to determine whether an error set is a FAS. We define a binary variable f_i to represent the situation of the check node c_i: f_i takes on the value 1 is c_i is unsatisfied and the value 0, otherwise. These values are set via the constraint set (5.9). The constraint set (5.10) guarantees that the number of odd-degree neighbors of a variable node v_j is less than half of its degree, i.e., v_j has mode even-degree neighbors than odd-degree neighbors. Finally, the constraint (5.11) is necessary to make the solution $x_j = 0$, $j = 1, 2, \ldots, n$, an infeasible choice.

Even though solving an IP problem is NP-hard, the results given in Table 5.2 for finding minimum FAS demonstrate that the runtime performance is quite high for

Table 5.2. Minimum FAS of LDPC codes.

		Minimum Fully Absorbing Sets			
Code Structure	Size	Optimal	Lower Bound	Upper Bound	Time
Algebraic (Tanner Codes)	55	4			0
	155	5			0
	310	5			1
	620	5			0
	775	4			2
	1085	4			5
	2170	4			13
	4340	4			67
	8680	4			311
Random (PEG)	54	3			0
	156	3			0
	312	3			0
	504	3			0
	756	3			1
	1008	3			2
	2400	3			9
	4200	3			26
	8010	3			106
Random (Permutation)	54	2			0
	156	2			0
	312	2			0
	504	2			1
	756	2			1
	1008	2			1
	2400	2			6
	4200	2			16
	8010	2			65

codes of practical lengths. It should be noted that, when the block length is less than 1,000, the computations take mere seconds in most cases. For block lengths up to 8,000, optimal solutions are obtained in less than five minutes. It should emphasized that finding and enumerating FAS is an offline task that is completed in the code design phase and the timing is not as big of a problem as in the real-time decoding implementations. Nevertheless, the proposed FAS IP models are shown to be solved efficiently.

5.1.3. Absorbing Set

After presenting the IP model of FAS, the IP model of AS can be defined more easily. The proposed IP model to find minimum AS over the BSC can be given as

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{5.14}$$

$$\text{s.t.:} \sum_{j=1}^{n} h_{i,j} x_j + f_i = 2d_i,\ i = 1, 2, \ldots, m \tag{5.15}$$

$$\sum_{i=1}^{m} h_{i,j} f_i \leq \sum_{i=1}^{m} h_{i,j}(1 - x_j/2), \quad j = 1, 2, \ldots, n \tag{5.16}$$

$$\sum_{i=1}^{m} f_i \geq 1 \tag{5.17}$$

$$f_i \in \{0, 1\},\ \ k_i \in \mathbb{Z}^+,\ \ i = 1, 2, \ldots, m \tag{5.18}$$

$$x_j \in \{0, 1\},\ \ j = 1, 2, \ldots, n. \tag{5.19}$$

Finding the minimum AS size is very similar to finding the minimum FAS size. The only difference is that, for the minimum AS size, we must consider only the variable nodes in the induced subgraph rather than the entire graph. Therefore, the constraint set (5.10) should be valid for the values of $j = 1, 2, \ldots, n$ for which $x_j = 1$. We achieve

this by replacing (5.10) by (5.16). If x_j is zero, then the constraint (5.16) becomes

$$\sum_{i=1}^{m} h_{i,j} f_i \leq \sum_{i=1}^{m} h_{i,j}, \tag{5.20}$$

which is satisfied for all possible $\mathbf{s}$ vectors and therefore has no effect. If x_j is one, however, the constraint becomes identical to (5.10).

Time performance of the IP model of AS is not as good as that of the IP model of FAS. However, For only one LDPC code with length 8,680, the optimal solution could not be found in allowed time. Therefore, it is fair to say that IP model of AS is efficient and can find the minimum AS for LDPC codes with practical lengths.

5.1.4. Minimum Elementary Absorbing Set

Elementary absorbing sets are AS with the elementary set condition, defined in Definition 3.4. The IP model of EAS is similar with AS and can be given as

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{5.21}$$

$$\text{s.t.:} \sum_{j=1}^{n} h_{i,j} x_j + f_i = 2d_i, \; i = 1, 2, \ldots, m \tag{5.22}$$

$$\sum_{i=1}^{m} h_{i,j} f_i \leq \sum_{i=1}^{m} h_{i,j}(1 - x_j/2), \quad j = 1, 2, \ldots, n \tag{5.23}$$

$$\sum_{i=1}^{m} f_i \geq 1 \tag{5.24}$$

$$f_i, d_i \in \{0, 1\}, \quad i = 1, 2, \ldots, m \tag{5.25}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n. \tag{5.26}$$

Table 5.3. Minimum AS of LDPC codes.

		AS			
Code Structure	Size	Optimal	Lower Bound	Upper Bound	Time
Algebraic (Tanner Codes)	55	3			0
	155	4			0
	310	4			1
	620	4			10
	775	4			31
	1085	4			51
	2170	4			203
	4340	4			1336
	8680	time	1	4	10800
Random (PEG)	54	3			0
	156	3			0
	312	3			1
	504	3			2
	756	3			5
	1008	3			13
	2400	3			98
	4200	3			599
	8010	3			7186
Random (Permutation)	54	2			0
	156	2			1
	312	2			1
	504	2			10
	756	2			11
	1008	2			15
	2400	2			129
	4200	2			547
	8010	2			5448

To be an EAS, the induced subgraph of an AS must have only check nodes with degrees one or two. Therefore, the degree of a check node in the induced subgraph, $d(c_i) = \sum_{j=1}^{n} h_{i,j}x_j$, cannot be greater than two for the check nodes in the induced subgraph. This condition is achieved by placing the constraint set (5.25), where the restriction on the value of d_i guarantees that the check node degrees cannot exceed two.

Although EAS is one of the most devastating error prone structures, Table 5.4 shows that they can be efficiently found in reasonable times.

Among the different classes of codes we considered, it can be seen that the proposed models are solved more efficiently for the algebraically constructed Tanner code family. This observation is especially true for minimum SS size calculations. Among the randomly constructed code families, it is interesting to see that this time the codes constructed by the PEG algorithm, although with more structure, correspond to a heavier load for the IP solver, presumably due to the larger minimum sizes for the error prone structures.

5.2. Enumerating All Error Prone Structures

The IP models developed in Section 5.1 for finding small error prone substructure sizes are useful tools to determine the error correction potential of a given code. However, in some applications, we further would like to list and enumerate all of these structures rather than knowing their sizes. Enumeration algorithms, such as the one proposed in this section, help the code designer tweak his code and/or decoder design. The proposed enumeration algorithm, presented in Figure 5.1, is quite generic and can be employed in conjunction with all of the IP models we have discussed so far.

The algorithm starts by solving the original IP problem to find the minimum error prone substructure and its size. Once a solution is obtained, its size is compared to the maximum search size a in Steps 5-7 and the algorithm is stopped if it exceeds a. The list of error prone structures it has found so far is given as output. Otherwise,

Table 5.4. Minimum EAS of LDPC codes.

		EAS			
Code Structure	Size	Optimal	Lower Bound	Upper Bound	Time
Algebraic (Tanner Codes)	55	3			0
	155	4			0
	310	4			2
	620	4			6
	775	4			13
	1085	4			26
	2170	4			168
	4340	4			708
	8680	time	1	4	10800
Random (PEG)	54	3			0
	156	3			0
	312	3			4
	504	3			2
	756	3			11
	1008	3			25
	2400	3			259
	4200	3			4530
	8010	3			6592
Random (Permutation)	54	2			0
	156	2			1
	312	2			1
	504	2			11
	756	2			14
	1008	2			14
	2400	2			64
	4200	2			824
	8010	2			6404

1: **Input:** Parity-check matrix $\mathbf{H}$ of the LDPC code, maximum size a, IP models

2: $size \leftarrow 0$;

3: **while**

4: Solve the IP, obtain the solution vector as $\mathbf{x}^*$;

5: **if** $w_{\mathrm{H}}(\mathbf{x}^*) \geq a$ **then**

6: break **while**;

7: **end if**

8: **if** $w_{\mathrm{H}}(\mathbf{x}^*) > size$ **then**

9: Restore the original IP model;

10: Add $\sum_{j=1}^{n} x_j \geq w_{\mathrm{H}}(\mathbf{x}^*)$ as a new constraint to the IP;

11: **end if**

12: Add $\mathbf{x}^*$ to the solution table;

13: Add $\sum_{j:x_j^*=0} x_j + \sum_{j:x_j^*=1} (1 - x_j) \geq 1$ as a new constraint to the IP model;

14: Set $size = w_{\mathrm{H}}(\mathbf{x}^*)$;

15: **end while**

16: **Output:** The solution table which presents the all error prone structures of size less than a.

Figure 5.1. Enumerating all error prone structures of size less than a.

if the size is smaller than a, then the obtained solution $\mathbf{x}^*$ is added to the list and a new constraint is added to the IP model to ensure that the same solution cannot be reached in subsequent runs (Steps 12 and 13). We also set the variable *size* to the substructure size to store this value in the subsequent runs (Step 14). An enumeration of all error prone structures is possible using this model, however, the number of additional constraints added to the model may be too large for large values of a. To overcome this problem and obtain a more efficient algorithm, Steps 8-11 are added for model simplification. In effect, Step 8 compares the size of the newest solution to the stored value. If the size has increased, there is no need to use the additional constraints added in Step 13 to make every one of the past solutions infeasible; we can simply restore the original IP model and add a single constraint to eliminate all solutions with size smaller than the current one. This allows us to avoid adding too many constraints that would result in reduced performance. The following example demonstrates the complexity savings achieved by these steps.

Example 5.1. *The smallest AS for the quasi-cyclic Tanner code of length* $n = 155$ *has size 4 and there are 465 distinct such substructures in the code's graph representation. Therefore, in order to find an AS of size 5, we would need some 465 additional constraints in our IP model, whereas a single constraint* $\sum_{j=1}^{n} x_j \geq 6$ *would easily replace these constraints.*

6. CONCLUSION

In this thesis, an efficient, general framework to find and enumerate error prone structures of any finite-length LDPC code is presented. This includes developing efficient integer programming models to describe and calculate the smallest stopping set size for the BEC, the smallest fully absorbing set, absorbing set, and elementary absorbing set sizes for the BSC and the AWGNC. The obtained results are provably optimal due to the use of integer programming.

The proposed integer programming models require only the knowledge of the parity-check matrix $\mathbf{H}$ and can be efficiently solved for a wide variety of practical code lengths as well as code structures, e.g., regular and irregular LDPC codes, randomly constructed and algebraically constructed codes, etc. With the knowledge of the dominant error prone structures, the error floor performance of LDPC codes can be estimated, as in the case of [6] and [8]. The obtained result can also be used to improve LDPC code designs, such as done in [24] with maximizing the stopping sets for BEC and in [9] with maximizing trapping sets for BSC. In addition to applications in code design and error performance estimations, the knowledge of the smallest error prone structures can also be used to generate adaptive cuts for LP-based decoding algorithms. In [19], cycles of the parity-check matrix of an LDPC code are used to improve the LP decoder performance. Since trapping sets are closely related to cycles, the proposed method can be an efficient tool for generating adaptive cutting techniques. An ongoing research on adaptive cuts with trapping sets has even shown a slight increase in the performance of the LP decoder.

APPENDIX A: Maximum Likelihood LP

For a transmitted codeword $\mathbf{x} \in \zeta$ and received vector $\mathbf{r}$, the maximum likelihood codeword 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

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

REFERENCES

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

2. 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.

3. 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.

4. Dolecek, L., Z. Zhang, V. Anantharam, M. Wainwright and B. Nikolic, "Analysis of Absorbing Sets for Array-Based LDPC Codes", *ICC '07. IEEE International Conference on Communications, 2007.*, pp. 6261–6268, 2007.

5. Di, C., D. Proietti, I. Telatar, T. Richardson and R. Urbanke, "Finite-length Analysis of Low-Density Parity-Check Codes on the Binary Erasure Channel", *IEEE Transactions on Information Theory*, Vol. 48, No. 6, pp. 1570–1579, 2002.

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

7. 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, 2006. GLOBECOM '06.*, pp. 1–6, 2006.

8. Dolecek, L., Z. Zhang, M. Wainwright, V. Ananthram and B. Nikolic, "Evaluation of the Low Frame Error Rate Performance of LDPC Codes Using Importance Sampling", *Process to Information Theory and Applications Workshop*, pp. 202–

207, 2007.

9. 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.

10. Keha, A. B. and T. M. Duman, "Minumum Distance Computation of LDPC Codes Using a Branch and Cut Algorithm", *IEEE Transactions on Communications*, Vol. 58, No. 4, pp. 1072–1079, 2010.

11. 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.

12. 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.

13. Karimi, M. and A. Banihashemi, "An Efficient Algorithm for Finding Dominant Trapping Sets of LDPC Codes", *Process to IEEE International Symposium on Turbo Codes and Iterative Information*, pp. 444–448, 2010.

14. Wang, C., S. Kulkarni and H. Poor, "Finding All Small Error-Prone Substructures in LDPC Codes", *IEEE Transactions on Information Theory*, Vol. 55, No. 5, pp. 1976–1999, 2009.

15. 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.

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

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

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

20. Wolsey, L. A., *Integer Programming*, Wiley-Interscience, New York, NY, 1998.

21. Luenberger, D. and Y. Ye, *Linear and Nonlinear Programming*, International Series in Operations Research & Management Science, Springer, New York, NY, 2008.

22. 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.

23. 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.

24. Wang, C.-C., "Code Annealing and the Suppressing Effect of the Cyclically Lifted LDPC Code Ensemble", *IEEE Information Theory Workshop, 2006. ITW '06 Chengdu.*, pp. 86–90, 2006.