STUDIES ON GRAPH-BASED CODING SYSTEMS.

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the
Graduate School of The Ohio State University

By

Jing Sun, M.S.

* * * * *

The Ohio State University

2004

Dissertation Committee:

Professor Oscar Y. Takeshita, Adviser
Professor Lee C. Potter
Professor Hesham El-Gamal

Approved by

Adviser

Department of Electrical Engineering
ABSTRACT

To make full use of the valuable radio spectrum, one of the targets of communications system design is to convey as much information as possible through the spectrum (the channel) allocated for the purpose. For a given channel, the amount of information that can be passed through it is upper bounded by the well-known Shannon channel capacity [40].

The invention of turbo codes in 1993 was a key step in the 50-year effort to design good coding schemes achieving the Shannon capacity. Since then, other coding schemes with similar performance, such as Low Density Parity Check (LDPC) codes [27] and turbo product codes [18] [34], have been re-discovered or invented. The common characteristics of these codes are that they all can be represented by a large (pseudo-)-random graph, and iteratively decoded [49].

In this dissertation, we treat three topics in the design and analysis of the two most important graph-based coding schemes: turbo codes and LDPC codes.

Together with two component convolutional codes, an interleaver is a key component of a turbo code. We introduce a class of deterministic interleavers for turbo codes based on permutation polynomials over $\mathbb{Z}_N$. It is observed that the performance of a turbo code using these permutation polynomial-based interleavers is usually dominated by a subset of input weight $2m$ error events. Due to the structure of these interleavers, we derive a simple method to find the weight spectrum of those error events. Therefore good permutation polynomials can be searched for a given component code to achieve better performance.
LDPC codes can be constructed using an interleaver. In a previous work, the use of maximum length linear congruential sequences (MLLCS) has been proposed for the construction of interleavers for regular LDPC codes with data node degree 3. Since the smallest loop size (girth) is a key characteristic of the graph of the LDPC code, a sufficient condition on the parameters of the MLLCS to generate a graph with girth larger than 4 is given. We extend the sufficient condition to general irregular LDPC codes and also provide sufficient conditions to guarantee even larger girth.

It is observed that the error floor of LDPC code (bit error performance at high signal-to-noise ratios) is usually caused by trapping sets, which are sets of data nodes that cannot be corrected by the iterative decoder. We develop an approximated linear system model for the iterative decoding process in a trapping set. Then the probability that the trapping set can be corrected can be estimated by observing the response of the linear system. Using the idea from the analysis of the linear system, the iterative decoder for regular LDPC codes can be slightly modified to greatly decrease the error floor.
To my parents, sisters and Xinning
I was fortunate to have Dr. Oscar Y. Takeshita as my adviser whose understanding and insights have made my Ph.D. study both enjoyable and fruitful. He patiently supported me to go through the whole process of doing research, from discovering the problems to finding different ways to solve them. He carefully guided me to think deeper and grasp the true understanding of the problems. He kindly encouraged me when I was frustrated by seemingly unsolvable problems. My Ph.D. research would have been impossible without his support.

Special thanks goes to Dr. Michael P. Fitz. He generously provided part of the funding to support my Ph.D. study. He also actively participated in all my research works. Many of my new ideas were generated in the interesting discussions with him. My achievements would have not been possible without his help.

Former and current students in IPS lab have helped make my OSU experience memorable. It was always my pleasure to have intellectually stimulating discussions with Youjian Liu, Xiaoxia Zhang, Ravikiran Rajagopal, Shengchao Li among others.

Finally, I would like to give my thanks to my family, whose love, care and support, have made this thesis possible.
VITA

July 1995 ................................. B.S. Electrical Engineering, Southeast University
July 1998 ................................. M.S. Electrical Engineering, Southeast University
1995-1998 ................................. Graduate Research Associate, Southeast University.
1998 ........................................ The Ohio State University Fellow
1999-present ............................. Graduate Research Associate, The Ohio State University.

PUBLICATIONS


**FIELDS OF STUDY**

Major Field: Electrical Engineering

Studies in:
- Communications and Signal Processing
- Control
- Mathematics
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Abstract</strong></td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td><strong>Dedication</strong></td>
<td>v</td>
</tr>
<tr>
<td></td>
<td><strong>Acknowledgments</strong></td>
<td>vi</td>
</tr>
<tr>
<td></td>
<td><strong>Vita</strong></td>
<td>vii</td>
</tr>
<tr>
<td></td>
<td><strong>List of Tables</strong></td>
<td>xii</td>
</tr>
<tr>
<td></td>
<td><strong>List of Figures</strong></td>
<td>xiii</td>
</tr>
<tr>
<td></td>
<td><strong>List of Symbols</strong></td>
<td>xvi</td>
</tr>
</tbody>
</table>

**Chapters:**

1. **INTRODUCTION** ................................................................. 1
   1.1 Historic Remarks ......................................................... 1
   1.2 Background of Turbo Codes ........................................... 3
   1.3 Background of LDPC Codes ............................................... 5

2. **PERMUTATION POLYNOMIAL-BASED INTERLEAVERS FOR TURBO CODES** ......... 10
   2.1 Introduction ............................................................. 10
   2.2 Permutation Polynomials over $\mathbb{Z}_N$ .......................... 13
   2.3 Permutation Polynomial-Based Interleavers ........................... 18
   2.4 Permutation Polynomials Search ........................................ 22
      2.4.1 Input Weight $2m$ Error Events ................................... 24
      2.4.2 Search for Good Interleavers using Effective Free Distance ... 28
2.4.3 Higher Input Weight Error Events

2.5 Results

2.6 Conclusions

3. MAXIMUM LENGTH LINEAR CONGRUENTIAL SEQUENCE-BASED INTERLEAVERS FOR LDPC CODES

3.1 Introduction

3.2 Loops in MLLCS-Based LDPC Codes

3.3 Sufficient conditions for $girth > 2m$

3.4 Conclusions

4. ANALYSIS OF TRAPPING SETS IN LDPC CODES USING APPROXIMATED LINEAR SYSTEM MODELS

4.1 Introduction

4.2 Analysis of a Trapping Set

4.3 Simple Trapping Sets

4.3.1 Eigen-Decomposition of Matrix $A$

4.3.2 Zero Input Response

4.3.3 Zero State Response

4.3.4 Combination of Zero Input Response and Zero State Response

4.4 Trapping Sets with External Data Nodes

4.5 Trapping Sets with Multiple Dominant Eigenvalues

4.6 Results

4.6.1 Regular LDPC Code Examples

4.6.2 Irregular LDPC Code Examples

4.7 Conclusions

5. CONCLUSIONS

Appendices:

A. PROOF OF THEOREM 2.8:

B. PROOF OF THEOREM 3.1:

C. PROOF OF THEOREM 3.2:
D. PROOF OF THEOREM 3.3: ......................................................... 119
E. PROOF OF LEMMA 4.10: ......................................................... 121
F. PROOF OF THEOREM 4.13: ......................................................... 123
Bibliography ................................................................. 126
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of $L$ for $b = 16$ with different $a$. Turbo code with 5/7 or 7/5 component code</td>
</tr>
<tr>
<td>2.2</td>
<td>Best permutation polynomials for different component codes. Frame size 256.</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>7</td>
</tr>
<tr>
<td>1.5</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>32</td>
</tr>
<tr>
<td>2.5</td>
<td>33</td>
</tr>
<tr>
<td>2.6</td>
<td>49</td>
</tr>
<tr>
<td>2.7</td>
<td>50</td>
</tr>
<tr>
<td>2.8</td>
<td>50</td>
</tr>
</tbody>
</table>

xiii
2.9 TC with 21/37 code. Frame size 256. ............................................. 51
2.10 TC with 37/25 code. Frame size 256. ............................................. 51
2.11 TC with 23/35 code. Frame size 256. ............................................. 52
2.12 TC with 5/7 code. Frame size 1024. ............................................... 53
2.13 TC with 5/7 code. Frame size 16384. ........................................... 53

4.1 A trapping set with no external data node ..................................... 68
4.2 A linear system model for messages in the trapping set ..................... 69
4.3 Examples: (a), Single length 2 loop; (b) Multiple loops with $A^k \neq 0$; (c),
Multiple loops with $A^k > 0$. ................................................................. 80
4.4 Consistent Gaussian Approximated Density Evolution, Irregular LDPC code 85
4.5 A trapping set with external data nodes ......................................... 91
4.6 A trapping set with multiple dominant eigenvalues ............................ 95
4.7 Mean and variance of $R_{zi}$ and $R_{zs}$ for trapping set as in Figure 4.1 in regular
(3,6) LDPC code, channel SNR 2dB. ....................................................... 98
4.8 Normalized mean of $R_{zi}$, $R_{zs}$ and $R$ for trapping set as in Figure 4.1 in regular (3,6) LDPC code, channel SNR 2dB. ....................... 99
4.9 Comparison of decoder using hard-limiter or approximated computation in
check nodes, regular (3,6) LDPC code. ..................................................... 101
4.10 Comparison of decoder using hard-limiter or approximated computation in
check nodes using Richardson’s method, Margulis code. .......................... 102
4.11 Mean and variance of $R_{zi}$ and $R_{zs}$ for trapping set as in Figure 4.1 in irregular LDPC code, channel SNR 2dB. ................................. 103
4.12 Normalized mean of $R_{zi}$, $R_{zs}$ and $R$ for trapping set as in Figure 4.1 in irregular LDPC code, channel SNR 2dB. ................................. 104
4.13 Error floor for the trapping set, simulated and analyzed .............. 105

B.1 Graph representations of a length 4 loop. ............................... 115

C.1 A length $2m$ loop .......................................................... 117


LIST OF SYMBOLS

\( \mathbb{Z}_N \) \hspace{1cm} \text{Ring over integer } \{0,1,\ldots,N-1\}

\( p_N \) \hspace{1cm} \text{Base of number } N. \text{ Defined as } p_N = [p_1, p_2, \ldots, p_m] \text{ when } N = \prod_{i=1}^{m} p_i^{n_{p_i}}, \text{ where } p_i \text{'s are primes.}

\( o_N \) \hspace{1cm} \text{Order of number } N. \text{ Defined as } o_N = [n_{p_1}, n_{p_2}, \ldots, n_{p_m}] \text{ when } N = \prod_{i=1}^{m} p_i^{n_{p_i}}, \text{ where } p_i \text{'s are primes.}

\( o_x|N \) \hspace{1cm} \text{Order of number } x \text{ with respect to } N. \text{ Defined as } o_x|N = [o_x[1], o_x[2], \ldots, o_x[m]]. \text{ It can be simplified to } o_x \text{ when } N \text{ is clear from the context.}

\( H \) \hspace{1cm} \text{Parity check matrix of a linear block code.}

\( \Sigma_i(x) \) \hspace{1cm} \text{Summation of series } \{x_n\} \text{ from element } i \text{ with step 2. } \Sigma_i(x) = x_i + x_{i+2} + x_{i+4} + \cdots. \)
CHAPTER 1

INTRODUCTION

1.1 Historic Remarks

In 1948, Shannon developed his well-known information theory [40]. He proved that asymptotically error free communication is possible in a given noisy channel up to a certain information transmission rate. This rate is called channel capacity, or Shannon capacity. From Shannon’s proof of the theory, it can be seen that a long random code can achieve capacity.

Unfortunately, a long random code has exponentially increasing complexity to decode and is not practical. One of the most important research areas in the information and coding theory is to find coding schemes with reasonable complexity that work as close to the channel capacity as possible. This task has been proved to be formidable. People even started to believe that the channel capacity is not achievable with a reasonable complexity.

The invention of turbo codes [6] in 1993 has since changed what people believed. When well-designed, a turbo code is able to achieve performance within a fraction of a dB from the channel capacity. There are two key features that make turbo codes perform very well. The first is the structure of the code and the second is the iterative decoder used. The code is simply constructed by parallel concatenating two recursive convolutional codes via an
interleaver. Due to the effect of the interleaver, a random-looking code is constructed using simple components. The good performance of the code is a direct result of this randomness. In the decoder, the optimum maximum-likelihood (ML) decoder has exponentially increasing complexity also due to this randomness. Fortunately, a simple sub-optimum iterative decoder was invented. The forward-backward maximum a posteriori (MAP) decoders for each convolutional code are run with information passed between them. Even though this decoder is suboptimum, the performance is close to the performance of an ML decoder.

It has been shown in [28] that the suboptimum iterative decoder is equivalent to a belief propagation decoder on the graph of the code [31]. It is known that when the graph of the code is loop-free, the belief propagation decoder is optimal. However, even if loops exist, in the case of turbo codes, the belief propagation decoder still performs very well.

The success of turbo codes motivated researchers to investigate other codes from another point of view with the experience gained from constructing turbo codes. Many other classes of well-performing codes were found, such as Serial Concatenated Convolutional Code (SCCC) [4], Repeat-Accumulate (RA) Codes [13], Turbo Product Code (TPC) and Low-Density Parity-Check Code (LDPC). All these codes can be regarded as codes over graphs, and their decoding algorithms are message passing algorithms in the corresponding graphs. Therefore, we call them graph-based codes.

Among all these graph based codes, the class of LDPC codes is another important one. Actually, this class of codes was invented a long time ago in 1963 by Gallager [16]. Unfortunately, its practical value was not well-recognized and has been ignored. After the invention of turbo codes, LDPC code was re-investigated by Mackay [27] and it was realized that an LDPC code is as good as a turbo code.
Our research is targeted on several areas in the graph-based codes including code design and performance analysis. This thesis focuses on three topics: permutation polynomial-based interleaver design for turbo codes, maximum length linear congruential sequence-based interleaver design for LDPC codes, and the approximated linear system analysis of trapping sets in LDPC codes.

The rest of the thesis is organized as follows. A background of turbo codes and LDPC codes are given in the rest of this chapter. In Chapter 2, our work on permutation polynomial-based interleaver design for turbo codes is presented. Chapter 3 focuses on the maximum length linear congruential sequence-based interleaver design for LDPC codes. Our work in the trapping set analysis for LDPC codes is covered in Chapter 4. Finally, in Chapter 5, we present our conclusions.

1.2 Background of Turbo Codes

A turbo code is composed of two simple recursive convolutional code encoders and an interleaver. A block diagram of a turbo code encoder is shown in Figure 1.1. The encoding process is as follows. The information sequence is encoded using the first recursive convolutional code. An interleaved version of the original information sequence is encoded using the other recursive convolutional code. Then the results of these two encoders (parities) and the original information sequence are multiplexed together (possibly punctured) and transmitted through the channel.

A suboptimum iterative decoder is used to decode the turbo code. A block diagram for such a decoder is shown in Figure 1.2. In the decoder, channel observations are separated into two streams, each for one convolutional decoder using a forward-backward MAP algorithm (the BCJR algorithm) [3]. The two decoders are alternatively activated. In each
iteration, decoder 1 is first activated by processing its channel observation and the extrinsic information from decoder 2, which is set to be zero for the first iteration. The extrinsic information is computed and sent to decoder 2. Then the decoder 2 is activated by processing its channel observation and extrinsic information from decoder 1. It also generates new extrinsic information that is passed to decoder 1.

A turbo code can be thought of as a code on a graph. The graph is shown in Figure 1.3. In the graph, the lower and upper lines of nodes represent the trellises of the first and
the second component codes respectively. The black circles are the states of the trellises. The squares represent the constraints in each trellis section. The white circles are the information sequence. For clarity of the graph, the parity check bits are absorbed into the check nodes. The iterative decoder described before can be regarded as a belief propagation decoder on this graph [28].

From the structure of a turbo code, one can easily see that the design of a turbo code can be separated into a component code design and an interleaver design. In this thesis, Chapter 2 is devoted to the interleaver design [46]. We have also investigated other problems in turbo coding, such as the encoder design [42], the highly parallel decoder design [43] and the methods for performance analysis [44]. Due to the limitation on the space, these works will not be presented here.

1.3 Background of LDPC Codes

An LDPC code is a linear block code. The special characteristic of the code that makes it good is that, in the parity check matrix, the density on 1’s is low. This is where the name LDPC come from [16].
The parity check matrix $H$ for the code is selected to be random-like. This helps the code to achieve near channel capacity performance. On the other hand, the density of 1’s in $H$ is selected to be low for the decoding complexity of the sub-optimum iterative decoder is linearly proportional to the number of 1’s. A linear-time decodable code is always preferrable. Therefore, the density of 1’s should go to zero as the frame size goes to infinity.

The parity check matrix $H$ can be represented by a bipartite graph. Assume $H$ has dimension $R \times N$, where $R$ is the number of checks and $N$ is the size of the frame. In the bipartite graph, there are $N$ data nodes and $R$ check nodes corresponding to $N$ columns and $R$ row of $H$ respectively. The degrees of the data nodes ($d_i$ for $i = 0, \ldots, N - 1$) and check nodes ($c_i$ for $i = 0, \ldots, R - 1$) are equal to the corresponding column and row weights. The rate of the code is $\frac{N-R}{N}$ if the rows of the parity check matrix are linearly independent. A data node $d_i$ is connected to check node $c_j$ if and only if $H_{j,i} = 1$.

There are two categories of LDPC codes. A regular LDPC code has constant data node degrees $d_\lambda$ and check node degrees $d_\rho$ and can be simply denoted by the integer pair $(d_\lambda, d_\rho)$. An irregular LDPC code can be described by two polynomials

$$
\lambda(x) = \sum_{i=2}^{d_\lambda} \lambda_i x^{i-1} \\
\rho(x) = \sum_{i=2}^{d_\rho} \rho_i x^{i-1}
$$

(1.1)

where $d_\lambda$ and $d_\rho$ are the maximum data node degree and check node degree respectively. The coefficients $\lambda_i$ and $\rho_i$ are the ratios of branches that are connected to degree $i$ data nodes or check nodes. A regular LDPC code is a special case of an irregular LDPC code such that the degree distributions are $\lambda(x) = x^{d_\lambda-1}$ and $\rho(x) = x^{d_\rho-1}$. 6
The bipartite graph of an LDPC code is shown in Figure 1.4. In the graph, circles are data nodes and squares are check nodes. We can interpret this bipartite graph as being formed by an interleaver as in Figure 1.5. In this graph, each node has a set of connectors. The number of connectors is equal to its degree. The connectors on the data node side and check node side are connected by an interleaver.

Figure 1.4: Bipartite graph representation of an LDPC code

From Figure 1.5, we can see that the design of an LDPC code can also be separated into two parts. The design of the degree distribution (selecting the degree of each node) and the design of the interleaver.

The design of the degree distribution has been considered in [36] [8] under the assumption that the frame size is infinitely long. The degree distribution is chosen such that the threshold is minimized, where the threshold is defined as a channel SNR such that the code can be corrected with probability one if and only if the true channel SNR is above the threshold.

The interleaver design is in some sense even more complex, for we do not have the infinite frame size assumption to simplify the model. Several ways have been tried to
Figure 1.5: Bipartite graph representation of an LDPC code constructed using an interleaver
design interleavers [27] [7] [23]. In Chapter 3, we follow [33] to use a maximum length
linear congruential sequence to form the interleaver [47].

Performance analysis of the iterative decoder for a given LDPC code is another ac-
tive research area about LDPC codes. The analysis of an iterative decoder can be further
separated into two cases: the infinite frame size analysis and finite frame size analysis.

When we assume the frame size goes to infinity, as we mentioned above, we can ob-
serve a threshold phenomenon [37]. When the channel SNR falls above the threshold,
the iterative decoder converges to the correct codeword with probability 1. On the other
hand, when the SNR is lower than the threshold, there is a non-zero error probability. The
thresholds are verified by the simulation of very long LDPC codes.
A direct result of the infinite frame size assumption is that there is no loop in the underlying graph of the code. This is not satisfied for practical codes, especially for codes with medium to short frame sizes. For finite frame size analysis, other techniques are needed.

The finite length analysis is more difficult. For this case, the performance of LDPC codes have two segments, a waterfall area where bit error rate (BER) decreases fast with respect to increasing SNR, and a error floor area where the BER decreases slower. Even though the error floor of an LDPC code is believed to be much lower than that of the turbo code, it still exists and its correct prediction is important in some applications. The error floor of the LDPC code is usually caused by a structure called trapping set in the code. Chapter 4 is devoted to an approximated linear system model-based analysis of trapping sets [45].
CHAPTER 2

PERMUTATION POLYNOMIAL-BASED INTERLEAVERS FOR TURBO CODES

2.1 Introduction

In this chapter, we consider the problem of interleaver design for turbo code. Interleavers for turbo codes have been extensively studied in [14] [39] [15] [11] [48]. They can in general be separated into two classes: random interleavers and deterministic interleavers.

The basic random interleavers permute the information bits in a pseudo-random manner. It was demonstrated in [6] that near Shannon limit performance can be achieved with these interleavers for large frame sizes. The S-random interleaver proposed in [14] is an improvement to the random interleaver. The S-random interleaver is a pseudo-random interleaver with the restriction that any two input positions within distance $S$ cannot be permuted to two output positions within distance $S$. Under this restriction, certain short error events in one component code will not be mapped to short error events in the other component code. Further improvements to the S-random interleaver have been presented in [39] [15]. In [39], an iterative decoding suitability (IDS) criterion is used to design 2-step
S-random interleavers. In [15], multiple error events are considered. These improved interleavers have better performance than S-random interleavers, especially for short frame sizes.

Turbo codes using random interleavers require an interleaver table to be stored both in the encoder and the decoder. This is not desirable in some applications, especially when the frame size is large or many different interleavers have to be stored. Deterministic interleavers can avoid interleaver tables since interleaving and de-interleaving can be performed algorithmically.

The simplest deterministic interleavers are block interleavers [32] and linear interleavers [48]. For some component codes and very short frame sizes, block and linear interleavers, with properly chosen parameters, perform better than random interleavers. However, for medium to long frame sizes, they have a high error floor [48] and random interleavers are still better.

In [48], quadratic interleavers have been introduced. Quadratic interleavers are a class of deterministic interleavers based on a quadratic congruence. For $N = 2^n$, it can be shown that $c_i = \frac{ki(i+1)}{2} \pmod{N}$ is a permutation of $i \in \{0, 1, \ldots, N-1\}$ for odd $k$. Then the quadratic interleaver is defined as

$$\pi_{Q_N} : c_i \mapsto c_{(i+1) \pmod{N}}, \quad \forall i,$$

which means that the data in position $c_i$ is interleaved to position $c_{(i+1) \pmod{N}}$, for all $i$. The quadratic interleavers perform well. Even though they are not as good as S-random interleavers, it is claimed in [48] that they achieve the average performance of random interleavers. Due to this characteristic, in the simulations in Section 2.5, in addition to S-random interleavers, we will compare our proposed interleavers to quadratic interleavers instead of the average of many randomly generated interleavers.
Recently, some other deterministic interleavers have been proposed. In [9], dithered relative prime (DRP) interleavers are used. Good minimum distance property can be achieved for short frame size cases by carefully selecting the parameters of the interleaving process. In [10], monomial interleavers are used, which is based on a special permutation polynomial in the form of $x^i$ over finite field. This class of interleavers can also achieve the similar performance as random interleavers. However, since the permutation polynomials are over finite field, the frame size can only be a power of a prime.

In this chapter, we will propose another class of deterministic interleavers. These interleavers are based on permutation polynomials over the ring of integers modulo $N$, $\mathbb{Z}_N$. By carefully selecting the coefficients of the polynomial, we can achieve a performance close to, or in some cases, even better than S-random interleavers.

It is observed that a subset of error events with input weight $2^m$, $m$ a small positive integer, usually dominates the performance, at least when the frame size is not very short. The parameter selection of the permutation polynomial is based on the minimum distance or the first few spectrum lines of this subset. For small $m$, we present a simple method to find these error events. This helps to reduce the complexity of searching for good permutation polynomial-based interleavers given the component code.

The chapter is organized as follows. In section 2.2, we briefly review permutation polynomials over $\mathbb{Z}_N$. Interleavers based on permutation polynomials are introduced in section 2.3. Minimum distance and spectrum based analysis are given in section 2.4. Simulation results are shown in section 2.5, and finally, we draw our conclusions in section 2.6.
2.2 Permutation Polynomials over $\mathbb{Z}_N$

Given an integer $N \geq 2$, a polynomial $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$, where $a_0, a_1, \ldots, a_m$ and $m$ are non-negative integers, is said to be a permutation polynomial over $\mathbb{Z}_N$ when $P(x)$ permutes the elements of $\mathbb{Z}_N$. In this paper, all the summations and multiplications are modulo $N$ unless explicitly stated. We further define the formal derivative of polynomial $P(x)$ to be a polynomial $P'(x)$ such that

$$P'(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots + ma_mx^{m-1}. \quad (2.1)$$

For the special case that $N = 2^n$, a necessary and sufficient condition for a polynomial to be a permutation polynomial was given in [38]. It is repeated in the following theorem.

**Theorem 2.1** Let $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$ be a polynomial with integer coefficients. $P(x)$ is a permutation polynomial over the integer ring $\mathbb{Z}_{2^n}$ if and only if (1) $a_1$ is odd, (2) $a_2 + a_4 + a_6 + \cdots$ is even, and (3) $a_3 + a_5 + a_7 + \cdots$ is even.

For the more general case $N = p^n$, where $p$ is any prime number, the necessary and sufficient condition is derived in [20].

**Theorem 2.2** $P(x)$ is a permutation polynomial over the integer ring $\mathbb{Z}_{p^n}$ if and only if $P(x)$ is a permutation polynomial over $\mathbb{Z}(p)$ and $P'(x) \not\equiv 0 \mod p$ for all integers $x \in \mathbb{Z}_{p^n}$.

For example, let $P(x) = 3x^2 + x$ and $p = 3$. Then $P(0) = 0 \mod 3$, $P(1) = 1 \mod 3$ and $P(2) = 2 \mod 3$, which implies that $P(x)$ is a permutation polynomial over $\mathbb{Z}(p)$. The formal derivative of $P(x)$ is $P'(x) = 6x + 1 \equiv 1 \mod 3$, which is a non-zero constant for all $x$ in $\mathbb{Z}_{p^n}$. Thus Theorem 2.2 is satisfied and $P(x)$ is a permutation polynomial over $\mathbb{Z}_{p^n}$.
It can be easily verified that when $p = 2$, the necessary and sufficient condition in Theorem 2.2 reduces to the form in Theorem 2.1. However, since in many of our design examples we use $N = 2^n$, we keep Theorem 2.1 for its simplicity.

For general $N$, we also have a necessary and sufficient condition for a polynomial to be a permutation polynomial, which is summarized in the following theorem.

**Theorem 2.3** For any $N = \prod_{i=1}^{m} p_i^{n_{p_i}}$, where $p_i$'s are distinct prime numbers, $P(x)$ is a permutation polynomial modulo $N$ if and only if $P(x)$ is also a permutation polynomial modulo $p_i^{n_{p_i}}$, $\forall i$.

**Proof:** First we prove the necessity.

Let $P(x)$ be a permutation polynomial over $\mathbb{Z}_N$. For any $l$,

$$P(x + lp_i^{n_{p_i}}) = a_0 + a_1(x + lp_i^{n_{p_i}}) + a_2(x + lp_i^{n_{p_i}})^2 + \cdots + a_m(x + lp_i^{n_{p_i}})^m$$

$$= a_0 + a_1x + a_2x^2 + \cdots + a_mx^m \mod p_i^{n_{p_i}}.$$

So $P(x + lp_i^{n_{p_i}}) = P(x) \mod p_i^{n_{p_i}}$.

Assume that $P(x)$ is not a permutation polynomial over $p_i^{n_{p_i}}$. Then there exist $x_1 \neq x_2$, $0 \leq x_1, x_2 < p_i^{n_{p_i}}$ such that $P(x_1) = P(x_2) = y \mod p_i^{n_{p_i}}$. Then $\forall l \in \{0, 1, \ldots, \frac{N}{p_i^{n_{p_i}}} - 1\}$, $P(x_1 + lp_i^{n_{p_i}}) = P(x_2 + lp_i^{n_{p_i}}) = y \mod p_i^{n_{p_i}}$. So a total of $\frac{N}{p_i^{n_{p_i}}}$ numbers are mapped to $y \mod p_i^{n_{p_i}}$. This cannot be true since $P(x)$ is a permutation polynomial modulo $N$ and it can map exactly $\frac{N}{p_i^{n_{p_i}}}$ numbers to $y \mod p_i^{n_{p_i}}$.

Next, we prove the sufficiency. Let $N_1$ and $N_2$ be co-prime. Let $P(x)$ be a permutation polynomial both modulo $N_1$ and $N_2$. Assume that $P(x)$ is not a permutation polynomial modulo $N_1N_2$. Then there exist $x$ and $x'$ such that $x \not\equiv x' \mod N_1N_2$ and $P(x) =$
\[ P(x') \mod N_1N_2. \] Define \( x_1 = x \mod N_1, x_2 = x \mod N_2 \) and \( x'_1 = x' \mod N_1, \] \( x'_2 = x' \mod N_2. \) Since \( P(x) = P(x') \mod N_1N_2, \) we have \( P(x_1) = P(x'_1) \mod N_1 \] and \( P(x_2) = P(x'_2) \mod N_2. \) Since \( P(x) \) is a permutation polynomial modulo \( N_1 \) and \( N_2, \) we must have \( x_1 = x'_1 \mod N_1 \) and \( x_2 = x'_2 \mod N_2. \) By the Chinese Remainder Theorem [2], this implies that \( x = x' \mod N_1N_2, \) which contradicts the assumption. Thus \( P(x) \) is also a permutation polynomial modulo \( N_1N_2. \)

\[ \square \]

Using this theorem, checking whether a polynomial is a permutation polynomial modulo \( N \) reduces to checking the polynomial modulo each \( p_i^{n_i} \) factor of \( N. \)

For \( p_i = 2, \) we can use Theorem 2.1 to check if the polynomial is a permutation polynomial, which is a simple test on the coefficients. For general \( p_i, \) we must use Theorem 2.2, which cannot be done by simply looking at the coefficients. In the following, we will derive some simple check criteria for some subclasses of permutation polynomials.

If we limit the polynomials to be of second degree, we have the following corollary.

**Corollary 2.4** A second degree polynomial of the form \( P(x) = ax + bx^2 \) is a permutation polynomial over \( \mathbb{Z}_{p^n} \) if and only if \( a \neq 0 \) and \( b = 0 \ mod \ p. \)

Proof: When \( p = 2, \) this corollary is a special case of Theorem 2.1. So we only need to consider the case that \( p \neq 2. \) In order to prove the corollary, we only need to check the conditions in Theorem 2.2.

Let \( P(x) \) be a permutation polynomial over \( \mathbb{Z}_{p^n}, \) then from the second condition in Theorem 2.2, the formal derivative \( a + 2bx \neq 0 \) modulo \( p \) for all \( x \in [0, \ldots, p - 1]. \) Let \( x = 0, \) the second condition implies that \( a \neq 0 \ mod \ p. \) Assume \( b \neq 0 \ mod \ p, \) then \( 2b \)
is co-prime to \( p \) and \( 2bx \) permutes \( [0, \ldots, p - 1] \). This implies that there exists a certain \( x \) such that \( a + 2bx = 0 \), which contradicts the condition. Thus \( b = 0 \mod p \).

On the other hand, if both \( a \neq 0 \mod p \) and \( b = 0 \mod p \) are satisfied, then \( P(x) \) can be reduced to \( P(x) = ax \mod \mathbb{Z}(p) \), which is a permutation polynomial. The formal derivative \( P'(x) = a \neq 0 \). The conditions for Theorem 2.2 are satisfied and \( P(x) \) is a permutation polynomial over \( \mathbb{Z}_p \).

\[ \square \]

Another class of permutation polynomials over \( \mathbb{Z}_{p^n} \) can be “generated” using permutation polynomial \( P_{GF}(x) = ax \) over \( \mathbb{Z}(p) \), where \( p \) is a prime number. This is summarized in the following corollary

**Corollary 2.5** If \( P_{GF}(x) = ax \) is a permutation polynomial over \( \mathbb{Z}(p) \) for a prime \( p \), define \( P_P(x) = \sum b_i x^i \), where \( b_i = 0 \mod p \). Then \( P_R(x) = P_{GF}(x) + P_P(x) \) is a permutation polynomial over \( \mathbb{Z}_{p^n} \).

**Proof:** Since \( P_{GF}(x) = ax \) is a permutation polynomial over \( \mathbb{Z}(p) \) for prime \( p \), we must have \( a \neq 0 \mod p \). Since \( P_P(x) = 0 \mod p \), then \( P_R(x) = P_{GF}(x) \mod p \) and \( P_R(x) \) is a permutation polynomial over \( \mathbb{Z}(p) \). The formal derivative \( P'_P(x) = \sum ib_i x^{i-1} = 0 \mod p \). Then \( P'_R(x) = P'_P(x) + a = a \neq 0 \mod p \). Both of the two conditions in Theorem 2.2 are satisfied.

\[ \square \]

To simplify the notations, we need to introduce some definitions.

**Definition 2.1 (Base)** For \( N = \prod_{i=1}^{m} p_i^{n_i} \), we define the base of \( N \) to be the vector \( p_N = [p_1, p_2, \ldots, p_m] \).
**Definition 2.2 (Order of \( N \))** For \( N = \prod_{i=1}^{m} p_i^{n_{p_i}} \), we define the order of \( N \) to be the vector 
\[ o_N = [n_{p_1}, n_{p_2}, \ldots, n_{p_m}] \].

With respect to \( N \), for any given number \( x \), we can write it in the form 
\[ x = x_0 \prod_{i=1}^{m} p_i^{o_x[i]} \],
where the \( p_i \)'s are the elements in \( p_N \) and \( x_0 \) is co-prime to \( p_i \) for any \( i \).

**Definition 2.3 (Order of \( x \) with respect to \( N \))** We can define the order of \( x \) (with respect to \( N \)) by 
\[ o_x | N = [o_x[1], o_x[2], \ldots, o_x[m]] \]. Can be simplified to \( o_x \) when \( N \) is clear.

The elements of any order vectors must be non-negative. We also define \( p_N^o_x \) to be \( \prod_{i=1}^{m} p_N[i]^{o_x[i]} \).

For example, if \( N = 720 = 2^4 \cdot 3^2 \cdot 5^1 \), then \( p_N = [2, 3, 5] \) and \( o_N = [4, 2, 1] \). If \( x = 12 = 2^2 \cdot 3^1 \cdot 5^0 \), then \( o_x = [2, 1, 0] \) and \( p_N^{o_x} = 2^2 \cdot 3^1 \cdot 5^0 = 12 \). If \( y = 35 = 2^0 \cdot 3^0 \cdot 5^1 \cdot 7^1 \), then \( o_y = [0, 0, 1] \) and \( p_N^{o_y} = 2^0 \cdot 3^0 \cdot 5^1 \cdot 7^1 = 5 \). In this paper, by default, the orders of all numbers are with respect to \( N \).

Since our computations are modulo \( N \), we need to define a special comparison of orders. First we define the comparison of orders in each dimension. Given \( N \) with base \( p_N \) and order \( o_N \), for two numbers \( x \) and \( y \), their orders are \( o_x \) and \( o_y \) respectively. Basically, the comparison for the \( i \)'th elements in \( o_x \) and \( o_y \) is based on the result of comparison of \( x \) and \( y \) modulo \( p_N^{o_N[i]} \). More precisely, we define \( o_x[i] = o_y[i] \) if both of them are not less than \( o_N[i] \). Otherwise, define the ordering of \( o_x[i] \) and \( o_y[i] \) by their numerical ordering.

Based on the comparisons of orders in each dimension, we can define comparisons of orders. We define \( o_x \leq o_y \) if \( o_x[i] \leq o_y[i] \) for all \( i \). We define \( o_x < o_y \) if \( o_x \leq o_y \) and for at least one \( i \), \( o_x[i] < o_y[i] \). Other comparisons like \( o_x > o_y \), \( o_x \geq o_y \) and \( o_x = o_y \) are similarly defined. Note that by the above definition, the order is a **partial ordered set**. \( o_x \nleq o_y \) does not imply \( o_x \geq o_y \). We further define \( o_x \) strictly less than \( o_y \), denoted by \( o_x < o_y \), to be the case that \( o_x[i] < o_y[i] \) for all \( i \). Strictly greater than is similarly defined.
We also define some operations on orders. The summation and subtraction of orders are defined to be element-wise. However, since orders must be non-negative, subtraction \( o_z = o_x - o_y \) is only defined when \( o_x \geq o_y \). Other operations like \( \max(\cdot, \cdot) \) and \( \min(\cdot, \cdot) \) are also defined to be element-wise.

Orders of numbers have some useful properties.

Property 2.1 If \( z = xy \), then \( o_z = o_x + o_y \).

Property 2.2 If \( x \) divides \( y \), let \( z = \frac{y}{x} \), then \( o_z = o_y - o_x \).

Property 2.3 Let \( z = x^y \), then \( o_z = yo_x \).

The proofs for these properties are simple. Furthermore, when \( o_x \gg o_y \), if \( z = x + y \), then \( o_z = o_y \). This result is not as trivial as the previous ones and we will give a simple proof. Let \( x = x_0 \prod_{i=1}^{m} p^{o_x[i]}_i \) and \( y = y_0 \prod_{i=1}^{m} p^{o_y[i]}_i \). By the definition of order, \( x_0 \) and \( y_0 \) are co-prime to all \( p_i \)'s. Since \( o_x \gg o_y \),

\[
\begin{align*}
z &= x + y \\
&= x_0 \prod_{i=1}^{m} p^{o_x[i]}_i + y_0 \prod_{i=1}^{m} p^{o_y[i]}_i \\
&= \prod_{i=1}^{m} p^{o_x[i]}_i (x_0 \prod_{i=1}^{m} p^{o_x[i]-o_y[i]}_i + y_0),
\end{align*}
\]

(2.2)

and \( o_x[i] - o_y[i] \) is positive for all \( i \). Since \( y_0 \) is co-prime to all \( p_i \)'s, the term \( x_0 \prod_{i=1}^{m} p^{o_x[i]-o_y[i]}_i + y_0 \) is also co-prime to all \( p_i \)'s. Thus we can observe from (2.2) that \( o_z = o_y \).

2.3 Permutation Polynomial-Based Interleavers

An interleaver needs to permute the numbers in \( \{0, 1, 2, \ldots, N - 1\} \). This is exactly what a permutation polynomial can do. For example, when \( N = 8 \), Polynomial \( P(x) = 2x^2 + x + 3 \) satisfies the conditions in Theorem 2.1 and is a permutation polynomial over
If we interleave $x$ to $P(x)$, then the sequence $\{0, 1, 2, 3, 4, 5, 6, 7\}$ will be interleaved to $\{3, 6, 5, 0, 7, 2, 1, 4\}$.

In general, if a polynomial $P(x)$ is a permutation polynomial over $\mathbb{Z}_N$, then an interleaver based on this permutation polynomial can be defined as

$$\pi_{P_N} : x \mapsto P(x), \quad \forall x.$$  

![Figure 2.1: Different interleavers of length 256: (a) Random, (b) S-random, (c) Quadratic, (d) Permutation polynomial based, $P(x) = 8x^2 + 3x$](image)

In Figure 2.1, plots of some interleavers are shown. In these plots, the x-axis is the input bit position and the y-axis is the output bit position. Comparing random and S-random interleavers (Figure 2.1(a) and (b)), it can be observed that the points in S-random interleaver are distributed more uniformly in the plane. This property can help to avoid
short error events in one component code to be interleaved to short error events in the other
component code. Figure 2.1(c) is for the quadratic interleaver. It resembles the random
interleaver in the way that we can also observe irregularity in the density of points in the
plane. Figure 2.1(d) corresponds to an interleaver generated by the permutation polynomial
$P(x) = 8x^2 + 3x$. The density of points in the plane is also not uniform. This characteristic
is similar to random and quadratic interleavers. We can also observe that there are some
periodic patterns in the plot. This regularity does not necessarily imply a bad performance,
which can be better explained by considering the input weight 2 error events.

An input weight 2 error event is defined as an error event with two information bits in
error. Each input weight 2 error event of a turbo code corresponds to one input weight 2
error event in each of the two component codes. We know that in turbo codes using random
interleavers, input weight 2 error events decide the effective free distance and further decide
the performance of the code in the error floor region (high SNR) [5]. For permutation
polynomial-based interleavers, this type of error events is also an important factor. An
input weight 2 error event in the first component code can be represented by $(x, x + t)$. (The pair indicates the locations of the two 1’s in the error event). These two positions will
be interleaved to $\pi(x)$ and $\pi(x + t)$ in the second component code. The distance between
$\pi(x)$ and $\pi(x + t)$ is denoted by

$$\Delta(x, t) = \pi(x + t) - \pi(x) \mod N.$$ 

If we fix $t$, we can plot $\Delta(x, t)$ with respect to $x$. We are interested in points where $\pi(x + t)$
and $\pi(x)$ are close to each other. When we talk about points near $\Delta(x, t) = 0$, we mean
points close to the line from both above and below (near the top of the plot).

For $t = 3$, the distance $\Delta(x, t)$’s for interleavers in Figure 2.1 are shown in Figure 2.2.
Figure 2.2(a), 2.2(b), 2.2(c), 2.2(d) are for random, S-random, quadratic and permutation
Figure 2.2: $\Delta(x, 3)$ for different interleavers of length 256: (a) Random, (b) S-random, (c) Quadratic, (d) Permutation polynomial based, $P(x) = 8x^2 + 3x$

polynomial-based interleavers respectively. The plots for random and quadratic interleaver are very similar and look random. At a first glance, the plot for the S-random interleaver is also similar to Figures 2.2(a) and 2.2(c). But after a closer look, one can notice that there are no points near the $\Delta(x, t) = 0$ line. This is because for this S-random interleaver with $S = 8$, since $t \leq S$, $S < \Delta(x, t) < N - S$ for all $x$. For a permutation polynomial-based interleaver, the $\Delta(x, t)$ is very regular. All the points are uniformly located along a few equally separated horizontal lines. Similar to the S-random interleaver, if the coefficients in $P(x)$ are carefully chosen, there are no points near the $\Delta(x, t) = 0$ line. The remaining problem is how to select good coefficients. This is discussed in the next section.
2.4 Permutation Polynomials Search

Interleavers based on different permutation polynomials have different performances. We are interested in finding the best permutation polynomial for a given component code and a given frame size. For a fixed frame size $N$, the only variables are the degrees and the coefficients of the permutation polynomials.

In this paper, we will focus on second degree polynomials $P(x) = bx^2 + ax + c$. The first reason for this selection is to have the lowest possible complexity. A first degree polynomial $P(x) = ax + c$ (linear interleavers) is the simplest one among all permutation polynomials. However, as shown in [48], linear interleavers have very bad input weight 4 error event characteristics, causing a high error floor at medium to long frame size cases. This leaves the second degree polynomial the next in the chain. Another reason to consider second degree polynomials is their relatively easy analysis. This will be shown in the following subsections.

Notice that the constant term $c$ just corresponds to a cyclic rotation in the interleaved sequence. It does not appear in the conditions for $P(x)$ to be a permutation polynomial and it does not affect the performance of the concatenated system if we disregard boundary effects. Therefore, we can simply let it be zero and consider polynomials of the form $P(x) = bx^2 + ax$.

In this section, we will use the minimum distance (or the first few spectrum lines) of a subset of error events of the turbo code as a criterion to select the coefficients of the polynomials. Namely, we will only consider some error events with an input weight $2^m$. Although these error events do not cover all possible error events, due to the structure of the permutation polynomial-based interleavers, these error events have high multiplicity and the performance of the turbo code is usually dominated by them, especially when the
frame size is not very short. The benefit of limiting to this subset is that these error events are relatively easy to be found and counted. The minimum distance for the limited set is usually close to the real minimum distance and can be used as an upper bound. We can also find the true minimum distance or the first few spectrum lines of the turbo code using the method in [17]. However, the complexity of doing this is relatively high.

In this paper, we assume that for each of the component convolutional code, a tail-biting trellis [24] is used. The end of the trellis is directly connected to the start of the trellis and no flushing bits are used. For tail-biting trellises, a cyclic shift of an error event in one component code is also an error event. Furthermore, it is possible that an error event starts near the end of the trellis and wraps over to the head of the trellis. Due to this phenomenon, we say that the error events are modulo $N$.

The tail-biting assumption is used to simplify our analysis. By using this assumption, we can ignore any boundary effects due to termination, which helps in finding and counting the error events we are interested in. Unfortunately, since we are using systematic recursive convolutional codes as component codes, a tail-biting trellis does not always exist [41]. Some form of termination has to be used. Some modulo $N$ error events will be broken by termination and the termination itself may introduce some extra error events. If we still find and count the error events modulo $N$, some error will be introduced. However, the proportion of the modulo $N$ error events broken by termination is very small and the low weight error events introduced by the termination are usually of low multiplicities. Therefore, when the frame size is not very short, the modulo $N$ error events dominate the performance. We can still count the error events modulo $N$ and ignore the error introduced in the weight spectrum estimation.
2.4.1 Input Weight $2^m$ Error Events

Long random interleavers can be approximated by uniform interleavers, which is a probabilistic device that interleaves a given input position to any possible output positions with an equal probability [5]. Using the uniform interleaver model, for very high SNR region, the decoding performance has been shown to be dominated by input weight 2 error events. The minimum distance associated with input weight 2 error events is called effective free distance $d_{ef}$ [14] and has been used as a design criterion to select component codes to have a good error floor performance.

Permutation polynomial-based interleavers are not random. In some sense, they cannot be well approximated by a uniform interleaver. So considering the effective free distance may not be enough. For example, for a linear permutation polynomial $P(x) = ax$, by carefully selecting the parameter $a$, the effective free distance can be designed to be very large. However, as analyzed in [48], for a certain type of input weight 4 error events, the corresponding Hamming distance is small. Furthermore, the number of these input weight 4 error events is proportional to the length of the frame, and cannot be controlled by selecting $a$. At least for a medium to high channel SNR, the performance is dominated by these input weight 4 error events.

We can think of the permutation polynomial-based interleavers as generalizations to the linear interleavers. It is observed that the most frequent error events are of input weight $2^m$ where $m = 1, 2, \ldots$, just like in the linear case. However, the number of these error events can be controlled by selecting the parameters $a$ and $b$. Given the component codes, we can find parameters pairs with a good performance.

A typical input weight $2^m$ error event is shown in Fig. 2.3. This error event is composed of $m$ input weight 2 error events (represented by dashed line segments) in each component
code and these error events are inter-connected via the interleaver (solid lines). In the figure, the \( i \)'th input weight 2 error event in the first component code begins at \( x_i \) and has length \( t_i + 1 \). The \( i \)'th input weight 2 error event in the second component code has length \( s_i + 1 \). Each input weight 2 error event can be represented by a pair of integers indicating the starting and ending positions in the corresponding trellis. Since we use the same systematic recursive convolutional code for both component codes, all \( t_i \)'s and \( s_i \)'s are multiples of the *cycle length* \( \tau \) of the convolutional code. Here the cycle length \( \tau \) is defined as the cycle of the output of the encoder when the input sequence is \([1, 0, 0, 0, \ldots]\). For example, for recursive convolutional code with generating polynomial \( \frac{1+D^2}{1+D+D^2} \) (from now on, we will use the simple octal notation, 5/7 for this example), with the previously mentioned input, the output sequence is \([1, 1, 1, 0, 1, 1, 0, 1, 1, 0, \ldots]\). There is a cycle of \([1, 1, 0]\) in the output and the cycle length \( \tau = 3 \). The cycle length equals to the length of the shortest input weight 2 error event minus 1. We define *error pattern* to be the length 2\( m \) vector \([t_1, t_2, \ldots, t_m, s_1, s_2, \ldots, s_m]\).
For this input weight $2m$ error event, we have

\[
P(x_2) - P(x_1) = s_1, \quad (2.3.1)
\]
\[
P(x_3) - P(x_1 + t_1) = s_2, \quad (2.3.2)
\]
\[
P(x_4) - P(x_2 + t_2) = s_3, \quad (2.3.3)
\]
\[- \vdots - \]
\[
P(x_{m-1}) - P(x_{m-3} + t_{m-3}) = s_{m-2}, \quad (2.3.m-2)
\]
\[
P(x_m) - P(x_{m-2} + t_{m-2}) = s_{m-1}, \quad (2.3.m-1)
\]
\[
P(x_m + t_m) - P(x_{m-1} + t_{m-1}) = s_m. \quad (2.3.m)
\]

For such an input weight $2m$ error event to be formed, all these $m$ equations need to be satisfied. There are a total of $3m$ variables in these $m$ equations. Among these $3m$ variables, the $x_i$’s take value from 0 to $N - 1$ and both $t_i$’s and $s_i$’s are multiples of the cycle length $\tau$. We will only consider $t_i$’s and $s_i$’s to be small multiples of $\tau$ since large values are associated with large Hamming distances.

To simplify the process of finding the error event, we will convert (2.3) to a form that is easier to analyze. This is done by computing $s_1 - s_2 + s_3 - s_4 + \cdots$. Using (2.3) and after some manipulations, we have

\[
s_1 - s_2 + s_3 - s_4 + \cdots = 2b(x_1 t_1 - x_2 t_2 + x_3 t_3 - x_4 t_4 \cdots )
\]
\[
+ b(t_1^2 - t_2^2 + t_3^2 - t_4^2 \cdots )
\]
\[
+ a(t_1 - t_2 + t_3 - t_4 \cdots ), \quad (2.4)
\]

or simply,

\[
\sum_{i=1}^{m} (-1)^{i-1}s_i = 2b \sum_{i=1}^{m} (-1)^{i-1}x_i t_i + b \sum_{i=1}^{m} (-1)^{i-1}t_i^2 + a \sum_{i=1}^{m} (-1)^{i-1}t_i. \quad (2.5)
\]
Of course, (2.5) by itself is not a necessary and sufficient condition for such an input weight $2m$ error event to appear. It needs to be combined with any $m - 1$ out of $m$ equations in (2.3). This set of equations will be used in the rest of the paper to check for an input weight $2m$ error event.

Given an error pattern, if the $2m$ first order error events do not overlap, the Hamming distance of the error event can be uniquely decided. This can be better explained by an example. Assume the component code is the recursive 5/7 code. The cycle length is $\tau = 3$.

Since all $t_i$’s and $s_i$’s are multiple of $\tau$, they are in the common form of $k\tau$. Let the input be $1 + D^{k\tau}$. Then the output is

$$
(1 + D^{3k}) \frac{1 + D^2}{1 + D + D^2} = (1 + D^3 + D^{2 \cdot 3} + \cdots + D^{3(k-1)}) \frac{1 + D^2}{1 + D + D^2} \quad (2.6)
$$

which is the summation of $k$ shifted versions of the output sequence with an input $1 + D^3$.

The weight of the output sequence of the input $1 + D^3$ is $w_0 = 2$. (It does not include the 1’s at the two ends). Then the weight of the output with the input $1 + D^{3k}$ is $2 + w_0k$. The total output weight for the error event is

$$
6m + \left( \frac{\sum |t_i|}{\tau} + \frac{\sum |s_i|}{\tau} \right) w_0, \quad (2.7)
$$

which contains the weight in both recursive component codes ($2m + \frac{\sum |t_i|}{\tau} w_0$ or $2m + \frac{\sum |s_i|}{\tau} w_0$) and the weight of the systematic bits ($2m$). For other component codes, only the term $w_0$ is different.

When some of the first order error events overlap, the corresponding Hamming distance is smaller than computed using (2.7) and can only be computed when all $x_i$, $t_i$ and $s_i$ are known. However, when $m$ is small, the number of such error events is small and can be
ignored for the same reason that we ignore the boundary effect. In the following part of the paper, we will just use (2.7) to compute the Hamming distance of an error pattern.

### 2.4.2 Search for Good Interleavers using Effective Free Distance

Although for deterministic interleavers, large $d_{ef}$ does not guarantee a good performance, small $d_{ef}$ is usually associated with a bad performance. We can use $d_{ef}$ as a simple criterion to rule out bad permutation polynomials.

In this subsection, we are interested in constructing interleavers that do not map a short input weight 2 error event in one component code to another short input weight 2 error event in the other component code.

For random interleavers and quadratic interleavers, we have no control over the input weight 2 error events. We can at best state that the probability for such error events to appear goes to zero as the interleaver size goes to infinity. For S-random interleavers, by definition, input weight 2 error events with length less than $S$ in both component codes can be avoided. Typically, an S-random interleaver with $S = \sqrt{N/2}$ can be found with reasonable complexity [14].

When $t \leq S$, an S-random interleaver will map $(x, x + t)$ to $(y, z)$ where $|y - z| > S$. But for a given component code, there are input weight 2 error events only for some values of $t$, namely, those $t$ that are multiples of the cycle length. Thus, in this sense, some of the ability of S-random interleaver is wasted. Using a permutation polynomial-based interleaver, we can select coefficients such that we only avoid input weight 2 error events tailored for a given component code. In this way, we may break even larger input weight 2 error events.
Let us restrict ourselves to polynomials of the form \( P(x) = bx^2 + ax \). By Corollary 2.4, \( P(x) = bx^2 + ax \) is a permutation polynomial if none of the elements in vector \( o_b \) (order of \( b \)) is zero, and the vector \( o_a \) (order of \( a \)) is an all zero vector.

Let \( t + 1 \) be the length of the input weight 2 error event in the first component code. Then \( t \) must be a multiple of the cycle length \( \tau \). Let the order of \( t \) be \( o_t \). Then the length of input weight 2 error event in the other component code minus 1 is

\[
\Delta(x, t) = P(x + t) - P(x) = 2bt x + bt^2 + at.
\] (2.8)

The coefficient of \( x \) is \( c_1 = 2bt \), which, by Property 2.1, has order \( o_{c_1} = o_2 + o_b + o_t \). For \( x \in \{0, 1, 2, \ldots, N - 1\} \), the first term of (2.8) is \( k \cdot p_{N}^{o_{c_1}} \) where \( k = 0, 1, \ldots, p_{N}^{o_{N} - o_{c_1}} - 1 \). Each of these values is taken \( p_{N}^{o_{c_1}} \) times. If we plot this term with respect to \( x \), we have \( p_{N}^{o_{N} - o_{c_1}} \) horizontal lines. The last two terms in (2.8) are not related to \( x \). So they provide an offset to the horizontal lines. This justifies the plot of \( \Delta(x, t) \) in the form of Figure 2.2(d).

To avoid short input weight 2 error events, when \( t \) is a small multiple of \( \tau \), we prefer all \( \Delta(x, t) \) that are also multiple of \( \tau \) to be far away from zero. In order to achieve this, we require the vector \( o_{c_1} \) to be relatively large, so that there are less horizontal lines in the \( \Delta(x, t) \) plot and we will be able to select the coefficients to move the \( \Delta(x, t) \) far away from zero. Since \( o_{c_1} \) is relatively large, we are only interested in the first line of \( \Delta(x, t) \) either above or below zero. Their distances to zero can be represented by

\[
s = \pm \Delta(x, t) \mod p_{N}^{o_{c_1}} = \pm (bt^2 + at) \mod p_{N}^{o_{c_1}}.
\] (2.9)

Given \( a, b \) and \( \tau \), we define \( L_{a,b,\tau} = \min(|t| + |s|) \), where \( t \) and \( s \) are multiples of \( \tau \), and use it as a criterion to select good interleavers. For a given component code, \( d_{ef} \) can be computed from \( L_{a,b,\tau} \). However, in the search for permutation polynomial-based interleaver
based on comparing effective free distance, the cycle length is the only information we need to know about the component code.

Before doing a search for good \(a\) and \(b\), it is useful to limit their ranges. This can be achieved by using following two Lemmas.

**Lemma 2.6** *For input weight 2 error event analysis, if we write \(b = b_1b_0 = b_1 \cdot p_N^{o_b}\), we can assume \(b_1 = 1\).*

Proof: From the assumption, \(b_1\) is co-prime to \(N\). By (2.9), for permutation polynomial \(P_1(x) = p_N^{o_b}x^2 + ax\), equation (2.9) becomes \(s_1 = p_N^{o_b}t^2 + at \mod p_N^{o_b+o_t+o_2}\). For another permutation polynomial \(P_2(x) = b_1p_N^{o_b}x^2 + ax\), equation (2.9) becomes \(s_2 = b_1p_N^{o_b}t^2 + at \mod p_N^{o_b+o_t+o_2}\). Compute \(s_2 - s_1\),

\[
s_2 - s_1 = (b_1 - 1)p_N^{o_b}t^2 \mod p_N^{o_b+o_t+o_2}. \tag{2.10}
\]

If 2 is a factor of \(N\), since \(b_1\) is co-prime to \(N\), \(b_1\) is odd and \(b_1 - 1\) is even. Then the right hand side of (2.10) has an order at least \(o_2 + o_b + 2o_t\). If 2 is not a factor of \(N\), the right hand side of (2.10) has an order at least \(o_b + 2o_t\) and the computation is modulo \(o_b + o_t\). In both these two cases, \(s_2 - s_1 = 0 \mod p_N^{o_b+o_t+o_2}\). This implies that \(P_1(x)\) and \(P_2(x)\) have the same input weight 2 error events, only their locations are different. From this point of view, these two polynomials are equivalent.

\[\Box\]

**Lemma 2.7** *For input weight 2 error event analysis, given \(b = b_1 \cdot p_N^{o_b}\), we only need to consider \(a\) such that \(1 \leq a < p_N^{o_b}\).*

Proof: We will use the result of Lemma 2.6 and let \(b = p_N^{o_b}\).
Let $a_0 = a \mod p_N^{\alpha_0+\alpha_2}$. Then $a$ can be written as $a = a_0 + l p_N^{\alpha_0+\alpha_2}$. We have

\[ s = \pm((b t^2 + (a_0 + l p_N^{\alpha_0+\alpha_2}) t) \mod p_N^{\alpha_0+\alpha_2} \]

\[ = \pm((b t^2 + a_0 t) \mod p_N^{\alpha_0+\alpha_2}, \]  
which implies that $L_{a,b,\tau} = L_{a_0,b,\tau}$.

When $2$ is not a factor of $N$, the above is sufficient for the proof. When $2$ is a factor of $N$, from the previous proof, without loss of generality, we can assume $1 \leq a < p_N^{\alpha_0+\alpha_2}$. Let $a_0 = p_N^{\alpha_0+\alpha_2} - a$. Then

\[ s = \pm((b t^2 + a_0 t) \mod p_N^{\alpha_0+\alpha_2+\alpha_t} \]

\[ = \pm((b t^2 + (p_N^{\alpha_0+\alpha_2} - a) t) \mod p_N^{\alpha_0+\alpha_2+\alpha_t} \]

\[ = \pm((b t^2 - a_0 t) \mod p_N^{\alpha_0+\alpha_2+\alpha_t} \]

\[ = \pm((b(-t)^2 + a_0(-t)) \mod p_N^{\alpha_0+\alpha_2+\alpha_t}, \] 
and $L_{a,b,\tau} = L_{a_0,b,\tau}$. This finishes our proof.

\[ \square \]

For given $\tau$ and fixed $o_b$, according to previous two Lemmas, we can calculate $L_{a,b,\tau}$ and select good parameters. For example, for $5/7$ code, $\tau = 3$. For $7/5$ code, $\tau = 2$. For $N = 2^n$, $p_N = [2]$ and $o_N = [n]$. If we fix $o_b = [4]$, which corresponds to $b = 16$, the search results are shown in Table 2.1.

<table>
<thead>
<tr>
<th>$a$</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>11</th>
<th>13</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L(5/7)$</td>
<td>12</td>
<td>18</td>
<td>12</td>
<td>24</td>
<td>24</td>
<td>18</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>$L(7/5)$</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>12</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of $L$ for $b = 16$ with different $a$. Turbo code with 5/7 or 7/5 component code
For a frame size of 1024, simulation results for turbo codes with 5/7 component code and 7/5 component code are shown in Figures 2.4 and 2.5, respectively. The orderings of the performance are the same as given by the $L_{a,b,\tau}$ criterion.

Once $o_b$ is fixed, it seems that the effective free distance is a simple way to select good $a$ and $b$ for the permutation polynomial. However, effective free distance does not provide enough information to select $o_b$. For example, it can be seen from (2.9) that the larger the $o_b$, the better the opportunity to select good $a$ to achieve larger effective free distance. But from computer simulations, it can be observed that the best performance of the permutation polynomial increases with $o_b$ up to a certain degree. Then it starts to decrease. In order to explain this and find more accurate ways to select the parameters, we need to investigate higher input weight error events.
2.4.3 Higher Input Weight Error Events

We know that input weight $2^m$ error events are important for permutation polynomial-based interleavers. In order to find the minimum distance of this subset of error events, we need to check these error events for small $m$. (Large $m$ is usually associated with large Hamming distance as shown in (2.7)).

As mentioned in the previous sections, for an input weight $2^m$ error event, among the $3^m$ variables ($x_i$, $t_i$ and $s_i$ for $i = 1, \ldots, m$) that describe the error event, $2^m$ of them are independent. One way to find the error event is first to fix the error pattern $[t_1, \ldots, t_m, s_1, \ldots, s_m]$ and then enumerate $x_1$. Given the error pattern and $x_1$, all other $x_i$ can be computed using any $m - 1$ out of $m$ equations in (2.3). Finally, these $3^m$ values can be applied to the un-used equation in (2.3) or (2.4) to verify if they form a valid error event.
Since permutation polynomial-based interleavers are highly structured, we show next that it is not necessary to enumerate all $x_1$ from 0 to $N - 1$. On the contrary, checking the range from 0 to $p_N^{a_N - a_2 - a_b} - 1$ is enough. Assume we have an error event as shown in Fig. 2.3. Each equation in (2.3) has the form

$$P(y_1) - P(y_2) = (by_1^2 + ay_1) - (by_2^2 + ay_2) = s$$  \hspace{1cm} (2.13)

In the first component code, if we cyclicly shift all $2m$ end points of the $m$ first order error events by any number $T$, we still have a valid input weight $2m$ error event under the tail-biting assumption. On the other side of the interleaver, after the cyclic shift, (2.13) becomes

$$P(y_1 + T) - P(y_2 + T) = (by_1^2 + ay_1) - (by_2^2 + ay_2) + 2bT(y_1 - y_2)$$ \hspace{1cm} (2.14)

When $2bT = 0 \mod N$,

$$P(y_1 + T) - P(y_2 + T) = s,$$ \hspace{1cm} (2.15)

and the interleaved points in the second component code also form a valid input weight $2m$ error event. The smallest $T$ that satisfies $2bT = 0 \mod N$ is $T = p_N^{a_N - a_2 - a_b}$. To summarize, the input weight $2m$ error events have a cyclic structure. If we cyclicly shift all $2m$ end points of the error event in the first component code by multiples of $p_N^{a_N - a_2 - a_b}$, we will still have a valid input weight $2m$ error event for the entire turbo code. Thus, in searching for error events, it is sufficient to enumerate all $x_1$ from 0 to $p_N^{a_N - a_2 - a_b} - 1$. When $p_N^{a_N - a_2 - a_b}$ is small, this method is efficient.

For large frame sizes, $p_N^{a_N - a_2 - a_b}$ is usually large, making it time-consuming to enumerate $x_1$. It is interesting to directly solve for $x_1$ given the error pattern $[t_1, \ldots, t_m, s_1, \ldots, s_m]$. 

34
In order to do this, we prefer to write the constraints of the input weight $2m$ error event with the given error pattern into a single equation.

In Section 2.4.1, an input weight $2m$ error event is defined by (2.5) and $m - 1$ equations in (2.3). Since we fix $t_i$ and $s_i$, there are $m$ non-linear equations and $m$ variables ($x_i$, $i = 1, \ldots, m$). However, in order to solve $x_1$ directly, we would like to transform these $m$ equations to a single equation with only one variable, $x_1$. In other words, we need to apply $m - 1$ equations in (2.3) to (2.5) and cancel $x_i$, $i = 2, \ldots, m$. In order to do this, we need to be able to solve the following problem: given $N, a, b$ and $s$, if $P(y) - P(x) = s$, write $y$ as a function of $x$.

Before proceeding, we need some more definitions. Given a permutation polynomial $P(x) = ax + bx^2$ and a fixed $s$, define a sequence $\{y_i\}$ such that

\[
\begin{align*}
    P(y_0) - P(0) &= s, \\
    P(y_1) - P(1) &= s, \\
    P(y_2) - P(2) &= s, \\
    \vdots \\
    \end{align*}
\]

(2.16)

Then recursively define $\Delta_0(i) \triangleq y_i - i$, $\Delta_1(i) \triangleq \Delta_0(i + 1) - \Delta_0(i)$, $\Delta_2(i) \triangleq \Delta_1(i + 1) - \Delta_1(i)$, etc. Note that all $y_i$ and $\Delta_k(i)$ are function of $a, b$ and $s$.

We have the following Theorem.

**Theorem 2.8** Given $N$, $P(x)$ and $s$, $\Delta_k(i)$ has the same order for all $i$. If the order of $\Delta_k(i)$ is denoted as $o_{\Delta_k}$, then $o_{\Delta_0} = o_s$ and $o_{\Delta_k} = o_{\Delta_{k-1}} + o_b + o_{2k}$ for $k > 0$, which implies that $o_{\Delta_k} = ko_b + ko_2 + \sum_{n=1}^k a_n + o_s$ for all $k$.

The proof for this theorem is in the Appendix.
Since the order of $\Delta_k$ is strictly increasing with respect to $k$, it will eventually be larger than $o_N$. If $K$ is the largest number such that $o_{\Delta K} \geq o_N$, then $\Delta_{K+1}(i) = 0 \mod N$. By the definition of $\Delta_{K+1}(i)$, $\Delta_K(i)$ is a constant for all $i$’s. This result is summarized in the following corollary.

**Corollary 2.9** For given $N$, $P(x)$ and $s$, if $K$ is the largest number such that $o_{\Delta K} \geq o_N$, then $\Delta_K(i) = \Delta_K$ is a constant for all $i$.

Note that $K$ is a function of $N$, $a$, $b$ and $s$. When the parameters are not clear from the context, we will explicitly write $K$ as a function, such as $K(s)$.

If the $K$ in Corollary 2.9 is found, for all $k > K$, $\Delta_k(i) = 0$. Furthermore, if for all $0 \leq k \leq K$, the $\Delta_k(0)$ is known, all $\Delta_k(i)$ can be computed for all $i$ using the definitions of $\Delta_k(i)$. For example, $\Delta_K(i) = \Delta_K(0)$, $\Delta_{K-1}(i) = \Delta_{K-1}(0) + i \Delta_K(0)$, etc. For the simplicity of the expressions, we will use $\Delta_k$ to represent $\Delta_k(0)$ when the context is clear. When $s$ needs to be specified, we will use the notation $\Delta_k(s)$.

Now we have the tool to find the relationship between $x$ and $y$ when we know $P(y) - P(x) = s$. This is summarized in the following theorem.

**Theorem 2.10** For given $N$, $P(x)$ and $s$, if $P(y) - P(x) = s$, then

$$y = x + \Delta_0(s) + x\Delta_1(s) + \frac{x(x-1)}{2!} \Delta_2(s) + \frac{x(x-1)(x-2)}{3!} \Delta_3(s) + \cdots.$$ (2.17)

We provide a sketch for a proof of this theorem. For the parameters given, we can compute all $\Delta_k(0)$ for $k = 0, \ldots, K(s)$. From Corollary 2.9, we know $\Delta_K(i)$ is a constant. Then we can recursively compute $\Delta_{K-1}(i)$, $\Delta_{K-2}(i)$, etc., using their definitions. All these computations are summation of series. Finally we can compute $\Delta_0(i)$ for all $i$. Then by the definition of $\Delta_0(i)$, $y_x = x + \Delta_0(x)$. 36
If we define \( \binom{x}{k} = 0 \) when \( x < k \), then (2.17) can be written as
\[
y = F(x, s) \triangleq x + \sum_{k=0}^{\infty} \binom{x}{k} \Delta_k(s) .
\] (2.18)

Since \( \binom{x}{k} \) is always an integer, \( F(x, s) \) will always be integer valued. However, it may not be a polynomial of \( x \). This is because \( \frac{\Delta_k(s)}{k!} \) may not be an integer. Since we do not know much about polynomials with fractional coefficients, it is preferred to somehow transform \( F(x, s) \) to an integer valued polynomial.

Given \( N \), define \( D(k) = \frac{k!}{p_N^{\alpha_2+\alpha_3+\cdots+\alpha_k}} \). \( D(k) \) can be written as \( D(k) = \prod_{i=2}^{k} \frac{i}{p_N^i} \). Each term of \( D(k) \) is in the form of \( \frac{1}{p_N^i} \). Since \( p_N^{\alpha_i} = \gcd(N,i) \), \( \frac{1}{p_N^i} \) is also an integer and it is either equal to 1 or co-prime to \( N \). The order of \( \frac{1}{p_N^i} \) is always a zero vector. Then the product \( D(k) \) is also an integer and it is either 1 or is co-prime to \( N \). If we multiply \( \frac{\Delta_k}{k!} \) by \( D(k) \), then we have \( \frac{\Delta_k}{p_N^{\alpha_2+\alpha_3+\cdots+\alpha_k}} \). Since \( \alpha_k = k o_b + k o_2 + \sum_{n=1}^{k} o_n + o_s \), it has a factor of \( p_N^{\alpha_2+\alpha_3+\cdots+\alpha_k} \) and \( D(k) \frac{\Delta_k}{k!} \) is an integer and its order is \( k o_b + k o_2 + o_s \). Next consider \( D(K) \frac{\Delta_k}{k!} \) for \( K > k \). Since \( D(K) = D(k) \prod_{i=k+1}^{K} \frac{i}{p_N^i} \),
\[
D(K) \frac{\Delta_k}{k!} = D(k) \frac{\Delta_k}{k!} \prod_{i=k+1}^{K} \frac{i}{p_N^i} .
\] (2.19)

As we have shown previously, both \( D(k) \frac{\Delta_k}{k!} \) and \( \prod_{i=k+1}^{K} \frac{i}{p_N^i} \) are integers. Then \( D(K) \frac{\Delta_k}{k!} \) for \( K > k \) is also an integer. Its order is still \( k o_b + k o_2 + o_s \).

Using this result, it can be easily seen that \( D(K)F(x, s) \) is a polynomial of \( x \) with integer coefficients. This property will be used in the following parts.

**Finding the Error Event by Solving Equations**

Using Theorem 2.10, given an input weight \( 2m \) error pattern \([t_1, t_2, \ldots, t_m, s_1, s_2, \ldots, s_m]\), searching for error events with the error pattern can be transformed into solving a polynomial equation.
An error event can be formed if (2.4) holds. Applying the result of Theorem 2.10 to the first \(m - 1\) equations in (2.3), we have

\[
\begin{align*}
x_2 &= F(x_1, s_1), \\
x_3 &= F(x_1 + t_1, s_2), \\
x_4 &= F(x_2 + t_2, s_3), \\
x_5 &= F(x_3 + t_3, s_4), \\
\vdots \\
x_m &= F(x_{m-2} + t_{m-2}, s_{m-1}),
\end{align*}
\]

(2.20)

Applying (2.20) in (2.4), in general, we have a polynomial equation of \(x_1\). For a general polynomial equation, the complexity to find the solutions or even find the number of solutions is high [2]. However, we have simpler methods to deal with polynomials associated with input weight 2 error events for small \(m\). In this subsection, we will only consider \(m = 1, 2\) and 3.

The input weight 2 error pattern is the simplest case. Here (2.20) is not used and (2.4) reduces to

\[
2bx_1 + bt_1^2 + at_1 - s_1 = 0.
\]

(2.21)

This is a linear polynomial of \(x_1\) in the form

\[
c_1x + c_0 = 0.
\]

(2.22)

We know that (2.22) has a solution if and only if \(o_{c_0} \geq o_{c_1}\). If this condition is satisfied, we can divide (2.22) by \(p_N^{o_{c_1}}\). The result is a first degree permutation polynomial modulo \(p_N^{o_N - o_{c_1}}\) and there is a unique solution. If \(x_0\) is the solution modulo \(p_N^{o_N - o_{c_1}}\), all \(x_0 + kp_N^{o_N - o_{c_1}}\) for any \(k\) are solutions to (2.22) modulo \(N\). Then for (2.22), there are exactly \(p_N^{o_{c_1}}\) solutions. In most cases, we are only interested in the Hamming distance of the error
event and multiplicity for such Hamming distance. It is not necessary to find out the exact location of the error event. So instead of solving equation (2.22), what we need to do is just check if $o_{c_0} \geq o_{c_2}$. If true, then add the multiplicity of $p_{N}^{o_{c_1}}$ to the corresponding spectrum line. The Hamming distance of the error event is directly decided by its error pattern using (2.7).

Equation (2.4) is more complex for $m > 1$ cases. However, we can show that, under some mild conditions, at least for $m = 2$ and $m = 3$, (2.4) can still be transformed to a permutation polynomial when (2.20) is applied. For larger $m$, we conjecture that the same is also true. Here, we do not consider larger $m$ cases in our minimum distance computation for they usually correspond to larger Hamming distance. We will satisfy ourselves by proving the cases for input weights 4 and 6.

For the input weight 4 case, using (2.20), (2.4) becomes

\[
2b \left[ x_1 t_1 - \left( x_1 + \sum_{k=0}^{\infty} \frac{x_1^k}{k!} \Delta_k(s_1) \right) t_2 \right] + b(t_1^2 - t_2^2) + a(t_1 - t_2) - (s_1 - s_2) = 0.
\] (2.23)

Collecting terms with respect to $x_1$, it becomes

\[
(-2b \Delta_0(s_1) t_2 + b(t_1^2 - t_2^2) + a(t_1 - t_2) - (s_1 - s_2))
\]

\[
+ 2b(t_1 - t_2 - \Delta_1(s_1) t_2) x_1
\]

\[- 2bt_2 \sum_{k=2}^{\infty} \frac{\Delta_k(s_1)}{k!} \prod_{m=0}^{k-1} (x_1 - m) = 0.
\] (2.24)

This is a polynomial of $x_1$ with possibly fractional coefficients. For $s_1$, we can find the $K$ in Corollary 2.9. Then multiply (2.24) by $D(K)$. From the analysis in the previous part, the result is a polynomial with integer coefficients in the form of

\[
c_0 + c_1 x_1 + c_2(x_1) = 0,
\] (2.25)
where
\[
c_0 = \left( -2b\Delta_0(s_1)t_2 + b(t_1^2 - t_2^2) + a(t_1 - t_2) \right) D(K),
\]
\[
c_1 = 2b(t_1 - t_2 - \Delta_1(s_1)t_2) D(K),
\]
\[
c_2(x) = -2bt_2 \sum_{k=2}^{\infty} \frac{\Delta_k(s_1) D(K)}{k!} \prod_{m=0}^{k-1} (x_1 - m).
\]

(2.26)

c_0 and c_1 are integers and c_2(x_1) is a polynomial of x_1. Furthermore, since D(k) is either 1 or is co-prime to N, multiplying by D(k) will not change the solutions for (2.24).

Now, let us check the order of terms in (2.26). We have shown that the order of \( \frac{\Delta_k(s_1)}{k!} D(K) \) is \( ko_b + ko_2 + os_1 \) for \( K \geq k \). If we expand the third term of \( c_2(x_1) \) to a polynomial of \( x_1 \), all the coefficients should have an order at least \( 3o_b + 3o_2 + os_1 + ot_2 \). If \( t_1 = t_2 \), \( oc_1 = 2o_b + 2o_2 + os_1 + ot_2 \). Since all the elements in \( o_b \) are positive, \( oc_1 \) is strictly less than the order of the orders of all terms in \( c_2(x_1) \). By Corollary 2.5, the polynomial of \( x_1 \) formed by \( \frac{c_1 x_1 + c_2(x_1)}{p_N^{oc_1}} \) is a permutation polynomial modulo \( p_N^{oc_1} \). In order to apply Corollary 2.5 to \( t_1 \neq t_2 \) case, we require that
\[
oc_1 \ll 3o_b + 3o_2 + os_1 + ot_2.
\]

(2.27)

Since we are interested in small Hamming distance error events, \( t_1 \) and \( t_2 \) are usually small, which implies that \( t_1 - t_2 \) is also small. The above condition is usually satisfied.

Given an error pattern \([t_1, t_2, s_1, s_2]\), first we need to verify that (2.27) holds. (In our search, we have never observed a case that this does not hold). Then similar to the input weight 2 case, we only need to check if \( oc_0 \geq oc_1 \) for the error pattern. If it is true, \( p_N^{oc_1} \) error events with this error pattern exist. If we are only interested in the spectrum, we do not need to solve the equation and only \( \Delta_0(s_1) \) and \( \Delta_1(s_1) \) need to be computed for the error pattern.
For the input weight 6 error event, we can use the same procedure as for the input weight 4 error event analysis. Equation (2.4) will be finally reduced to the same form as in (2.25). Similar to the input weight 4 case, a factor $D$ needs to be multiplied to the equation to transform it into a polynomial with integer coefficients. Here $D = D(\max(K(s_1), K(s_2)))$. Then in (2.25), $c_1 = 2b(t_1 - t_2 + t_3 - \Delta_1(s_1)t_2 + \Delta_1(s_2)t_3)D$ and the coefficients of $c_2(x)$ have orders at least equal to $\min(3o_b + 3o_2 + o_{s_1} + o_{t_2}, 3o_b + 3o_2 + o_{s_2} + o_{t_3})$. The condition to apply Corollary 2.5 is that

$$o_{c_1} \ll \min(3o_b + 3o_2 + o_{s_1} + o_{t_2}, 3o_b + 3o_2 + o_{s_2} + o_{t_3}).$$

(2.28)

This is usually the case when $t_i$’s are small. Then $\frac{c_1x_1 + c_2(x_1)}{p_{N-oc_1}}$ is a permutation polynomial modulo $p_N^{oc_1}$. As in the input weight 4 case, we can compare $o_{c_0}$ and $o_{c_1}$ to see if error events with such error pattern exist and find the multiplicities of the error events. If one is only interested in the spectrum, there is no need to solve the equation and only $\Delta_0(s_1)$, $\Delta_0(s_2)$, $\Delta_1(s_1)$, $\Delta_1(s_2)$ need to be computed for the given error pattern.

**Upper Bounds on $o_b$**

From input weight 2 error event analysis, we have the result that, given $N$, the larger the $o_b$, the larger the flexibility we have to choose good interleavers and the better the performance. But from simulations, we can observe that the best possible performance for permutation polynomial interleaver grows with $o_b$ up to a certain value. Then the performance degrades again. We can use Theorem 2.10 to find some simple upper bounds for $o_b$. 

41
Upper bound on $o_b$ from input weight 4 error events  We start from (2.25). When $m = 2$, assume condition (2.27) holds, then when $o_{c_0} \geq o_{c_1}$, if a solution of $x_1$ exists for (2.25), an error event exists for the given error pattern starting at the $x_1$.

One special case is that, for some error pattern, every $x_1$ from 0 to $N - 1$ is a solution. This will lead to a spectrum line with high multiplicity, which may dominate the performance even if it is not the minimum distance. Let us consider $t_1 = t_2 = t$ and $s_1 = s_2 = s$. Then $c_0$ and $c_1$ in (2.26) become $-2b\Delta_0(s)tD(K)$ and $-2b\Delta_1(s)tD(K)$ respectively. It is easy to see that $o_{c_1} \geq o_{c_0}$. All coefficients in $c_2(x_1)$ have a larger order than $o_{c_1}$. If $o_{c_0} = o_N$, (which contains the case that some elements in $o_{c_0}$ are larger than the corresponding elements in $o_N$ numerically), then (2.25) becomes an all-zero polynomial. Trivially, every possible $x_1$ is a solution to the polynomial.

This special case places an upper bound for $o_b$ such that we require that $o_N$ has at least one element larger than the corresponding element in $o_{c_0} = o_b + o_2 + o_s + o_t$.

An interesting special case exceeding this upper bound is $o_b = o_N$, which implies that $b = 0 \mod N$. The permutation polynomial reduces to a linear interleaver. Now the error event can be described as

$$a(t_1 - t_2) = s_1 - s_2.$$  \hfill (2.29)

It is easy to see that when $t_1 = t_2$ and $s_1 = s_2$, equation (2.29) trivially holds. Obviously every $x_1$ is a solution. This is exactly why a linear interleaver has many input weight four error events and can not work well at medium to long frame sizes.
Upper bound on $o_b$ from input weight 6 error events

For the input weight 6 error event case, we also start from (2.25). Assume the condition (2.28) holds, then

$$c_0 = \left[ 2b[t_3\Delta_0(s_2) - t_2\Delta_0(s_1)] + 2bt_3t_1\Delta_1(s_2) \
+ 2bt_1t_3 + b(t_1^2 - t_2^2 + t_3^2) \
+ a(t_1 - t_2 + t_3) - (s_1 - s_2 + s_3) \right]D, \tag{2.30}$$

$$c_1 = [2b(t_1 - t_2 + t_3) + 2b(t_3\Delta_1(s_2) - t_2\Delta_1(s_1))]D, \tag{2.31}$$

where $D = D(\max(K(s_1), K(s_2)))$, and the coefficients of $c_2(x_1)$ have higher order than $o_{c_1}$. We are still interested in the error pattern that every $x_1$ is a solution.

We discovered that the error patterns of the form $[2t, t, -t, s, -s, -2s]$ are critical. (Since we require $t_1 \geq |t_2|$ and $t_1 \geq |t_3|$ to avoid some double counting, many other critical error patterns are equivalent to this one). The error pattern that corresponds to the smallest Hamming distance is when $t = \tau$ and $s = \pm t$.

For this error pattern, we have $t_1 - t_2 + t_3 = s_1 - s_2 + s_3 = 0$. Then $c_0$ and $c_1$ reduce to

$$c_0 = \left[ -2bt[\Delta_0(-s) + \Delta_0(s)] - 4bt^2\Delta_1(-s) \right]D \tag{2.32}$$

$$c_1 = -2bt(\Delta_1(-s) + \Delta_1(s))D \tag{2.33}$$

In order to proceed, we need two lemmas.

**Lemma 2.11** $\Delta_0(-s) + \Delta_0(s)$ has order $o_b + o_2 + 2o_s$.

**Proof:** Let $P(y) = s$ and $P(x) = -s$, where $P(x) = bx^2 + ax$. By the definition of $\Delta_0(i)$, we know $y = \Delta_0(s)$ and $x = \Delta_0(-s)$. Define $c = y + x$, then $y = c - x$. Since $P(y) + P(x) = s + (-s) = 0$, we have

$$b(c - x)^2 + a(c - x) + bx^2 + ax = 0. \tag{2.34}$$
Rewriting it as a polynomial of $c$, we have

$$bc^2 + (-2bx + a)c + 2bx^2 = 0,$$  \hfill (2.35)

which is a permutation polynomial of $c$ and the solution for $c$ has the same order as term $2bx^2$. The order of $x$, or $\Delta_0(-s)$, is known by Theorem 2.8 to be $o_s$. Thus $c$ has an order $o_b + o_2 + 2o_s$.

\[\square\]

**Lemma 2.12** $\Delta_1(-s) + \Delta_1(s)$ has an order at least $o_b + 2o_2 + 2o_s$.

**Proof:** Assume we have $y_0, y_1, x_0$ and $x_1$ such that

$$P(y_0) = by_0^2 + ay_0 = s, \hfill (2.36)$$

$$P(y_1) - P(1) = by_1^2 + ay_1 - b - a = s, \hfill (2.37)$$

$$P(x_0) = bx_0^2 + ax_0 = -s, \hfill (2.38)$$

$$P(x_1) - P(1) = bx_1^2 + ax_1 - b - a = -s. \hfill (2.39)$$

Then $\Delta_1(s) = y_1 - y_0 - 1$ and $\Delta_1(-s) = x_1 - x_0 - 1$. Define $c \triangleq y_0 + x_0$ and $d = \Delta_1(s) + \Delta_1(-s) = y_1 + x_1 - c - 2$. Then $y_1 = d + c - x_1 + 2$. Applying this in (2.37) and (2.39), we have

$$bd^2 + (-2bx_1 + 2bc + 4b + a)d + (bc^2 - 2bc(x_1 - 1)$$

$$+ 2bc + 2b(x_1 - 1)^2 + ac) = 0. \hfill (2.40)$$

This is a permutation polynomial of $d$. From the previous lemma, we know $c$ has an order $o_b + o_2 + 2o_s$. Then in the constant term in (2.40), both $2b(x_1 - 1)^2$ and $ac$ have an order $o_b + o_2 + 2o_s$ and the other terms have higher orders. So the order of $d$ is at least $o_b + 2o_2 + 2o_s$. Stronger results may exist but this is sufficient for our current purpose.

44
Now we return to (2.32). The order of $c_0$ and $c_1$ can be found using the above two lemmas. $o_{c_0}$ is at least $2o_b + 2o_2 + 3o_r$ and $o_{c_1}$ is at least $2o_b + 3o_2 + 3o_r$. When each element in $o_N$ is not larger than the corresponding element in $2o_b + 2o_2 + 3o_r$, both $c_0$ and $c_1$ are zero modulo $N$ and all $x_1$’s are solutions to the equation. This can serve as another upper bound for $o_b$.

This upper bound in $o_b$ is not stringent. If the error pattern corresponds to a Hamming distance large enough, then even though the multiplicity is high, the effect in the performance can be low. As shown in (2.7) the Hamming distance of the error event is $18 + 6w_0$. For example, for the recursive 5/7 code, $w_0 = 2$. The Hamming distance is 30. For the recursive 23/35 code, $w_0 = 3$, the Hamming distance is 36 and for the recursive 37/23 code, $w = 8$ and the Hamming distance is 66. Sometimes, for a code with a large cycle length, this upper bound on $o_b$ can be relaxed.

**Range of $a$ and $b$ to search**

In order to search for good second degree permutation polynomials for the interleaver, we need to select the range of $a$ and $b$ to be searched. It is shown in Lemma 2.6 and Lemma 2.7 that, for input weight 2 error events, if $o_b$ is fixed, we can pick $b = b_0 = p^{o_b}_N$ and consider $a$ from 1 to $b_0$. Unfortunately, for general error events, we only have a similar result for the range of $a$.

**Theorem 2.13** For second degree permutation polynomial-based interleavers, we only need to consider $a$ in the range between 1 and $2b$.

**Proof**: Consider $P(x) = bx^2 + ax$ and $P'(x) = bx^2 + (a + 2b)x$. Assume we have an error event of input weight $m$ for the code using $P(x)$ based interleaver. The positions of
the errors are at \( x_1, x_2, \ldots, x_m \) in the first component code and \( P(x_1), P(x_2), \ldots, P(x_m) \) in the second component code. Consider the code based on \( P'(x) \). Assume there are errors at \( x_1 - 1, x_2 - 1, \ldots, x_m - 1 \) in the first component code. Then \( P'(x_i - 1) = b(x_i - 1)^2 + (a + 2b)(x_i - 1) = bx_i^2 + ax_i - (a + b) \). \( P'(x_i - 1) \) is a shift of \( P(x_i) \) by \( a + b \). Thus the same error event can be found in a code using both interleavers. The only difference is that the error positions are cyclicly shifted to the left by 1 in the first component code and by \( a + b \) in the second component code. The spectrums of the two codes are the same.

For input weight 4 error events, in general, we have to search all \( b \) and all \( 1 \leq a \leq 2b \) for given \( o_b \). This is tedious. However, it can be shown that under some conditions, we can still use the limited ranges of \( b \) given in Lemma 2.6. More precisely, we only need to consider \( b = b_0 = p_N^o \). This is equivalent to showing that, under some conditions, if we replace \( b \) by \( b_0 \), the same error pattern exists in both cases with the same multiplicity.

If we replace \( b \) by \( b_0 \) in (2.25), it becomes

\[
c_0' + c_1' x_1 + c_2'(x_1) = 0. \tag{2.41}
\]

Since \( \Delta_k(s) \)'s are functions of \( a \) and \( b \), in \( c_1' \) and \( c_0' \), \( \Delta_k'(s) \) will be used instead of \( \Delta_k(s) \). However, \( \Delta_k'(s) \) and \( \Delta_k(s) \) have the same order.

In order to show that the same error pattern exists in both cases, it is equivalent to showing that (2.25) and (2.41) have the same number of solutions for the error pattern. This is further equivalent to show that \( o_{c_1} = o_{c_1}' \) and \( o_{c_0} \geq o_{c_1} \iff o_{c_0}' \geq o_{c_1}' \). One sufficient
condition for this to happen is both

\( t_1 = t_2, \text{ or } \) \hspace{1cm} \( t_1 = t_2, \text{ or } \) \hspace{1cm} (2.42) \n
\( o_{t_1 - t_2} \ll o_b + o_2 + o_{s_1} + o_{t_2}, \) \hspace{1cm} (2.43) \n
and

\( s_1 = s_2, \text{ or } \) \hspace{1cm} (2.44) \n
\( o_{s_1 - s_2} \ll o_b + o_2 + o_{s_1} + o_{t_2}. \) \hspace{1cm} (2.45) \n
Inequality (2.43) and (2.45) are used to guarantee that the orders of \( c_0 \) and \( c_1 \) are dominated by \( s_1 - s_2 \) and \( t_1 - t_2 \) respectively when \( t_1 \neq t_2 \) and \( s_1 \neq s_2 \). In the following analysis, we assume that these conditions are satisfied.

When \( t_1 = t_2 \), applying this to (2.26), we have

\[ c_0 = (-2b\Delta_0(s_1)t_2 - (s_1 - s_2))D(K) \]

and

\[ c_1 = -2b\Delta_1(s_1)t_2D(K). \]

Then \( o_{c_1} = 2o_b + 2o_2 + o_{s_1} + o_{t_2}. \) If \( s_1 = s_2 \), \( o_{c_0} = o_b + o_2 + o_{s_1} + o_{t_2}. \) If (2.45) holds, \( o_{c_0} = o_{s_1 - s_2} \ll o_b + o_2 + o_{s_1} + o_{t_2}. \) In both these two cases, \( o_{c_0} \ll o_{c_1} \) and no solution exists for (2.25). Same result can be derived for (2.41).

When \( t_1 \neq t_2 \) and (2.43) holds, \( o_{c_1} = o_{c_1'} = o_b + o_2 + o_{t_1 - t_2}. \) Let us compute \( c_0 - c_0' \),

\[ c_0 - c_0' = -2(b\Delta_0(s_1) - b_0\Delta_0'(s_1))t_2D(K) \]

\[ + (b - b_0)(t_1 - t_2)(t_1 + t_2)D(K). \]

(2.46)

The two terms in (2.46) have orders at least \( o_b + 2o_2 + o_{s_1} + o_{t_2} \) and \( o_b + o_2 + o_{t_1 - t_2} \) respectively. They are both higher than the \( o_{c_1}. \) Since the difference between \( c_0 \) and \( c_0' \) has higher order than \( o_{c_1}, o_{c_0} \geq o_{c_1} \iff o_{c_0}' \geq o_{c_1}. \) Thus we proved that \( b \) can be limited to the ranges in Lemma 2.6 when conditions in (2.43) and (2.45) hold.

For input weight 6 error patterns, similar sufficient condition can be derived. However, the form of the sufficient condition becomes very complex and will therefore be omitted.
Although not all input weight 4 and 6 error patterns satisfy these sufficient conditions, it seems that most of the error patterns that correspond to the first few spectrum lines do satisfy. Furthermore, in our permutation polynomial search, it can be observed that extending to more general $b$ and $a$ usually cannot generate significantly better permutation polynomials. Therefore, in our search for the polynomials using input weight $2m$ error events spectrum, we will only consider $b = p_N^{o_b}$ and $1 \leq a < 2b$.

2.5 Results

Given the frame size $N$ and the component code, our search for good permutation polynomial-based interleavers is basically an enumeration of $a$ and $b$ for the polynomial. First we need to enumerate $o_b$. From the analysis in Section 2.4, $p_N^{o_b}$ should be relatively large but bounded by some constraints due to some special input weight 4 and 6 error events. Once we fix $o_b$, we let $b = p_N^{o_b}$ and enumerate all the $a$ is in the range specified in Theorem 2.13. In this section, we will present some examples for the polynomial search for different component codes.

We consider six different systematic recursive component codes shown in the first column of Table 2.2. Their corresponding cycle lengths are from 2 to 7 respectively. In this section, we will only consider frame sizes $N = 2^n$. Then $p_N = 2$ and all the orders are scalars.

For the frame size $N = 256$, the best permutation polynomials found for the turbo code using these component codes are shown in Table 2.2. The corresponding minimum distance of the input weight $2m$ error events and its multiplicity are also shown. Computer simulation results comparing with S-random interleaver and quadratic interleaver are shown in Figures 2.6-2.11.
<table>
<thead>
<tr>
<th>Component code</th>
<th>Cycle length ($\tau$)</th>
<th>Best poly.</th>
<th>$d_{min}$ (multiplicity)</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/5</td>
<td>2</td>
<td>$15x + 16x^2$</td>
<td>18(512)</td>
<td>2.6</td>
</tr>
<tr>
<td>5/7</td>
<td>3</td>
<td>$15x + 32x^2$</td>
<td>28(512)</td>
<td>2.7</td>
</tr>
<tr>
<td>37/21</td>
<td>4</td>
<td>$7x + 8x^2$</td>
<td>24(256)</td>
<td>2.8</td>
</tr>
<tr>
<td>21/37</td>
<td>5</td>
<td>$15x + 32x^2$</td>
<td>28(512)</td>
<td>2.9</td>
</tr>
<tr>
<td>37/25</td>
<td>6</td>
<td>$15x + 16x^2$</td>
<td>24(512)</td>
<td>2.10</td>
</tr>
<tr>
<td>23/35</td>
<td>7</td>
<td>$15x + 32x^2$</td>
<td>36(512)</td>
<td>2.11</td>
</tr>
</tbody>
</table>

Table 2.2: Best permutation polynomials for different component codes. Frame size 256.

Figure 2.6: TC with 7/5 code. Frame size 256.
Figure 2.7: TC with 5/7 code. Frame size 256.

Figure 2.8: TC with 37/21 code. Frame size 256.
Figure 2.9: TC with 21/37 code. Frame size 256.

Figure 2.10: TC with 37/25 code. Frame size 256.
In all these simulations, the permutation polynomial-based interleavers perform better than quadratic interleavers, which have a similar performance as the average of random interleavers. Comparing to S-random interleaver, the permutation polynomial-based interleavers also have a better performance for both bit error rate and frame error rate.

We also find the best permutation polynomial-based interleavers for turbo code with recursive 5/7 component codes at frame sizes 1024 and 16384. They are based on polynomials $P(x) = 31x + 64x^2$ and $P(x) = 15x + 32x^2$, respectively. The simulation results are shown in Figures 2.12 and 2.13. For very long frame sizes, the performance of permutation polynomial-based interleavers are not as good as the S-random interleavers but much better than quadratic interleavers.
Figure 2.12: TC with 5/7 code. Frame size 1024.

Figure 2.13: TC with 5/7 code. Frame size 16384.
2.6 Conclusions

In this chapter, we introduced permutation polynomial-based interleavers for turbo codes. This is a class of deterministic interleavers, and the interleaving (de-interleaving) can be done using a simple arithmetic computation instead of a table lookup. Given the parameters of the generator polynomial for the interleaver, we can use polynomial solving to find the minimum distance (or the first few spectrum lines) of an important subset of error events, which can be used to well approximate the real minimum distance. Based on the approximated minimum distance, a limited search for good interleavers has been performed. The new interleaver is compared with the S-random interleaver and the quadratic interleaver. For short frame sizes, good interleavers that outperform the S-random interleavers have been found. For long frame sizes, the permutation polynomial-based interleavers have a performance close to the S-random interleavers. The new interleavers outperform quadratic interleavers (which are known to have a similar performance as random interleavers) for all frame sizes.
3.1 Introduction

In this chapter, we focus on the construction of LDPC codes.

There are many ways to construct LDPC codes. MacKay et al. proposed some constructions concatenating small random permutation matrices to form parity check matrices of regular LDPC codes [27] and irregular LDPC codes [26]. In [7], the Bit-Filling method is proposed, which basically generates the columns of the parity check matrix randomly under some constraints. The construction of LDPC codes based on finite geometry is investigated in [23]. The construction of regular LDPC codes using combinatorial designs is considered in [21].

As mentioned in Chapter 1, all LDPC codes can be defined as a bipartite graph as in Figure 1.5. Given the degree distribution, the construction of the LDPC code becomes the construction of the interleaver \( y = \pi(x) \). In [33], a deterministic construction of regular LDPC codes is proposed. The construction uses maximal-length linear congruential sequences (MLLCS) defined as

\[
A_{n+1} = (aA_n + b) \mod M, \quad 0 \leq n \leq M - 2, \quad (3.1)
\]
where \(a, b\) and \(A_0\) are integer parameters in the range of \(0, 1, \ldots, M - 1\). It is shown in [22] that an MLLCS can be formed if \(b\) is relatively prime to \(M\), \((a - 1)\) is a multiple of \(p\) for every prime \(p\) dividing \(M\), and if \(M\) is a multiple of 4, \((a - 1)\) is also a multiple of 4. Given an MLLCS \(\{A_n\}\), the interleaver required in the LDPC code construction can be formed by using \(\pi(n) = A_n\) for \(n = 0, 1, \ldots, M - 1\).

The main result of [33] is a sufficient condition for a regular LDPC code with \(d_\lambda = 3\) to have \(girth > 4\), where girth is defined as the length of the smallest loop in a graph.

In this chapter, we develop two sufficient conditions on the girth of irregular LDPC codes using MLLCS. Regular LDPC codes are included as special cases. When applied to data node degree 3 regular LDPC codes, the first sufficient condition is similar to the one in [33], but has a much simpler proof. The second sufficient condition is tighter and can be used to make the girth even larger.

The chapter is organized as follows: In Section 3.2, MLLCS-based construction of LDPC code is analyzed. Some propositions to derive sufficient conditions are formally stated. In Section 3.3, sufficient conditions for different girth are developed and design examples are given. Finally, we draw our conclusions in Section 3.4.

3.2 Loops in MLLCS-Based LDPC Codes

An MLLCS is defined as (3.1). However, after some simple manipulations, for the non-trivial case with \(a \neq 1\), we can write \(A_n\) as

\[
A_n = \frac{a^{n+n_0} - 1}{a - 1} b \mod M, \tag{3.2}
\]

where \(n_0\) is uniquely decided by \(A_0\).

Our target is to select parameters such that no length \(2m\) loop exists for small \(m\). Before we proceed, it is useful to state some sufficient conditions to check if a loop of length \(2m\) is
not formed. First, for an LDPC code with degree distribution as in (1.1), two branches with
check node side labels $x_1$ and $y_1$ are connected to distinct check nodes if $|x_1 - y_1| \geq d_\rho$. For
the length 4 loop case, if $x_1, x_2, y_1$ and $y_2$ are check node side labels and branches $x_1$ and $y_1$
are connected to a given check node (or at least, $|x_1 - y_1| \leq d_\rho - 1$), then the other pair of
branches $x_2$ and $y_2$ are connected to different check nodes if $|(y_2 - x_2) + (y_1 - x_1)| \geq 2d_\rho - 1$
or $|(y_2 - x_2) - (y_1 - x_1)| \geq 2d_\rho - 1$. These sufficient conditions can be further extended to
larger loops and similar sufficient conditions can be derived when $x_i$ and $y_i$ are data node
side labels. These statements can be proved easily using triangular inequality.

In the following, we define $a_0 = \gcd(M, a - 1)$. Then $a - 1 = a_1 a_0$, and $a_1$ is co-prime
to $M$. We will also frequently use the binomial identity $(x + 1)^n = \sum_{i=0}^n \binom{n}{i} x^i$. If not
specified, all the computations are modulo $M$.

### 3.3 Sufficient conditions for $girth > 2m$

In this section, we develop two sufficient conditions for the girth of an LDPC code. The
proofs of these results are given in the Appendix.

Using the representation of the MLLCS in (3.2), we can develop sufficient condition for $girth > 4$.

**Theorem 3.1** An MLLCS-based LDPC code with degree distribution (1.1) have girth $> 4$
if (1). $2d_\rho - 1 \leq (b\Delta \mod a_0) \leq a_0 - (2d_\rho - 1)$, where $a_0 = \gcd(M, a - 1)$, for
$\Delta = 1, 2, \ldots, 2(d_\lambda - 1)$, and (2). for any $c = 1, 2, \ldots, d_\rho - 1$, $(a^l - 1)c \neq 0 \mod M$ for
all $l = 1, 2, \ldots, d_\lambda - 1$.

We have an even tighter sufficient condition to guarantee that no length $2m$ loop exists.

**Theorem 3.2** An MLLCS-based LDPC code with degree distribution (1.1) has no length
$2m$ loop if for all $t_i$ and $s_i$, (1). if $a_{\Sigma_1(t)} = a_{\Sigma_2(t)}$, $-a_{\Sigma_1(t)} - a_{\Sigma_2(t)} + b + s_1a_{\Sigma_2(t)} - s_2a_{\Sigma_3(t)} + \ldots$
For regular LDPC codes, Theorem 3.1 can be tightened using the regularity of the code.

**Theorem 3.3** For a regular \((d_\lambda, d_\rho)\) MLLCS-based LDPC code with even \(\frac{M}{a_0}\), the condition (1) in Theorem 3.1 can be reduced to checking \(\Delta = 1, 2, \ldots, 2d_\lambda - 3\) instead of \(\Delta = 1, 2, \ldots, 2d_\lambda - 2\).

It can be easily verified that if Theorem 3.3 is applied to the regular \((3, d_\rho)\) LDPC code, it contains the sufficient condition in [33].

Using these theorems, we can find MLLCS satisfies the given girth requirement for a given frame size. For example, for a rate 1/2 irregular LDPC code with \(M = 7000\) and degree distribution 
\[ \lambda(x) = 0.5x^2 + 0.5x^3, \quad \rho(x) = x^6, \]
we find that for \((a, b) = (281, 27)\), the corresponding graph has girth 6. For a regular \((3, 6)\) LDPC code with \(M = 6144\), the girth is 6 for \((a, b) = (97, 41)\). Comparing to the sufficient condition in [33], our sufficient conditions usually provides more candidate MLLCS, which leaves more space for further optimizations. In some cases, our sufficient condition can still find some valid MLLCS, while the sufficient condition in [33] can not.

Good MLLCS-based interleavers are not always possible for all \(M\). Let \(M = \prod_{i=1}^{n} p_i^{k_i}\), where \(p_i\) are distinct prime numbers. From the conditions for a linear congruential sequence to be maximum length, one can see that \(a_0 = \gcd(M, a - 1)\) divides \(M\) and \(\prod_{i=1}^{n} p_i \leq a_0 < M\). Intuitively, when \(\frac{a_0}{\prod_{i=1}^{n} p_i}\) or \(\frac{M}{a_0}\) is too small, the regularity of the MLLCS-based interleaver is too strong and the performance of the induced LDPC code
is bad. Therefore, we prefer to have $M$ that can be decomposed into many prime factors (counting multiplicity) and at least one of the prime factors has relatively high multiplicity. The choice of $a_0$ needs to guarantee that both $\frac{a_0}{\prod_{i=1}^n p_i}$ and $\frac{M}{a_0}$ are relatively large.

3.4 Conclusions

In this chapter, we follow [33] and use maximum length linear congruential sequences to construct LDPC codes. We develop sufficient conditions on the parameters of MLLCS such that the girth of the corresponding graph is larger than 4. Contrary to [33], where similar sufficient conditions for regular LDPC codes were limited to data node degree 3, our sufficient conditions can be applied to general irregular LDPC codes, and as a particular case, regular LDPC codes with arbitrary data node degrees. The proofs for our sufficient conditions are also more straightforward than the proof in [33].
CHAPTER 4

ANALYSIS OF TRAPPING SETS IN LDPC CODES USING APPROXIMATED LINEAR SYSTEM MODELS

4.1 Introduction

The practical decoder for LDPC code is the iterative decoder. When the underlying graph of the code does not contain loops, the iterative decoder converges to the ML result. Since all good LDPC codes have loops, the iterative decoder is suboptimum. However, it has been demonstrated that iterative decoder performs almost as good as an optimum decoder.

Even though the decoding complexity of an iterative decoder is low, the analysis of this decoder is difficult. The existing analysis methods can be separated into two cases: the infinite frame size analysis and finite frame size analysis. The infinite frame size analysis can predict the threshold of the code. However, for small to intermediate frame size applications, the threshold is not good enough. We also want to estimate the position of the error floor, i.e., the high SNR performance of the code.

For the special case of simple binary erasure channel, in [12], the concept of stopping set is developed. The stopping set is defined as a set of data nodes whose spanned subgraph does not contain any check node with degree one. It can be shown that the iterative decoder
will stop at the unique maximum stopping set covered by the erasures. A combinatorial method is developed to find the distribution of stopping sets for some regular LDPC code ensembles. The technique is extended to general regular LDPC codes and irregular LDPC codes in [50] and [30] respectively.

In [35], Richardson extended the idea of stopping set to *trapping set* in general binary symmetric channel. A trapping set is defined as the collection of the data nodes that can not always be decoded to the correct value after any given number of iterations. From this definition, the data nodes in the trapping set can be either always decoded wrong after enough iterations or the hard decisions of these bits form some kind of oscillation.

Richardson’s method is basically a technique to estimate the error floor of the performance curve. First, the trapping sets are enumerated. For each trapping set, the probability that it is triggered is heuristically associated with a parameter $s$ defined as the average of the log-likelihood ratios of the channel observations at the data nodes in the trapping set. The association is based on the intuition that the larger the error in these log-likelihood ratios, the more likely the trapping set can be triggered. It is not convenient to associate the triggering probability with all log-likelihood ratios. Therefore, the average value $s$ is used instead. Given $s$, the conditional probability that the trapping set is triggered is found using inline simulation. Then the unconditional triggering probability can be found by averaging the conditional probability over the distribution of $s$, which is a Gaussian.

Richardson’s method is a computation technique relies heavily on simulations. Even though the position of the error floor can be predicted, there still lacks a link between the structure of the trapping set and the corresponding component in the error floor.

In this chapter, we are trying to develop such a link. More precisely, given the structure of a trapping set, we want to estimate its triggering probability at high SNR. To simplify
the problem, some approximations are used to linearize the computations within the trapping set. Then the messages in the trapping set are separated into zero input response and zero state response. The distributions of these two responses can be tracked analytically using Gaussian models. Then the triggering probability is associated with the tail of the distribution of the summation of these two responses.

The trapping sets can be separated into two types. For the first type, all data nodes involved in the trapping set are either in loops or in paths connecting different loops. The second type of trapping set is an extension of the first one. In these trapping sets, we also have data nodes that are not in any loops but are connected to loops from only one of its outgoing branches. These two types of trapping sets are analyzed separately.

The chapter is organized as follows. The trapping set is analyzed in Section 4.2. The simple trapping sets with all data nodes in loops or on paths connecting different loops are analyzed in Section 4.3. The more complex trapping sets are discussed in Sections 4.4 and 4.5. Analysis results are compared with simulation result in Section 4.6. Finally, we draw our conclusions in Section 4.7.

### 4.2 Analysis of a Trapping Set

It was observed in [25] that when an iterative decoder is used, the error floor of LDPC codes is usually not caused by small weight error events, but instead by trapping sets (TS), a concept introduced in [35]. As mentioned in the previous section, a trapping set is defined to be a set of data nodes that has non-zero probability to be not correctable by the given decoder. It is clear that the trapping set is decoder dependent. For an ML decoder, trapping sets are simply error events. In this chapter, we are only interested in the typical iterative decoder.
In this chapter, we estimate the probability that a given trapping set is triggered by tracking the messages in the decoder. Before doing that, we need a few definitions. We define the graph spanned by a trapping set, $G$, to be the graph formed by the collection of the data nodes in the trapping set, all branches from these data nodes, and the check nodes that those branches are connected to. The graph $G$ is a subgraph of the bipartite graph for the code. For a check node in $G$, some of its branches are connected to data nodes in $G$. We call those branches inner branches. The remaining branches are called outer branches. For data nodes in the trapping set, inner branches and outer branches are defined as branches that are connected to inner degree 2 and inner degree 1 check nodes respectively, where inner degree and outer degree of a check node are defined as the number of its inner and outer branches. As we will see, loops play an important part in forming trapping sets. Given $G$, we define its subgraph formed by branches either in a loop or in a path connecting different loops to be the root of $G$. We denote this subgraph by $R$. We call all data nodes that are in $G$ but not in $R$ external data nodes. One important property of an external data node is that there is a unique path in $G$ that connects it to $R$. We define a simple trapping set to be a trapping set without external data nodes. It can be seen that a general trapping set can be reduced to a simple trapping set by removing all the external data nodes and their connected branches.

We need a few assumptions to make the analysis tractable.

**Assumption 4.1** We assume that in $G$, all check nodes are of inner degrees either 1 or 2.

It is noted in [35] that most of the observed trapping sets are of this type.

**Assumption 4.2** We assume an infinite frame size outside the trapping set.

63
Trapping set analysis is basically a finite frame size analysis. Otherwise, if an infinite frame size is assumed, the girth of the LDPC code would go to infinity and a trapping set would not exist. However, if we let the frame size to be finite, the complex structure of the code needs to be considered. To simplify the problem, we use a combination of these two models, the trapping set is kept and the part of the graph outside the trapping set is assumed to be loop free (infinite frame size). Under this assumption, the density evolution method [37] applies on the messages on branches outside the trapping set. Since we are interested in the error floor area, at high SNR, these messages go to infinity as iteration goes on. Furthermore, the input messages to the trapping set can be assumed to be independent and they are also independent to the messages within the trapping set.

In order to track the messages within $G$, we have to investigate the computation rules at both the data node side and the check node side. At the data node side, the outgoing message at a branch is the summation of the incoming messages on all other branches and the local observation. The computation is linear and simple. At the check node side, the computation is much more complex. The outgoing message at branch $i$ must satisfy

$$\tanh(O_i/2) = \prod_{j \neq i} \tanh(I_j/2),$$

where $I_j$ and $O_j$ are the incoming and outgoing messages on branch $j$ respectively [37].

It is hard to analyze the message evolution due to the complexity of this computation. We need to simplify the problem using some more assumptions.

**Assumption 4.3** *We assume the absolute values of the messages to be large.*

This assumption is valid when the channel SNR is high enough, or the iteration number is large.
By definition, 
\[ \tanh(x/2) = \frac{e^{x} - 1}{e^{x} + 1}. \] (4.2)

When \( e^{x} \gg 1 \) (or \( x \gg 0 \)), 
\[ \frac{e^{x} - 1}{e^{x} + 1} = 1 - 2e^{-x} + O(e^{-2x}). \] (4.3)

When \( 0 < e^{x} \ll 1 \) (or \( x \ll 0 \)), 
\[ \frac{e^{x} - 1}{e^{x} + 1} = -1 + 2e^{x} - O(e^{2x}). \] (4.4)

Using these approximations, for a message \( M \), if \( M \gg 0 \) or \( M \ll 0 \), we have 
\[ \tanh(M/2) = \text{sgn}(M)(1 - 2e^{-|M|} + O(e^{-2|M|})). \] (4.5)

Using this in the check node computation, we have 
\[ \text{sgn}(O_{i}) = \prod_{j \neq i} \text{sgn}(I_{j}) \] 
\[ (1 - 2e^{-|O_{i}|} + O(e^{-2|O_{i}|})) = \prod_{j \neq i} (1 - 2e^{-|I_{j}|} + O(e^{-2|I_{j}|})). \] (4.6)

With the assumption that \( |I_{j}| \) and \( |O_{i}| \) are large, and by ignoring the high order terms, the second equation in (4.6) can be simplified to 
\[ e^{-|O_{i}|} \approx \sum_{j \neq i} e^{-|I_{j}|}. \] (4.7)

Define \( |I|_{\text{min}} = \min_{j \neq i} |I_{j}| \), then (4.7) can be converted to 
\[ |O_{i}| = |I|_{\text{min}} - \log(\sum_{j \neq i} e^{-|I_{j}|} + |I|_{\text{min}}). \] (4.8)

The absolute value of \( O_{i} \) is dominated by the incoming message with the smallest absolute value, especially when the absolute values of other incoming messages are considerably larger than the minimum one. We can find bounds of \( |O_{i}| \), which are 
\[ |I|_{\text{min}} \geq |O_{i}| \geq |I|_{\text{min}} - \log(K - 1), \] (4.9)
where $K$ is the degree of the check node. The output $|O_i|$ achieves the upper bound when all $|I_j|, j \neq i$, goes to infinity. The lower bound is achieved when all incoming messages have the same absolute value. Since we assume $|I_j|$'s are large and $K$ is limited, one can see that $|I|_{\min}$ is a good approximation to $|O_i|$. This approximation is also used in [19] to reduce the computation complexity. However, we just use this approximation for the derivation of our analysis model. It is not used for simulations.

Now, the computation in the check node is much simpler. However, it is still not linear due to the "take the minimum" operation. To further simplify that, we will make use of another two assumptions.

**Assumption 4.4** For an inner degree 1 check node, instead of taking the minimum of all the messages from the outer branches, we randomly pick one of the messages.

**Assumption 4.5** For an inner degree 2 check node, the outgoing message on one of the inner branches is set to be the incoming message from the other inner branch.

For the inner degree 1 check node, by the infinite frame size assumption, all messages from the outer branches have the same distribution. Since these messages go to infinity, the introduced error will be small. For the inner degree 2 check node, the assumption is valid when all incoming outer messages are positive and they are (much) larger than the magnitude of the incoming message from the other inner branch.

We will see that the Assumption 4.5 is not always valid for all the LDPC codes. For some degree distributions, a large error will be introduced in the error floor predicted. Since we can think of this assumption as a suboptimal computation rule in the check node, the performance of the resulting decoder should be worse than the suboptimum iterative
decoder. Therefore, the error floor predicted using this assumption is an approximated upper bound (can be loose) of the true error floor.

With the above simplifications, we finally have a linear system. The system can be modeled as follows. For a trapping set with graph $G$, define $M$ and $N$ to be the numbers of branches connected to inner degree 2 and 1 check nodes respectively. Then by Assumption 4.1, the total number of branches in the $G$ is $M + N$. Define an $M \times 1$ vector $r$ to be the check-to-data message from inner degree 2 check nodes to data nodes. Similarly, define an $N \times 1$ vector $o$ to be the check-to-data message from inner degree 1 check nodes to data nodes. Define an $M \times 1$ vector $l$ to be the data-to-check message from data nodes in the trapping set to inner degree 2 check nodes. We are not interested in the message from data nodes to check nodes with inner degree 1 for these messages will not be fed back to the trapping set due to Assumption 4.2. Define $B_r$ to be the $M \times M$ matrix of the linear transform to compute $l$ from $r$. Define $B_o$ to be the $M \times N$ matrix of the linear transform to compute $l$ from $o$. Define $F$ to be the linear transform to compute $r$ from $l$. Since all check nodes involved in $F$ have inner degree 2, it is simply a permutation matrix. Define an $M \times 1$ vector $c$ to be the channel observation component in the data-to-check message. Since the channel observation at one data node appears in all its output messages, elements in $c$ corresponding to the same data node are of the same value.

Using these definitions, the system can be described by the following recursive form.

$$\begin{align*}
l^{(k)}(t) &= B_r r^{(k-1)} + B_o o^{(k-1)} + c \\
r^{(k)}(t) &= F l^{(k)},
\end{align*}$$

(4.10)

where the superscripts are the iteration numbers. The first equation describes the computation at the data nodes side and the second equation is for the check node side.
We can expand the recursive form in (4.10) and have the general form as in the following equation.

\[
\begin{align*}
1^{(K)} &= \left[ \sum_{k=1}^{K} A^{k-1} \right] c + \sum_{k=1}^{K-1} A^{K-1-k} B_o o^{(k)} \\
r^{(k)} &= F 1^{(k)}.
\end{align*}
\] (4.11)

where we define \( A = B_r F \) to simplify the equation.

A trapping set with three degree 3 data nodes is shown in Figure 4.1. The labels in the figure are the indexes of the branches in vectors \( I \) and \( r \) respectively. For this example,

\[
\begin{align*}
B_r &= \begin{bmatrix}
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 
\end{bmatrix}, \\
B_o &= \begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
0 \\
0 \\
0 \\
0 
\end{bmatrix}, \\
F &= \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 
\end{bmatrix}.
\] (4.12)

Figure 4.1: A trapping set with no external data node

\( M = 8, N = 1 \). Matrices \( B_r, B_o \) and \( F \) are
and $A$ is

$$A = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 
\end{bmatrix}, \quad (4.13)$$

Equation (4.11) can be shown as a linear feedback system as in Figure 4.2. In the figure, we only show the computation for $l^{(k)}$. The message in the other direction $r^{(k)}$ is just a permutation of $l^{(k)}$ and we are not interested in it.

![Diagram](image)

**Figure 4.2:** A linear system model for messages in the trapping set

Since this model is linear, we can separate $l^{(k)}$ into two parts, the zero input response $l_{zi}^{(k)}$ and the zero state response $l_{zs}^{(k)}$.

$$l^{(k)} = l_{zi}^{(k)} + l_{zs}^{(k)} \quad (4.14)$$

and

$$l_{zi}^{(K)} = \sum_{k=1}^{K} A^{k-1} c, \quad (4.15)$$

$$l_{zs}^{(K)} = \sum_{k=1}^{K-1} A^{K-1-k} B o^{(k)}. \quad (4.16)$$
To further investigate the zero input response and the zero state response, we first consider simple trapping sets and then extend the results to the general trapping set cases. We will discuss these two types of trapping sets in the next two sections.

4.3 Simple Trapping Sets

In this section, we analyze simple trapping sets with no external data node. By definition, all data nodes in these trapping sets are either in a loop or in a path that connects different loops. One example is shown in Figure 4.1. These trapping sets are the basis of more general trapping sets with external data nodes, for if we remove all the external data nodes and the branches they are connected to in any given trapping set, it can be reduced to a trapping set in this category.

In this section, we will consider the zero input response and the zero state response separately first and then combine them to form the total response. Since both responses are closely related to the eigen-decomposition of matrix $A$ in (4.11), we will investigate the properties of $A$ first.

4.3.1 Eigen-Decomposition of Matrix $A$

We only consider the case that the Jordan canonical form of $A$ is a diagonal matrix. (This is the only case we observed). Then the eigen-decomposition of $A$ can be written as

$$A = V\Lambda V^{-1},$$

where $V$ is a full rank matrix and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_M)$ is diagonal. Without loss of generality, we assume $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_M|$. Even though $A$ is a real matrix with elements either 0 or 1, matrices $V$ and $\Lambda$ are complex in general. Let $v_i$ be the $i$’th column of $V$ and let $u_i$ be the $i$’th row of $V^{-1}$. By definition, $v_i$ and $u_i$ are the (right-) eigenvector
and left-eigenvector corresponding to the eigenvalue $\lambda_i$ respectively. Then (4.17) can be further written as

$$A = \sum_{i=1}^{M} \lambda_i v_i u_i.$$  

(4.18)

One important property we will use is

$$A^k = \sum_{i=1}^{M} \lambda_i^k v_i u_i,$$  

(4.19)

where $k \geq 1$.

One can see that (4.19) is dominated by the terms corresponding to the eigenvalues with the largest amplitude. We call these eigenvalues *dominant eigenvalues*. The corresponding eigenvectors are called *dominant eigenvectors*.

There are a few properties for the eigen-decomposition of $A$. They are listed in the following lemmas.

**Lemma 4.1** An eigenvalue of $A$ is real if and only if the corresponding eigenvector (left-eigenvector) is real.

**Lemma 4.2** If $\lambda_i$ and $\lambda_j$ are complex conjugates of each other, then $(v_i, v_j)$ and $(u_i, u_j)$ form two complex conjugate pairs.

These properties are based on the fact that $A$ is a real matrix.

Since $A$ is non-negative, the Perron-Frobenius Theory in matrix analysis can be applied. Some useful results are repeated here. The detailed proofs can be found in [29].

A matrix $A$ is defined to be non-negative ($A \geq 0$) if and only if all entries in $A$ are non-negative. Similarly, we can define $A$ to be positive ($A > 0$) if all entries in $A$ are positive. Similar definitions can be applied to vectors.

When $A > 0$, the Perron’s Theorem applies.
Theorem 4.3 (Perron’s Theorem) For a positive (square) matrix $A$, there is a unique dominant eigenvalue, which is real and positive. The corresponding eigenvector is also positive.

The non-negative matrices can be separated into two classes. The reducible matrix $A$ is defined to be a matrix such that there exists a permutation matrix $P$ such that

$$P^T AP = \begin{bmatrix} X & Y \\ 0 & Z \end{bmatrix},$$

(4.20)

where $X$ and $Z$ are square matrices. Other non-negative matrices are called irreducible. It can be shown that the irreducible matrix has the following graph-based interpretation. Given a directed graph with $M$ nodes. Define the associated adjacency matrix to be an $M \times M$ matrix such that the $(i, j)$ entry is 1 if and only if there is a directed branch from node $j$ to node $i$ in the graph. Then the adjacency matrix is irreducible if and only if for any pair of nodes $(s, e)$, there exists a directed path from $s$ to $e$.

For irreducible matrices, we have the following result.

Theorem 4.4 (Perron-Frobenius Theorem) If $A \geq 0$ is irreducible, then there exists a positive dominant eigenvalue with multiplicity 1. The corresponding eigenvector is also positive.

The difference between irreducible matrices and positive matrices is that, for the irreducible case, we can have multiple dominant eigenvalues. However, similar to the positive case, we have a unique positive dominant eigenvalue with positive corresponding eigenvector.

Irreducible matrices can be further separated into two classes according to the number of dominant eigenvalues. The irreducible matrices with only one dominant eigenvalue is called primitive. Otherwise, if $h > 1$ dominant eigenvalues exist, the matrices are called imprimitive.
Primitive matrices can be associated to the positive matrices as in the following theorem.

**Theorem 4.5 (Frobenius's Test for Primitivity)** $A \geq 0$ is primitive if and only if $A^k > 0$ for some $k$.

The $h > 1$ dominant eigenvalues for imprimitive matrix are known to satisfy the following relationship.

**Theorem 4.6** For an imprimitive irreducible matrix $A$ with $h$ dominant eigenvalues, these $h$ dominant eigenvalues are uniformly located on a circle in the complex plane. More precisely, they are $|\lambda|, |\lambda|e^{2\pi i/h}, \ldots, |\lambda|e^{2\pi(i-1)/h}$, where $|\lambda|$ is the magnitude of the dominant eigenvalues.

For general non-negative matrices, the following result holds.

**Theorem 4.7** When $A$ is nonnegative, it possesses a non-negative dominant eigenvalue $\lambda$ belonging to a non-negative eigenvector $\mathbf{v} \geq 0$, but there may exist other eigenvalues of the same amplitude.

Note that here we can only guarantee the existence of non-negative dominant eigenvalue (not positive) and the corresponding eigenvector is also non-negative (not positive).

In order to further investigate properties of $A$, we need to develop a physical interpretation for it. By the definition of $B_r$ and $F$, the matrix $A = B_rF$ can be interpreted as follows. We have an entry $A_{i,j} = 1$ if and only if for the computation of the data-to-check message on branch $i$, the data-to-check message on branch $j$ in the previous iteration is used. Furthermore, for matrix $A^k$, we have an entry $A^k_{i,j} = 1$ if and only if the data-to-check message on branch $j$ is used in the computation of the data-to-check message on
branch $i$ after $k$ iterations. For this situation, we say that branch $j$ can reach branch $i$ through a path of length $k$, where the length is counted on the graph formed by using $A$ as the adjacency matrix. For the special case that $i = j$, the path is a loop of length $k$.

Using this interpretation of $A^k$, we can show the following result.

**Theorem 4.8** For a simple trapping set, the $A$ matrix is either irreducible, or is equivalent to a block diagonal matrix (under relabeling of branches), in which each diagonal block is irreducible.

Proof: It is equivalent to show that, if a branch $i$ can reach branch $j$ (in a certain number of steps), then branch $j$ can also reach branch $i$. By definition, this is true when the matrix is irreducible. When the matrix is equivalent to a block-diagonal matrix with each block on the diagonal to be irreducible, within each block, any two branches can reach each other. Since it is block-diagonal, no two branches from two different blocks can reach each other. On the other hand, if two branches can reach each other, we can relabel the branches to put them in the same block. In this way, we either have a block-diagonal matrix, or a irreducible matrix.

For a simple trapping set, the branches described in $A$ are exactly the branches in the root of the trapping set. More precisely, all these branches are either in a loop, or on a path connecting different loops. We can show that such an $A$ matrix satisfies our condition above. Let us denote by $c_j$ the check node that the branch $j$ is connected to, and assume that branch $i$ can reach branch $j$. We can remove both branches $i$ and $j$ from $\mathcal{G}$ to form a subgraph $\mathcal{G}'$. Denote by $d_i$ the data node that the branch $i$ is connected to. If in $\mathcal{G}'$, there is a path that connects $c_j$ and $d_i$, then branch $j$ can reach branch $i$ through the path. On the other hand, if such path does not exist, then $c_j$ and $d_i$ must be in two disconnected parts of $\mathcal{G}'$. Denote these two parts by $\mathcal{G}_j$ and $\mathcal{G}_i$ respectively. Since the trapping set does not
contain external data nodes, \( G_i \) and \( G_j \) must contain loops. Then there must be a path in \( G_j \) that starts and ends at \( c_j \), and there must be a path in \( G_i \) that starts and ends at \( d_i \). By combining these two paths and a reversed copy of the original path from branch \( i \) to branch \( j \), we can construct a path from branch \( j \) to branch \( i \). Therefore, we proved that the \( A \) matrix for a trapping set without external data nodes satisfies the condition that a branch \( i \) can reach a branch \( j \) if and only if the branch \( j \) reaches the branch \( i \).

\[ \Box \]

For the case that \( A \) is equivalent to a block-diagonal matrix, the eigenvalues of \( A \) are the collection of eigenvalues of each block. The eigenvector of \( A \) can be formed by adding zero entries to the eigenvectors of each block. Therefore, we can separate the problem and consider each irreducible block independently.

### 4.3.2 Zero Input Response

In this section, we mainly discuss the zero input response in (4.15). Using the eigen-decomposition in (4.15), we have

\[
1^{(K)}_{zi} = V(\sum_{k=1}^{K} \Lambda^{k-1})V^{-1}c.
\]  

(4.21)

Since \( \Lambda^k = \text{diag}(\lambda^k_1, \lambda^k_2, \ldots, \lambda^k_M) \), equation (4.21) can be written as

\[
1^{(K)}_{zi} = \sum_{i=1}^{M} c'(i)(\sum_{k=0}^{K-1} \lambda^k_i)v_i,
\]  

(4.22)

where \( c'(i) = u_i c \).

By observing (4.22), we see that \( 1^{(K)}_{zi} \) is dominated by the terms corresponding to the dominant eigenvalues.

The channel observation \( c \) is associated with the zero input response (and further with the total response) through \( c'(i) = u_i c \). Note that this is different from Richardson’s
method [35], where the average of the channel observations on the data nodes in the trapping set is used. We believe that averaging can be regarded as a heuristic approximation to $c'(i)$.

We can find the lower and upper bounds on the magnitude of eigenvalues. To show this result, we need to use the physical interpretation of $A^k$ developed in the previous subsection. According to this interpretation, the number of 1’s in row $i$ of $A$ is the inner degree of the corresponding data node minus one. In other words, for all the $K$ branches from a given data node of degree $K$, in the $K$ corresponding rows of $A$, the row weights are $K - 1$. Furthermore, all the 1’s in these $K$ rows are located in $K$ columns, each with column weight also $K - 1$. With proper column and row permutations, the matrix $A$ can be transformed into a block diagonal form $A'$, with each block corresponding to a data node. For a degree $K$ data node, the corresponding block on the diagonal of $A'$ is a $K \times K$ matrix with 0 on the diagonal and 1 on all off-diagonal positions. All these results can be verified for the example in Figure 4.1. For this example, $A$ can be column permuted to form

$$A' = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$  \hspace{1cm} (4.23)

A lower bound on the eigenvalues of $A$ for the simple trapping set is summarized in the following theorem.

**Theorem 4.9** For a simple trapping set, all eigenvalues have magnitudes at least one.

Proof: By definition, $Av = \lambda v$, with $||v|| = 1$. Then $||Av|| = ||\lambda v|| = |\lambda|$. To show that $|\lambda| \geq 1$, it is enough to show that $||Av|| \geq 1$ for any eigenvector $v$. It is further enough
to show that $||Av|| \geq ||v||$ for any vector $v$, which is proved in Lemma 4.10 using the structure of $A$ and $A'$.

\[\square\]

**Lemma 4.10** For a simple trapping set, $||Av|| \geq ||v||$ for any vector $v$.

The proof of this Lemma is given in the appendix.

Theorem 4.9 provides a lower bound on the magnitudes of the eigenvalues of $A$ for the simple trapping set. We also have an upper bound for the magnitude using the following theorem.

**Theorem 4.11 (Gerschgorin Circles)** The eigenvalues of an $n \times n$ complex matrix $A$ are contained in the union $G_r$ of the $n$ Gerschgorin circles defined by $|z - A_{i,i}| = r_i$, where $r_i = \sum_{j=1, j \neq i}^{n} |A_{i,j}|$ for $i = 1, 2, \ldots, n$.

Note that in our $A$ matrix, the diagonal elements are always 0. Also the maximum row weight in $A$ is $D - 1$ where $D$ is the maximum data node degree in the root of the simple trapping set. Therefore, an upper bound of the magnitudes of eigenvalues of $A$ is $D - 1$.

Combining the results of Theorem 4.9 and Theorem 4.11, we know that all eigenvalues satisfy $1 \leq |\lambda_i| \leq D - 1$.

Given the range of the eigenvalue, we want to find how the zero input response is related to the eigenvalues. Define the coefficient of the $i$'th eigenvector term in (4.22) to be $R_{zi}(i)$.

Then

$$R_{zi}(i) = c'(i)(\sum_{k=0}^{K-1} \lambda_i^k) = \begin{cases} c'(i) \frac{\lambda_i^K - 1}{\lambda_i - 1}, & \text{if } \lambda_i \neq 1 \\ c'(i)K, & \text{if } \lambda_i = 1 \end{cases} \quad (4.24)$$
and it is a scaled version of the Gaussian random variable \( c'(i) \) in both cases. Then, as \( K \) goes to infinity, the mean and variance of \( R_{zi}(i) \) satisfy

\[
E\{R_{zi}(i)\} = \begin{cases} 
O(\lambda^K), & \text{if } |\lambda_i| > 1 \\
O(K), & \text{if } \lambda_i = 1 \\
O(1), & \text{if } |\lambda_i| = 1, \lambda_i \neq 1,
\end{cases}
\]

\[
Var\{R_{zi}(i)\} = \begin{cases} 
O(\lambda_i^{2K}), & \text{if } |\lambda_i| > 1 \\
O(K^2), & \text{if } \lambda_i = 1 \\
O(1), & \text{if } |\lambda_i| = 1, \lambda_i \neq 1.
\end{cases}
\]

As \( K \) increases, the zero input response will be dominated by the terms corresponding to the dominant eigenvalues. In this section and the next one, we only consider the case that there is only one dominant eigenvalue (\( A \) is primitive). The discussion for the case with multiple dominant eigenvalues will be delayed to Section 4.5.

By Theorem 4.5, matrix \( A \) is primitive if and only if \( A^k > 0 \) for some large enough \( k \). By using the physical interpretation of \( A^k \), one necessary and sufficient condition for \( A^k \) to be all positive for \( k \geq K \) is that, for any branches \( i \) and \( j \) in \( \mathcal{R} \),

1. \( i \) can reach \( j \).
2. There exists a positive integer \( K_i \) such that \( i \) can reach itself in \( k_i \) steps, for any \( k_i \geq K_i \).

If these conditions hold, for all \( i \) and \( j \), by combining the loop from \( i \) to itself and the known path from \( i \) to \( j \), we can form a path from \( i \) to \( j \) with any large enough length by selecting the length of the loop part. On the other hand, if \( A^k \) is positive for large enough \( k \), the positive diagonal elements imply that any branch can reach itself and positive off-diagonal elements imply that any two branches can reach each other.

Let us consider the first condition. We assume the root \( \mathcal{R} \) to be connected. Then the message from any branch \( i \) can be passed to any data node \( d \) through a certain path. If the
branch $j$ on $d$ is not on the last section of the path, then $i$ can reach $j$ in the next iteration. The difficult part is when branch $j$ is on the last section of the path. If we remove branch $j$, and $d$ is still in a graph with loops, then we extend the path to the loop, circle around the loop, and go back to node $d$. This time, branch $j$ is no longer on the last section of the path and $i$ can reach $j$ in the next iteration. On the other hand, if $d$ is not in a loop when $j$ is removed, then $j$ must be in a single loop in the root formed by inner degree 2 data nodes and check nodes. In this case, it can be easily verified that the message from one of the branches can only reach half of the branches in $\mathcal{R}$ and $A^k$ cannot be positive for any $k$. Therefore, $i$ can reach $j$ if $j$ is not in a single loop.

Now, we consider the second condition. For a given branch $i$, if we know that the shortest loop is of length $g$, and if we can find $g$ consecutive numbers, $K_i, K_i + 1, \ldots, K_i + g - 1$, such that there exists a loop from branch $i$ to itself of length $k_i$, where $K_i \leq k_i \leq K_i + g - 1$, then there exist loops from $i$ to itself of length $k_i$ for all $k_i \geq K_i$. This can be shown as follows. Suppose we want to find a loop of length $k + K_i$, where $k \geq 0$. Note that $k$ can be written as $k'_i g + k''_i$, where $0 \leq k''_i \leq g - 1$. By the assumption, there is a loop of length $g$ (or any multiples of that) and a loop of length $K_i + k''_i$. By combining these two loops, we can form a loop of length $k + K_i$.

For the branch $i$, a sufficient condition for loops with $g$ continuous lengths to be found is, the branch is in two loops with length $g_1$ and $g_2$ such that $g_1$ and $g_2$ are co-prime. Since $g$ is the shortest loop that branch $i$ is in, it is obvious that $g_1 \geq g$ and $g_2 \geq g$. We prove this by construction. Consider $g_1$ consecutive numbers $g_1g_2, g_1g_2 + 1, \ldots, g_1g_2 + g_1 - 1$. The general form is $g_1g_2 + x$, where $x = 0, 1, \ldots, g_1 - 1$. We only need to show that $g_1g_2 + x$ can be written in the form $m_1g_1 + m_2g_2$ for any $x$. We can show this as follows. Computing $g_1g_2 + x - m_2g_2$ where $m_2 = 1, \ldots, g_1$, we have $x, x + g_2, x + 2g_2, \ldots, x + (g_1 - 1)g_2$. 

79
If two numbers $x + n_1 g_2$ and $x + n_2 g_2$ are the same modulo $g_1$, then $(n_1 - n_2) g_2 = 0$ modulo $g_1$. Since $g_1$ and $g_2$ are co-prime, $g_1$ must divide $n_1 - n_2$. However, since $0 \leq n_1, n_2 \leq (g_1 - 1)$ and $n_1 \neq n_2$, we must have $0 < |n_1 - n_2| < g_1$. Therefore, $x, x + g_2, x + 2g_2, \ldots, x + (g_1 - 1)g_2$ are not the same modulo $g_1$ and form a permutation of $0, 1, \ldots, g_1 - 1$. Then at least one of them is 0 modulo $g_1$. Let it be $x + n_1 g_2$ and it equals to $n_2 g_1$. Then $x + g_1 g_2 = (g_1 - n_1) g_2 + n_2 g_1 = m_1 g_1 + m_2 g_2$, where $m_1 = n_2$ and $m_2 = g_1 - n_1$.

To summarize, for a given trapping set, there exists a $K$ such that $A^k > 0$ for all $k \geq K$ if for any branch $i$, it participates in two loops of co-prime lengths. This condition contains the condition that any branch is not in a single loop. A few examples are given in Figure 4.3.

![Figure 4.3: Examples: (a), Single length 2 loop; (b) Multiple loops with $A^k \neq 0$; (c), Multiple loops with $A^k > 0$.](image)

In the figure, circles are data nodes and squares are check nodes. In Figure 4.3(a), all branches form a single loop. For this example, branch 1 can only reach branches 4 and 1, but not branches 2 and 3. Therefore, $A^k$ cannot be positive. In Figure 4.3(b), using
symmetry, we only need to consider branches 1, 2 and 9. Branches 1 and 2 are in loops of lengths 3 and 4. Branch 9 is in loops of lengths 3 and 7. The co-prime condition is satisfied and \( A^k > 0 \) for large enough \( k \). In Figure 4.3(c), once more using symmetry, we only need to consider branches 1, 5, 6 and 7. Since \( R \) is not a single loop, these branches can reach all branches. However, all these branches are in loops of even sizes. Therefore, the co-prime condition is not satisfied and \( A^k \) is not always positive.

When we only consider these primitive irreducible \( A \) matrices, the dominant eigenvalue is \( \lambda_1 \) and the zero input response in (4.22) becomes

\[
l^{(K)}_{z_1} = c'(1)(\sum_{k=0}^{K-1} \lambda_1^k) v_1.
\]

(4.26)

Note that a unique dominant eigenvalue assumption implies that \( |\lambda_1| > 1 \). Otherwise, by Theorem 4.9, all the eigenvalues are of magnitude 1 and there are multiple dominant eigenvalues. Furthermore, Theorem 4.3 states that the dominant eigenvector is real and positive. This implies that, for the zero input response, the messages in the root of a simple trapping set have only two modes of evolution. They either go to infinity together, or go to negative infinity together, which is decided by the sign of \( c'(1) \).

### 4.3.3 Zero State Response

Using the eigen-decomposition of \( A \) in the zero state response in (4.16), we have

\[
l^{(K)}_{z_0} = \sum_{k=1}^{K-1} V \Lambda^{K-1-k} V^{-1} B_o o^{(k)}.
\]

(4.27)

Define \( B = V^{-1} B_o \). \( B \) is a \( M \times N \) matrix. Define \( b_i \) to be the \( i \)'th row of \( B \), then \( b_i = u_i B_o \). Then equation (4.27) can be further simplified to

\[
l^{(K)}_{z_0} = \sum_{k=1}^{K-1} \sum_{i=1}^{M} b_i o^{(k)} \lambda_i^{K-1-k} v_i,
\]

\[
= \sum_{i=1}^{M} \left[ \sum_{k=1}^{K-1} b_i o^{(k)} \lambda_i^{K-1-k} \right] v_i.
\]

(4.28)
We define the coefficient of the $i$’th eigenvector to be

$$R_{zs}(i) = \sum_{k=1}^{K-1} b_i o^{(k)}(i) \lambda_i^{K-1-k}. \quad (4.29)$$

Comparing to the zero input response, the zero state response is more difficult to analyze. It is a function of inputs from outside of the trapping set for all previous iterations. Generally, there are no closed form models for these inputs and they are correlated.

Under the infinite frame size assumption, the input $o^{(k)}$’s are independent Gaussian, therefore we can use Gaussian approximated density evolution to find the parameters of $o^{(k)}$. Following [8], we assume the Gaussian random variables to be consistent, i.e., the variance of the Gaussian random variable equals two times the mean. In this way, we only need to track the mean in the evolution.

In [8], it is shown that, the mean of the check-to-data messages is

$$m(k) = \phi^{-1}\left(1 - \left[1 - \phi(m_0 + (d_v - 1)m_{(k-1)})\right]^{d_v-1}\right) \quad (4.30)$$

for a regular LDPC code with check node degree $d_c$ and data node degree $d_v$. Here $k$ is the iteration number, $m_0$ is the mean of the channel observation in likelihood ratio form, and

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{\pi x}} \int_{-\infty}^{\infty} \tanh \frac{u}{2} e^{-\frac{(u-x)^2}{4x}} du, & \text{if } x > 0, \\ 1, & \text{if } x = 0. \end{cases} \quad (4.31)$$

For irregular LDPC codes with degree distribution $(\lambda(x), \rho(x))$, the mean evolution is

$$m(k) = \sum_{j=2}^{d_c} \rho_j \phi^{-1}\left(1 - \left[1 - \sum_{i=2}^{d_v} \lambda_i \phi(m_0 + (i-1)m_{(k-1)})\right]^{j-1}\right) \quad (4.32)$$

We can reduce (4.30) and (4.32) to simpler forms when $m_{(k-1)}$ is large.

It was observed in [8] that when $x \gg 0$, the function $\phi(x)$ can be closely approximated by

$$\phi(x) \approx \sqrt{\frac{\pi}{x}} e^{-x/4}. \quad (4.33)$$
For the regular case, using the approximation in (4.33) and observing $(1 - x)^n \approx 1 - nx$ for $0 < x \ll 1$, (4.30) becomes
\[
\sqrt{\frac{\pi}{m(k)}} \exp\left(-\frac{m(k)}{4}\right) = (d_v - 1) \sqrt{\frac{\pi}{(d_v - 1)m(k-1)}} \exp\left(-\frac{(d_v - 1)m(k)}{4}\right) \tag{4.34}
\]
Taking logarithm on both sides, we have
\[
m(k) + 2 \log(m(k)) = (d_v - 1)m(k-1) + 2 \log(m(k-1)) - 4 \log(d_v - 1) + 2 \log(d_v - 1). \tag{4.35}
\]
Note that the $m(k)$ term increases much faster than the $\log(m(k))$ term, therefore we can say that
\[
m(k) = (d_v - 1)m(k-1) + \text{some small value terms}, \tag{4.36}
\]
and
\[
m(k) = O((d_v - 1)^k), \tag{4.37}
\]
For the non-trivial case that $d_v \geq 3$, the mean $m(k)$ increases to infinity exponentially.

For the irregular LDPC case, the mean evolution equation is slightly more complex. The evolution rule in (4.32) is actually the average of the means of outgoing messages from check nodes of different degrees. For a check node of degree $j$, the mean of the output is
\[
m^{(j)}(k) = \phi^{-1}\left(1 - \left[1 - \sum_{i=2}^{d_t} \lambda_i \phi(m_0 + (i - 1)m(k-1))\right]^{j-1}\right) \tag{4.38}
\]
Following the same technique for the regular case, when $m(k-1)$ is large, we have
\[
m^{(j)}(k) + 2 \log(m^{(j)}(k)) = -4 \log(j - 1) - 4 \log\left(\sum_{i=2}^{d_t} \lambda_i e^{-(m_0 + (i - 1)m(k-1))/4} \sqrt{m_0 + (i - 1)m(k-1)}\right) \tag{4.39}
\]
As $m(k-1)$ goes to infinity, the second term on the right hand side is dominated by the term in the summation corresponding to the smallest $i$ with non-zero $\lambda_i$.

If the smallest check node degree is 2, then
\[
m^{(j)}(k) = m(k-1) + \text{small terms}. \tag{4.40}
\]
Averaging over the check node degree distribution, we have

\[ m_{(k)} = m_{(k-1)} + \text{small terms}. \]  \hspace{1cm} (4.41)

Therefore,

\[ m_{(k)} = O(k). \]  \hspace{1cm} (4.42)

If the smallest check node degree is not 2, but \( d \geq 3 \), then

\[ m_{(k)}^{(j)} = (d-1)m_{(k-1)} + \text{small terms}. \]  \hspace{1cm} (4.43)

Averaging over the check node degree distribution, we have

\[ m_{(k)} = (d-1)m_{(k-1)} + \text{small terms}. \]  \hspace{1cm} (4.44)

Therefore,

\[ m_{(k)} = O((d-1)^k). \]  \hspace{1cm} (4.45)

In summary, when the smallest check node degree is 2, the mean increases linearly. Otherwise, the mean increases exponentially. Note the above analysis is based on the assumption that the \( m_{(k-1)} \) is large. Therefore, the linear or exponential increasing rate is only valid for large \( m_{(k)} \). An example is shown in Figure 4.4 for different \( \lambda_2 \).

We can apply (4.37), (4.42) and (4.45) back to (4.28) to find the mean of \( l_{zs}^{(K)} \). For regular LDPC codes with data node degree \( d \) and irregular LDPC codes with smallest data node degree \( d \geq 3 \),

\[ m_{(k)} \approx C_1(d-1)^k, \]  \hspace{1cm} (4.46)

where \( C_1 \) is a constant. As can be observed in Figure 4.4, the approximation is accurate for \( k \) large. With this approximation, by equation (4.28), the mean of \( R_{zs}(i) \) is

\[ E\{R_{zs}(i)\} = \sum_{k=1}^{K-1} \left( \sum_{j=1}^{N} b_i(j) \right) C_1(d-1)^k \lambda_1^{K-1-k} \]  \hspace{1cm} (4.47)
When $\lambda_i \neq d - 1$,
\[
E\{R_{zs}(i)\} = \left(\sum_{j=1}^{N} b_i(j)\right)C_1\frac{(d - 1)(\lambda_i^{K-1} - (d - 1)^{K-1})}{\lambda_i - (d - 1)}.
\] (4.48)

Otherwise, when $\lambda_i = d - 1$,
\[
E\{R_{zs}(i)\} = \left(\sum_{j=1}^{N} b_i(j)\right)C_1(K - 1)(d - 1)^{K-1}.
\] (4.49)

Asymptotically, the mean of $R_{zs}(i)$ is
\[
E\{R_{zs}(i)\} = \begin{cases} 
O(\lambda^K), & \text{if } |\lambda_i| > d - 1, \\
O(K(d - 1)^K), & \text{if } \lambda_i = d - 1, \\
\text{Oscillation with envelop } O((d - 1)^K), & \text{if } |\lambda_i| = d - 1, \lambda_i \neq d - 1 \\
O((d - 1)^K), & \text{if } |\lambda_i| < d - 1,
\end{cases}
\] (4.50)

Similarly, we can find the asymptotic variance of $R_{zs}(i)$. Since we assume the messages outside the trapping set to be consistent, the variance of each element in $o^{(k)}$ is $2C_1(d - 1)^k$. 

85
Then the variance of $R_{zs}(i)$ is

\[
\text{Var}\{R_{zs}(i)\} = \sum_{k=1}^{K-1} 2\left(\sum_{j=1}^{N} b_i^2(j)\right) C_1 (d-1)^k |\lambda_i|^{2(K-1-k)}
\]

\[
= 2\left(\sum_{j=1}^{N} b_i^2(j)\right) C_1 \frac{(d-1)(|\lambda_i|^{2(K-1)} - (d-1)^{K-1})}{|\lambda_i|^2 - (d-1)},
\]

when $|\lambda_i|^2 \neq d-1$. On the other hand, when $|\lambda_i|^2 = d-1$,

\[
\text{Var}\{R_{zs}(i)\} = 2\left(\sum_{j=1}^{N} b_i^2(j)\right) C_1 (K-1)(d-1)^{K-1},
\]

(4.52)

Asymptotically, the variance of the $R_{zs}(i)$ is

\[
\text{Var}\{R_{zs}(i)\} = \begin{cases} 
O(\lambda_i^{2K}), & \text{if } |\lambda_i|^2 > d-1, \\
O(K(d-1)^K), & \text{if } |\lambda_i|^2 = d-1, \\
O((d-1)^K), & \text{if } |\lambda_i|^2 < d-1,
\end{cases}
\]

(4.53)

For an irregular LDPC code with smallest data node degree 2, equation (4.42) can be approximated by

\[
m_{(k)} \approx C_2 k,
\]

(4.54)

where $C_2$ is a constant. As in the previous case, the approximation is accurate for $k$ large. Then the mean of $R_{zs}(i)$ is

\[
E\{R_{zs}(i)\} = \sum_{k=1}^{K-1} \left(\sum_{j=1}^{N} b_i(j)\right) C_2 k \lambda_i^{K-1-k}
\]

\[
= \begin{cases} 
\left(\sum_{j=1}^{N} b_i(j)\right) C_2 \left(\frac{\lambda_i^{K-1}}{|\lambda_i| - 1} - \frac{K}{\lambda_i - 1}\right), & \lambda_i \neq 1 \\
\left(\sum_{j=1}^{N} b_i(j)\right) C_2 \frac{K(K-1)}{2}, & \lambda_i = 1.
\end{cases}
\]

(4.55)

For the variance of $R_{zs}(i)$,

\[
\text{Var}\{R_{zs}(i)\} = 2\sum_{k=1}^{K-1} \left(\sum_{j=1}^{N} b_i^2(j)\right) C_2 k |\lambda_i|^{2(K-1-k)}
\]

\[
= \begin{cases} 
2\left(\sum_{j=1}^{N} b_i^2(j)\right) C_2 \left(\frac{|\lambda_i|^{2(K-1)} - 1}{|\lambda_i| - 1}\right) & |\lambda_i| \neq 1 \\
2\left(\sum_{j=1}^{N} b_i^2(j)\right) C_2 \frac{K(K-1)}{2} & |\lambda_i| = 1
\end{cases}
\]

(4.56)
Asymptotically,
\[
E\{R_{zs}(i)\} = \begin{cases} 
O(\lambda^K), & \text{if } |\lambda_i| > 1, \\
O(K^2), & \text{if } \lambda_1 = 1, \\
O(K), & \text{if } |\lambda_i| = 1, \lambda_i \neq 1,
\end{cases}
\]
\[
(4.57)
\]
\[
\text{Var}\{R_{zs}(i)\} = \begin{cases} 
O(\lambda_i^{2K}), & \text{if } |\lambda_i| > 1, \\
O(K^2), & \text{if } |\lambda_i| = 1.
\end{cases}
\]
\[
(4.58)
\]

As in the zero input response case, the zero state response is also dominated by the terms corresponds to the dominant eigenvalues. When \( A \) is primitive, there is a unique dominant eigenvalue, \( \lambda_1 \), which is positive real and \( 1 < \lambda_1 \leq D - 1 \). Then we only need to consider \( R_{zs}(1) \).

### 4.3.4 Combination of Zero Input Response and Zero State Response

To check if a trapping set is corrected, we need to check the elements in the total response vector, which is the summation of zero input and zero state response vectors. If the elements in the total response are always positive after a certain number of iterations, the trapping set is corrected. Otherwise, some errors cannot be corrected by the iterative decoder. In this case, we say the trapping set is triggered. For the primitive \( A \) we consider in this section, by Theorem 4.3, the dominant eigenvector is positive. Therefore, we only need to consider \( R = R_{zi}(1) + R_{zs}(1) \).

The triggering probability is \( Pr\{R < 0\} \). The magnitude of the zero input response \( R_{zi}(1) \) is a Gaussian random variable and we approximate \( R_{zs}(1) \) by another Gaussian random variables. Then \( R \) is also Gaussian. We can trace the distribution of \( R \) with respect to iterations using the method described in the previous subsections. Some examples are given in Section 4.6.

We give some asymptotic results in this section. For the zero input response, the asymptotes of the mean and variance of \( R_{zi}(1) \) are given in (4.25). The asymptotes of
the mean and variance of $R_{zs}(1)$ are shown in (4.50) and (4.53) for $d > 2$ case and in (4.57) and (4.58) for $d = 2$ case. To combine the zero input and zero state responses, we just need to add the corresponding mean and variance for they are independent. Then $E\{R\} = E\{R_{zi}(1)\} + E\{R_{zs}(1)\}$ and $Var\{R\} = Var\{R_{zi}(1)\} + Var\{R_{zs}(1)\}$. The summation is dominated by the larger one of the two.

For $d > 2$ case, we have five different cases:

\[
\begin{align*}
E\{R\} &= O(\lambda^K), Var\{R\} = O(\lambda^{2K}), \quad \text{if } d - 1 < \lambda_1 \\
E\{R\} &= O(K\lambda^K), Var\{R\} = O(\lambda^{2K}), \quad \text{if } \lambda_1 = d - 1 \\
E\{R\} &= O((d - 1)^K), Var\{R\} = O(\lambda^{2K}), \quad \text{if } \sqrt{d - 1} < \lambda_1 < d - 1 \\
E\{R\} &= O((d - 1)^K), Var\{R\} = O(K(d - 1)^K), \quad \text{if } \lambda_1 = \sqrt{d - 1}, \\
E\{R\} &= O(((d - 1)^K), Var\{R\} = O((d - 1)^K), \quad \text{if } \lambda_1 < \sqrt{d - 1},
\end{align*}
\]

Both the mean and variance of $R$ go to infinity exponentially. It is easier to see the shape of the distribution by normalizing $R$ to have variance 1. The normalized version of $R$ is denoted by $\hat{R}$. The asymptotical mean of $\hat{R}$ is

\[
E\{\hat{R}\} = \begin{cases} 
O(1), & \text{if } d - 1 < \lambda_1 \\
O(K), & \text{if } \lambda_1 = d - 1 \\
O((d - 1)^K), & \text{if } \sqrt{d - 1} < \lambda_1 < d - 1 \\
O((d - 1)^K/\sqrt{K}), & \text{if } \lambda_1 = \sqrt{d - 1}, \\
O((d - 1)^{K/2}), & \text{if } \lambda_1 < \sqrt{d - 1},
\end{cases}
\]

For the first case, the mean and variance of the zero state response grow as fast as the mean and variance of the zero input response. However, since the constants $C_1$ and $C_2$ in (4.46) and (4.54) are usually small, the dominant term in $R$ is $R_{zi}$ and $Pr\{R < 0\} \approx Pr\{R_{zi} < 0\} > 0$. The triggering probability is dominated by the zero input response.

For the other cases, the mean of the $\hat{R}$ goes to infinity and the tail probability goes to zero. In cases 2 and 3, the variances of the zero input response and zero state response grow at the same rate. However, the mean of the zero state response grows faster. For large $K$,
the mean of $R$ will be dominated by the mean of $R_{zs}$ and the variance will be dominated by the variance of $R_{zi}$. The mean of $\hat{R}$ is $O((d - 1)^K/\lambda_1^K)$ and goes to infinity. This implies that the triggering probability of the trapping set goes to zero. For the last two cases, the mean and variance of the zero state response increase faster. The dominant term in $R$ is $R_{zs}$. Since the mean of $\hat{R}_{zs}$ goes to infinity, the triggering probability goes to zero.

There is an intuitive explanation for this result. For the first case, $\lambda_1$ is large compared to $d - 1$. This usually happens when the loop structure in the trapping set is relatively complex. Therefore, the positive feedback within the trapping set is strong and the zero input response is dominant. For the other cases, $\lambda_1$ is small compared to $d - 1$. This implies that the loop structure is simple and the positive feedback is weak. Eventually the zero state response (the influence from the outside) will be dominant and the triggering probability goes to zero.

For the $d = 2$ case, we can similarly combine the zero input and zero state responses. The asymptotic mean and variance of $R$ are

$$E\{R\} = O(\lambda_1^K), \quad Var\{R\} = O(\lambda_1^{2K}).$$

(4.61)

The total response is dominated by the zero input response.

Similarly, we can normalize these distributions to have variance 1. The asymptotic mean of the normalized version $\hat{R}$ is

$$E\{\hat{R}\} = O(1).$$

(4.62)

The normalized mean is finite and there is a finite triggering probability.
4.4 Trapping Sets with External Data Nodes

Now we consider the case that there are external data nodes in the trapping set. We can show that adding external data nodes to a simple trapping set corresponds to adding zero eigenvalues to the eigen-structure. To show this, we need a few more definitions. For an external data node, by definition, one of its branches is in a unique path that eventually connects the node to the root of the trapping set. We call this branch an upstream branch (or that it points to the upstream direction) and all other branches downstream branches (or that they point to the downstream direction). For a matrix $A$, we call a certain column or row to be upstream or downstream when the corresponding branch in the graph is upstream or downstream. The upstream or downstream element in a vector is similarly defined.

To show our result, we need to sequentially label the branches using the upstream and downstream concept. First, we label the branches in the root of the trapping set. Then we label the remaining branches using a depth-first enumeration starting from the root. In this way, for any branch not in the root, all other branches in its downstream direction have larger labels and all other branches in its upstream direction have smaller labels.

Given this labeling system, for the constructed $A$, we have the following property: For a downstream branch $i$, the data-to-check message on the branch will only appear in the computation of messages on the downstream branches on the downstream side of branch $i$. So, in the $i$’th column of matrix $A$, the first $i$ elements must be zero. On the other hand, for an upstream branch $i$, to compute the message on the branch, only the messages from some upstream branches in the downstream direction are used. Therefore, in the $i$’th row of $A$, the first $i$ elements are always zero. This property can be verified in an example shown in Figure 4.5. This example is constructed by adding two external data nodes to Figure 4.1. For this example, $A$ is
Figure 4.5: A trapping set with external data nodes

\[ A = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \]  

(4.63)

The fore-mentioned properties of \( A \) can be easily verified in this example.

When we compute the eigenvalues, we form the matrix \( C_M = A - xI \), where \( M \) is the total number of branches connected to the inner degree 2 check nodes. The diagonal elements of \( C_M \) are \(-x\). Define \( M_1 \) to be the number of branches in the root and \( M_2 = M - M_1 \). We need to write \(|C_M|\) as a polynomial of \( x \) and solve \(|C_M| = 0\) to find the
eigenvalues. We define the sub-matrix of $C_M$ formed by the first $M_1$ rows and $M_1$ columns to be $C_{M_1}$. We consider the last column and row of $C_M$. If this last branch is not in the root, then either all elements above the $-x$ on the diagonal are zero (the branch is a downstream one), or all elements to the left of $-x$ are zero (the branch is is an upstream one). We can use the property that $|\text{diag}(A, B)| = |A| \cdot |B|$ and have $|C_M| = |C_{M-1}|(-x)$. In this way, we can recursively remove the last column and row and have $|C_M| = |C_{M_1}|(-x)^{M_2}$. Here $|C_{M_1}|$ is the characteristic polynomial of the root part of the subgraph and the $(-x)^{M_2}$ term corresponds to $M_2$ zero eigenvalues. Then the roots of $|C_M|$ are the roots of $|C_{M_1}|$ and $M_2$ zero roots.

For the previous example,

$$
A - xI = 
\begin{bmatrix}
-x & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & -x & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -x & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -x & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & -x & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & -x & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -x & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & -x & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -x & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -x & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x
\end{bmatrix} \quad (4.64)
$$

For the last four columns and rows, either all elements above the $-x$ elements are zeros, or all elements on the left of $-x$ are zeros. This verifies our analysis result.

This result can be summarized in the following theorem

**Theorem 4.12** The non-zero eigenvalues of a general trapping set are eigenvalues of the induced simple trapping set.

Denote the sub-matrix formed by the first $M_1$ columns and rows of $A$ by $A'$. Then $A'$ corresponds to the root of the trapping set. The Jordan canonical form of $A'$ is denoted
by $\Lambda$ and is assumed to be diagonal. In general, the Jordan canonical form $J$ of $A$ is not diagonal. However, it has a close relationship with $\Lambda$ as stated in the following theorem.

**Theorem 4.13** Let the Jordan canonical form of $A$ and $A'$ to be $J$ and $\Lambda$ respectively, where $\Lambda$ is diagonal. Then $J = \begin{bmatrix} \Lambda & 0 \\ 0 & J_0 \end{bmatrix}$. Here $J_0$ is a block diagonal matrix composed of Jordan blocks in the form of

$$J_{0,i} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & 1 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix},$$

(4.65)

where $i$ is the index for Jordan blocks in $J_0$.

The proof for the theorem is in the appendix.

Since the Jordan canonical forms of $A$ and $A'$ are closely related, we can still use a similar technique developed in the previous sections for these trapping sets. Even though we cannot write $A$ as in (4.18), it is still true that

$$A^L = V \begin{bmatrix} \Lambda^L & 0 \\ 0 & J_0^L \end{bmatrix} V^{-1}. \quad (4.66)$$

For an $m \times m$ Jordan block $J_{0,i}$ in (4.65), one important property is that $J_{0,i}^m = 0$, an all-zero $m \times m$ matrix. Then, if $L$ is not smaller than the largest Jordan block size, $A^L$ becomes

$$A^L = V \begin{bmatrix} \Lambda^L & 0 \\ 0 & 0 \end{bmatrix} V^{-1}, \quad (4.67)$$

which is still diagonal and the expansion in (4.65) can be used to write $A^L$ as

$$A^L = \sum_{i=1}^{M_1} \lambda_i^L v_i u_i. \quad (4.68)$$

Then, the zero input and zero state responses are

$$z^{(K)}_{z1} = V \left( \sum_{k=1}^{L-1} J^{k-1} \right) V^{-1} c + \sum_{i=1}^{M_1} c'(i) \left( \sum_{k=L}^{K-1} \lambda_i^k \right) v_i, \quad (4.69)$$
\[ \mathbf{1}^{(K)}_{zs} = \sum_{k=K-L}^{K-1} A^{K-1-k} B_o \mathbf{o}^{(k)} + \sum_{i=1}^{M_1} \left( \sum_{k=1}^{K-L-1} b_i \mathbf{o}^{(k)} \lambda_i^{K-1-k} \right) \mathbf{v}_i. \] (4.70)

When \( K \) is large, we can ignore the first terms in both (4.69) and (4.70) without changing the asymptotic growth rates of the means and variances of the responses. Therefore, we can ignore the fact that \( A \) cannot be diagonalized and use the techniques developed in the previous section to analyze trapping sets with external data nodes.

### 4.5 Trapping Sets with Multiple Dominant Eigenvalues

In this section, we investigate trapping sets with multiple dominant eigenvalues. Since we only consider irreducible matrices, the matrix \( A \) is imprimitive. By Theorem 4.6, when multiple dominant eigenvalues exist, they are located uniformly on a circle in the complex plane. By Theorem 4.4, one of the dominant eigenvalue is positive real and the corresponding eigenvector is positive. The \( h \) dominant eigenvalues are \( \{ \lambda, \lambda e^{j2\pi/h}, \ldots, \lambda e^{j2\pi(h-1)/h} \} \), where \( \lambda > 0 \).

Each dominant eigenvector corresponds to a mode of response and the total response is the mixture of different modes. How these modes are mixed depends on the channel observation and the input from outside the trapping set. For \( K \) large enough, the zero input response and zero state response can be simplified to

\[ \mathbf{1}^{(K)}_{zi} = \sum_{i=1}^{h} R_{zi}(i) \mathbf{v}_i, \] (4.71)

where

\[ R_{zi}(i) = c'(i) \left( \sum_{k=0}^{K-1} \lambda_i^k \right), \] (4.72)

and

\[ \mathbf{1}^{(K)}_{zs} = \sum_{i=1}^{h} R_{zs}(i) \mathbf{v}_i, \] (4.73)
where

\[ R_{zs}(i) = \sum_{k=1}^{K-1} b_i o^{(k)} \lambda_i^{K-1-k}. \]  \hspace{1cm} (4.74)

The two responses are in the direction within the subspace spanned by the corresponding eigenvectors.

An example is shown in Figure 4.6. There are two dominant eigenvalues, \( \lambda_1 = 3 \) and \( \lambda_2 = -3 \). These two eigenvalues correspond to two different modes that the messages evolve. The first mode corresponds to the first eigenvector \( v_1 \), whose elements are all positive. This is the case that the data nodes in the trapping are either all correctly detected or all incorrectly detected. The second mode corresponds to the second eigenvector \( v_2 \) whose first 4 elements are positive and the next 4 elements are negative. In this mode, the messages form an oscillation. For example, in a certain iteration, all four inner outgoing messages are positive for the first data node and negative for the second data node. Then the signs of messages flip for every iteration and an oscillation on the hard decisions of the
data nodes is formed. The actual messages will be a combination of these two modes. The dominant mode is decided by the channel observation $c$ and messages from outside $o^{(k)}$.

Unlike the primitive $A$ case, the dominant eigenvectors are not all positive. Therefore, it is not sufficient to analyze $R(i)$ terms only. We have to directly consider the total response

$$ l^{(k)} = l^{(k)}_{zi} + l^{(k)}_{zs}. $$

For the zero input response,

$$ l^{(K)}_{zi} = \sum_{i=1}^{h} \left[ \sum_{k=0}^{K-1} \lambda^k e^{j2\pi k(i-1)/h} \right] c'(i) v_i, $$

where $c'(i) = u_i c$. Define the matrix $U'$ to be the first $m$ rows of $U$. Define $L_{zi}$ to be an $h \times h$ diagonal matrix with $L_{zi}(i, i) = \sum_{k=0}^{K-1} \lambda^k e^{j2\pi k(i-1)/h}$. Then the coefficients of the terms in (4.75) form a vector $R_{zi} = L_{zi} U' c$. Since we know that $c$ is a Gaussian vector with distribution $\mathcal{N}(M_c, \Sigma_c)$, the distribution of $R_{zi}$ is

$$ \mathcal{N}(M_{zi}, \Sigma_{zi}) = \mathcal{N}(L_{zi} U' M_c, L_{zi} U' \Sigma_c (U')^H L_{zi}^H). $$

For the zero state response,

$$ l^{(K)}_{zs} = \sum_{i=1}^{h} \sum_{k=1}^{K-1} b_i o^{(k)} \lambda^{K-1-k} e^{j2\pi i(K-1-k)/h} v_i $$

$$ = \sum_{k=1}^{K-1} \left[ \sum_{i=1}^{h} b_i o^{(k)} \lambda^{K-1-k} e^{j2\pi i(K-1-k)/h} v_i \right], $$

where $b_i = u_i B_o$. Define matrix $L^{(k)}_{zs}$ to be an $h \times h$ diagonal matrix with $L^{(k)}_{zs}(i, i) = \lambda^k e^{j2\pi (i-1)k/h}$. Then the coefficients of the terms in (4.77) for each $k$ form a vector $R^{(k)}_{zs} = L^{(K-k-1)}_{zs} U' B_o o^{(k)}$. The distribution of $o^{(k)}$ can be found using density evolution. Since we are using a consistent Gaussian vector model, the distribution is in the form of $\mathcal{N}(M_\delta^{(k)}, \Sigma_\delta^{(k)})$. Then the distribution of $R^{(k)}_{zs}$ is

$$ \mathcal{N}(L^{(K-k-1)}_{zs} U' B_o M_\delta^{(k)}, L^{(K-k-1)}_{zs} U' B_o \Sigma_\delta^{(k)} (L^{(K-k-1)}_{zs} U' B_o)^H). $$
Since we assume all $o^{(k)}$'s are independent, the vector $R_{zs} = \sum_{k=1}^{K-1} R_{zs}^{(k)}$ has distribution

$$
\mathcal{N}(M_{zs}, \Sigma_{zs}) = \mathcal{N}\left(\sum_{k=1}^{K-1} L_{zs}^{(K-k-1)} U' B_o M_o^{(k)}, \sum_{k=1}^{K-1} L_{zs}^{(K-k-1)} U' B_o \Sigma_o^{(k)} (L_{zs}^{(K-k-1)} U' B_o)^H \right).
$$

(4.78)

Define vector $R = R_{zi} + R_{zs}$. Then $R$ is also a Gaussian vector with distribution

$$
\mathcal{N}(M_R, \Sigma_R) = \mathcal{N}(M_{zi} + M_{zs}, \Sigma_{zi} + \Sigma_{zs}).
$$

(4.79)

Note that $R$ is the total response in the form of coefficient in each dominant eigenvector direction, we still need to find the distribution of the total response $l$. Define $V'$ to be the first $m$ columns of $V$. Then $l = V'R$ and its distribution is

$$
\mathcal{N}(M_l, \Sigma_l) = \mathcal{N}(V'M_R, V'\Sigma_R (V')^H).
$$

(4.80)

The triggering probability is the probability that at least one of the terms in $l$ is not always positive after any number of iterations. Given the distribution of $l$, this probability can be computed using a $d$-dimensional integration. However, the complexity of the method is high. As an alternative, we can use a union bound to upper-bound this probability,

$$
\Pr(\cup_{i=1}^{M} l(i) < 0) \leq \sum_{i=1}^{M} \Pr(l(i) < 0),
$$

(4.81)

which is a simple computation using the marginal distribution of $l$.

### 4.6 Results

In this section, we apply the techniques developed in the previous sections to estimate the triggering probability for a few trapping sets. Due to limited simulation resources, when the error floor is too low, we cannot directly compare the analysis result with the simulation results. For such cases, we will compare the analysis result with the error floor found using Richardson’s result [35].
4.6.1 Regular LDPC Code Examples

The first example is the simple trapping set in Figure 4.1. Since it contains very short loops, its error floor is relatively high, which let us compare the analysis with the simulation results.

Since there is only one dominant eigenvalue, it is enough to consider the distribution of $R_z(1) + R_s(1)$. We can find the distribution of $R_z$ directly using the channel SNR. The distribution of $R_s$ can be computed using the distributions of $o^{(k)}$, which can be further found using the consistent Gaussian approximated density evolution.

We consider a regular (3,6) LDPC code with 3600 branches. The means and variances of $R_z$ and $R_s$ for channel SNR 2dB are shown in Figure 4.7. It can be observed from the figure that all of them increase exponentially.

![Figure 4.7: Mean and variance of $R_z$ and $R_s$ for trapping set as in Figure 4.1 in regular (3,6) LDPC code, channel SNR 2dB.](image)

98
To visualize the tail probability, we normalize all distributions to have variances 1. The mean of the normalized $R_{zi}$, $R_{zs}$ and $R$ are shown in Figure 4.8. The mean of the normalized total response increases exponentially, which implies an asymptotically zero triggering probability and no error floor.

One important observation from Figure 4.7 is that there is a cross between the means of zero input response and zero state response. The cross is located at the point where the means approximately equal $10^{10}$. Before the messages evolve to this order of magnitude (the first about 40 iterations), they are dominated by the zero input response. The mean of the normalized total response is flat as shown in Figure 4.8. However, after that cross point, the zero state response becomes the dominant term and the mean of the normalized total response starts to increase to infinity.

Figure 4.8: Normalized mean of $R_{zi}$, $R_{zs}$ and $R$ for trapping set as in Figure 4.1 in regular (3,6) LDPC code, channel SNR 2dB.
Unfortunately, in most iterative decoders for LDPC codes, due to the limitation of the numerical precision of a computer, the key function $\tanh(x)$ cannot be accurately computed for $|x|$ large. In most realizations, a hard-limiter is used. Before computing $\tanh(x)$, the value $|x|$ is compared with a pre-defined limit $L$. If $|x| > L$, then $x$ is replaced by $\text{sgn}(x)L$.

For example, the IEEE 754 double precision floating point representation [1] only provides 53 bits of resolution, which corresponds to $1.11 \times 10^{-16}$. Then $\tanh(20) = 1 - 8.50 \times 10^{-18}$ is well above the resolution and will simply return 1.

If a hard-limiter is used, the evolution of messages stops well before they reach the value of $10^{10}$. The evolution of the mean of the normalized total response stops at the flat area, which implies that an error floor will appear due to the usage of the hard-limiter. To solve this problem, we propose to use the approximated computation rule in (4.5) to compute $\tanh(x)$ for $|x|$ large (larger than 10). For small $|x|$, we still directly compute $\tanh(x)$. The simulation results using the approximated computation and the hard-limiter are compared in Figure 4.9. The decoder using the approximated computation has about 0.3dB performance gain for the bit error rate curve. The performance gain in frame error rate curve is even larger. For the trapping set in 4.1, at 3dB, we observe a triggering rate of $1.8 \times 10^{-4}$ for the decoder using a hard-limiter. For the decoder using the approximated computation, we have not observed a single case that the trapping set being triggered.

Another trapping set example was discovered in [25] and analyzed in [35]. This trapping set is in a Margulis code (a (3,6) regular LDPC code) with information frame size $n = 2640$. The trapping set has 12 data nodes. Four checks are not satisfied. It was reported in [35] that the triggering probability is $6.5 \times 10^{-10}$ at 2.8dB. In [35], a 5-bit quantized decoder is used, which can be thought of as a quantized version of a decoder with a hard-limiter. We implemented Richardson’s method using both a hard-limiter decoder and
an approximation decoder. The estimated error floors are compared in Figure 4.10. One can see that the error floor for the approximation decoder is much lower.

It is worthwhile noting that no error floor is predicted if we apply our analysis method to this trapping set. Actually, for a regular LDPC code with data node degree at least 3, we can show that our method predicts no error floor for all trapping sets except those that are also error events (all checks are satisfied). This is because, by (4.60), the only case that we have a non-zero error floor is when the dominant eigenvalue is larger than $d - 1$, where $d$ is the data node degree. However, by Theorem 4.11, the dominant eigenvalue is upper bounded by $D - 1$, where $D$ is the largest inner degree of the data nodes in the trapping set. Since we are considering a regular code, we must have $D \leq d$. Therefore the dominant eigenvalue will never be larger than $d - 1$ and an error floor will not appear.
Figure 4.10: Comparison of decoder using hard-limiter or approximated computation in check nodes using Richardson’s method, Margulis code.

Contradicting to our prediction, even though we use the approximation decoder, Richardson’s method still predicts an error floor. We believe this is because we use an infinite frame size assumption. The Margulis code has a frame size of 2640. The messages outside the trapping set may not follow the density evolution result and hence the zero state response may not be strong enough to dominate the total response.

4.6.2 Irregular LDPC Code Examples

For irregular LDPC codes with smallest data node degree 2, we consider an example with the degree distribution

\[
\begin{align*}
\lambda(x) &= 0.27684x + 0.28342x + 0.43974x^8, \\
\rho(x) &= 0.01568x^5 + 0.85244x^6 + 0.13188x^7.
\end{align*}
\]
For this degree distribution, there are degree 2 data nodes and \( d - 1 = 1 \). For any trapping set with a dominant eigenvalue larger than 1, there will be an error floor.

Let us apply our method for the trapping set in Figure 4.1. For SNR=2dB, the means and variances of \( R_{zi} \) and \( R_{zs} \) are shown in Figure 4.11. The means of the normalized versions are shown in Figure 4.12.

![Figure 4.11: Mean and variance of \( R_{zi} \) and \( R_{zs} \) for trapping set as in Figure 4.1 in irregular LDPC code, channel SNR 2dB.](image)

As can be observed in Figure 4.12, the normalized mean is upper bounded and there is no point in using the approximated computation in the decoder. A hard-limiter decoder is sufficient for decoding this code.

From the mean of the normalized total response, the triggering probability can be estimated. It is compared with the simulated triggering rate in Figure 4.13.
There is a gap between the estimated triggering probability and the simulated one. We believe it is due to the Assumption 4.5 used in deriving the linear model. As we know, the Assumption 4.5 is valid if the magnitudes of messages from outer branches are larger than those from the inner branches. For irregular LDPC codes with minimum data node degree 2, this is usually not satisfied. Therefore, our analysis result can be used as an upper bound to the true error floor.

4.7 Conclusions

In this chapter, we addressed the problem of estimating the triggering probability of any trapping sets. For that purpose, we introduced an approximated linear system model for a trapping set. By investigating the zero input and zero state responses of the linear system, we can track the Gaussian modeled messages in the trapping set with respect to iterations.
We discovered that the properties of the zero input and zero state responses are closely related to the eigen-structure of a matrix $A$ describing the structure of the trapping set, especially to the dominant eigenvalues and eigenvectors. The eigen-structures of different types of trapping sets are discussed and the asymptotic properties are analyzed.

Using this technique, one can estimate the error floor of LDPC codes without using simulations. As a by-product, it is pointed out that, for some LDPC codes, the hard-limiter used in many practical iterative decoders to provide numerical stability introduces a high error floor and should be avoided. Instead, an approximated computation is proposed to replace the hard-limiter.
CHAPTER 5

CONCLUSIONS

In this thesis, we investigated several topics in the design and analysis of graph-based codes.

We proposed to use permutation polynomial-based interleavers for turbo codes. Using properties of the polynomials, the performance of the turbo code can be associated with the coefficients of the polynomials using some simple methods. Then a search for good polynomials can be performed with relatively low complexity. Using this method, we found some permutation polynomial-based interleavers that outperforms the commonly used S-random interleavers.

We also considered an interleaver design for LDPC codes. An MLLCS-based interleaver is used. Since the girth of the graph is one of the performance-deciding characteristic of the code, we developed a sufficient condition that guarantees the girth to be larger than 4 for any degree distributions. We also developed another set of sufficient conditions that can avoid even larger loops.

To analyze trapping sets of LDPC codes, we proposed an approximated linear system model-based method. After using some approximations, the message evolution in the trapping set is linearized. The triggering probability of the trapping set can be estimated by investigating the zero input and zero state responses of the system, which are both closely
related to the eigen-decomposition of a matrix $A$ in the linear system. Based on the analysis result, we proposed a way to modify the iterative decoder for regular LDPC codes that can greatly decrease the error floor. For an irregular LDPC code, an approximated upper bound for the error floor can be found.
APPENDIX A

PROOF OF THEOREM 2.8:

Proof: Induction is extensively used in the proof.

Theorem 2.8 claims that, for fixed $N$, $P(x)$ and $s$:

1. $\Delta_k(i)$ has the same order for all $i$,

2. $o_{\Delta_0} = o_s$, and

3. $o_{\Delta_k} = o_{\Delta_{k-1}} + o_b + o_{2k}$, for $k > 0$, \hspace{1cm} (A.1)

which implies

$$o_{\Delta_k} = ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s$$, for $k > 0$. \hspace{1cm} (A.2)

These can be proved by using induction on $k$.

As the first step of the induction, we need to show that $\Delta_0(m)$ has an order $o_s$ and $\Delta_1(m)$ has an order $o_b + o_s + o_2$. By definition,

$$\Delta_1(i) = \Delta_0(i+1) - \Delta_0(i)$$

$$= [y_{i+1} - (i + 1)] - (y_i - i)$$

$$= y_{i+1} - y_i - 1,$$

108
we have

\[ y_{i+1} = \Delta_1(i) + y_i + 1. \]  

(A.4)

Applying this to \( P(y_{i+1}) - P(i + 1) = s = P(y_i) - P(i) \), we have

\[ E_1(i) \triangleq (a + 2by_i + 2b)\Delta_1(i) + b\Delta_1^2(i) + 2b\Delta_0(i) \]

\[ = 0, \]  

(A.5)

which is a permutation polynomial of \( \Delta_1(i) \) and \( \Delta_1(i) \) has the same order as \( 2b\Delta_0(i) \).

Since \( \Delta_0(0) = y_0 \) and \( P(y_0) = ay_0 + by_0^2 = s \) is a permutation polynomial, the order of \( \Delta_0(0) \) is \( o_s \). Then by (A.5), the order of \( \Delta_1(0) \) is \( o_b + o_s + o_2 \). From the definition of \( \Delta_1(i) \), we have

\[ \Delta_1(i) = \Delta_0(i + 1) - \Delta_0(i). \]  

(A.6)

As \( \Delta_0(0) \) has an order \( o_s \) and \( \Delta_1(0) \) has an order \( o_b + o_s + o_2 \), which is strictly larger than \( o_s \), \( \Delta_0(1) \) must also have order \( o_s \). By alternately using (A.5) and (A.6), we can show that all \( \Delta_0(i) \) has an order \( o_s \) and all \( \Delta_1(i) \) has an order \( o_b + o_s + o_2 \).

Next, we need to find the order of \( \Delta_{k+1}(i) \) using the order of \( \Delta_0(i) \), \( \Delta_1(i) \), \ldots, \( \Delta_k(i) \).

Starting from (A.5), we can define \( E_k(i) \triangleq E_{k-1}(i + 1) - E_{k-1}(i) \) and use \( y_{i+1} = \Delta_1(i) + y_i + 1, \Delta_0(i + 1) = \Delta_0(i) + \Delta_1(i), \Delta_1(i + 1) = \Delta_1(i) + \Delta_2(i), \ldots, \) we can inductively show that \( E_k(i) \) has the form of

\[ E_k(i) = \left[ a + 2kb + 2by_i + \sum_{j=1}^{k-1} 2g_{k,j}b\Delta_j(i) \right] \Delta_k(i) \]

\[ + b\Delta_k^2(i) + C(k, i), \]  

(A.7)

where \( C(k, i) \) is a term that contains \( \Delta_1(i), \ldots, \Delta_{k-1}(i) \), but not \( \Delta_k(i) \) and \( g_{k,j} \)'s are integers defined for \( k \geq 2 \) and \( j = 1, \ldots, k - 1 \). For convenience, for all other \( k \) and \( j \), we define \( g_{k,j} = 0 \).
We can verify that (A.5) satisfies (A.7) with $C(1, i) = 2b\Delta_0(i)$. For $E_{k+1}(i)$, we have

\[E_{k+1}(i) = E_k(i + 1) - E_k(i)\]

\[= \left[ a + 2kb + 2by_{i+1} + \sum_{j=1}^{k-1} 2g_{k,j}b\Delta_j(i + 1) \right] \Delta_k(i + 1)\]

\[+ b\Delta_k^2(i + 1) + C(k, i + 1)\]

\[- \left[ a + 2kb + 2by_i + \sum_{j=1}^{k-1} 2g_{k,j}b\Delta_j(i) \right] \Delta_k(i)\]

\[- b\Delta_k^2(i) - C(k, i)\]

\[= \left[ a + 2(k + 1)b + 2by_i + 2b\Delta_1(i) + 2b\Delta_k(i) \right.\]

\[+ \sum_{j=1}^{k-1} 2g_{k,j}(\Delta_{j+1}(i) + \Delta_j(i)) \Delta_{k+1}(i)\]

\[+ b\Delta_{k+1}^2(i) + \left[ 2b\Delta_1(i) + 2b\right.\]

\[+ \sum_{j=1}^{k-1} 2g_{k,j}\Delta_{j+1}(i) \Delta_k(i) + C(k, i + 1) - C(k, i)\]

\[= \left[ a + 2(k + 1)b + 2by_i + \sum_{j=1}^{k} 2g_{k+1,j}b\Delta_j(i) \right.\]

\[+ b\Delta_{k+1}^2(i) + C(k + 1, i),\]

where $g_{2,1} = 2$, and

\[
\begin{align*}
\left\{ 
\begin{array}{l}
g_{k+1,1} = g_{k,1} + 1, \\
g_{k+1,j} = g_{k,j} + g_{k,j-1} \quad \text{for } j = 2, 3, \ldots, k - 1, \\
g_{k+1,k} = g_{k,k-1} + 1
\end{array}
\right.
\]

(A.9)

It can be verified that $g_{k,j} = \binom{k}{j}$.

The $C(k + 1, i)$ term in (A.8) satisfies

\[C(k + 1, i) = \left[ 2b + \sum_{j=0}^{k-1} 2\binom{k}{j}b\Delta_{j+1}(i) \right] \Delta_k(i) + C(k, i + 1) - C(k, i).\]

(A.10)

It contains $\Delta_1(i), \ldots, \Delta_k(i)$, but not $\Delta_{k+1}(i)$. Thus we proved that (A.7) is true for all $k$. 

110
By regarding (A.7) as a second degree polynomial of $\Delta_k(i)$, it is easy to see that all $E_k(i)$ are permutation polynomials and $\Delta_k(i)$ has the same order as the term $C(k, i)$.

Before finding the orders of all $C(k, i)$’s, we need to derive a compact expression for $C(k, i)$.

From (A.5), $C(1, i) = 2b\Delta_0(i)$. We can use (A.10) to inductively compute $C(k, i)$. For example,

\[
C(2, i) = 2b\Delta_1^2(i) + 4b\Delta_1(i), \quad (A.11)
\]
\[
C(3, i) = 6b\Delta_2(i) + 6b\Delta_1(i)\Delta_2(i) + 6b\Delta_2^2(i). \quad (A.12)
\]

\[
\vdots
\]

It can be observed that

\[
C(k, i) = 2kb\Delta_{k-1}(i) + \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} C^{(k)}_{m,n} b\Delta_m(i)\Delta_n(i), \quad (A.13)
\]

where $C^{(k)}$ forms a $(k-1)\times(k-1)$ matrix. If we define $\Delta^{(k)}(i) = [\Delta_1(i), \Delta_2(i), \ldots, \Delta_{k-1}(i)]$, equation (A.13) can be written in a matrix form.

\[
C(k, i) = 2kb\Delta_{k-1}(i) + \Delta^{(k)} C^{(k)} (\Delta^{(k)})^T. \quad (A.14)
\]

Inductively, we can show that $C^{(k)}_{m,n} = 0$ when $m + n < k$ and for other $m$ and $n$, if we define $\binom{x}{y} = 0$ and if $x < y$,

\[
C^{(k)}_{m,n} = \binom{k}{m} \binom{m}{k-n}. \quad (A.15)
\]
It can be verified that \( C^{(2)} = [2] \) and \( C^{(3)} = \begin{bmatrix} 0 & 3 \\ 3 & 6 \end{bmatrix} \) satisfy (A.13). Applying (A.13) in (A.10),

\[
C(k + 1, i) = \left[ 2b + \sum_{j=0}^{k-1} k \binom{k}{j} b \Delta_{j+1}(i) \right] \Delta_k(i) + 2kb\Delta_{k-1}(i + 1) \\
+ \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} C^{(k)}_{m,n} b \Delta_m(i + 1) \Delta_n(i + 1) - 2kb\Delta_k(i) \\
- \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} C^{(k)}_{m,n} b \Delta_m(i) \Delta_n(i)
\]

\[
= 2(k + 1)b\Delta_k(i) + \sum_{m=1}^{k-1} \sum_{n=1}^{k-1} bC^{(k)}_{m,n} \left( \Delta_{m+1}(i) \Delta_n(i) \right. \\
+ \Delta_m(i) \Delta_{n+1}(i) + \Delta_{m+1}(i) \Delta_{n+1}(i) \left. \right) \\
+ 2\sum_{j=0}^{k-1} k \binom{k}{j} b \Delta_{j+1}(i) \Delta_k(i) \\
= 2(k + 1)b\Delta_k(i) \\
+ \Delta^{(k+1)}(D_1 + D_2 + D_3 + D_4 + D_5)(\Delta^{(k+1)})^T
\]

\[
= 2(k + 1)b\Delta_k(i) + \Delta^{(k)}C^{(k+1)}(\Delta^{(k)})^T,
\]

where \( D_h \)'s are \( k \times k \) matrices for \( h = 1, \ldots, 5 \). Define \( g^{(k)} \) to be a \( 1 \times k \) vector such that

\[
g^{(k)} = \begin{pmatrix} \binom{k}{0}, \binom{k}{1}, \ldots, \binom{k}{k-1} \end{pmatrix},
\]

Then

\[
D_1 = \begin{pmatrix} 0_{(k-1)\times 1} & 0_{1\times (k-1)} \\ C^{(k)} & 0 \end{pmatrix}, \\
D_2 = \begin{pmatrix} 0_{(k-1)\times 1} & 0 \end{pmatrix}, \\
D_3 = \begin{pmatrix} 0 & 0_{1\times (k-1)} \\ 0_{(k-1)\times 1} & C^{(k)} \end{pmatrix}, \\
D_4 = \begin{pmatrix} 0_{k\times (k-1)} & g^T \end{pmatrix}, \\
D_5 = \begin{pmatrix} 0_{(k-1)\times k} \\ g \end{pmatrix}
\]

(A.17)
More precisely,
\[
\begin{align*}
C_{m,n}^{(k+1)} &= 0, & m + n < k + 1 \\
C_{k,1}^{(k+1)} &= C_{k-1,1}^{(k)} + 1, \\
C_{1,k}^{(k+1)} &= C_{1,k-1}^{(k+1)} + 1, \\
C_{k,n}^{(k+1)} &= C_{k-1,n-1}^{(k)} + C_{k,n-1}^{(k)} + \binom{k}{n-1}, & 1 < n < k \\
C_{m,k}^{(k+1)} &= C_{m-1,k-1}^{(k)} + C_{m,k-1}^{(k)} + \binom{k}{m-1}, & 1 < m < k \\
C_{k,k}^{(k+1)} &= C_{k-1,k-1}^{(k)} + 2k, \\
C_{m,n}^{(k+1)} &= C_{m-1,n}^{(k)} + C_{m,n-1}^{(k)} + C_{m-1,n-1}^{(k)}, & \text{otherwise.}
\end{align*}
\]

(A.18)

It is straightforward to check that (A.15) satisfies (A.18). So we proved that \( C(k,i) \) satisfies (A.13).

Now we can find the order of all \( C(k,i) \)'s. Note that at this step of the induction, the orders of \( \Delta_0(i), \Delta_1(i), \ldots, \Delta_{k-1}(i) \) are known and satisfy (A.1) and (A.2).

We claim that the order of \( C(k,i) \) is decided by the first term of (A.13), \( 2kb\Delta_{k-1}(i) \).

From the previous step in the induction, we know \( o_{\Delta_{k-1}} = (k-1)o_b + (k-1)o_2 + \sum_{n=1}^{k-1} o_n + o_s \). Then the first term of (A.13) has an order \( ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s \). It is enough to show that all other terms have larger order. Consider \( n \)th column of \( C^{(k)} \), the first non-zero element is \( C_{k-n,n}^{(k)} \). For \( m > k - n \), \( \frac{C_{m,n}^{(k)}}{C_{k-n,n}^{(k)}} = \frac{n!}{(k-m)!(n-k+m)!} = \binom{n}{n-k+m} \) is an integer. This implies that \( C_{m,n}^{(k)} \) is a multiple of \( C_{k-n,n}^{(k)} \) and \( o_{C_{m,n}^{(k)}} \geq o_{C_{k-n,n}^{(k)}} \). Also, since \( m < k \) and \( k - n < k \), we know the order of \( \Delta_m \) and \( \Delta_{k-n} \) by the previous steps of the induction and \( o_{\Delta_m} \gg o_{\Delta_{k-n}} \) for \( m > k - n \). Combining these two results, we can see that in (A.13), the order of term \( C_{m,n}^{(k)} b\Delta_m(i) \Delta_n(i) \) is strictly larger than the order of \( C_{k-n,n}^{(k)} b\Delta_{k-n}(i) \Delta_n(i) \) for \( m > k - n \). Then, in order to find the order of \( C(k,i) \), we only need to consider the first term in (A.13) and terms in the form \( C_{k-n,n}^{(k)} b\Delta_{k-n}(i) \Delta_n(i) \) for \( n = 1, \ldots, k-1 \). By (A.2), the first term has order \( ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s \). Term in the form of \( C_{k-n,n}^{(k)} b\Delta_{k-n}(i) \Delta_n(i) \) has an order \( (k+1)o_b + ko_2 + \sum_{n=1}^{k} o_n + 2o_s \), which is strictly larger than the order of the
first term. Thus, the order of the first term in (A.13) is the order of $\Delta_k(i)$, which is

$$o_{\Delta_k(i)} = ko_b + ko_2 + \sum_{n=1}^{k} o_n + o_s, \quad \forall i. \quad (A.19)$$

This concludes our proof.
APPENDIX B

PROOF OF THEOREM 3.1:

Proof: A length 4 loop is shown in Figure B.1. Note that a parallel branch is a special length 4 loop such that \( l_1 = l_2 + t_2 \) and \( l_2 = l_1 + t_1 \). We only need to develop the sufficient condition to avoid length 4 loops. Assume \( |l'_1 - l_1| \leq d_\lambda - 1 \), \( |l'_2 - l_2| \leq d_\lambda - 1 \) and \( |A_{l_2} - A_{l_1}| \leq d_\rho - 1 \). In order for no length 4 loop to exist, it is sufficient to show that \( |A_{l_2'} - A_{l_1'}| \geq d_\rho \). By triangular inequality, it is further enough to show that

\[
|(A_{l_2'} - A_{l_1'}) - (A_{l_2} - A_{l_1})| \geq 2d_\rho - 1.
\]

(B.1)

Figure B.1: Graph representations of a length 4 loop.
Using \( a = a_1a_0 + 1 \), and expanding terms in the form of \((a_1a_0 + 1)^n\),

\[
(A_{l_2} - A_{l_1}) - (A_{l_2} - A_{l_1}) = b \left[ \sum_{i=2}^{l_2+n_0} \binom{l_2 + n_0}{i} a_1^{i-1}a_0^{i-1} - \sum_{i=2}^{l_1+n_0} \binom{l_1 + n_0}{i} a_1^{i-1}a_0^{i-1} \right. \\
\left. - \sum_{i=2}^{l_2+n_0} \binom{l_2 + n_0}{i} a_1^{i-1}a_0^{i-1} + \sum_{i=2}^{l_1+n_0} \binom{l_1 + n_0}{i} a_1^{i-1}a_0^{i-1} \right] + b(l_2' - l_1' - l_2 + l_1).
\]

(B.2)

The first term is a multiple of \( a_0 \) modulo \( M \). In the second term, let \( \Delta = l_2' - l_1' - l_2 + l_1 \), then \(-2(d_\lambda - 1) \leq \Delta \leq 2(d_\lambda - 1)\). When \( \Delta \neq 0 \), a sufficient condition for a length four loop not to be formed is \( 2d_\rho - 1 \leq (b\Delta \mod a_0) \leq a_0 - (2d_\rho - 1) \). Note that this condition implies that \( a_0 \geq 4d_\rho - 2 \). Due to symmetry, it is enough to consider \( \Delta = 1, 2, \ldots, 2(d_\lambda - 1) \). This is the first condition in the theorem.

When \( \Delta = 0 \), \( l_2' - l_2 = l_1' - l_1 \) and \( b(l_2' - l_1' - l_2 + l_1) = 0 \). Since the first term in (B.2) is a multiple of \( a_0 \) and \( a_0 \geq 4d_\rho - 2 \) by the previous condition, the only case that

\[
| (A_{l_2} - A_{l_1}) - (A_{l_2} - A_{l_1}) | < 2d_\rho - 1 \text{ is when } |(A_{l_2} - A_{l_1}) - (A_{l_2} - A_{l_1})| = 0.
\]

We need to find the condition that this can not happen, i.e., \( A_{l_2} - A_{l_1} \neq A_{l_2} - A_{l_1} \) when \( l_2' - l_2 = l_1' - l_1 = l \).

Since

\[
A_{l_2} - A_{l_1} = \frac{a^{l_2+n_0} - a^{l_1+n_0}}{a-1} b = \frac{a^{l_2+l+n_0} - a^{l_1+l+n_0}}{a-1} b = (A_{l_2} - A_{l_1})a^l,
\]

(B.3)

\[
(A_{l_2} - A_{l_1}) - (A_{l_2} - A_{l_1}) = (a^l - 1)(A_{l_2} - A_{l_1}).
\]

Without loss of generality, we can assume \( A_{l_2} - A_{l_1} = c, 1 \leq c \leq d_\rho - 1 \). We need \((a^l - 1)c \neq 0 \mod M\) for all \( l = [1, 2, \ldots, d_\lambda - 1] \), which is the second condition in the theorem. This concludes our proof.

\[\square\]
APPENDIX C

PROOF OF THEOREM 3.2:

Proof: A general length $2m$ loop is shown in Figure C.1. For this loop, we can write $m$

![Figure C.1: A length 2m loop](image)

equations.

\[
\frac{a^{l_2} - a^{l_1}}{a - 1} b = s_1 \quad (C.1.1)
\]

\[
\ldots
\]

\[
\frac{a^{l_{i+1}} - a^{l_{i-1}+t_{i-1}}}{a - 1} b = s_i \quad (C.1.i)
\]

\[
\ldots
\]

\[
\frac{a^{l_m+t_m} - a^{l_{m-1}+t_{m-1}}}{a - 1} b = s_m \quad (C.1.m)
\]
After some straightforward manipulations, equations in (C.1) can be combined as

\[ A_{l_1}(a^{\Sigma_1(t)} - a^{\Sigma_2(t)}) = s_1 a^{\Sigma_2(t)} - s_2 a^{\Sigma_3(t)} + s_3 a^{\Sigma_4(t)} - \cdots - \frac{a^{\Sigma_1(t)} - a^{\Sigma_2(t)}}{a - 1} b, \]  

which is a function of \( l_1 \) with parameters \( t_i \) and \( s_i, i = 1, \ldots, m \).

In order for no length \( 2m \) loop to exist, it is sufficient that for any \( t_i \) and \( s_i \), there is no solution of \( l_1 \) for (C.2), which is summarized in the statement of the theorem.
APPENDIX D

PROOF OF THEOREM 3.3:

Proof: We start from (B.2). Equation (B.2) can be rewritten as

\[
(A_l' - A_l) - (A_l' - A_l) = b \left[ \sum_{i=3}^{l_2+n_0} \binom{l_2 + n_0}{i} a_1^{i-1} a_0^{i-1} - \sum_{i=3}^{l_1+n_0} \binom{l_1 + n_0}{i} a_1^{i-1} a_0^{i-1} \right] - \sum_{i=3}^{l_2+n_0} \binom{l_2 + n_0}{i} a_1^{i-1} a_0^{i-1} + \sum_{i=3}^{l_1+n_0} \binom{l_1 + n_0}{i} a_1^{i-1} a_0^{i-1} 
+ b \left[ \binom{l_2 + n_0}{2} - \binom{l_1 + n_0}{2} - \binom{l_2 + n_0}{2} - \binom{l_1 + n_0}{2} \right] a_1 a_0 + b[(l_2 - l_2) - (l_1 - l_1)].
\] (D.1)

The first term is a multiple of \(a_0^2\). When \(\Delta = (l_2' - l_2) - (l_1' - l_1) = 2(d_\lambda - 1)\), we must have \(l_1 = k_1 d_\lambda + d_\lambda - 1, l_1' = k_1 d_\lambda, l_2 = k_2 d_\lambda\) and \(l_2' = k_2 d_\lambda + d_\lambda - 1\) for some \(k_1\) and \(k_2\). Then the second term of (D.1) becomes

\[
b[(k_1 + k_2) d_\lambda (d_\lambda - 1) + 2n_0 (d_\lambda - 1) + (d_\lambda - 1) (d_\lambda - 2)] a_1 a_0. \] (D.2)

If \(\frac{M}{a_0}\) is even, equation (D.2) is a multiple of \(2a_0\) and so is the first term of (D.1). Then for \(\Delta = 2(d_\lambda - 1)\), the sufficient condition in Theorem 3.1 can be reduced to \(2d_\rho - 1 \leq 2(d_\lambda - 1) b \mod 2a_0 \leq 2a_0 - (2d_\rho - 1)\), which is equivalent to \(d_\rho \leq (d_\lambda - 1) b \mod a_0 \leq a_0 - d_\rho\). This is a weaker condition than \(2d_\rho - 1 \leq \Delta b \mod a_0 \leq a_0 - (2d_\rho - 1)\) for \(\Delta = (d_\lambda - 1),\)
which is already included in Theorem 3.1. Thus when $\frac{M}{m_0}$ is even, we only need to check $\Delta = 1, 2, \ldots, 2d \lambda - 3$ in Theorem 3.1.
APPENDIX E

PROOF OF LEMMA 4.10:

Proof: We know $||v||^2 = \sum_{i=1}^{M} |v_i|^2$ and the value does not change under a permutation of elements in $v$. We also know that $||Av||$ does not change under a row permutation of $A$. We can permute columns and rows of $A$ to form the block diagonal matrix $A'$ and perform the same column permutation to $v$ to form $v'$. Then $||Av|| = ||A'v'||$. We only need to show $||A'v'|| \geq ||v'||$ for any vector $v'$.

Due to the block diagonal form of $A'$, we can consider each block separately. Pick one block of $A'$ and assume it is $m \times m$. We denote that block by $A_0$. Then $A_0$ is an all one matrix with the exception of the elements on the diagonal. It is sufficient to show that

$||A_0x|| \geq ||x||$, which can be expanded into the form

$|x_2 + x_3 + \cdots + x_m|^2 + |x_1 + x_3 + \cdots + x_m|^2 + \cdots + |x_1 + x_2 + \cdots + x_{m-1}|^2$

$\geq |x_1|^2 + |x_2|^2 + \cdots + |x_m|^2$  \hspace{1cm} (E.1)

First we consider the case that $x$ is a real vector. Now we need to show

$(x_2 + x_3 + \cdots + x_m)^2 + (x_1 + x_3 + \cdots + x_m)^2 + \cdots + (x_1 + x_2 + \cdots + x_{m-1})^2$

$\geq x_1^2 + x_2^2 + \cdots + x_m^2.$  \hspace{1cm} (E.2)

for real $x_i$'s. Define

$f(x_1, x_2, \ldots, x_m) = (x_2 + x_3 + \cdots + x_m)^2 + (x_1 + x_3 + \cdots + x_m)^2 + \cdots$

$+ (x_1 + x_2 + \cdots + x_{m-1})^2 - (x_1^2 + x_2^2 + \cdots + x_m^2)$

$= \sum_{i=1}^{m} (m-2)x_i^2 + \sum_{i,j,i \neq j} (m-2)x_ix_j$.  \hspace{1cm} (E.3)
Then
\[ \frac{\partial f}{\partial x_i} = 2(m - 2) \sum_{i=1}^{m} x_i. \]  
(E.4)

Since
\[ \frac{\partial^2 f}{\partial x_i^2} = 2(m - 2) > 0, \]  
(E.5)
the minimum point satisfies
\[ \sum_{i=1}^{m} x_i = 0. \]  
(E.6)

Therefore, \( x_1 = -\sum_{i=2}^{m} x_i \). Applying this back to (E.3), we have
\[ f(-\sum_{i=2}^{m} x_i, x_2, \ldots, x_m) = (m - 2) \left[ \sum_{i=1}^{m} x_i^2 + \sum_{i,j \neq j} x_i x_j \right] \]
\[ = \ldots = 0 \]  
(E.7)

So, \( f(x_1, x_2, \ldots, x_m) \geq 0 \).

Next, we extended the result to a general complex vector case and prove (E.1). A complex vector \( x \) can be represented by two real vectors \( y \) and \( z \) such that \( x = y + iz \).

Then (E.1) becomes
\[ (y_2 + y_3 + \cdots + y_m)^2 + (z_2 + z_3 + \cdots + z_m)^2 + (y_1 + y_3 + \cdots + y_m)^2 \]
\[ + (z_1 + z_3 + \cdots + z_m)^2 + \cdots + (y_1 + y_2 + \cdots + y_{m-1})^2 + (z_1 + z_2 + \cdots + z_{m-1})^2 \]
\[ \geq y_1^2 + z_1^2 + y_2^2 + z_2^2 + \cdots + y_m^2 + z_m^2 \]  
(E.8)

To show this, we only need to use the real case result in (E.2) twice for \( y \) and \( z \) respectively.

This concludes our proof.

\[ \square \]
APPENDIX F

PROOF OF THEOREM 4.13:

Proof: Using the physical interpretation of \( A^L \), it is not difficult to understand the following observations.

1. The sub-matrix formed by the first \( M_1 \) columns and rows of \( A^L \) is exactly \( (A')^L \), for any \( L \geq 1 \).

2. For \( L \) large enough, all downstream columns of \( A^L \) are zero vectors.

3. For \( L \) large enough, all upstream rows of \( A^L \) are zero vectors.

In the last two observations, \( L \) only needs to be larger than half the largest depth of external data nodes, the length of the path that connects the external data node back to the root of the trapping set.

Assume \( v_i \) is an eigenvector of \( A \) corresponding to a non-zero eigenvalue \( \lambda_i \). Then if branch \( j \) is an upstream branch, the \( j \)'th element in \( v_i \) is zero. This can be shown as follows.

Since \( Av_i = \lambda_i v_i \), for any \( L \geq 1 \), vector \( v_i \) is also the eigenvector of \( A^L \) corresponding to eigenvalue \( \lambda_i^L \). Then \( A^L v_i = \lambda_i^L v_i \). For \( L \) large enough, the elements in the \( j \)'th row of \( A^L \) are all zero. Then the \( j \)'th element in \( A^L v_i \) must be zero. Therefore, the \( j \)'th element in \( v_i \) is zero.

123
We have a similar result for the left-eigenvector. If \( u_i \) is a left-eigenvector of \( A \) corresponding to a non-zero eigenvalue \( \lambda_i \). Then if branch \( j \) is a downstream branch, the \( j \)'th element in \( u_i \) is zero. The proof is similar to the proof for the right-eigenvector using the third observation above.

Given an eigen-pair \( \lambda_i \) and \( v'_i \) for \( A' \), the eigen-pair of \( A \) is \( \lambda_i \) and \( v_i \), where \( v_i \) is formed by appending another vector \( v''_i \) to \( v'_i \), \( v_i = \begin{bmatrix} v'_i \\ v''_i \end{bmatrix} \). This can be shown as follows.

We can write \( A \) as
\[
A = \begin{bmatrix} A' & A_1 \\ A_2 & A_3 \end{bmatrix}.
\] (F.1)

As demonstrated in the derivation of Theorem 4.12, submatrices \( A_1, A_2 \) and \( A_3 \) have the following properties: The downstream columns in \( A_1 \) are zero vectors; The upstream rows of \( A_2 \) are zero vectors; The diagonal elements of \( A_3 \) are zeros; In a downstream column of \( A_3 \), all elements above the diagonal element are zeros; In an upstream row of \( A_3 \), all elements on the left of the diagonal element are zeros.

We only need to show that, given \( v'_i \), we can always find \( v''_i \) to form \( v_i \). Assume \( v_i \) to be the eigenvector. We can write
\[
\begin{bmatrix} A' & A_1 \\ A_2 & A_3 \end{bmatrix} \begin{bmatrix} v'_i \\ v''_i \end{bmatrix} = \lambda_i \begin{bmatrix} v'_i \\ v''_i \end{bmatrix}.
\] (F.2)

We can expand this and form
\[
A'v'_i + A_1v''_i = \lambda_i v'_i 
\] (F.3)
\[
A_2v'_i + A_3v''_i = \lambda_i v''_i.
\] (F.4)

Let the upstream elements in \( v''_i \) be zero. The downstream elements are still to be found. Then since the downstream columns in \( A_1 \) are zero vectors, (F.3) can be reduced to
\[
A'v'_i = \lambda_i v'_i.
\] (F.5)
This satisfies the condition that \( \lambda_i \) and \( v'_i \) form an eigen-pair of \( A' \). Given \( v'_i \), in (F.4), the term \( A_2 v'_i \) can be computed and the upstream elements in \( A_2 v'_i \) are zeros. We only need to show that we can solve

\[
A_2 v'_i = (\lambda_i I - A_3) v''_i
\]  

(F.6)

for the downstream elements of \( v''_i \). Note that since we have already assumed that the upstream elements of \( v''_i \) are zeros, it is not enough to show that \( \lambda_i I - A_3 \) is invertible.

It is known that the downstream elements in upstream rows of \( \lambda_i I - A_3 \) are zeros. Since the upstream elements in \( v''_i \) are zeros, the upstream elements of \( (\lambda_i I - A_3) v''_i \) are zeros. Then the upstream part for (F.6) is satisfied. Again, since the upstream elements in \( v''_i \) are zeros, the upstream columns of \( \lambda_i I - A_3 \) do not appear in (F.6). Therefore, we only need to show that the sub-matrix of \( \lambda_i I - A_3 \) formed by removing all upstream columns and rows is invertible. This is true, for the sub-matrix is a lower triangle matrix with diagonal elements \( \lambda_i \neq 0 \).

Using the above method, given the eigenvector \( v'_i \) for \( A' \), we can always compute \( v''_i \) and form \( v_i \) as an eigenvector for \( A \). Since we assume the Jordan canonical form of \( A' \) is diagonal, for an eigenvalue \( \lambda_i \) with multiplicity \( k \), there are \( k \) corresponding eigenvectors for matrix \( A' \). For each of these eigenvectors, we can append the vector and form \( k \) eigenvectors for \( A \). These \( k \) eigenvectors are independent for the sub-vectors formed by the first \( M_1 \) elements are independent. Therefore, the Jordan block for the eigenvalue \( \lambda_i \) is diagonal. The non-diagonal Jordan blocks are only possible for zero eigenvalues. This concludes our proof.

\[ \square \]
BIBLIOGRAPHY


[47] Jing Sun, Oscar Y. Takeshita, and Michael P. Fitz. On low-density parity-check codes constructed with maximum length linear congruential sequences. accepted for publication in IEEE Transactions on Communications.

