This Dissertation

entitled

PSEUDOCODEWORDS, EXPANDER GRAPHS, AND THE
ALGEBRAIC CONSTRUCTION OF LOW-DENSITY
PARITY-CHECK CODES

typeset with \texttt{nddiss2e} v1.0 (2004/06/15) on April 21, 2006 for

Christine A. Kelley

This \LaTeX\ class file conforms to the University of Notre Dame style guidelines established in Spring 2004. However it is still possible to generate a non-conformant document if the published instructions are not followed! Be sure to refer to the published Graduate School guidelines at \url{http://graduateschool.nd.edu} as well.

It is YOUR responsibility to ensure that the Chapter titles and Table caption titles are put in CAPS LETTERS. This class file does \textit{NOT} do that! This way, you have total control over how you want the symbols and sub-/superscripts in titles and captions look like.

\textit{This summary page can be disabled by specifying the nosummary option to the class invocation. (i.e., \texttt{\textbackslash documentclass[...],nosummary,...}\texttt{nddiss2e})}

\textbf{THIS PAGE IS NOT PART OF THE THESIS, BUT MUST BE TURNED IN TO THE PROOFREADER!}

\texttt{nddiss2e} documentation can be found at these locations:

\url{http://www.gsu.nd.edu}
\url{http://graduateschool.nd.edu}
PSEUDOCODEWORDS, EXPANDER GRAPHS, AND THE ALGEBRAIC CONSTRUCTION OF LOW-DENSITY PARITY-CHECK CODES

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Christine A. Kelley, B.S., M.S.

______________________________
Joachim Rosenthal, Director

Graduate Program in Mathematics
Notre Dame, Indiana
April 2006
In 1948, Claude Shannon provided a sound mathematical foundation for communication, and proved the existence of good codes. Low-density parity-check (LDPC) codes are a family of error-correcting codes that achieve reliable performance (i.e., arbitrarily low probability of decoding error) near the Shannon-limit when decoded iteratively via very efficient graph-based message-passing decoders, the complexity of which grows only linearly in the block-length. Despite their tremendous success, LDPC codes lack a strong theoretical foundation and few explicit constructions of them are known that outperform their random counterparts.

The first part of this thesis investigates how the structure of a graph representing an LDPC code affects the decoding performance. In particular, lower bounds on the minimum pseudocodeword weight, a key parameter in predicting iterative decoding performance, are obtained which link the performance of the code to the structural properties of its graph. One bound is a tree bound and is extended to generalized and $p$-ary LDPC codes. To this end, a suitable definition of pseudocodeword weight is derived for the $p$-ary symmetric channel. Moreover, the relationship between pseudocodeword weight distributions and different graph
representations is examined. Bounds on the minimum pseudocodeword weight are also presented for several classes of expander codes based on the eigenvalues of the adjacency matrix of the Tanner graph.

The second part of this thesis constructs LDPC codes having desirable properties. The first construction is combinatorial and uses permutations and mutually orthogonal Latin squares. One type results in the well-known projective-geometry-based LDPC codes. Another type yields a new family of codes having comparable parameters and pseudocodeword weights suitable for iterative decoding. The resulting performance of these codes is superior in comparison with that of comparable random LDPC codes, a feature attributed to the lack of low-weight bad pseudocodewords. For the second construction, unbalanced expander graphs are obtained by modifying the original zig-zag graph product to take bi-regular expander graphs as components, with suitable constraints on the degrees. Graphs resulting from the zig-zag and replacement product graphs are then used in the design of generalized LDPC codes having compact representation and pseudocodeword weights lower-bounded by the eigenvalue bounds.
This thesis is dedicated

To my father, Ronald Kelley,

and

In loving memory of my mother, Ute Kelley (1944-2005).
CONTENTS

FIGURES ................................................................. vii

TABLES ................................................................. xi

ACKNOWLEDGMENTS ..................................................... xii

CHAPTER 1: INTRODUCTION ............................................. 1
  1.1 Linear Block Codes .............................................. 5
  1.2 LDPC Codes and Tanner Graphs .................................. 7
  1.3 Communication Channels ........................................ 10
  1.4 Decoding Algorithms ............................................ 12
    1.4.1 Maximum-likelihood decoding ............................... 12
    1.4.2 Iterative decoding .......................................... 13
    1.4.2.1 Min-sum decoding ....................................... 15
    1.4.2.2 Sum-product decoding ................................... 16
    1.4.3 Linear programming decoding .............................. 17
  1.5 Thesis Overview ................................................. 18

CHAPTER 2: PSEUDOCODEWORDS ........................................ 20
  2.1 Computation Tree Interpretation ............................... 20
  2.2 Graph Covers Interpretation .................................... 24
    2.2.1 Polytope representation .................................. 26
  2.3 Relation to Stopping Sets ....................................... 27
  2.4 Irreducible Pseudocodewords ................................. 28
  2.5 Pseudocodewords and Iterative Decoding Behavior .......... 29
  2.6 Graph-Covers-Polytope Approximation ....................... 31
  2.7 Proofs .......................................................... 37

CHAPTER 3: BOUNDS ON MINIMAL PSEUDOCODEWORD WEIGHTS
  FOR THE BSC AND AWGN CHANNELS .............................. 39
  3.1 Pseudocodeword Weights ....................................... 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 Tree Bounds</td>
<td>41</td>
</tr>
<tr>
<td>3.3 Relation to Max-Fractional Weight</td>
<td>43</td>
</tr>
<tr>
<td>3.4 Bounds in Terms of Support Size and Maximum Component Value</td>
<td>44</td>
</tr>
<tr>
<td>3.5 Proofs</td>
<td>46</td>
</tr>
<tr>
<td>CHAPTER 4: PSEUDOCODEWORD WEIGHTS OF NON-BINARY LDPC CODES</td>
<td>53</td>
</tr>
<tr>
<td>4.1 ( p )-ary LDPC Codes</td>
<td>53</td>
</tr>
<tr>
<td>4.2 Pseudocodewords of ( p )-ary Codes</td>
<td>55</td>
</tr>
<tr>
<td>4.2.1 Derivation of pseudocodeword weight for ( p )-ary LDPC codes on the ( p )-ary symmetric channel</td>
<td>57</td>
</tr>
<tr>
<td>4.3 Tree Bound on ( w_{\text{min}}^{\text{PSC}} )</td>
<td>59</td>
</tr>
<tr>
<td>4.4 Tree Bound on ( w_{\text{min}}^{\text{AWGN}} )</td>
<td>60</td>
</tr>
<tr>
<td>4.5 Proofs</td>
<td>60</td>
</tr>
<tr>
<td>CHAPTER 5: STRUCTURE OF PSEUDOCODEWORDS</td>
<td>67</td>
</tr>
<tr>
<td>5.1 Structure of the Pseudocodeword Vectors</td>
<td>67</td>
</tr>
<tr>
<td>5.2 Examples</td>
<td>73</td>
</tr>
<tr>
<td>5.3 Pseudocodeword Sets With Respect To Different Channels</td>
<td>79</td>
</tr>
<tr>
<td>5.4 Proofs</td>
<td>81</td>
</tr>
<tr>
<td>CHAPTER 6: GRAPH REPRESENTATIONS AND PSEUDOCODEWORD WEIGHT DISTRIBUTIONS</td>
<td>90</td>
</tr>
<tr>
<td>6.1 Case Study I: The [7, 4, 3] Hamming Code</td>
<td>90</td>
</tr>
<tr>
<td>6.2 Case Study II: The [15, 11, 3] Hamming Code</td>
<td>95</td>
</tr>
<tr>
<td>6.3 Conclusions</td>
<td>96</td>
</tr>
<tr>
<td>CHAPTER 7: LIFT DEGREES</td>
<td>102</td>
</tr>
<tr>
<td>7.1 Realizing Pseudocodewords</td>
<td>102</td>
</tr>
<tr>
<td>7.2 Proofs</td>
<td>105</td>
</tr>
<tr>
<td>CHAPTER 8: TREE-BASED CONSTRUCTION OF LDPC CODES</td>
<td>107</td>
</tr>
<tr>
<td>8.1 Preliminaries</td>
<td>108</td>
</tr>
<tr>
<td>8.1.1 Permutations</td>
<td>108</td>
</tr>
<tr>
<td>8.1.2 Mutually orthogonal Latin squares (MOLS)</td>
<td>109</td>
</tr>
<tr>
<td>8.2 Tree-based Construction: Type 1</td>
<td>110</td>
</tr>
<tr>
<td>8.2.1 Type 1-A</td>
<td>111</td>
</tr>
<tr>
<td>8.2.2 Type I-B</td>
<td>113</td>
</tr>
<tr>
<td>8.3 Tree-based Construction: Type II</td>
<td>117</td>
</tr>
<tr>
<td>8.3.1 Type II, ( \ell = 3 )</td>
<td>118</td>
</tr>
<tr>
<td>8.3.2 Relation to finite geometry codes</td>
<td>120</td>
</tr>
<tr>
<td>8.3.3 Type II, ( \ell = 4 )</td>
<td>123</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>8.3.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>8.4</td>
<td>Simulation Results</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Performance with min-sum iterative decoding</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Performance of Type I-B and Type II LDPC codes with sum-product iterative decoding</td>
</tr>
<tr>
<td>8.5</td>
<td>Summary of Constructions</td>
</tr>
<tr>
<td>8.6</td>
<td>Proofs</td>
</tr>
<tr>
<td>9.1</td>
<td>$p$-ary Type I-B LDPC codes</td>
</tr>
<tr>
<td>9.2</td>
<td>$p$-ary Type II LDPC Codes</td>
</tr>
<tr>
<td>9.2.1</td>
<td>$p$-ary finite geometry codes</td>
</tr>
<tr>
<td>9.3</td>
<td>Performance of $p$-ary Type I-B and Type II LDPC Codes over the $p$-ary Symmetric Channel</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Tables of code parameters</td>
</tr>
<tr>
<td>9.4</td>
<td>Proofs</td>
</tr>
<tr>
<td>10.1</td>
<td>Expander Graphs</td>
</tr>
<tr>
<td>10.2</td>
<td>Graph Products</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Tensor product</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Replacement product</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Zig-zag product</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Generalized zig-zag product</td>
</tr>
<tr>
<td>10.3</td>
<td>Properties of the Product Graphs</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Degree</td>
</tr>
<tr>
<td>10.3.2</td>
<td>Expansion</td>
</tr>
<tr>
<td>10.3.3</td>
<td>Diameter and girth</td>
</tr>
<tr>
<td>10.3.4</td>
<td>Remarks</td>
</tr>
<tr>
<td>10.4</td>
<td>LDPC codes from Zig-zag Product Graphs</td>
</tr>
<tr>
<td>10.5</td>
<td>Existing Minimum Distance Bounds</td>
</tr>
<tr>
<td>10.6</td>
<td>Performance of Zig-zag LDPC Codes</td>
</tr>
<tr>
<td>10.7</td>
<td>Conclusions</td>
</tr>
<tr>
<td>10.8</td>
<td>Proofs</td>
</tr>
<tr>
<td>11.1</td>
<td>Distance Bounds For Expander Codes</td>
</tr>
<tr>
<td>11.2</td>
<td>Extension to Stopping Number</td>
</tr>
<tr>
<td>11.3</td>
<td>Extension to Minimum Pseudocode Weight</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHAPTER 9: $p$-ARY FINITE GEOMETRY AND TREE-BASED LDPC CODES</td>
</tr>
<tr>
<td>11.1</td>
<td>Distance Bounds For Expander Codes</td>
</tr>
<tr>
<td>11.2</td>
<td>Extension to Stopping Number</td>
</tr>
<tr>
<td>11.3</td>
<td>Extension to Minimum Pseudocode Weight</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
11.4 A Parity-oriented Bound on Pseudocodeword Weight .......... 192
11.5 Pseudocodeword Properties of Zig-Zag LDPC Codes .......... 192
11.6 Proofs .................................................. 193

CHAPTER 12: CONCLUSIONS .................................... 201

BIBLIOGRAPHY .................................................. 204
### FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A Communication System.</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Channel coding.</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Tanner graph of the parity check matrix $H$.</td>
<td>9</td>
</tr>
<tr>
<td>1.4</td>
<td>Binary Erasure Channel (BEC).</td>
<td>10</td>
</tr>
<tr>
<td>1.5</td>
<td>Binary Symmetric Channel (BSC).</td>
<td>11</td>
</tr>
<tr>
<td>1.6</td>
<td>Additive White Gaussian Noise (AWGN) Channel.</td>
<td>11</td>
</tr>
<tr>
<td>1.7</td>
<td>Maximum Likelihood Decoder.</td>
<td>13</td>
</tr>
<tr>
<td>1.8</td>
<td>Message passing in iterative decoding.</td>
<td>13</td>
</tr>
<tr>
<td>2.1</td>
<td>A graph and its computation tree after two iterations of message passing.</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>An assignment corresponding to a codeword.</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>An assignment corresponding to a pseudocodeword.</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>A pseudocodeword in the base graph (or a valid codeword in a lift).</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>A stopping set $S = {v_0, v_1, v_3, v_5}$ in $G$.</td>
<td>28</td>
</tr>
<tr>
<td>2.6</td>
<td>Example of a reducible pseudocodeword $p$.</td>
<td>29</td>
</tr>
<tr>
<td>2.7</td>
<td>A Tanner graph and computation tree (CT) for the $[4,1,4]$ repetition code.</td>
<td>32</td>
</tr>
<tr>
<td>2.8</td>
<td>Modified Tanner graph and CT for the $[4,1,4]$ repetition code.</td>
<td>35</td>
</tr>
<tr>
<td>2.9</td>
<td>Modified Tanner graph and CT for the $[4,1,4]$ repetition code.</td>
<td>35</td>
</tr>
<tr>
<td>2.10</td>
<td>Performance of different representations of $[4,1,4]$ repetition code over</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>the binary input additive white Gaussian noise channel (BI-AWGNC).</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Tanner graph with $t$-value equal to 2.</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>A $p$-ary symmetric channel.</td>
<td>56</td>
</tr>
</tbody>
</table>
8.3 Tree construction of Type I-B LDPC code. (Shaded nodes are imaginary nodes and dotted lines are imaginary lines.) 116
8.4 Type I-B LDPC constraint graph having degree $d = 4$ and girth $g = 6$. 116
8.5 Tree construction of girth 6 Type II ($\ell = 3$) LDPC code. 121
8.6 Type II LDPC constraint graph having degree $d = 4$ and girth $g = 6$. 121
8.7 Tree construction of girth 8 Type II ($\ell = 4$) LDPC code. 127
8.8 Type II LDPC constraint graph having degree $d = 3$ and girth $g = 8$. (Shaded nodes highlight a minimum weight codeword.) 127
8.9 Performance of Type I-A versus Random LDPC codes on the BI-AWGNC with min-sum iterative decoding. 131
8.10 Performance of Type I-B versus Random LDPC codes on the BI-AWGNC with min-sum iterative decoding. 132
8.11 Performance of Type II $\ell = 3$ versus Random LDPC codes on the BIAWGNC with min-sum iterative decoding. 132
8.12 Performance of Type II $\ell = 4$ versus Random LDPC codes on the BIAWGNC with min-sum iterative decoding. 133
8.13 Performance of Type I-B versus Random LDPC codes on the BI-AWGNC with sum-product iterative decoding. 133
8.14 Performance of Type II $\ell = 3$ versus Random LDPC codes on the BIAWGNC with sum-product iterative decoding. 134
8.15 Performance of Type II $\ell = 4$ versus Random LDPC codes on the BIAWGNC with sum-product iterative decoding. 134
9.1 Performance of Type I-B versus Random 3-ary LDPC codes on the 3-ary symmetric channel with sum-product iterative decoding. 145
9.2 Performance of Type II $\ell = 3$ versus Random 3-ary LDPC codes on the 3-ary symmetric channel with sum-product iterative decoding. 146
9.3 Performance of Type II $\ell = 4$ versus Random 3-ary LDPC codes on the 3-ary symmetric channel with sum-product iterative decoding. 147
10.1 Tensor product of two graphs. 155
10.2 Replacement product of two graphs. 156
10.3 Zig-Zag product of two graphs. 159
10.4 Zig-Zag product of two unbalanced bipartite graphs. 160
10.5 LDPC codes from zig-zag product graphs based on Example 10.4.1. 170
10.6 LDPC codes from zig-zag product graphs based on Example 10.4.2. 171
10.7 Performance of LDPC codes from zig-zag product graphs based on
Example 10.4.3. .................................................. 172
11.1 Expander code: Case A ............................................. 184
11.2 Expander code: Case B ............................................. 185
11.3 Expander code: Case C ............................................. 185
11.4 Expander code: Case D ............................................. 186
TABLES

8.1 PERMUTATIONS FOR TYPE I-A CONSTRUCTION. . . . . . . 112
8.2 THE FUNCTION $f$ FOR THE TYPE II $\ell = 4$ CONSTRUCTION. 126
9.1 SUMMARY OF TYPE I-A CODE PARAMETERS. . . . . . . . . 146
9.2 SUMMARY OF TYPE I-B CODE PARAMETERS. . . . . . . . . 148
9.3 SUMMARY OF TYPE II, $\ell = 3$ CODE PARAMETERS. . . . . . 149
9.4 SUMMARY OF TYPE II, $\ell = 4$ CODE PARAMETERS. . . . . . 149
ACKNOWLEDGMENTS

There are many people I would like to acknowledge who helped make this work possible. First, I would like to thank my Ph.D. supervisor, Professor Joachim Rosenthal, for his advice and expertise during the duration of my graduate studies. My interest in coding theory originated from our discussions.

Secondly, I am very grateful to Dr. Deepak Sridhara, my friend and colleague, who has provided unlimited professional and emotional support over the last few years, particularly when my mother was sick. He enabled me to remain focused on work and has been an invaluable resource for me. In addition, I would like to thank him for running the simulations that are present in this thesis.

I would also like to thank the coding theory professors in the Department of Electrical Engineering at Notre Dame, especially Professor Daniel Costello for all of his help and useful suggestions. I am also grateful to Dr. Pascal Vontobel for his useful comments, insights, and suggestions on papers, as well as the anonymous reviewers for their comments. I would also like to thank my undergraduate advisor, Professor Rob Beezer, at the University of Puget Sound, for his exciting Calculus class which got me interested in mathematics, and for his encouragement and advice.

Moreover, I would like to thank Professor Mark Alber and Professor Andrew Sommese for their time spent on my defense committee and their suggestions.

For financial support, I would like to thank the Department of Mathematics
and the Center for Applied Mathematics at the University of Notre Dame, as well as ITR Grant no. CCR-0205310. I am also grateful to the Department of Mathematics administrative assistants for their practical help, and for providing a friendly environment in the department.

Finally, I would like to thank my family and friends for their support and encouragement. In particular, I am indebted to my parents for always being there for me and for teaching me the value of education and kindness.
CHAPTER 1
INTRODUCTION

Error-correcting codes are an integral part of any communication system and are found in numerous practical applications such as digital-storage media, wireless networks, and deep-space communication systems. During communication, a message is transmitted from one point in space to another or a message is stored in a medium at one instant in time and retrieved at another. In this channel coding framework, a message may be altered during transmission due to noise. Thus, when a message travels from a source to a destination, either over space or over time, both the sender and the receiver would like to ensure that the message be received free of error. This can be achieved by means of encoding the message prior to transmission. That is, redundancy is added to the information message in such a way that any two encoded messages differ significantly. By doing this, a relatively small amount of errors will not alter an encoded message by a large amount, and consequently, these errors may potentially be corrected. One of the primary objectives in coding is to develop codes (i.e., encoders and decoders) to ensure reliable and efficient transmission of information.

The general problem of communication is depicted in the block diagram in Figure 1.1. The source information may be a video image, a stream of bits from a computer, etc., that needs to be transmitted reliably over a noisy channel such as a telephone line, a wireless link, or the internet. A message \( u \) from the
output of the source encoder is encoded by a channel encoder as \( x \). Typically, the message \( u \) is a \( k \)-tuple over a finite field \( \mathbb{F}_q \), the channel encoder is a linear transformation that can be described by a matrix \( G \), and the encoded message \( x = Gu \) is an \( n \)-tuple over \( \mathbb{F}_q \). In this setting, the set of encoded messages (i.e., the code) form a \( k \)-dimensional subspace of \( \mathbb{F}_q^n \). The problem of decoding is one of estimating the message \( u \) upon receiving a noisy version \( y \) of the encoded message \( x \) from the channel. The information rate of the channel encoder is given by the fraction \( \frac{k}{n} \).

In a landmark paper in 1948, Shannon [42] proved that there is a theoretical limit called the capacity \( C \geq 0 \) such that there exist encoding and decoding procedures at transmission rates below \( C \) that guarantee that the decoded error probability can be made arbitrarily small. However, Shannon’s result was an existence result using probabilistic arguments and did not provide any explicit encoding and decoding procedures to demonstrate his result.

Since Shannon’s result, research in coding has often been aimed at constructing classes of codes that, when optimally decoded, could reach information rates
close to Shannon’s limit while achieving acceptably low probability of error. Furthermore, these codes must admit practical encoding and decoding algorithms. To achieve arbitrarily low probability of decoding error and operate close to the Shannon capacity, it was recognized that one must use very long block length codes. But this presented another problem since it was shown that the decoding of a general linear block code poses an NP-hard problem [5]. Hence, the primary objective is to find classes of codes that both perform well and are practical to implement.

The emergence of graph-based code designs and iterative decoding methods over the last decade have revolutionalized the field of coding. It has been realized that coding techniques near the Shannon limit can be achieved using these methods. The development of turbo codes in 1993 [6] and low density parity check (LDPC) codes (since 1995) showed that there are coding techniques that offer performance over various communication channels that is far better than all the error control codes known previously.

The focus of this dissertation is on low-density parity-check (LDPC) codes and iterative decoding methods. Low density parity check (LDPC) codes were originally introduced by Gallager [20] in his doctoral thesis in 1963. An LDPC code is a linear block code \( \mathcal{C} \subseteq \mathbb{F}^n \) which can be described as

\[
\mathcal{C} = \{ x \in \mathbb{F}^n \mid xH^T = 0 \},
\]

where the parity check matrix \( H \) is ‘sparse’ — i.e., contains few nonzero entries. Unless stated otherwise, \( \mathbb{F} = \mathbb{F}_2 \) is the binary field, and \( \mathcal{C} \) is a binary LDPC code. Gallager [20] was also the first to present several iterative decoding algorithms for these codes and showed that in general they have very good error-correcting
capabilities. Subsequently after, Tanner [48] gave a graphical interpretation to these codes, and showed how to view the iterative decoding algorithms presented by Gallager as message-passing algorithms on these graphs.

An LDPC code can be represented graphically by what is known as a Tanner graph, which is simply a bipartite graph whose incidence matrix is the parity check matrix of the LDPC code. The columns of the parity-check matrix are represented by a set of left nodes called variable nodes which correspond to the codeword positions of a codeword of the LDPC code. The rows of the parity-check matrix are represented by a set of right nodes called constraint nodes which correspond to the set of constraints that are imposed on the codeword positions of the LDPC code. Similarly, a turbo code can also be described graphically. Thus, both LDPC codes and turbo codes belong to the family of Codes on Graphs [35].

In the existing literature, LDPC codes have predominantly been designed by choosing randomly generated sparse parity-check matrices. However, the lack of any structure in the codes obtained from these random constructions makes them less desirable for practical use. For instance, the encoding complexity increases as $O(n^2)$ which makes them computationally intensive for implementation when the block length $n$ is very large. It is therefore advantageous to develop algebraic or combinatorial constructions of LDPC codes that impose structure in the code. This may simplify encoding, provide efficient representation and practical implementation, and further yield good performance when decoded iteratively using message passing decoders. For example, some of the finite-geometry based LDPC codes of [29] admit a simple encoding procedure, as the corresponding parity-check matrices are in the form of circulant matrices. In addition, codes with underlying structure may be amenable to analysis that can help predict their behavior.
The construction of finite (short to moderate) block length LDPC codes with systematic techniques is the principal motivation for this dissertation. For several practical applications, algebraically designed codes on graphs of moderate to large block lengths (1,000-10,000) that have good performance with iterative decoding are of interest.

While algebraic designs of graph-based codes are desired, it is important to first consider the parameters necessary to optimize in these designs. Several researchers have considered designing graphs with large girth, large expansion, small diameter, or more recently, large stopping sets. This dissertation provides more accurate analysis of the iterative decoding algorithm and shows that the so-called pseudocodewords of LDPC constraint graphs dominate the performance of the iterative decoder. Thus, an important parameter to consider for designing an LDPC constraint graph is the set of pseudocodewords determined by the corresponding LDPC graph. In this dissertation, we consider the role of pseudocodewords in affecting the performance of the min-sum and linear programming decoder and obtain lower bounds on the minimum pseudocodeword weight of LDPC Tanner graphs. Further, we present some conditions for the LDPC Tanner graphs so that potentially bad pseudocodewords are avoided. From these considerations, we develop specific LDPC code constructions that are good with respect to the distribution of pseudocodewords.

1.1 Linear Block Codes

In channel coding, a message $\mathbf{x} = (x_1, \ldots, x_k), x_i \in \mathbb{F}_q, i \in \{1, \ldots, k\}$ over the alphabet $\mathbb{F}_q$, is encoded into a codeword $\mathbf{c} = (c_1, \ldots, c_n), n > k$ using an injective map. A code $C$ consists of the image of all messages $x \in \mathbb{F}_q^k$. If the code is a linear
subspace of $\mathbb{F}_q^n$, then $C$ is a linear code. A $k \times n$ matrix $G$ of full rank such that the code $C$ is spanned by the rows of $G$ is called a generator matrix for the code.

$$C = \{ c = xG | x \in \mathbb{F}_q^k \}.$$

An $r \times n$ matrix $H$ of rank $n - k$ such that $GH^T = 0$ is called a parity-check matrix for $C$. Alternatively, the code $C$ may be defined as the nullspace of $H$,

$$C = \{ c \in \mathbb{F}_q^n | cH^T = 0 \}.$$

Let $d_H(x, y)$ denote the Hamming distance between two codewords $x, y \in C$, i.e. the number of positions where $x$ and $y$ differ, and let $wt_H(x)$ denote the Hamming weight of a codeword $x$, i.e. the number of nonzero positions in $x$.

**Definition 1.1.1** The minimum distance of a code $C$, denoted by $d_{\text{min}}$, is the minimum number of positions where any two codewords differ. For linear codes, this is the same as the minimum weight, or the minimum number of nonzero positions of any nonzero codeword in the code.

$$d_{\text{min}} = \min_{c,c' \in C,c \neq c'} d_H(c,c') = \min_{c \neq 0 \in C} wt_H(c).$$
**Definition 1.1.2** An \([n, k, d]\) linear block code is a \(k\)-dimensional linear subspace of \(\mathbb{F}_q^n\) having blocklength \(n\) and minimum distance \(d_{\text{min}} = d\).

### 1.2 LDPC Codes and Tanner Graphs

A sequence of \(m \times n\) matrices will be called \(c\)-sparse if as \(m, n \to \infty\), the number of nonzero entries in these matrices is always less than \(c \cdot \max(m, n)\).

**Definition 1.2.1** A **low-density parity-check (LDPC) code** is a linear block code with a sparse parity-check matrix \(H\). That is, \(H\) belongs to a \(c\)-sparse sequence of matrices where \(c \ll 0.5\). The sequence of matrices representing LDPC codes will be called an **LDPC code ensemble**.

A family of binary LDPC codes has the ratio of ones to zeros tending to zero as the blocklength \(n\) tends to infinity. It makes sense to also enforce that a sequence of LDPC code matrices has an upper bound on the number of ones per row and per column. Consider a sequence of matrices with increasing dimensions where every matrix in the sequence has at most three ones per column and at most six ones per row. Then such a sequence describes a family of LDPC matrices or LDPC codes. On the other hand, consider a sequence of matrices with \(\frac{n}{1+\log(n)}\) ones per row and \(\frac{n\log(n)}{1+\log(n)}\) zeros per row. Then, although the fraction of non-zero entries tends to zero as \(n\) increases, each matrix is still not sparse enough to be practical for graph-based iterative decoding algorithms.

Gallager [20] used the sparseness of these matrices to establish results on their distances and error-correction capabilities. However, he focussed on **regular matrices** to simplify his analysis. An LDPC matrix of block length \(N\) is called \((N, j, k)\) regular if it has a block length of \(N\) and is described by a parity check matrix containing \(j\) non-zero entries in every column and \(k\) non-zero entries in every row.
Gallager showed that regular LDPC codes of column weight \( j \geq 3 \) are asymptotically good. That is, the distance of LDPC codes with column weight \( j \geq 3 \) grows linearly with the blocklength. In addition, he showed that almost all codes in a random ensemble of \((N, j, k)\) LDPC codes had a minimum distance of at least \( N\delta(j, k) \) when \( N \) is large, for some \( \delta(j, k) > 0 \). Thus, the above family of sparse matrices yields LDPC codes with good minimum distances.

Any code may be represented by a bipartite graph that is obtainable from the parity-check matrix \( H \). Namely, \( H \) is the incidence matrix of the graph. One set of vertices, called variable nodes, corresponds to the columns of \( H \), and the other set of vertices, called parity-check nodes or simply, check nodes, corresponds to the rows of \( H \). An edge connects a variable node \( j \) to a check node \( i \) if and only if \( h_{ij} = 1 \) in \( H \). For LDPC codes, the sparsity of \( H \) means that the corresponding graph is sparse in the number of edges. The graphical representation of LDPC codes was first proposed by Tanner in a landmark paper in 1981 [48] and hence, the corresponding graphs are called Tanner graphs.

<table>
<thead>
<tr>
<th>parity ( p ) ( \downarrow ) bit ( \rightarrow )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
<th>( x_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( p_2 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( p_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( p_4 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
cH^T = 0 \Rightarrow (x_1, x_2, \ldots, x_8)H^T = 0
\]

Tanner graphs provide a sparse graphical representation of LDPC codes. Namely, the code \( C \) may be viewed as the set of all binary sequences assigned to the variable nodes such that all of the constraints imposed by the parity-check nodes are
Figure 1.3. Tanner graph of the parity check matrix $H$. 

satisfied (See Figure 1.3).

**Definition 1.2.2** A *generalized LDPC code* is an LDPC code represented by a Tanner graph $G$ wherein the constraint nodes of degree $k_s$ are not simple parity-checks but, rather, are constraints of a $[k_s, k'_s, \epsilon k_s]$ sub-code\(^1\).

Suppose a Tanner graph has all constraint nodes of degree $k$, then it is customary to obtain a generalized LDPC code by allowing the constraint nodes to represent constraints of a $[k, k', \epsilon k]$ sub-code. Suppose an LDPC matrix is of size $m \times n$ and there are $k$ ones in each row. Then, allowing the rows to represent constraints of a $[k, k', \epsilon k]$ sub-code yields an LDPC code of rate at least $1 - \frac{m(k-k')}{n}$.

By generalizing the constraint equations in this manner, the rate of the overall code is lowered whereas the minimum distance is increased.

\(^1\)Note that $k_s$ and $\epsilon$ are the minimum distance and the relative minimum distance, respectively of the sub-code.
1.3 Communication Channels

The communication channel determines the type of error and probability of error during transmission. In this section, we look at three basic communication channel models. (See Figures 1.4, 1.5, and 1.6.) First, the *Binary Erasure Channel (BEC)* is a binary-input channel where an error takes the form of an erasure. That is, a bit will be erased with probability $\delta$, and transmitted without error with probability $1 - \delta$. One nice feature of this channel is that the receiver can identify the location of all errors. That is, any position that has a 1 or a 0 in the received word is guaranteed to be correct.

The *Binary Symmetric Channel (BSC)* is a binary-input channel where the errors take the form of bit flips. That is, a bit is flipped (changed from 1 to 0, or 0 to 1) with probability $\epsilon$. In this case, the location of the errors is unknown at the receiver.

Last, the *Additive White Gaussian Noise (AWGN) channel* is a binary-input channel where the noise has a Gaussian distribution. The deep space communication channel is a real-world example for this type of channel.
Figure 1.5. Binary Symmetric Channel (BSC).

Figure 1.6. Additive White Gaussian Noise (AWGN) Channel.
1.4 Decoding Algorithms

1.4.1 Maximum-likelihood decoding

In maximum-likelihood (ML) decoding, the decoder estimates the most likely transmitted message by finding the codeword $x$ in the code $C$ that maximizes the probability of receiving the sequence $y$ from the channel, given that $x$ was transmitted. That is, the ML decoder estimates the codeword as

$$\hat{x}_{ML} = \arg\max_{x \in C} Pr(y|x).$$

The quantity $Pr(y|x)$ is the probability of receiving the sequence $y$ from the channel given that the codeword $x$ was transmitted. This quantity depends on the channel model; that is, the type of errors the channel introduces in the transmitted message.

For the most common channel which is the additive white Gaussian noise (AWGN) channel, the quantity $Pr(y|x)$ has a Gaussian distribution. This practically means that the ML decoder’s estimate is the codeword $x \in C$ that minimizes the Euclidean distance between the received sequence $y$ and $x$.

Although, the ML decoding problem has a simple description and is an optimal algorithm in terms of minimizing the probability of decoding error, its complexity increases exponentially with the code length for decoding general linear codes. This is because there is an exponentially large number of codewords in terms of the code length that are compared with respect to the received sequence in implementing the function $\arg\max_{x \in C} Pr(y|x)$. Thus, the ML decoding problem is in general NP-Hard. This motivates the question of finding simplified, albeit sub-optimal, algorithms for decoding codes.
1.4.2 Iterative decoding

The feature that makes LDPC codes attractive is the existence of computationally simple decoding algorithms. These algorithms either converge iteratively to a solution that may or may not be the maximum likelihood solution, or they do not converge at all. The algorithms are therefore suboptimal, however, preferred due to their decoding complexity. The most common of these algorithms are the min-sum and the sum-product algorithms [18, 30]. These two algorithms are graph-based message-passing algorithms applied on the LDPC constraint graph.
More recently, linear programming (LP) decoding has been applied to decode LDPC codes. Although LP decoding is more complex, it has the advantage that when it decodes to a codeword, the codeword is guaranteed to be the maximum-likelihood codeword (see [13]).

A message-passing decoder exchanges messages along the edges of the code’s constraint graph. For binary LDPC codes, the variable nodes assume the values one or zero; hence, a message can be represented either as the probability vector $[p_0, p_1]$, where $p_0$ is the probability that the variable node assumes the value 0, and $p_1$ is the probability that the variable node assumes the value 1, or as a log-likelihood ratio (LLR) $\log\left(\frac{p_0}{p_1}\right)$, in which case the domain of the message is $\mathbb{R} \cup \{\pm\infty\}$. The two decoding algorithms, min-sum and sum-product, are best described by the following update rules [30]:

- At a variable node $v$ (Figure 1.8), a message sent out along an edge $e$ is the result of a function whose input parameters are the messages received at $v$ on edges other than $e$, inclusive of the messages received from any external nodes connected to $v$ (such as the one shown as a blank square in Figure 1.8). (The external node in Figure 1.8 is introduced in the existing bipartite constraint graph strictly for convenience; it distinguishes messages that are obtained from the channel from messages that are obtained from neighboring constraint nodes.) A message from the external node in this case corresponds to channel information associated with the variable node $v$.

- At a constraint node $c$ (Figure 1.8), a message sent to a variable node along an edge $e$ is the result of a local function at $c$ whose input parameters are the messages received at $c$ on edges other than $e$. 
Let $\mu_{[x \rightarrow f]}(x)$ denote the message (a log-likelihood ratio (LLR)) sent from the variable node $x$ to the constraint node $f$, and let $\mu_{[f \rightarrow x]}(x)$ be the message sent from node $f$ to node $x$. Let $n(v)$ denote the set of neighbors of node $v$ (i.e., nodes connected to $v$ by an edge), and let $n(v)\backslash\{u\}$ denote the set of neighbors of $v$ excluding node $u$.

1.4.2.1 Min-sum decoding

The update rules in min-sum (MS) decoding are described below. For notation, let $y$ be the channel observation corresponding to a variable node $x$ and for convenience, we also use $x$ as the random variable associated with the variable node $x$. Further, let $Y$ be the set of random variables corresponding to the neighboring variable nodes of the constraint node $f$, i.e., $N(f)$, and let $g(Y)$ be an indicator function that is either 0 or $\infty$ depending on whether the arguments in $Y$ satisfy the constraint equation imposed by the node $f$ or not. We use the symbol $\sim \{x\}$ to indicate the set of all variables in $Y$ other than $x$. The update rules in MS decoding are then as follows:

- At a variable node at $x$:

\[
\mu_{[x \rightarrow f]}(x) = -\log \left( \frac{P(y|x=1)}{P(y|x=0)} \right) + \sum_{h \in n(x) \backslash \{f\}} \mu_{[h \rightarrow x]}(x),
\]

- At a constraint node $f$:

\[
\mu_{[f \rightarrow x]}(x) = \min_{\sim \{x\}} (g(Y) \sum_{z \in n(f) \backslash \{x\}} \mu_{[z \rightarrow f]}(z)).
\]
Final decision at a variable node: After each iteration, the estimate at a variable node $x$ is

$$\hat{x} = \begin{cases} 
0 & \text{if } -\log \left( \frac{P(y|x=1)}{P(y|x=0)} \right) + \sum_{h \in n(x)} \mu_{[h \rightarrow x]}(x) \geq 0 \\
1 & \text{otherwise}
\end{cases}$$

Let $c = (c_1, \ldots, c_n)$ be a codeword and let $w = (w_1, \ldots, w_n)$ be the input to the decoder from the channel. That is, the LLR’s from the channel for the codebits $v_1, \ldots, v_n$ are $w_1, \ldots, w_n$, respectively. Then the optimal min-sum decoder (or, ML decoder) estimates the codeword

$$c^* = \arg \min_{c \in C} (c_1 w_1 + c_2 w_2 + \cdots + c_n w_n) = \arg \min_{c \in C} cw^T.$$ 

1.4.2.2 Sum-product decoding

The update rules in sum-product (SP) decoding are as follows:

- At a variable node:

  $$\mu_{x \rightarrow f}(x) = -\log \left( \frac{P(y|x=1)}{P(y|x=0)} \right) + \sum_{h \in n(x) \setminus \{f\}} \mu_{h \rightarrow x}(x),$$

- At a constraint node:

  $$\mu_{f \rightarrow x}(x) = \sum_{Y \sim \{x\}} g(Y) \prod_{z \in n(f) \setminus \{x\}} \mu_{z \rightarrow f}(z).$$

At a constraint node, the SP decoder computes the *aposteriori* probability of an adjoining variable node based on the estimates received from the remaining variable nodes that are connected to the constraint node. The SP decoder is
sub-optimal in the sense that this computation is performed at every constraint independent of the other constraints. However, this leads to the optimal solution in the special case where the constraint graph has no closed paths (or, cycles).

The reader familiar with convolutional codes will observe that min-sum versus sum-product decoding of a graph-based code is analogous to Viterbi versus BCJR decoding of a trellis code [4] [16].

1.4.3 Linear programming decoding

Maximum-likelihood decoding may also be viewed as a combinatorial optimization problem [13], which, as mentioned above, is complex to implement for large block length codes. In [13], Feldman introduces a novel linear programming (LP) decoder which simplifies the ML problem by using a relaxed polytope instead of the polytope formed by the convex hull of the codewords of the code. This relaxation in the polytope description results in a sub-optimal and a simpler algorithm in comparison with the ML decoder. The integral points or vertices of this relaxed polytope are precisely the codewords of the code, whereas the fractional vertices of the polytope are the pseudocodewords of the LP decoder. The LP decoder always converges to a vertex of the polytope and one of the attributes of the LP decoder is that it has the ML certificate property. That is, if the LP decoder converges to an integral vertex, then the estimate of the LP decoder is guaranteed to be the ML decoder’s estimate. The complexity of the LP decoder is in general significantly smaller than that of the ML decoder, but typically also significantly larger than that of an iterative decoder such as the min-sum decoder. The pseudocodewords of the LP decoder are exactly the pseudocodewords arising from graph covers for the min-sum decoder and are described in greater detail in
Chapter 2. Furthermore, the fractional weight of pseudocodewords, a parameter which controls the performance of the LP decoder, is related to the pseudocodeword weight, a parameter that controls the performance of the min-sum iterative decoder, in Section 3.3.

1.5 Thesis Overview

This dissertation is organized as follows. Pseudocodewords of Tanner graphs are introduced in Chapter 2, and the graph-covers-polytope definition of [28] is examined using the [4,1,4] repetition code as an example. Lower bounds on the pseudocodeword weight for the BSC and AWGN channels are derived in Chapter 3, including extensions to generalized LDPC codes. Chapter 4 introduces the notion of pseudocodeword weight for the p-ary symmetric channel, as well as some bounds on the minimum pseudocodeword weight. Chapter 5 investigates the structure of pseudocodewords realizable in lifts of general Tanner graphs. In Chapter 6, pseudocodeword distributions for different graph representations are examined, and the performance of iterative decoding on different Tanner graph representations of individual codes is presented. It is shown for two example codes that redundancy in representation improves iterative decoding performance. Chapter 7 continues the analysis in Chapter 5 by providing a bound on the minimum degree lift needed to realize a given lift-realizable irreducible pseudocodeword of a Tanner graph. In Chapter 8, new constructions of LDPC codes are presented that have good minimum pseudocodeword weights. Chapter 9 discusses p-ary finite geometry and tree-based LDPC codes. In Chapter 10, a new zigzag graph product creating unbalanced expander graphs is presented, and a construction for LDPC codes based on zigzag product graphs is shown. Chapter 11 presents sev-
eral bounds on the pseudocodeword weight for different types of expander codes, and continues the analysis of the zig-zag product codes. Chapter 12 concludes the dissertation. For readability, the proofs for each chapter have been moved to the last section of each chapter.
CHAPTER 2

PSEUDOCODEWORDS

In this chapter we establish the necessary terminology and notation regarding pseudocodewords that will be used in this thesis, including an overview of pseudocodeword interpretations and their effect on iterative decoding. The accuracy of the graph-covers approximation is also examined. Let \( V \) denote a set of \( n \) variable nodes and let \( U \) denote a set of \( m \) constraint nodes and let \( G = (V, U; E) \) be a bipartite graph comprising of variable nodes \( V \), constraint nodes \( U \), and edges \( E \subseteq \{(v, u) | v \in V, u \in U\} \), and representing a binary LDPC code \( \mathcal{C} \) with minimum distance \( d_{\text{min}} \).

2.1 Computation Tree Interpretation

As originally introduced by Wiberg [51], let \( C(G) \) be the computation tree of the base LDPC constraint graph \( G \) corresponding to the \textit{min-sum} iterative decoder. The tree is formed by enumerating the Tanner graph from an arbitrary variable node, called the \textit{root} of the tree, down through the desired number of layers corresponding to decoding iterations. A computation tree enumerated for \( \ell \) iterations and having variable node \( v_i \) acting as the root node of the tree is denoted by \( C_i(G)_\ell \). The shape of the computation tree is dependent on the scheduling of message passing used by the iterative decoder on the Tanner graph \( G \).
Figure 2.1. A graph and its computation tree after two iterations of message passing.

Figure 2.1, the computation tree $C_2(G)_2$ is shown for the flooding schedule [20].

The computation tree aids in the analysis of iterative decoding by describing the message-passing algorithm using a cycle-free graph. Since iterative decoding is exact on cycle-free graphs, the computation tree is a valuable tool in the exact analysis of iterative decoding on finite-length LDPC codes with cycles.

A valid assignment on a computation tree is an assignment of zeros and ones to the variable nodes such that all constraint nodes are satisfied. A codeword $c$ corresponds to a valid assignment on the computation tree, where for each $i$, all $v_i$ nodes in the computation tree are assigned the same value (Figure 2.2); a pseudocodeword $p$, on the other hand, corresponds to a valid assignment on the computation tree, where for each $i$, not all $v_i$ in the computation tree need be assigned the same value (Figure 2.3). In other words, if an assignment on $C(G)$ corresponds to a codeword, then the local codeword configuration at a check node $c_j$ on $C(G)$ is the same as at any other copy of $c_j$ on $C(G)$. However, a valid assignment on $C(G)$ corresponds to a pseudocodeword, when the local codeword-
configurations may differ at different copies of the checks $c_j$ on $C(G)$.

A local configuration at a check node $c_j$ in the base Tanner graph $G$ is obtained by taking the average of the local-codeword-configurations at all copies of $c_j$ on the computation tree. If the local configurations at all the check nodes of the graph $G$ are consistent\(^1\), then they yield a pseudocodeword vector $p$ that lies in the polytope of [28] (see equation 2.2). Such a pseudocodeword $p$ is realizable as a codeword on a lift graph of $G$ [28]. If the local configurations are not consistent among all the check nodes, then there is no well-defined vector for the valid configuration

\(^1\)That is, if a variable node $v_i$ participates in checks $c_1, c_2, \ldots, c_k$, then the component value of $v_i$ in each of the local codeword-configurations at $c_1, c_2, \ldots, c_k$ is the same.
(or, pseudocodeword) on the computation tree, and this pseudocodeword is not realizable on a lift graph of $G$.

Let $S$ be a subset of $\{1, 2, \ldots, n\}$ and let $\mathbf{p}_S$ denote the vector obtained by restricting the vector $\mathbf{p}$ to only those components indicated by $S$. Further, let $N(j)$ denote the set of variable node neighbors of the check node $c_j$ in $G$. We introduce the notion of a vector representation for a pseudocodeword in the following manner. Given a valid assignment on the computation tree, a pseudocodeword vector $\mathbf{p}$ may be defined as $\mathbf{p} \in [0, 1]^n$ such that, for all $j$, $\mathbf{p}_{N(j)}$ is a local configuration (possibly scaled) at check $c_j$, and if $v_i$ is a variable node in $N(j_1)$ and $N(j_2)$, then $p_i$ assumes the same value in $p_{N(j_1)}$ and $p_{N(j_2)}$, for all variable nodes and their check node neighbors. If no such vector exists for a valid assignment on the computation tree, then we say that the corresponding pseudocodeword configuration has no vector representation.

**Example 2.1.1** In Figure 2.2, the local-codeword-configuration at each copy of $c_1$ is $(0, 1, 1)$ corresponding to variable nodes $(v_1, v_2, v_3)$, and by averaging the local-codeword-configurations at all copies of $c_1$, we get that $(0, 1, 1)$ is the local configuration at $c_1$. Similarly, the local-configuration at $c_2$ is $(0, 1, 1)$, corresponding to $(v_1, v_3, v_4)$, and at $c_3$ is $(0, 1, 1)$, corresponding to $(v_1, v_2, v_4)$. Thus, the vector $\mathbf{p} = (0, 1, 1, 1)$ is a pseudocodeword vector that describes the assignment in Figure 2.2 and, when restricted to the variable nodes participating in a particular check node, gives the local-configuration at that check node.

**Example 2.1.2** In Figure 2.3, the local configuration at $c_1$ is $(1, 0.5, 0.5)$ corresponding to $(v_1, v_2, v_3)$, at $c_2$ is $(1, 0.75, 0.25)$ corresponding to $(v_1, v_3, v_4)$, and at $c_3$ is $(1, 0.5, 0.5)$ corresponding to $(v_1, v_2, v_4)$. The local-configurations are not consistent to yield a vector describing this pseudocodeword. That is, there is no
vector that, if restricted to the components corresponding to the variable node neighbors of a particular check node, would yield the local-configuration at that check node. However, if the assignment on the computation tree were altered slightly so as to yield a local-configuration of \((1, 0.5, 0.5)\) at \(c_2\) (corresponding to \((v_1, v_3, v_4)\)), while retaining the same local-configurations at \(c_1\) and \(c_3\) as above, then the pseudocodeword can be described by the vector \(p = (1, 0.5, 0.5, 0.5)\) corresponding to \((v_1, v_2, v_3, v_4)\).

2.2 Graph Covers Interpretation

Wiberg originally formulated the idea of pseudocodewords, using the terminology of deviation sets, in terms of the computation tree, as described in [51]. This work was later extended by Frey et. al. in [17]. More recently in [28], Koetter and Vontobel introduce a graph covers interpretation of pseudocodewords, as described in this section. A degree \(\ell\) cover (or, lift) \(\hat{G}\) of \(G\) is defined as follows:

**Definition 2.2.1** A finite degree \(\ell\) cover of \(G = (V, U; E)\) is a bipartite graph \(\hat{G}\) where for each vertex \(x_i \in V \cup U\), there is a cloud \(\hat{X}_i = \{\hat{x}_{i1}, \hat{x}_{i2}, \ldots, \hat{x}_{i\ell}\}\) of vertices in \(\hat{G}\), with \(\text{deg}(x_i) = \text{deg}(\hat{x}_i)\) for all \(1 \leq j \leq \ell\), and for every \((x_i, x_j) \in E\), there are \(\ell\) edges from \(\hat{X}_i\) to \(\hat{X}_j\) in \(\hat{G}\) connected in a 1–1 manner.

Figure 2.4 shows a base graph \(G\) and a degree four cover of \(G\).

**Definition 2.2.2** Suppose that \(\hat{c} = (\hat{c}_{1,1}, \hat{c}_{1,2}, \ldots, \hat{c}_{1,\ell}, \hat{c}_{2,1}, \ldots, \hat{c}_{2,\ell}, \ldots)\) is a codeword in the Tanner graph \(\hat{G}\) representing a degree \(\ell\) lift of \(G\). A pseudocodeword \(p\) of \(G\) is a vector \((p_1, p_2, \ldots, p_n)\) obtained by reducing a codeword \(\hat{c}\), of the code in the lift graph \(\hat{G}\), in the following way:
Figure 2.4. A pseudocodeword in the base graph (or a valid codeword in a lift).

\[
\mathbf{\hat{c}} = (\hat{c}_{1,1}, \ldots, \hat{c}_{1,\ell}, \hat{c}_{2,1}, \ldots, \hat{c}_{2,\ell}, \ldots) \to \\
(\hat{c}_{1,1} + \hat{c}_{1,2} + \cdots + \hat{c}_{1,\ell}, \hat{c}_{2,1} + \hat{c}_{2,2} + \cdots + \hat{c}_{2,\ell}, \ldots) = (p_1, p_2, \ldots, p_n) = \mathbf{p},
\]

where \(p_i = (\hat{c}_{i,1} + \hat{c}_{i,2} + \cdots + \hat{c}_{i,\ell})\) and ‘+’ denotes real addition.

Note that each component of the pseudocodeword is merely the number of 1-valued variable nodes in the corresponding variable cloud, and that any codeword \(\mathbf{c}\) is trivially a pseudocodeword as \(\mathbf{c}\) is a valid codeword configuration in a degree-one lift. Equivalently, pseudocodewords realizable in graph covers can be defined by taking as components the fraction of one-valued variable nodes in every cloud. This definition yields a pseudocodeword \(\mathbf{p} = (p_1, p_2, \ldots, p_n)\) that is a vector of rational entries such that \(\mathbf{p} \in [0, 1]^n\), and

\[
p_i = \frac{\hat{c}_{i,1} + \hat{c}_{i,2} + \cdots + \hat{c}_{i,\ell}}{\ell},
\]

where ‘+’ denotes real addition. A pseudocodeword with integer entries as in the former definition is referred to as an unscaled pseudocodeword in [28], and we adopt that terminology in this thesis. The set of lift-realizable pseudocodewords is a subset of those arising in the computation tree. These sets of pseudocodewords are compared in Section 2.6.
Definition 2.2.3 A pseudocodeword that does not correspond to a codeword in
the base Tanner graph is called a non-codeword pseudocodeword, or nc-pseudocodeword,
for short.

2.2.1 Polytope representation

The set of all pseudocodewords associated with a given Tanner graph $G$ has an
elegant geometric description [13, 28]. In [28], Koetter and Vontobel characterize
the set of pseudocodewords via the fundamental cone. For each parity check $j$ of
degree $\delta_j$, let $C_j$ denote the $(\delta_j, \delta_j - 1, 2)$ simple parity check code, and let $P_{\delta_j}$ be
a $2^{\delta_j-1} \times \delta_j$ matrix with the rows being the codewords of $C_j$. The fundamental
polytope at check $j$ of a Tanner graph $G$ is then defined as:

$$P^{GC}(C_j) = \{\omega \in \mathbb{R}^{\delta_j} : \omega = xP_{\delta_j}, x \in \mathbb{R}^{2^{\delta_j-1}}, 0 \leq x_i \leq 1, \sum_i x_i = 1\}, \quad (2.1)$$

and the fundamental polytope of $G$ is defined as:

$$P^{GC}(G) = \{\omega \in \mathbb{R}^n : \omega_{N(j)} \in P^{GC}(C_j), j = 1, \ldots, m\}. \quad (2.2)$$

We use the superscript GC to refer to the pseudocodewords arising from graph
covers and the notation $\omega_{N(j)}$ to denote the vector $\omega$ restricted to the coordinates
of the neighbors of check $c_j$. The fundamental polytope gives a compact charac-
terization of all possible lift-realizable pseudocodewords of a given Tanner graph
$G$. Incorporating multiplicities of vectors, the fundamental cone $F(G)$ associated
with $G$ is obtained as:

$$F(G) = \{\mu \omega \in \mathbb{R}^n : \omega \in P^{GC}(G), \mu \geq 0\}.$$
Though the fundamental cone is the same for all channels, the worst case pseudocodewords depend on the channel [28].

**Definition 2.2.4** A pseudocodeword \( p \) of the Tanner graph \( G \) is said to be lift-realizable if \( p \) is obtained by reducing a valid codeword configuration on some lift (or, cover) graph \( \hat{G} \) of \( G \) as described in Definition 2.2.2.

In other words, a lift-realizable pseudocodeword \( p \) corresponds to a point in the graph-covers polytope \( P^{GC}(G) \).

In [13], Feldman also uses a polytope to characterize the pseudocodewords in linear programming (LP) decoding and this polytope has striking similarities with the polytope of [28]. The feasible set of the LP decoder is given by:

\[
P^{LP}(G) = \{ c \in \mathbb{R}^n : x_j \in \mathbb{R}^{2^{d_j}-1}, \sum_{S \in C_j} x_{j,S} = 1, c_i = \sum_{S \in C_j, i \in S} x_{j,S} \forall i \in N(j), 0 \leq x_{j,S} \leq 1, \forall S \in C_j, j \in \{1, \ldots, m\} \}.
\]

**Remark 2.2.1** It can be shown that the polytopes of [28] and [13] are equivalent, i.e., \( P^{GC}(G) = P^{LP}(G) \) [9].

### 2.3 Relation to Stopping Sets

**Definition 2.3.1** The support of a vector \( x = (x_1, \ldots, x_n) \), denoted \( \text{supp} \, x \), is the set of indices \( i \) where \( x_i \neq 0 \).

**Definition 2.3.2** [12] A stopping set in \( G \) is a subset \( S \) of \( V \) where for each \( s \in S \), every neighbor of \( s \) is connected to \( S \) at least twice.
The size of a stopping set $S$ is equal to the number of elements in $S$. A stopping set is said to be \textit{minimal} if there is no smaller sized nonempty stopping set contained within it. The smallest minimal stopping set is called a \textit{minimum} stopping set, and its size is denoted by $s_{\text{min}}$. Note that a minimum stopping set is not necessarily unique.

**Example 2.3.1** Figure 2.5 shows a stopping set in the graph. Observe that $\{v_4, v_7, v_8\}$ and $\{v_3, v_5, v_9\}$ are two minimum stopping sets of size $s_{\text{min}} = 3$, whereas $\{v_0, v_1, v_3, v_5\}$ is a minimal stopping set of size 4.

One useful observation is that that the support of a lift-realizable pseudocodeword forms a stopping set in $G$. This is also implied in [13, 28].

**Lemma 2.3.1** The support of a lift-realizable pseudocodeword $p$ of $G$ is the incidence vector of a stopping set in $G$.

2.4 Irreducible Pseudocodewords

**Definition 2.4.1** An unscaled pseudocodeword $p = (p_1, \ldots, p_n)$ is \textit{irreducible} if it cannot be written as a sum of two or more nonzero codewords or pseudocodewords. Otherwise the pseudocodeword $p$ is said to be \textit{reducible}.
Example 2.4.1 Consider the graph $G$ in Figure 2.6. The reducible pseudocodeword $p = (1, 2, 1)$ can be decomposed as $p = (1, 1, 0) + (0, 1, 1)$, i.e., as a sum of two codewords, each of which is an irreducible pseudocodeword.

Note that irreducible pseudocodewords are called *minimal* pseudocodewords in [28] as they correspond to vertices of the polytope $P^{GC}(G)$. Frey et. al [17] have shown that the min-sum iterative decoder will always converge to an irreducible pseudocodeword.

2.5 Pseudocodewords and Iterative Decoding Behavior

Let $\mathcal{P}$ be the set of all pseudocodewords (including all codewords) of the graph $G$. Then the graph-based min-sum decoder described above estimates \[ x^* = \arg\min_{x \in \mathcal{P}} xw^T. \]

We will refer to the dot product $xw^T$ as the *cost-function* of the vector $x$ with respect to the channel input vector $w$. The graph-based iterative min-sum (MS) decoder tries to estimate the pseudocodeword with the lowest cost, whereas the graph-based sum-product (SP) decoder also takes into account the cost associated with all pseudocodewords of the given graph, not merely that with the lowest \[ ^2 \text{Sum under real addition, as this definition refers to pseudocodewords with integer entries.} \]
cost, in its estimate. In this rest of this section, we will focus our attention on the graph-based min-sum (MS) iterative decoder, since it is simpler to analyze than the sum-product (SP) decoder and provides a sound estimate of the performance under SP-decoding. The following definition characterizes the iterative decoder behavior, providing conditions when the MS decoder may fail to converge to a valid codeword.

An input weight vector $w$ to an iterative decoder will be the vector of log-likelihood ratios (LLRs) corresponding to the estimates of the variable nodes obtained from the corresponding values received from the channel.

**Definition 2.5.1** [22] A pseudocodeword $p = (p_1, p_2, \ldots, p_n)$ is good if for all input weight vectors $w = (w_1, w_2, \ldots, w_n)$ to the min-sum iterative decoder, there is a codeword $c$ that has lower overall cost than $p$, i.e., $cw^T < pw^T$.

Suppose the all-zeros codeword is the maximum-likelihood (ML) codeword for an input weight vector $w$, then all non-zero codewords $c$ have a positive cost, i.e., $cw^T > 0$.

**Definition 2.5.2** A pseudocodeword $p$ is bad if there is a weight vector $w$ such that for all non-zero codewords $c$, $cw^T > pw^T$.

Note that a pseudocodeword $p$ that is bad on one channel is not necessarily bad on other channels since the set of weight vectors that are possible depend on the channel. Again, in the case where the all-zeros codeword is the maximum-likelihood (ML) codeword, it is equivalent to say that a pseudocodeword $p$ is bad if there is a weight vector $w$ such that for all codewords $c$, $cw^T \geq 0$ but $pw^T < 0$.

Using the terminology of deviation sets, Wiberg [51, Theorem 4.2] states a
necessary condition for the min-sum iterative-decoder to fail to converge, which can be rephrased using our terminology as follows:

**Theorem 2.5.1** A necessary condition for a decoding error to occur is that the cost of some irreducible pseudocodeword \( p \) which is not a codeword is non-positive.

On the erasure channel, pseudocodewords of \( G \) are essentially stopping sets in \( G \) [13, 19], and thus, the non-convergence of the iterative decoder is attributed to the presence of stopping sets. As an example, in Figure 2.5, if the received word from the channel has erasures in positions corresponding to \( v_0, v_1, v_3, \) and \( v_5 \), then none of the values will be determinable since each neighboring vertex of the stopping set \( S \) has at least two incoming unknowns. Moreover, any stopping set may potentially give rise to a bad pseudocodeword on the erasure channel since any stopping set can potentially prevent the iterative decoder from converging. In subsequent chapters, we will examine the MS decoder on the BSC and the AWGN channel and the structure of pseudocodewords over these channels.

2.6 Graph-Covers-Polytope Approximation

In this section, we examine the graph-covers-polytope definition of [28] in characterizing the set of pseudocodewords of a Tanner graph with respect to min-sum iterative decoding. Consider the \([4, 1, 4]\)-repetition code which has a Tanner graph representation as shown in Figure 2.7. The corresponding computation tree for three iterations of message passing is also shown in the figure. The only lift-realizable pseudocodewords for this graph are \((0, 0, 0, 0)\) and \((k, k, k, k)\), for some positive integer \( k \); thus, this graph has no non-codeword pseudocodewords. Even on the computation tree, the only valid assignment assigns the same value for all
the nodes on the computation tree. Therefore, there are no non-codeword pseudocodewords on the graph’s computation tree as well. In Figure 2.7, the Tanner graph is shown for the LDPC code having parity-check matrix

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

Figure 2.7. A Tanner graph and computation tree (CT) for the [4,1,4] repetition code.

Suppose we add an additional redundant check node to the graph, then we obtain a new LDPC constraint graph, shown in Figure 2.8, for the same code. Even on this graph, the only lift realizable pseudocodewords are \((0, 0, 0, 0)\) and \((k, k, k, k)\), for some positive integer \(k\). Therefore, the polytope of [28] contains \((0, 0, 0, 0)\) and \((1, 1, 1, 1)\) as the vertex points and has no bad pseudocodewords.\(^3\)

\(^3\)Note that the original graph has one redundant check node.
(as in Definition 2.5.2). However, on the computation tree, there are several valid assignments that do not have an equivalent representation in the graph-covers-polytope. The assignment where all nodes on the computation tree are assigned the same value, say 1, (as highlighted in Figure 2.8) corresponds to a codeword in the code. For this assignment on the computation tree, the local configuration at check $c_1$ is $(1,1)$ corresponding to $(v_1, v_2)$, at check $c_2$ it is $(1,1)$ corresponding to $(v_2, v_3)$, at check $c_3$ it is $(1,1)$ corresponding to $(v_3, v_4)$, at check $c_4$ it is $(1,1)$ corresponding to $(v_1, v_4)$, and at check $c_5$ it is $(1,1,1,1)$ corresponding to $(v_1, v_2, v_3, v_4)$. Thus, the pseudocodeword vector $(1, 1, 1, 1)$ corresponding to $(v_1, v_2, v_3, v_4)$ is consistent locally with all the local configurations at the individual check nodes.

However, an assignment where some nodes are assigned different values compared to the rest (as highlighted in Figure 2.9) corresponds to a non-codeword pseudocodeword on the Tanner graph. For the assignment shown in Figure 2.9, the local configuration at check $c_1$ is $(1,1)$, corresponding to $(v_1, v_2)$, as there are two check nodes $c_1$ in the computation tree with $(1,1)$ as the local codeword at each of them. Similarly, the local configuration at check $c_2$ is $(2/3, 2/3)$, corresponding to $(v_2, v_3)$, as there are three $c_2$ nodes on the computation tree, two of which have $(1,1)$ as the local codeword and the third has $(0,0)$ as the local codeword. Similarly, the local configuration at check $c_3$ is $(1/3, 1/3)$ corresponding to $(v_3, v_4)$, the local configuration at check $c_4$ is $(1/2, 1/2)$ corresponding to $(v_1, v_4)$, and the local configuration at check $c_5$ is $(1/3, 1, 0, 2/3)$ corresponding to $(v_1, v_2, v_3, v_4)$. Thus, there is no pseudocodeword vector that is consistent locally with all of the above local configurations at the individual check nodes. Hence, the pseudocodeword is not realizable on a finite-degree cover.
Clearly, as the computation tree grows with the number of decoding iterations, the number of non-codeword pseudocodewords in the graph grows exponentially with the depth of the tree. Thus, even in the simple case of the repetition code, the graph-covers-polytope of [28] fails to capture all min-sum-iterative-decoding-pseudocodewords of a Tanner graph.

To a large extent, empirical evidence indicates that analyzing lift-realizable pseudocodewords provides a decent estimate of the min-sum iterative decoder’s performance, and designing Tanner graphs having large minimum pseudocodeword weight is advantageous. It is also important to note that the weight definitions in [19] and Definition 3.1.1 assume the lift-realizable case. We restrict attention to lift-realizable pseudocodewords for the remaining part of this thesis, since they have an elegant mathematical characterization which allows for simple analysis. However, it would be useful to obtain a mathematical characterization of pseudocodewords on the iterative decoder’s computation tree. Last, we remark that the lift-realizable set exactly characterizes the pseudocodewords in LP decoding, and consequently, much of the analysis in this thesis is extendable to that case.

Figure 2.10 shows the performance of min-sum iterative decoding on the constraint graphs of Figures 2.7 and 2.8 when simulated over the binary input additive white Gaussian noise channel (BIAWGNC) with signal to noise ratio $E_b/N_0$. The maximum-likelihood (ML) performance of the code is also shown as reference. With a maximum of 10,000 decoding iterations, the performance obtained by the iterative decoder on the single cycle constraint graph of Figure 2.7 is the same as the optimal ML performance (the two curves are one on top of the other), thereby confirming that the graph has no nc-pseudocodewords. The iterative decoding
performance deteriorates when a new degree four check node is introduced as in Figure 2.8. A significant fraction of detected errors, i.e., errors due to the decoder not being able to converge to any valid codeword within 10,000 iterations, were obtained upon simulation of this new graph. (Detected error rates are primarily attributed to the existence of pseudocodewords.) To de-emphasize the effect of non-codeword pseudocodewords, arising out of the computation tree of Figure 2.8, on the iterative decoder, the message log-likelihood ratios were suitably scaled\textsuperscript{4}, as suggested in [17], and this pushed the performance to the optimal ML performance.

\textsuperscript{4}Scaling allows for messages at earlier iterations to have greater influence than less reliable messages at later iterations.
This example illustrates that the fundamental polytope of [28] does not capture the entire set of min-sum-iterative-decoding-pseudocodewords of a Tanner graph. In general, we state the following results:

**Lemma 2.6.1** A bipartite graph $G$ representing an LDPC code $C$ contains no non-codeword irreducible pseudocodewords on the computation tree $C(G)$ of any depth if and only if either (i) $G$ is a tree, or (ii) $G$ contains only degree two check nodes.

**Lemma 2.6.2** A bipartite graph $G$ representing an LDPC code $C$ contains either exactly one or zero non-codeword irreducible lift-realizable pseudocodewords if either (i) $G$ is a tree, or (ii) there is at least one path between any two variable nodes in $G$ that traverses only via check nodes having degree two.
Note that condition (ii) in Lemma 2.6.2 states that if there is at least one path between every pair of variable nodes that has only degree two check nodes, then $G$ contains no non-codeword irreducible lift-realizable pseudocodewords. However, condition (ii) in Lemma 2.6.1 requires that every path between every pair of variable nodes has only degree two check nodes. Unlike Lemma 2.6.1 which gives a necessary and sufficient condition, Lemma 2.6.2 only provides a necessary condition for a graph $G$ to contain no non-codeword irreducible pseudocodewords.

The above two lemmas state some necessary conditions under which a graph has no non-codeword irreducible pseudocodewords with respect to min-sum decoding. In Section 5.1, we give some sufficient conditions for bad pseudocodewords to exist. However, it would be interesting to find conditions that are both necessary and sufficient for a graph to have no bad pseudocodewords with respect to the min-sum iterative decoder.

2.7 Proofs

**Lemma 2.6.1**  **Proof:** Let $G$ represent the constraint graph of an LDPC code $C$. Suppose $G$ is a tree, then clearly, any pseudocodeword of $G$ can be expressed as a linear combination of codewords of $G$. Hence, suppose $G$ is not a tree, and suppose all check nodes in $G$ are of degree two. Then the computation tree contains only check nodes of degree two and hence, for a valid assignment on the computation tree, the value of any child variable node $v_1$ on the computation tree that stems from a parent check node $h$ is the same as the value of the variable node $v_2$ which is the parent node of $h$. Thus, the only local codeword configurations at each check node is the all-ones configurations when the root node of the tree is assigned the value one. Hence, the only valid solutions on the computation
tree correspond to the all ones vector and the all zeros vector – which are valid codewords in $C$.

Conversely, suppose $G$ is not a tree and suppose there is a check node $h$ of degree $k$ in $G$. Let $v_1, v_2, \ldots, v_k$ be the variable node neighbors to $h$. Then, enumerate the computation tree rooted at $v_1$ for a sufficient depth such that the node $h$ appears several times in the tree and also as a node in the final check node layer of the tree. Then, there are several possible valid assignments in the computation tree, where the values assigned to the leaf nodes that stem from $h$ yield a solution that is not a valid codeword in $G$. Thus, $G$ contains non-codeword irreducible pseudocodewords on its computation tree.

Lemma 2.6.2 Proof: Let $G$ represent the constraint graph of an LDPC code $C$. Suppose $G$ is a tree, then clearly, any pseudocodeword of $G$ can be expressed as a linear combination of codewords of $G$. Hence, suppose $G$ is not a tree, and between every pair of variable nodes in $G$ there is a path that contains only degree two check nodes in $G$. Then $G$ contains only lift-realizable pseudocodewords of the form $(k, k, \ldots, k)$, where $k$ is a positive integer. Note that the all ones-vector is a valid codeword in $G$. Hence, the only irreducible pseudocodewords in $G$ are the all-zero vector $(0, 0, \ldots, 0)$, and either the all-ones vector $(1, 1, \ldots, 1)$ or the all-twos vector $(2, 2, \ldots, 2)$ if the all-ones vector is not a codeword.
CHAPTER 3

BOUNDS ON MINIMAL PSEUDOCODEWORD WEIGHTS FOR THE BSC
AND AWGN CHANNELS

As in classical coding where the distance between codewords affects error correction capabilities, the distance between pseudocodewords affects iterative decoding capabilities. Analogous to the classical case, the distance between a pseudocodeword and the all-zeros codeword is captured by weight. In this chapter we introduce the notion of pseudocodeword weight and provide some lower bounds on the minimum pseudocodeword weights. These bounds are important as the calculation of the actual minimum pseudocodeword weight is a computationally difficult task.

3.1 Pseudocodeword Weights

The weight of a pseudocodeword depends on the channel, as noted in the following definition.

**Definition 3.1.1** [19] Let \( p = (p_1, p_2, \ldots, p_n) \) be a pseudocodeword of the code \( C \) represented by the Tanner graph \( G \). Then the weight of \( p \) is:

- \( w_{\text{BEC}}(p) = |\text{supp}(p)| \) for the binary erasure channel (BEC);
Let $e$ be the smallest number such that the sum of the $e$ largest $p_i$’s is at least $\sum_{i=1}^{e} p_i$. Then $w_{BSC}(p)$ for the binary symmetric channel (BSC) is:

$$w_{BSC}(p) = \begin{cases} 
2e, & \text{if } \sum_{i=1}^{e} p_i = \frac{\sum_{i=1}^{n} p_i}{2} \\
2e - 1, & \text{if } \sum_{i=1}^{e} p_i > \frac{\sum_{i=1}^{n} p_i}{2},
\end{cases}$$

where $\sum_{i=1}^{e} p_i$ is the sum of the $e$ largest $p_i$’s.

- $w_{AWGN}(p) = \frac{(p_1 + p_2 + \ldots + p_n)^2}{(p_1^2 + p_2^2 + \ldots + p_n^2)}$ for the additive white Gaussian noise (AWGN) channel.

Note that the weight of a pseudocodeword of $G$ reduces to the traditional Hamming weight when the pseudocodeword is a codeword of $G$, and that the weight is invariant under scaling of a pseudocodeword. The minimum pseudocodeword weight of $G$ is the minimum weight over all pseudocodewords of $G$ and is denoted by $w_{\text{BEC}}^{\text{min}}$ for the BEC (and likewise, for other channels). The minimum pseudocodeword weight $w_{\text{min}}$ is of fundamental importance as it plays an analogous role in iterative decoding as the minimum distance $d_{\text{min}}$ in ML-decoding.

**Remark 3.1.1** The definition of pseudocodeword and pseudocodeword weights are the same for generalized Tanner graphs, wherein the constraint nodes represent subcodes instead of simple parity-check nodes. The difference is that as the constraints impose more conditions to be satisfied, there are fewer possible non-codeword pseudocodewords. Therefore, a code represented by an LDPC constraint graph having stronger subcode constraints will have a larger minimum distance than a code represented by the same graph having weaker subcode constraints.
3.2 Tree Bounds

In this section, we derive lower bounds on the pseudocodeword weight for the BSC and AWGN channel, following Definition 3.1.1. We establish the following lower bounds for the minimum pseudocodeword weight:

**Theorem 3.2.1** Let $G$ be a regular bipartite graph with girth $g$ and smallest left degree $d$. Then the minimum pseudocodeword weight is lower bounded by

$$w_{\text{min}}^{\text{BSC/AWGN}} \geq \begin{cases} 1 + d + d(d - 1) + \ldots + d(d - 1)^{\frac{g-4}{4}}, & g \text{ odd} \\ 1 + d + \ldots + d(d - 1)^{\frac{g-8}{4}} + (d - 1)^{\frac{g-4}{4}}, & g \text{ even} \end{cases}.$$

Note that this lower bound holds analogously for the minimum distance $d_{\text{min}}$ of $G$ [48], and also for the size of the smallest stopping set, $s_{\text{min}}$, in a graph with girth $g$ and smallest left degree $d$ [37].

Observe that as the girth $g$ increases, the lower bound on $w_{\text{min}}$ increases. This supports the general assumption that large girth benefits the performance of codes when decoded via iterative graph-based decoding methods. Intuitively, large girth is desired so that the messages remain independent for as many decoding iterations as possible.

For generalized LDPC codes (see Definition 1.2.2), wherein the right nodes in $G$ of degree $k$ represent constraints of a $[k, k', \epsilon k]$ sub-code\(^1\), the above result is extended as:

\(^1\)Note that $\epsilon k$ and $\epsilon$ are the minimum distance and the relative minimum distance, respectively, of the sub-code.
Theorem 3.2.2 Let \( G \) be a \( k \)-right-regular bipartite graph with girth \( g \) and smallest left degree \( d \) and let the right nodes represent constraints of a \([k, k', ek]\) subcode, and let \( x = (ek - 1) \). Then:

\[
\begin{align*}
\wmin_{\text{BSC/AWGN}} \geq \begin{cases} 
1 + dx + d(d-1)x^2 + \ldots + d(d-1)^{\frac{g-6}{k-1}}x^{\frac{g-2}{k-1}}, & \frac{g}{2} \text{ odd} \\
1 + dx + \ldots + d(d-1)^{\frac{g}{k-1}}x^{\frac{g-4}{k-1}} + (d-1)^{\frac{g-4}{k-1}}x^{\frac{g}{k-1}}, & \frac{g}{2} \text{ even}
\end{cases}
\end{align*}
\]

Furthermore, we extend the notion of stopping sets to the generalized case as follows:

Definition 3.2.1 A stopping set \( S \) for a generalized LDPC code, where the right nodes in \( G \) of degree \( k \) represent constraints of a \([k, k', ek]\) sub-code, is a set of variable nodes \( S \) whose neighbors are each connected at least \( ek \) times to \( S \) in \( G \).

This definition makes sense since an optimal decoder on an erasure channel can recover at most \( ek - 1 \) erasures in a linear code of length \( k \) and minimum distance \( ek \). Thus if all constraint nodes are connected to a set \( S \) of variable nodes at least \( ek \) times, and if all the bits in \( S \) are erased, then the iterative decoder will not be able to recover any erasure bit in \( S \). Note that Definition 3.2.1 assumes that there are no idle components in the subcode, i.e. components that are 0 in all of the codewords of the subcode. By the above definition of a stopping set in a generalized Tanner graph, a similar lower bound holds for \( s_{\min} \) also.
Lemma 3.2.1 The minimum stopping set size $s_{\text{min}}$ in a $k$-right-regular bipartite graph $G$ with girth $g$ and smallest left degree $d$, wherein the right nodes represent constraints of a $[k, k', \epsilon k]$ subcode having no idle components, is lower bounded as:

\[
s_{\text{min}} \geq \begin{cases} 
1 + dx + d(d - 1)x^2 + \ldots + d(d - 1)^{\frac{g-5}{4}}x^{\frac{g-2}{4}}, & \frac{g}{2} \text{ odd} \\
1 + dx + \ldots + d(d - 1)^{\frac{g-5}{4}}x^{\frac{g-4}{4}} + (d - 1)^{\frac{g-3}{4}}x^{\frac{g-2}{4}}, & \frac{g}{2} \text{ even}
\end{cases}
\]

where $x = (\epsilon k - 1)$.

3.3 Relation to Max-Fractional Weight

The max-fractional weight of a vector $x = [x_1, \ldots, x_n]$ is defined as $w_{\text{max-frac}}(x) = \frac{\sum x_i^{\max}}{\max x_i}$. The max-fractional weight of pseudocodewords in LP decoding (see [13]) is analogous to the pseudocodeword weight in MS decoding.

It is worth noting that for any pseudocodeword $p$, the pseudocodeword weight of $p$ on the BSC and AWGN channels relates to the max-fractional weight of $p$ as follows:

Lemma 3.3.1 For any pseudocodeword $p$, $w_{\text{BSC/AWGN}}^{\text{BSC/AWGN}}(p) \geq w_{\text{max-frac}}(p)$.

It follows that $w_{\text{min}}^{\text{BSC/AWGN}} \geq d_{\text{frac}}^{\text{max}}$, the max-fractional distance which is the minimum max-fractional weight over all $p$. Consequently, the bounds established in [13] for $d_{\text{frac}}^{\text{max}}$ are also lower bounds for $w_{\text{min}}$. One such bound is given by the following theorem.

Theorem 3.3.1 (Feldman [13]) Let $\deg_l^-$ (respectively, $\deg_r^-$) denote the smallest left degree (respectively, right degree) in a bipartite graph $G$. Let $G$ be a Tanner graph with $\deg_l^- \geq 3$, $\deg_r^- \geq 2$, and girth $g$, with $g > 4$. Then

\[
d_{\text{frac}}^{\text{max}} \geq (\deg_l^- - 1)^{\left\lceil \frac{g}{2} \right\rceil - 1}.
\]
Corollary 3.3.2 Let $G$ be a Tanner graph with $\deg_l^{-} \geq 3$, $\deg_r^{-} \geq 2$, and girth $g$, with $g > 4$. Then

$$w_{\min}^{BSC/\text{AWGN}} \geq (\deg_l^{-} - 1)^{\lceil \frac{g}{4} \rceil - 1}.$$

Note that Corollary 3.3.2, which is essentially the result obtained in Theorem 3.2.1, makes sense due to the equivalence between the LP polytope and GC polytope (see chapter 2).

3.4 Bounds in Terms of Support Size and Maximum Component Value

The support size of a pseudocodeword $p$ has been shown to upper bound its weight on the BSC/AWGN channel [19], i.e.

$$w^{BSC/\text{AWGN}}(p) \leq |\text{supp}(p)|.$$

Hence, from Lemma 2.3.1, it follows that $w_{\min}^{BSC/\text{AWGN}} \leq s_{\min}$.

We now bound the weight of a pseudocodeword $p$ based on its maximum component value $t$ and its support size $|\text{supp}(p)|$. In Chapter 7, we show that all irreducible pseudocodewords realizable in a finite cover cannot have any component larger than some fixed value $t$; this value depends on the structure of the graph.

Definition 3.4.1 Let $G$ be a Tanner graph. Then the maximum component value an irreducible pseudocodeword of $G$ can have is called the $t$-value of $G$, and will be denoted by $t$. 
Example 3.4.1 Consider the graph in Figure 3.1. The pseudocodeword $p = (1,1,1,2,1,1,1)$ is an irreducible pseudocodeword of $G$ having maximum component 2, and any pseudocodeword of $G$ having a component larger than 2 is reducible. The $t$-value of this graph is therefore equal to two.

Figure 3.1. Tanner graph with $t$-value equal to 2.

Thus, when $t$ is known, we can bound the weight of all irreducible pseudocodewords as follows:

Lemma 3.4.1 Suppose in an LDPC constraint graph $G$ every irreducible lift-realizable pseudocodeword $p = (p_1, p_2, \ldots, p_n)$ with support set $V$ has components $0 \leq p_i \leq t$, for $1 \leq i \leq n$, then: (a) $w_{AWGN}(p) \geq \frac{2t^2}{(1+t^2)(t-1)+2t} |V|$, and (b) $w_{BSC}(p) \geq \frac{1}{t} |V|$.

For some graphs with large $|V|$, the $t$-value may still be small which makes the above lower bound large. Since the support of any pseudocodeword is a stopping set (Lemma 2.3.1), $w_{\text{min}}$ can be lower bounded in terms of $s_{\text{min}}$ and $t$. Thus, stopping sets are also important in the BSC and the AWGN channel.

Further, we can bound the weight of good and bad pseudocodewords (see Definitions 2.5.1, 2.5.2) separately, as shown below:
**Theorem 3.4.1** For an \([n, k, d_{\text{min}}]\) code represented by an LDPC constraint graph \(G\): (a) if \(p\) is a good pseudocodeword of \(G\), then \(w_{\text{BSC/AWGN}}^{\text{BSC/AWGN}}(p) \geq w_{\text{max}}(p) \geq d_{\text{min}}\), and (b) if \(p\) is a bad pseudocodeword \([22]\) of \(G\), then \(w_{\text{BSC/AWGN}}(p) \geq w_{\text{max}}(p) \geq \frac{s_{\text{min}}}{t}\), where \(t\) is as in Lemma 3.4.1.

Intuitively, it makes sense for good pseudocodewords (see Definition 2.5.1), i.e., those pseudocodewords that are not problematic for iterative decoding, to have a weight larger than the minimum distance of the code, \(d_{\text{min}}\). However, we note that bad pseudocodewords can also have weight larger than \(d_{\text{min}}\).

### 3.5 Proofs

**Theorem 3.2.1**  
**Proof:**

![Single parity check code](image)

\[ \alpha_i \leq \sum_{j \neq i} \alpha_j \]

\(\alpha_0\), \(\alpha_1\), \(\alpha_2\), \(\alpha_{i-1}\)

Single parity check code.

![Local tree structure for a \(d\)-left regular graph](image)

\[ d\alpha_0 \leq \sum_{j \in L_0} \alpha_j, \]
\[ d(d - 1)\alpha_0 \leq \sum_{j \in L_1} \alpha_j, \]
\[ \vdots \]

Local tree structure for a \(d\)-left regular graph.
Case: $\frac{g}{2}$ odd. At a single constraint node, the following inequality holds:

$$\alpha_i \leq \sum_{j \neq i} \alpha_j.$$  

Applying this in the LDPC constraint graph enumerated as a tree with the root node corresponding to the dominant pseudocodeword component $\alpha_0$, we have

$$\alpha_1 \leq \sum_{j \in L_0} \alpha_j,$$

where $L_0$ corresponds to variable nodes in the first level (level 0) of the tree. Similarly, we have

$$d(d - 1)\alpha_1 \leq \sum_{j \in L_1} \alpha_j,$$

and so on, until,

$$d(d - 1)\left(\frac{g-6}{4}\right)\alpha_1 \leq \sum_{j \in L_\frac{g-6}{4}} \alpha_j.$$  

Since the LDPC graph has girth $g$, the variable nodes up to level $L_{\frac{g-6}{4}}$ are all distinct. The above inequalities yield:

$$[1 + d + d(d - 1) + \cdots + d(d - 1)\left(\frac{g-6}{4}\right)]\alpha_1 \leq \sum_{i \in \{0\} \cup L_0 \cup \cdots \cup L_{\frac{g-6}{4}}} \alpha_i \leq \sum_{\text{all}} \alpha_i.$$  

Without loss of generality, let us assume, $\alpha_1, \ldots, \alpha_e$ to be the $e$ dominant components in $p$. That is, $\alpha_1 + \alpha_2 + \cdots + \alpha_e \geq \frac{\sum_i \alpha_i}{2}$. Since, each is at most $\alpha_1$, we

\footnote{Note that $L_i$ refers to the level for which the exponent of the $(d - 1)$ term is $i$.}
have $\sum_{i=1}^{e} \alpha_i \leq e\alpha_1$. This implies that

$$e\alpha_0 \geq \sum_{i=0}^{e-1} \alpha_i \geq \sum_i \frac{\alpha_i}{2} \geq \frac{[1 + d + d(d - 1) + \cdots + d(d - 1) \frac{g - 6}{d - 1}]}{2}.$$

$$\Rightarrow e \geq \frac{[1 + d + d(d - 1) + \cdots + d(d - 1) \frac{g - 6}{d - 1}]}{2}.$$

Since $w_{BSC} = 2e$, the result follows. (The case when $\frac{g}{2}$ is even is treated similarly.)

**AWGN case:** Let $x = \frac{[1 + d + d(d - 1) + \cdots + d(d - 1) \frac{g - 6}{d - 1}]}{2}$. Since,

$$\sum_{i=1}^{e} \alpha_i \geq \frac{[1 + d + d(d - 1) + \cdots + d(d - 1) \frac{g - 6}{d - 1}]}{2} \alpha_1,$$

we can write $\sum_{i=1}^{e} \alpha_i = (x + y)\alpha_1$, where $y$ is some non-negative quantity. Suppose $\alpha_1 + \cdots + \alpha_e = \alpha_{e+1} + \cdots \alpha_n$. Then,

$$w = \frac{(\sum_{i=1}^{n} \alpha_i)^2}{\sum_{i=1}^{n} \alpha_i^2} \geq \frac{(2 \sum_{i=1}^{e} \alpha_i)^2}{2 \sum_{i=1}^{e} \alpha_i^2}.$$

Since we have $\sum_{i=1}^{n} \alpha_i^2 \leq 2 \sum_{i=1}^{e} \alpha_i^2 \leq 2\alpha_1(\sum_{i=1}^{e} \alpha_i) = 2(x + y)\alpha_1^2$, we get,

$$w \geq \frac{(2(x + y)\alpha_1)^2}{2(x + y)\alpha_1^2} = \frac{4(x + y)^2\alpha_1^2}{2(x + y)\alpha_1^2} = 2(x + y) \geq 2x.$$

(The case $\alpha_1 + \cdots + \alpha_e > \alpha_{e+1} + \cdots \alpha_n$ is treated similarly.)

**Theorem 3.2.2**  
**Proof:** As in the proof of Theorem 3.2.1, where we note that for a single constraint with neighbors having pseudocodeword components $\alpha_1, \ldots, \alpha_k$, we have the following relation (for $\alpha_i, i = 1, \ldots, k$):

$$(\epsilon k)\alpha_i \leq \sum_{j=1}^{k} \alpha_j.$$
The result follows by applying this inequality at every constraint node as in the proof of Theorem 3.2.1.

**Lemma 3.2.1 Proof:** Let \( \frac{g}{2} \) be odd. Consider the LDPC graph enumerated as a tree from a root node. Without loss of generality, let the root node participate in a stopping set. Then, at the first level of constraint nodes, there must be at least \( \epsilon k - 1 \) variable nodes connecting each constraint node that the root node is connected to, and all these variable nodes also participate in the same stopping set. Hence there are at least \( d(\epsilon k - 1) \) other variable nodes at the first variable level from the root node, also belonging to the same stopping set. The constraint nodes that these variable node are connected to in the subsequent level must each have \( \epsilon k - 1 \) additional variable node neighbors belonging to the stopping set, and so on. Enumerating in this manner up to the \( L_{\frac{g-6}{4}} \) level of variable nodes from the root node, where \( L_{\frac{g-6}{4}} \) is as in Theorem 3.2.1, the number of variable nodes belonging to the stopping set that includes the root node is at least \( 1 + d(\epsilon k - 1) + d(d - 1)(\epsilon k - 1)^{2} + \cdots + d(d - 1)^{\frac{g-6}{4}}(\epsilon k - 1)^{\frac{g-2}{4}} \). Since we began with an arbitrary root node, we have \( s_{\min} \geq 1 + d(\epsilon k - 1) + d(d - 1)(\epsilon k - 1)^{2} + \cdots + d(d - 1)^{\frac{g-6}{4}}(\epsilon k - 1)^{\frac{g-2}{4}} \). The case when \( \frac{g}{2} \) is even is treated similarly.

**Lemma 3.3.1 Proof:** Let \( \mathbf{p} = (\alpha_1, \ldots, \alpha_n) \) be a pseudocodeword of \( G \), and without loss of generality, let \( \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_n \). To establish the inequality for the AWGN channel, we need to show

\[
\frac{\left( \sum_{i=1}^{n} \alpha_i \right)^2}{\sum_{i=1}^{n} \alpha_i^2} \geq \frac{\sum_{i=1}^{n} \alpha_i}{\max_i \alpha_i} = \frac{\sum_{i=1}^{n} \alpha_i}{\alpha_1}.
\]
Since $\sum_i \alpha_i^2 \leq \alpha_1^2 + \alpha_1 \alpha_2 + \cdots + \alpha_1 \alpha_n = \alpha_1 (\sum_i \alpha_i)$, this implies

$$\left(\sum_{i=1}^n \alpha_i\right)^2 \geq \left(\frac{\sum_{i=1}^n \alpha_i}{\alpha_1 \sum_{i=1}^n \alpha_i}\right) \geq \sum_{i=1}^n \alpha_i \alpha_1.$$

Hence, $w_{AWGN}(p) \geq w_{\text{max-frac}}(p)$.

To establish the bound for the BSC, let $e$ be the smallest number such that $\sum_{i=1}^e \alpha_i = \sum_{i=e+1}^n \alpha_i$. Then $w(BSC)(p) = 2e$. Moreover, $w_{\text{max-frac}}(p) = \sum_{i=1}^e \frac{\alpha_i}{\alpha_1} = \frac{2\sum_{i=1}^e \alpha_i}{\alpha_1}$. Each $\alpha_i \leq \alpha_1 \Rightarrow w_{\text{max-frac}}(p) \leq \frac{2\alpha_1}{\alpha_1} = 2e = w(BSC)(p)$. Now suppose $\sum_{i=1}^e \alpha_i > \sum_{i=e+1}^n \alpha_i$. Then, for some $\delta > 0$, we have $\sum_{i=1}^e \alpha_i = \sum_{i=e+1}^n \alpha_i + \delta$. We have $w_{\text{max-frac}}(p) = \sum_{i=1}^e \frac{\alpha_i}{\alpha_1} = \frac{\sum_{i=1}^e \alpha_i + \sum_{i=e+1}^n \alpha_i}{\alpha_1}$. Note that $\sum_{i=1}^e \alpha_i + \sum_{i=e+1}^n \alpha_i < 2\sum_{i=1}^e \alpha_i < (2e)\alpha_1$. Thus, $w_{\text{max-frac}}(p) < \frac{2\alpha_1}{\alpha_1} = 2e$.

**Corollary 3.3.2**  
**Proof:** Follows from Lemma 3.3.1 and Theorem 3.3.1.

**Lemma 3.4.1**  
**Proof:** (a) **AWGN case:** Let $n_k$ be the number of $p_i$’s that are equal to $k$, for $k = 1, \ldots, t$. The pseudocodeword weight is then equal to:

$$w_{\text{AWGN}}(p) = \frac{(n_1 + 2n_2 + \cdots + tn_t)^2}{(n_1 + 2^2n_2 + \cdots + t^2n_t)}.$$

Now, we have to find a number $r$ such that $w_{\text{AWGN}}(p) \geq r|\text{supp}(p)|$. Note however, that $|\text{supp}(p)| = n_1 + n_2 + \cdots + n_t$. This implies that for an appropriate choice of $r$, we have

$$\frac{(n_1 + 2n_2 + \cdots + tn_t)^2}{(n_1 + 2^2n_2 + \cdots + t^2n_t)} \geq r(n_1 + \cdots + n_t),$$

or

$$\sum_{i=1}^t i^2 n_i^2 \geq \sum_{i=1}^t \sum_{j=i+1}^t \frac{(i^2 + j^2) r - 2ij}{1 - r} n_i n_j.$$

(*)
Note that $r < 1$ in the above. Clearly, if we set $r$ to be the minimum over all $1 \leq i < j \leq n$ such that $(i^2 + j^2)r \leq 2ij$, then it can be verified that this choice of $r$ will ensure that $(\ast)$ is true. This implies $r = \frac{2t}{1+t^2}$ (for $i = 1, j = t$).

However, observe that left-hand-side (LHS) in $(\ast)$ can be written as the following LHS:

$$\frac{1}{t-1} \sum_{i=1}^{t} \sum_{j=i+1}^{t} (i^2n_i^2 + j^2n_j^2) \geq \sum_{i=1}^{t} \sum_{j=i+1}^{t} \frac{(i^2 + j^2)r - 2ij}{1-r} n_i n_j.$$  

Now, using the inequality $a^2 + b^2 \geq 2ab$, $r$ can be taken as the minimum over all $1 \leq i < j \leq n$ such that $\frac{1}{t-1} (i^2n_i^2 + j^2n_j^2) \geq \frac{r(i^2 + j^2) - 2in_i n_j}{1-r}$. This gives $r = \frac{2t^2}{(1+t^2)(t-1)+2t}$, thereby proving the lemma in the AWGN case.

**BSC case:** Since $w_{BSC}(p) \geq w_{\text{max-frac}}(p)$, we have from Lemma 3.3.1 that $w_{BSC}(p) \geq p_1 + \ldots + p_n \geq \frac{|\text{supp}(p)|}{t}$.

**Theorem 3.4.1 Proof:** Let $p$ be a good pseudocodeword. This means that if for any weight vector $w$ we have $cw^T \geq 0$ for all $0 \neq c \in C$, then, $pw^T \geq 0$.

Let us now consider the BSC and the AWGN cases separately.

**BSC case:** Suppose at most $\frac{d_{\min}}{2}$ errors occur in channel. Then, the corresponding weight vector $w$ will have $\frac{d_{\min}}{2}$ or fewer components equal to $-1$, and the rest of the components equal to $+1$. This implies that the cost of any $0 \neq c \in C$ (i.e., $cw^T$) is at least 0 since there are at least $d_{\min}$ 1’s in support of any $0 \neq c \in C$. Since $p$ is a good pseudocodeword, it must also have positive cost, i.e. $pw^T \geq 0$. Let us assume the worst case, that the $-1$’s occur in the dominant $\frac{d_{\min}}{2}$ positions of $p$, and without loss of generality, assume $p_1 \geq p_2 \geq \ldots \geq p_n$. (Therefore, $w = (-1, -1, \ldots, -1, +1, +1, \ldots, +1)$.) Positive cost of $p$ implies $p_1 + \ldots + p_{d_{\min}/2} \leq p_{(d_{\min}/2)+1} + \ldots + p_n$. So we have $e \geq \frac{d_{\min}}{2}$, where $e$ is as defined
in the pseudocodeword weight of $p$ for the BSC. The result follows.

**AWGN case:** Without loss of generality, let $p_1$ be dominant component of $p$. Set the weight vector $w = (1-d_{\min}, 1, \ldots, 1)$. Then it can be verified that $cw^T \geq 0$ for any $0 \neq c \in C$. Since $p$ is a good pseudocodeword, this implies $p$ also must have positive cost. Cost of $p$ is $(1-d_{\min})p_1 + p_2 + \ldots + p_n \geq 0 \Rightarrow d_{\min} \leq \frac{p_1 + \ldots + p_n}{p_1}$.

Note, that the right-hand-side (RHS) is $w_{\max-frac}(p)$; hence, the result follows from Lemma 3.3.1.

Now let us consider $p$ to be a bad pseudocodeword. From Lemma 2.3.1, we have $|\text{supp}(p)| \geq s_{\min}$. Therefore, $w_{\max-frac}(p) \geq \frac{s_{\min}}{t}$ (since $p_1 = t$ is the maximum component of $p$), and hence, the result follows by Lemmas 3.3.1 and 3.4.1.
CHAPTER 4

PSEUDOCODEWORD WEIGHTS OF NON-BINARY LDPC CODES

For some applications, such as magnetic recording, codes over larger alphabets are desirable. This chapter investigates non-binary LDPC codes, and extends the previous decoding analysis to this case. In particular, pseudocodewords for $p$-ary LDPC codes and the notion of pseudocodeword weight for the $p$-ary symmetric channel are introduced, and bounds are obtained for the minimum pseudocodeword weight. These results hold for alphabets of size $p$, where $p$ is a prime, as well as for alphabets of prime power order, $q = p^m$.

4.1 $p$-ary LDPC Codes

Let $H$ be a parity check matrix representing a $p$-ary LDPC code $C$. As it is a low-density code, the parity-check matrix is sparse in the number of non-zero entries. The corresponding LDPC constraint graph $G$ that represents $H$ is an incidence graph of the parity check matrix as in the binary case. However, each edge of $G$ is now assigned a weight which is the value of the corresponding non-zero entry in $H$. In [10, 11], LDPC codes over $GF(q)$ are considered for transmission over binary modulated channels, whereas in [44], LDPC codes over integer rings are considered for higher-order modulation signal sets.
For convenience, we consider the special case wherein each of these edge weights are equal to one. This is the case when the parity check matrix has only zeros and ones. We will refer to such a graph as a binary LDPC constraint graph representing a $p$-ary LDPC code $C$.

We first show that if the LDPC graph corresponding to $H$ is $d$-left (variable-node) regular, then the same tree bound of Theorem 3.2.1 holds. That is,
Lemma 4.1.1 If $G$ is a $d$-left regular bipartite LDPC constraint graph with girth $g$ and represents a $p$-ary LDPC code $C$, then, the minimum distance of the $p$-ary LDPC code $C$ is lower bounded as

$$d_{\min} \geq T(d, g).$$

We note here that in general this lower bound is not met and typically $p$-ary LDPC codes that have the above graph representation have minimum distances larger than the above lower bound.

4.2 Pseudocodewords of $p$-ary Codes

Recall from [25, 28] that a pseudocodeword of an LDPC constraint graph $G$ is a valid codeword in some finite cover of $G$. To define a pseudocodeword for a $p$-ary LDPC code, we will restrict the discussion to LDPC constraint graphs that have edge weights of unity among all their edges – in other words, binary LDPC constraint graphs that represent $p$-ary LDPC codes. A finite cover of a graph is defined in a natural way as in [28] wherein all edges in the finite cover also have an edge weight of unity. For the rest of this section, let $G$ be a LDPC constraint graph of a $p$-ary LDPC code $C$ of block length $n$, and let the weights on every edge of $G$ be unity. We define a pseudocodeword $F$ of $G$ as a $n \times p$ matrix of the form

$$F = \begin{bmatrix}
  f_{0,0} & f_{0,1} & f_{0,2} & \cdots & f_{0,p-1} \\
  f_{1,0} & f_{1,1} & f_{1,2} & \cdots & f_{1,p-1} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  f_{n-1,0} & f_{n-1,1} & f_{n-1,2} & \cdots & f_{n-1,p-1} 
\end{bmatrix}.$$
where the pseudocodeword $F$ forms a valid codeword $\hat{c}$ in a finite cover $\hat{G}$ of $G$ and $f_{i,j}$ is the fraction of variable nodes in the $i^{th}$ variable cloud, for $0 \leq i \leq n - 1$, of $\hat{G}$ that have the assignment (or, value) equal to $j$, for $0 \leq j \leq p - 1$, in $\hat{c}$.

A $p$-ary symmetric channel is shown in Figure 4.1. The input and the output of the channel are random variables belonging to a $p$-ary alphabet that can be denoted as $\{0, 1, 2, \ldots, p - 1\}$. An error occurs with probability $\epsilon$, which is parameterized by the channel, and in the case of an error, it is equally probable for an input symbol to be altered to any one of the remaining symbols.

Following the definition of pseudocodeword weight for the binary symmetric channel [19], we provide the following definition for the weight of a pseudocodeword on the $p$-ary symmetric channel. For a pseudocodeword $F$, let $F'$ be the sub-matrix obtained by removing the first column in $F$. (Note that the first column in $F$ contains the entries $f_{0,0}, f_{1,0}, f_{2,0}, \ldots, f_{n-1,0}$.) Then the weight of a pseudocodeword $F$ on the $p$-ary symmetric channel is defined as follows.
Definition 4.2.1 Let $e$ be a number such that the sum of the $e$ largest components in the matrix $F'$, say, $f_{i_1,j_1}, f_{i_2,j_2}, \ldots, f_{i_e,j_e}$, exceeds $\sum_{i \neq i_1, i_2, \ldots, i_e} (1 - f_{i,0})$. Then the weight of $F$ on the $p$-ary symmetric channel is defined as

$$w_{PSC}(F) = \begin{cases} 
2e, & \text{if } f_{i_1,j_1} + \cdots + f_{i_e,j_e} = \sum_{i \neq i_1, i_2, \ldots, i_e} (1 - f_{i,0}), \\
2e - 1, & \text{if } f_{i_1,j_1} + \cdots + f_{i_e,j_e} > \sum_{i \neq i_1, i_2, \ldots, i_e} (1 - f_{i,0}). 
\end{cases}$$

Note that in the above definition, none of the $j_k$’s, for $k = 1, 2, \ldots, e$, are equal to zero, and all the $i_k$’s, for $k = 1, 2, \ldots, e$, are distinct. That is, we choose at most one component in every row of $F'$ when picking the $e$ largest components. (An explanation on the above definition of “weight” is given at the end of this section.)

We note that, in general, this bound is rather loose. (The inequality in (4.2), in the proof of Lemma 4.3.1, is typically not tight.) Moreover, we expect that $p$-ary LDPC codes to have larger minimum pseudocodeword weights than corresponding binary LDPC codes. By corresponding binary LDPC codes, we mean the codes obtained by interpreting the given LDPC constraint graph as one representing a binary LDPC code.

4.2.1 Derivation of pseudocodeword weight for $p$-ary LDPC codes on the $p$-ary symmetric channel

Suppose the all-zero codeword is sent across a $p$-ary symmetric channel and the vector $r = (r_0, r_1, \ldots, r_{n-1})$ is received. Then errors occur in positions where $r_i \neq 0$. Let $S = \{i \mid r_i \neq 0\}$ and let $S^c = \{i \mid r_i = 0\}$. The distance between $r$ and
a pseudocodeword $F$ is defined as

$$d(r, F) = \sum_{i=0}^{n-1} \sum_{k=0}^{p-1} \chi(r_i \neq k) f_{i,k} = \sum_{i=0}^{n-1} (1 - f_{i,r_i}), \quad (4.1)$$

where $\chi(P)$ is an indicator function that is equal to 1 if the proposition $P$ is true and is equal to 0 otherwise.

The distance between $r$ and the all-zero codeword 0 is

$$d(r, 0) = \sum_{i=0}^{n-1} \chi(r_i \neq 0)$$

which is the Hamming weight of $r$ and can be obtained from Equation (4.1).

The iterative decoder chooses in favor of $F$ instead of the all-zero codeword 0 when $d(r, F) \leq d(r, 0)$. That is, if

$$\sum_{i \in S_c} (1 - f_{i,0}) + \sum_{i \in S} (1 - f_{i,r_i}) \leq \sum_{i \in S} 1.$$

The condition for choosing $F$ over the all-zero codeword reduces to

$$\left\{ \sum_{i \in S_c} (1 - f_{i,0}) \leq \sum_{i \in S} f_{i,r_i} \right\}.$$

Hence, we define the weight of a pseudocodeword $F$ in the following manner.

Let $e$ be a number such that the sum of the $e$ largest components in the matrix $F'$, say, $f_{i_1,j_1}, f_{i_2,j_2}, \ldots, f_{i_e,j_e}$, exceeds $\sum_{i \neq i_1, i_2, \ldots, i_e} (1 - f_{i,0})$. Then the weight of $F$ on the $p$-ary symmetric channel is defined as

$$w_{PSC}(F) = \begin{cases} 2e, & \text{if } f_{i_1,j_1} + \cdots + f_{i_e,j_e} = \sum_{i \neq i_1, i_2, \ldots, i_e} (1 - f_{i,0}) \\ 2e - 1, & \text{if } f_{i_1,j_1} + \cdots + f_{i_e,j_e} > \sum_{i \neq i_1, i_2, \ldots, i_e} (1 - f_{i,0}) \end{cases}$$
Note that in the above definition, none of the $j_k$'s, for $k = 1, 2, \ldots, e$, are equal to zero, and all the $i_k$'s, for $k = 1, 2, \ldots, e$, are distinct. That is, we choose at most one component in every row of $F'$ when picking the $e$ largest components. The received vector $r = (r_0, r_1, \ldots, r_{n-1})$ that has the following components: $r_{i_1} = j_1, r_{i_2} = j_2, \ldots, r_{i_e} = j_e, r_i = 0$, for $i \notin \{i_1, i_2, \ldots, i_e\}$, will cause the decoder to make an error and choose $F$ over the all-zero codeword.

Observe that for a codeword, the above weight definition reduces to the Hamming weight. If $F$ represents a codeword $c$, then exactly $w = \text{wt}_H(c)$, the Hamming weight of $c$, rows in $F'$ contain the entry 1 in exactly one column, and contain zeros in the remaining entries. Furthermore, the matrix $F$ has the entry 0 in the first column of these $w$ rows and has the entry 1 in the first column of the remaining rows. Therefore, from the weight definition of $F$, $e = \frac{w}{2}$ and the weight of $F$ is $2e = w$.

4.3 Tree Bound on $w_{\text{min}}^{PSC}$

We define the $p$-ary minimum pseudocodeword weight of $G$ (or, minimum pseudoweight) as in the binary case, i.e., as the minimum weight of a pseudocodeword among all finite covers of $G$, and denote this as $w_{\text{min}}(G)$ or $w_{\text{min}}$ when it is clear that we are referring to the graph $G$.

**Lemma 4.3.1** Let $G$ be a $d$-left regular bipartite graph with girth $g$ that represents a $p$-ary LDPC code $C$. Then the minimum pseudocodeword weight $w_{\text{min}}$ on the $p$-ary symmetric channel is lower bounded as

$$w_{\text{min}} \geq T(d, g) = \begin{cases} (1 + d + d(d - 1) + d(d - 1)^2 + \ldots + d(d - 1)^{\frac{q-2}{2}}), & \frac{q}{2} \text{ odd} , \\ (1 + d + d(d - 1) + \ldots + d(d - 1)^{\frac{q-2}{2}} + (d - 1)^{\frac{q-4}{2}}), & \frac{q}{2} \text{ even} . \end{cases}$$
4.4 Tree Bound on $w_{\text{min}}^{\text{AWGN}}$

Following the definition of effective distance $d_{\text{eff}}^2(F, c)$, between a pseudocodeword $F$ and a codeword $c$ on the AWGN channel, presented in [19], the weight of a pseudocodeword $F$ is given by $d_{\text{eff}}^2(F, 0)$. On simplifying the expression in [19], the weight of pseudocodeword $F$ on the AWGN channel is given by

$$w_{q-\text{AWGN}}(F) = \frac{(\sum_{i=0}^{n-1} \sum_{m=0}^{q-1} f_{i,m} m^2)^2}{\sum_{i=0}^{n-1} (\sum_{m=0}^{q-1} f_{i,m} m)^2}$$

The above weight definition assumes $q$-ary pulse amplitude modulation, i.e., the symbols sent across the channel belong to the signal set \{0, 1, 2, \ldots, q - 1\}.

**Theorem 4.4.1** Let $G$ be a $d$-left regular bipartite graph with girth $g$ that represents a $q$-ary LDPC code $C$. Then the minimum pseudocodeword weight $w_{\text{min}}$ on the AWGN channel is lower bounded as

$$w_{\text{min}} \geq T(d, g) = \begin{cases} 1 + d + d(d - 1) + d(d - 1)^2 + \ldots + d(d - 1)^{\frac{g-6}{4}}, & g \text{ odd}, \\ 1 + d + d(d - 1) + \ldots + d(d - 1)^{\frac{g-8}{4}} + (d - 1)^{\frac{g-4}{4}}, & g \text{ even}. \end{cases}$$

4.5 Proofs

**Lemma 4.1.1** Proof: The proof is essentially the same as in the binary case. Enumerate the graph as a tree starting at an arbitrary variable node. Furthermore, assume that a codeword in $C$ contains the root node in its support. The
root variable node (at layer $L_0$ of the tree) connects to $d$ constraint nodes in the next layer (layer $L_1$) of the tree. These constraint nodes are each connected to some sets of variable nodes in layer $L_2$, and so on. Since the graph has girth $g$, the nodes enumerated up to layer $L_{\frac{g-6}{2}}$ when $\frac{g}{2}$ is odd (respectively, $L_{\frac{g-4}{2}}$ when $\frac{g}{2}$ is even) are all distinct. Since the root node belongs to a codeword, say $c$, it assumes a non-zero value in $c$. Since the constraints must be satisfied at the nodes in layer $L_1$, at least one node in Layer $L_2$ for each constraint node in $L_1$ must assume a non-zero value in $c$. (This is under the assumption that an edge weight times a (non-zero) value, assigned to the corresponding variable node, is not zero in the code alphabet. Since we have chosen the edge weights to be unity, such a case will not arise here. But also more generally, such cases will not arise when the alphabet and the arithmetic operations are that of a finite field. However, when working over other structures, such as finite integer rings and more general groups, such cases could arise.)

Under the above assumption, that there are at least $d$ variable nodes (i.e., at least one for each node in layer $L_1$) in layer $L_2$ that are non-zero in $c$. Continuing this argument, it is easy to see that the number of non-zero components in $c$ is at least $1 + d + d(d - 1) + \cdots + d(d - 1)^{\frac{g-6}{4}}$ when $\frac{g}{2}$ is odd, and $1 + d + d(d - 1) + \cdots + d(d - 1)^{\frac{g-8}{4}} + (d - 1)^{\frac{g-4}{4}}$ when $\frac{g}{2}$ is even. Thus, the desired lower bound holds.
Lemma 4.3.1 Proof:

\[
(1 - f_i,0) \leq \sum_{j \neq i} (1 - f_j,0)
\]

Single constraint code.

Case: \( \frac{d}{2} \) odd. Consider a single constraint node with \( r \) variable node neighbors as shown in Figure 4.2. Then, for \( i = 0, 1, \ldots, r - 1 \) and \( k = 0, 1, \ldots, p - 1 \), the following inequality holds:

\[
f_{i,k} \leq \sum_{j \neq i} \sum_{\sigma_j, \sum \sigma_j + k = 0 \mod p} \frac{f_{j,\sigma_j} \sigma_j}{\sum \sigma_j}
\]

where the middle summation is over all possible assignments \( \sigma_j \in \{0, 1, \ldots, p - 1\} \) to the variable nodes \( j \neq i \) such that \( k + \sum_{j \neq i} \sigma_j = 0 \mod p \), i.e., this is a valid assignment for the constraint node. The innermost summation in the denominator is over all \( j \neq i \).

Figure 4.2. Inequalities among the pseudocodeword components.
However, for $i = 0, 1, \ldots, r - 1$, the following (weaker) inequality also holds:

\[ (1 - f_{i,0}) \leq \sum_{j \neq i} (1 - f_{j,0}) \]  

(4.2)

Now let us consider a $d$-left regular LDPC constraint graph representing a $p$-ary LDPC code. We will enumerate the LDPC constraint graph as a tree from an arbitrary root variable node, as shown in Figure 4.2. Let $F$ be a pseudocodeword matrix for this graph. Without loss of generality, let us assume that the component $(1 - f_{0,0})$ corresponding to the root node is the maximum among all $(1 - f_{i,0})$ over all $i$.

Applying the inequality in (4.2) at every constraint node in first constraint node layer of the tree, we obtain

\[ d(1 - f_{0,0}) \leq \sum_{j \in L_0} (1 - f_{j,0}), \]

where $L_0$ corresponds to variable nodes in first level of the tree. Subsequent application of the inequality in (4.2) to the second layer of constraint nodes in the tree yields

\[ d(d - 1)(1 - f_{0,0}) \leq \sum_{j \in L_1} (1 - f_{j,0}), \]

Continuing this process until layer $L_{\frac{d-6}{4}}$, we obtain

\[ d(d - 1)^{\frac{a-6}{4}} (1 - f_{0,0}) \leq \sum_{j \in L_{\frac{d-6}{4}}} (1 - f_{j,0}). \]
Since the LDPC graph has girth $g$, the variable nodes up to level $L_{g+6}$ are all distinct. The above inequalities yield:

\[ [1 + d + d(d-1) + \cdots + d(d-1)^{g+6}](1 - f_{0,0}) \leq \sum_{i \in \{0\} \cup L_{g+1} \cup \cdots \cup L_{g+6}} (1 - f_{i,0}) \leq \sum_{\text{all } i} (1 - f_{i,0}) \quad (4.3) \]

Let $e$ be the smallest number such that there are $e$ maximal components $f_{i_1,j_1}, f_{i_2,j_2}, \ldots, f_{i_e,j_e}$, for $i_1, i_2, \ldots, i_e$ all distinct and $j_1, j_2, \ldots, j_e \in \{1, 2, \ldots, p-1\}$, in $F'$ (the sub-matrix of $F$ excluding the first column in $F$) such that

\[ f_{i_1,j_1} + f_{i_2,j_2} + \cdots + f_{i_e,j_e} \geq \sum_{i \notin \{i_1, i_2, i_3, \ldots, i_e\}} (1 - f_{i,0}). \]

Then, since none of the $j_k$'s, $k = 1, 2, \ldots, e$, are zero, we clearly have

\[ (1 - f_{i_1,0}) + (1 - f_{i_2,0}) + \cdots + (1 - f_{i_e,0}) \geq f_{i_1,j_1} + f_{i_2,j_2} + \cdots + f_{i_e,j_e} \geq \sum_{i \notin \{i_1, i_2, i_3, \ldots, i_e\}} (1 - f_{i,0}). \]

Hence we have that

\[ 2((1 - f_{i_1,0}) + (1 - f_{i_2,0}) + \cdots + (1 - f_{i_e,0})) \geq \sum_{\text{all } i} (1 - f_{i,0}). \]

We can then lower bound this further using the inequality in (4.3) as

\[ 2((1 - f_{i_1,0}) + (1 - f_{i_2,0}) + \cdots + (1 - f_{i_e,0})) \geq [1 + d + d(d-1) + \cdots + d(d-1)^{g+6}](1 - f_{0,0}). \]

Since we assumed that $(1 - f_{0,0})$ is the maximum among $(1 - f_{i,0})$ over all $i$, we have

\[ 2e(1 - f_{0,0}) \geq 2((1 - f_{i_1,0}) + (1 - f_{i_2,0}) + \cdots + (1 - f_{i_e,0})) \geq [1 + d + d(d-1) + \cdots + d(d-1)^{g+6}](1 - f_{0,0}). \]
\[ [1 + d + d(d - 1) + \cdots + d(d - 1)^\frac{q-6}{4}] (1 - f_{0,0}). \]

This yields the desired bound

\[ w_{PSC}(F) = 2e \geq 1 + d + d(d - 1) + \cdots + d(d - 1)^{\frac{q-6}{4}}. \]

Since the pseudocodeword \( F \) was arbitrary, we also have \( w_{\min} \geq 1 + d + d(d - 1) + \cdots + d(d - 1)^{\frac{q-6}{4}} \). The case \( \frac{q}{2} \) even is treated similarly.

**Theorem 4.4.1** Proof: Let \( F \) be a pseudocodeword in \( G \). Without loss of generality, let \( (1 - f_{0,0}) \) be the maximum of \( (1 - f_{0,i}) \) over all \( i \). We will first lower bound the weight \( w_{q-\text{AWGN}}(F) \) as

\[
w_{q-\text{AWGN}}(F) = \frac{(\sum_{i=0}^{n-1} \sum_{m=0}^{q-1} f_{i,m}m^2)^2}{\sum_{i=0}^{n-1} (\sum_{m=0}^{q-1} f_{i,m})^2} \\
\geq \frac{(\sum_{i=0}^{n-1} \sum_{m=0}^{q-1} f_{i,m}m^2)}{1 - f_{0,0}}. \quad (*)
\]

This lower bound is obtained by showing that the denominator in the weight expression can be upper bounded by using the Cauchy-Schwartz inequality as follows

\[
\sum_{i=0}^{n-1} \sum_{m=0}^{q-1} (f_{i,m})^2 \leq (\sum_{i=0}^{n-1} (f_{i,1} + f_{i,2} + \cdots + f_{i,q-1})) (\sum_{i=0}^{n-1} \sum_{m=0}^{q-1} f_{i,m}m^2).
\]

Further, since \( f_{i,1} + f_{i,2} + \cdots + f_{i,q-1} = 1 - f_{i,0} \leq 1 - f_{0,0} \), we obtain the lower bound in \((*)\).

Since \( \sum_{i=0}^{n-1} \sum_{m=0}^{q-1} f_{i,m}m^2 \geq \sum_{i=0}^{n-1} (f_{i,1} + f_{i,2} + \cdots + f_{i,q-1}) = \sum_{i=0}^{n-1} (1 - f_{i,0}) \), we have

\[
w_{q-\text{AWGN}}(F) \geq \frac{\sum_{i=0}^{n-1} (1 - f_{i,0})}{1 - f_{0,0}}.
\]
Now, using the inequality (4.3) from the proof of Theorem 4.3.1 yields the desired lower bound $w_{q-AWGN}(F) \geq 1 + d + d(d - 1) + \cdots + d(d - 1)^{\frac{g-6}{2}}$ for the case $g/2$ odd. (The case $g/2$ even follows similarly.)
CHAPTER 5

STRUCTURE OF PSEUDOCODEWORDS

In this chapter we again restrict attention to the min-sum decoder. We examine the structure of lift-realizable pseudocodewords and identify sufficient conditions for certain pseudocodewords to potentially cause the min-sum iterative decoder to fail to converge to a codeword. Some of these conditions relate to subgraphs of the base Tanner graph. We also present three different examples of Tanner graphs giving rise to different types of pseudocodewords. As in chapter 3, let $G$ be a bipartite graph representing a binary LDPC code $C$, with $|V| = n$ left (variable) nodes, $|U| = m$ right (check) nodes, and edges $E \subseteq \{(v, u) | v \in V, u \in U\}$. We recall that we are only considering the set of pseudocodewords that arise from finite degree lifts of the base graph, and that by Definition 2.2.2, the pseudocodewords have non-negative integer components and so are unscaled pseudocodewords.

5.1 Structure of the Pseudocodeword Vectors

**Lemma 5.1.1** Let $p = (p_1, p_2, \ldots, p_n)$ be an unscaled pseudocodeword in the graph $G$ that represents the LDPC code $C$. Then the vector $x = p \mod 2$, obtained by reducing the entries in $p$, modulo 2, corresponds to a codeword in $C$. 
The following implications follow from Lemma 5.1.1:

- If a pseudocodeword $\mathbf{p}$ has at least one odd component, then it has at least $d_{\min}$ odd components.

- If a pseudocodeword $\mathbf{p}$ has a support size $|\text{supp}(\mathbf{p})| < d_{\min}$, then it has no odd components.

- If a pseudocodeword $\mathbf{p}$ does not contain the support of any non-zero codeword in its support, then $\mathbf{p}$ has no odd components.

**Lemma 5.1.2** A pseudocodeword $\mathbf{p} = (p_1, \ldots, p_n)$ can be written as $\mathbf{p} = \mathbf{c}^{(1)} + \mathbf{c}^{(2)} + \cdots + \mathbf{c}^{(k)} + \mathbf{r}$, where $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(k)}$, are $k$ (not necessarily distinct) codewords and $\mathbf{r}$ is some residual vector, not containing the support of any nonzero codeword in its support, that remains after removing the codewords $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(k)}$ from $\mathbf{p}$. Either $\mathbf{r}$ is the all-zero vector, or $\mathbf{r}$ is a vector comprising of 0 or even entries only.

This lemma describes a particular composition of a pseudocodeword $\mathbf{p}$. Note that the above result does not claim that $\mathbf{p}$ is reducible even though the vector $\mathbf{p}$ can be written as a sum of codeword vectors $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(k)}$, and $\mathbf{r}$. Since $\mathbf{r}$ need not be a pseudocodeword, it is not necessary that $\mathbf{p}$ be reducible structurally as a sum of codewords and/or pseudocodewords (as in Definition 2.4.1).

It is also worth noting that the decomposition of a pseudocodeword, even that of an irreducible pseudocodeword, is not unique.

**Example 5.1.1** For representation B of the $[7, 4, 3]$ Hamming code as shown in Figure 5.1, label the vertices clockwise from the top as $v_1, v_2, v_3, v_4, v_5, v_6,$ and $v_7$. The vector $\mathbf{p} = (p_1, \ldots, p_n) = (1, 2, 1, 1, 1, 0, 2)$ is an irreducible pseudocodeword and may be decomposed as $\mathbf{p} = (1, 0, 1, 0, 0, 1) + (0, 0, 1, 1, 0, 1) + $
Figure 5.1. The [7,4,3] Hamming code, Representation B.

(0,2,0,0,0,0,0) and also as $p = (1,0,1,1,1,0,0) + (0,2,0,0,0,0,2)$. In each of these decompositions, each vector in the sum is a codeword except for the last vector which is the residual vector $r$.

**Theorem 5.1.1** Let $p = (p_1,\ldots,p_n)$ be a pseudocodeword. If there is a decomposition of $p$ as in Lemma 5.1.2 such that $r = 0$, then $p$ is a good pseudocodeword as in Definition 2.5.1.

**Theorem 5.1.2** The following are sufficient conditions for a pseudocodeword $p = (p_1,\ldots,p_n)$ to be bad, as in Definition 2.5.2:

1. $w_{\text{BSC/AWGN}}(p) < d_{\text{min}}$.

2. $|\text{supp}(p)| < d_{\text{min}}$.

3. If $p$ is a non-codeword irreducible pseudocodeword and $|\text{supp}(p)| \geq \ell + 1$, where $\ell$ is the number of distinct codewords whose support is contained in $\text{supp}(p)$.

Subgraphs of the LDPC constraint graph may also give rise to bad pseudocodewords. For a pseudocodeword $p = (p_1,\ldots,p_n)$ with support $S = \{i_1,\ldots,i_k\}$, let $G_{|S}$ be the subgraph of $G$ induced by $\{v_{i_1},\ldots,v_{i_k}\}$ in $G$. For example, in Figure 2.5, the subgraph $G_{|S}$ induced by the set $S = \{v_0, v_1, v_3, v_5\}$ is shown in Figure 5.2.
Definition 5.1.1  A stopping set $S$ has property $\Theta$ if $S$ contains at least one pair of variable nodes $u$ and $v$ that are not connected by any path that traverses only via degree two check nodes in the subgraph $G_{|S}$ of $G$ induced by $S$ in $G$.

Example 5.1.2  In Figure 5.8 in section 5.2, the set $\{v_1, v_2, v_4\}$ is a minimal stopping set and does not have property $\Theta$, whereas the set $\{v_1, v_3, v_4, v_5, v_6, v_7, v_{10}, v_{11}, v_{12}, v_{13}\}$ is not minimal but has property $\Theta$. The graph in Figure 5.3 is a minimal stopping set that has property $\Theta$.

Definition 5.1.2  A variable node $v$ in an LDPC constraint graph $G$ is said to be problematic if there is a stopping set $S$ containing $v$ that is not minimal but nevertheless has no stopping set $S' \subseteq S$ for which $v \in S'$.
Examples of graphs containing problematic nodes are presented in section 5.3. Note that if a graph $G$ has a problematic node, then $G$ necessarily contains a stopping set with property $\Theta$.

The following result classifies bad non-codeword pseudocodewords, with respect to the AWGN channel, using the graph structure of the underlying pseudocodeword supports, which, by Lemma 2.3.1, are stopping sets in the LDPC constraint graph.

**Theorem 5.1.3** Let $G$ be an LDPC constraint graph representing an LDPC code $C$, and let $S$ be a stopping set in $G$. Then, the following hold:

1. If there is no non-zero codeword in $C$ whose support is contained in $S$, then all non-codeword pseudocodewords of $G$, having support equal to $S$, are bad as in Definition 2.5.2. Moreover, there exists a bad pseudocodeword in $G$ with support equal to $S$.

2. If there is at least one codeword $c$ whose support is contained in $S$, then we have the following cases:

   (a) if $S$ is minimal,

      (i) there exists a non-codeword pseudocodeword $p$ with support equal to $S$ iff $S$ has property $\Theta$.

      (ii) all non-codeword pseudocodewords with support equal to $S$ are bad.

   (b) if $S$ is not minimal,

      (i) and $S$ contains a problematic node $v$ such that $v \notin S'$ for any stopping set $S' \subset S$, then there exists a bad pseudocodeword $p$ with support $S$. Moreover, any non-codeword irreducible pseudocodeword $p$ with support $S$ is bad.
(ii) and \( S \) does not contain any problematic nodes, then every variable node in \( S \) is contained in a minimal stopping set within \( S \). Moreover, there exists a bad non-codeword pseudocodeword with support \( S \) iff either one of these minimal stopping sets is not the support of any non-zero codeword in \( C \) or one of these minimal stopping sets has property \( \Theta \).

**Example 5.1.3** The graph in Figure 5.4 is an example of case 2(b)(ii) in Theorem 5.1.3. Note that the stopping set in the figure is a disjoint union of two codeword supports and therefore, there are no non-codeword irreducible pseudocodewords.

![Figure 5.4. A non-minimal stopping set as in case 2(b)(ii) of Theorem 5.1.3.](image)

**Example 5.1.4** The graph in Figure 5.5 is an example of case 2(a). The graph has property \( \Theta \) and therefore has non-codeword pseudocodewords, all of which are bad. For example, the pseudocodeword \( p = (3, 3, 3, 1, 1, 1) \) is a bad pseudocodeword on this graph.
Figure 5.5. A minimal stopping set as in case 2(a) of Theorem 5.1.3.

5.2 Examples

In this section we present three different examples of Tanner graphs which give rise to different types of pseudocodewords.

Example 5.2.1 Figure 5.6 shows a graph that has no pseudocodeword with weight less than $d_{\text{min}}$ on the BSC and AWGN channel. For this code (or more precisely, LDPC constraint graph), the minimum distance, the minimum stopping set size, and the minimum pseudocodeword weight on the AWGN channel, are all equal to 4, i.e., $d_{\text{min}} = s_{\text{min}} = w_{\text{min}} = 4$, and the $t$-value (see Definition 3.4.1) is 2. A non-codeword irreducible pseudocodeword with a component of value 2 may be observed by assigning value 1 to the nodes in the outer and inner rings and assigning value 2 to exactly one node in the middle ring, and zeros elsewhere.

Figure 5.7 shows the performance of this code on a binary input AWGN channel, with signal to noise ratio (SNR) $E_b/N_o$, with min-sum, sum-product, and maximum-likelihood decoding. The min-sum and sum-product iterative decoding was performed for 50 decoding iterations on the LDPC constraint graph. It is evident that all three algorithms perform almost identically. Thus, this LDPC code does not have low weight (relative to the minimum distance) bad pseudocodewords, implying that the performance of the min-sum decoder, under i.i.d. Gaussian noise, will be close to the optimal ML performance.
Figure 5.6. A graph with all pseudocodewords having weight at least $d_{\text{min}}$.

Figure 5.7. Performance of Example 5.2.1 - LDPC code: Min-Sum, Sum-Product, ML decoding over the BIAWGNC.
Figure 5.8. A graph with good and bad pseudocodewords.

**Example 5.2.2** Figure 5.8 shows a graph that has both good and bad pseudocodewords. Consider $p = (1, 0, 1, 1, 1, 3, 0, 0, 1, 1, 1, 0)$. Letting $w = (1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0)$, we obtain $pw^T = -2$ and $cw^T \geq 0$ for all codewords $c$. Therefore, $p$ is a bad pseudocodeword for min-sum iterative decoding. In particular, this pseudocodeword has a weight of $w^{BSC/AWGN}(p) = 8$ on both the BSC and the AWGN channel. This LDPC graph results in an LDPC code of minimum distance $d_{\text{min}} = 8$, whereas the minimum stopping set size and minimum pseudocodeword weight (AWGN channel) of the graph are 3, i.e., $s_{\text{min}} = w_{\text{min}} = 3$, and the $t$-value is 8.

Figure 5.9 shows the performance of this code on a binary input additive white Gaussian noise channel with min-sum, sum-product, and maximum-likelihood decoding. The min-sum and sum-product iterative decoding was performed for 50 decoding iterations on the LDPC constraint graph. It is evident in the figure that the min-sum and the sum-product decoders are inferior in performance in comparison to the optimal ML decoder. Since the minimal pseudocodeword weight $w_{\text{min}}$ is much less than the minimum distance of the code $d_{\text{min}}$, the performance of the min-sum iterative decoder at high signal to noise ratios (SNRs) is dominated by low-weight bad pseudocodewords.
Figure 5.9. Performance of Example 5.2.2 - LDPC code: Min-Sum, Sum-Product, ML decoding over the BIAWGNC.

**Example 5.2.3** Figure 5.10 shows a graph on \( m + 1 \) variable nodes, where the set of all variable nodes except \( v_1 \) form a minimal stopping set of size \( m \), i.e., \( s_{\text{min}} = m \). When \( m \) is even, the only irreducible pseudocodewords are of the form \((k, 1, 1, \ldots, 1)\), where \( 0 \leq k \leq m \) and \( k \) is even, and the only nonzero codeword is \((0, 1, 1, \ldots, 1)\). When \( m \) is odd, the irreducible pseudocodewords have the form \((k, 1, 1, \ldots, 1)\), where \( 1 \leq k \leq m \), and \( k \) is odd, or \((0, 2, 2, \ldots, 2)\), and the only nonzero codeword is \((1, 1, \ldots, 1)\). In general, any pseudocodeword of this graph is a linear combination of these irreducible pseudocodewords. When \( k \) is not 0 or 1, then these are nc-irreducible pseudocodewords; the weight vector \( \mathbf{w} = (w_1, \ldots, w_{m+1}) \), where \( w_1 = -1, w_2 = +1, \) and \( w_3 = \ldots = w_{m+1} = 0 \), shows that these pseudocodewords are bad. When \( m \) is even or odd, any reducible pseudocodeword of this graph that includes at least one nc-irreducible pseudocodeword in its sum, is also bad (according to Definition 2.5.2). We also observe that for both the BSC and AWGN channel, all of the irreducible pseudocodewords have weight at most \( d_{\text{min}} = m \) or \( m + 1 \), depending on whether \( m \) is even or odd.
The minimum pseudocodeword weight is \( w_{\text{AWGN}}^{\text{min}} = 4m/(m+1) \), and the LDPC constraint graph has a \( t \)-value of \( m \).

Figures 5.11 and 5.12 show the performance of the code for odd and even \( m \), respectively, on a binary input additive white Gaussian noise channel with min-sum, sum-product, and maximum-likelihood decoding. The min-sum and sum-product iterative decoding was performed for 50 decoding iterations on the respective LDPC constraint graphs. The performance difference between the min-sum (respectively, the sum-product) decoder and the optimal ML decoder is more pronounced for odd \( m \). (In the case of even \( m \), \((0, 2, 2, \ldots, 2)\) is not a bad pseudocodeword, since it is twice a codeword, unlike in the case for odd \( m \); thus, one can argue that, relatively, there are a fewer number of bad pseudocodewords when \( m \) is even.) Since the graph has low weight bad pseudocodewords, in comparison to the minimum distance, the performance of the min-sum decoder in the high SNR regime is clearly inferior to that of the ML decoder.

Note that the graph in Example 5.2.1 has no minimal stopping sets with property \( \Theta \), and all stopping sets have size at least \( d_{\text{min}} \). However, all graphs have problematic nodes and conditions 1 and 3 in Theorem 5.1.2 are met in Examples 5.2.2 and 5.2.3. Specifically, the problematic nodes are the variable nodes in the inner ring of Example 5.2.1, \( v_7, v_8 \) in Example 5.2.2, and \( v_1 \) in Example 5.2.3.
Figure 5.11. Performance of Example 5.2.3 - LDPC code for $m = 11$: Min-Sum, Sum-Product, ML decoding over the BIAWGNC.

Figure 5.12. Performance of Example 5.2.3 - LDPC code for $m = 10$: Min-Sum, Sum-Product, ML decoding over the BIAWGNC.
This section has demonstrated three particular LDPC constraint graphs having different types of pseudocodewords, leading to different performances with iterative decoding in comparison to optimal decoding. In particular, we observe that the presence of low weight nc-irreducible pseudocodewords, with weight relatively smaller than the minimum distance of the code, can adversely affect the performance of iterative decoding.

5.3 Pseudocodeword Sets With Respect To Different Channels

In [17], Frey et. al show that the max-product iterative decoder (equivalently, the min-sum iterative decoder) will always converge to an irreducible pseudocodeword (as in Definition 2.4.1) on the AWGN channel. However, their result does not explicitly show that for a given irreducible pseudocodeword $p$, there is a weight vector $w$ such that the cost $pw^T$ is the smallest among all possible pseudocodewords. In the previous subsection, we have given sufficient conditions under which such a weight vector can explicitly be found for certain irreducible pseudocodewords. However, finding such a weight vector $w$ for any irreducible pseudocodeword $p$ may not always be possible.

In particular, we state the following definitions and results.

**Definition 5.3.1** A *truncated AWGN channel*, parameterized by $L$ and denoted by $TAWGN(L)$, is an AWGN channel whose output log-likelihood ratios corresponding to the received values from the channel are truncated, or limited, to the interval $[-L, L]$.

In light of [14, 15], there are fewer problematic lift-realizable pseudocodewords on the BSC than on the truncated AWGN channel or the AWGN channel.
Definition 5.3.2 For an LDPC constraint graph $G$ that defines an LDPC code $\mathcal{C}$, let $P_{\text{AWGN}}^B(G)$ be the set of lift-realizable pseudocodewords of $G$ where for each pseudocodeword $p$ in the set, there exists a weight vector $w$ such that the cost $pw^T$ on the AWGN channel is the smallest among all possible lift-realizable pseudocodewords in $G$.

Let $P_{\text{BSC}}^B(G)$ and $P_{\text{TAWGN(L)}}^B(G)$ be defined analogously for the BSC and the truncated AWGN channel, respectively. Then, we have the following result:

Theorem 5.3.1 For an LDPC constraint graph $G$, and $L \geq 1$, we have

$$P_{\text{BSC}}^B(G) \subseteq P_{\text{TAWGN(L)}}^B(G) \subseteq P_{\text{AWGN}}^B(G).$$

The above result implies that there may be fewer problematic irreducible pseudocodewords for the BSC than over the TAWGN(L) channel and the AWGN channel. In other words, min-sum iterative decoding may be more accurate for the BSC than over the AWGN channel. Thus, quantizing or truncating the received information from the channel to a smaller interval before performing min-sum iterative decoding may be beneficial. Since the set of lift-realizable pseudocodewords for min-sum iterative decoding is the set of pseudocodewords for linear-programming (LP) decoding (see chapter 1), the same analogy carries over to LP decoding as well. Indeed, at high enough signal-to-noise ratios, the above observation has been shown true for the case of LP decoding in [15] and more recently in [14].
5.4 Proofs

Lemma 5.1.1 Proof: Consider a graph $H$ having a single check node which is connected to variable nodes $v_1, \ldots, v_k$. Suppose $b = (b_1, \ldots, b_k)$ is a pseudocodeword in $H$, then $b$ corresponds to a codeword in a lift $\hat{H}$ of $H$. Every check node in $\hat{H}$ is connected to an even number of variable nodes that are assigned value 1, and further, each variable node is connected to exactly one check node in the check cloud. Since the number of variable nodes that are assigned value 1 is equal to the sum of the $b_i$’s, we have $\sum_i b_i \equiv 0 \mod 2$.

Let $\hat{G}$ be the corresponding lift of $G$ wherein $p$ forms a valid codeword. Then each check node in $\hat{G}$ is connected to an even number of variable nodes that are assigned value 1. From the above observation, if nodes $v_{i_1}, \ldots, v_{i_k}$ participate in the check node $c_i$ in $G$, then $p_{i_1} + \cdots + p_{i_k} \equiv 0 \mod 2$. Let $x_i = p_i \mod 2$, for $i = 1, \ldots, n$ (n being the number of variable nodes, i.e., the block length of $C$, in $G$). Then, at every check node $c_i$, we have $x_{i_1} + \cdots x_{i_k} \equiv 0 \mod 2$. Since $x = (x_1, \ldots, x_n) = p \mod 2$ is a binary vector satisfying all checks, it is a codeword in $C$.

Lemma 5.1.2 Proof: Suppose $c \in C$ is in the support of $p$, then form $p' = p - c$. If $p'$ contains a codeword in its support, then repeat the above step on $p'$. Subtracting codewords from the pseudocodeword vector in this manner will lead to a decomposition of the vector $p$ as stated. Observe that the residual vector $r$ contains no codeword in its support. From Lemma 5.1.1, $x = p \mod 2$ is a codeword in $C$. Since $p = c^{(1)} + \cdots c^{(k)} + r$, we have $x = (c^{(1)} + \cdots + c^{(k)}) \mod 2 + r \mod 2$. But since $x \in C$, this implies $r \mod 2 \in C$. However, since $r$ contains no codeword in its support, $r \mod 2$ must be the all-zero codeword. Thus, $r$ contains only even (possibly 0) components.
Theorem 5.1.1  Proof: Let \( p \) be a pseudocodeword of a code \( C \), and suppose \( p \) may be decomposed as \( p = c^{(1)} + c^{(2)} + \ldots + c^{(k)} \), where \( \{c^{(i)}\}_{i=1}^{k} \) is a set of not necessarily distinct codewords. Suppose \( p \) is bad. Then there is a weight vector \( w \) such that \( pw^T < 0 \) but for all codewords \( c \in C, cw^T \geq 0 \). Having \( pw^T < 0 \) implies that \( c^{(1)}w^T + c^{(2)}w^T + \ldots + c^{(k)}w^T = -x \), for some positive real value \( x \). So there is at least one \( i \) for which \( c^{(i)}w^T < 0 \), which is a contradiction. Therefore, \( p \) is a good pseudocodeword.

Theorem 5.1.2  Proof: Let \( M \) be a sufficiently large finite positive integer.

1. If \( w_{t_{BSC/AWG}}(p) < d_{\text{min}} \), then \( p \) is a bad pseudocodeword by Theorem 3.4.1.

2. If \( |\text{supp}(p)| < d_{\text{min}} \), then there is no codeword in the support of \( p \), by Lemma 5.1.1. Let \( w = (w_1, \ldots, w_n) \), be a weight vector where for \( i = 1, 2, \ldots, n \),

\[
    w_i = \begin{cases} 
        -1 & \text{if } v_i \in \text{supp}(p), \\
        M & \text{if } v_i \notin \text{supp}(p)
    \end{cases}
\]

Then \( pw^T < 0 \) and for all codewords \( c \in C, cw^T \geq 0 \).

3. Suppose \( p \) is a non-codeword irreducible pseudocodeword. Without loss of generality, assume \( p = (p_1, p_2, \ldots, p_s, 0, 0, \ldots, 0) \), i.e., the first \( s \) positions of \( p \) are non-zero and the rest are zero. Suppose \( p \) contains \( \ell \) distinct codewords \( c^{(1)}, c^{(2)}, \ldots, c^{(\ell)} \) in its support. Then if \( s \geq \ell + 1 \), we define a weight vector \( w = (w_1, w_2, \ldots, w_n) \) as follows. Let \( w_i = M \) for \( i \notin \text{supp}(p) \). Solve for \( w_1, w_2, \ldots, w_s \) from the following system of linear equations:

\[
    c^{(1)}w^T = +1, \\
    c^{(2)}w^T = +1,
\]
\[
\begin{align*}
\vdots \\
c^{(t)} w^T &= +1, \\
p w^T &= p_1 w_1 + p_2 w_2 + \cdots + p_s w_s = -2,
\end{align*}
\]

The above system of equations involves \( s \) unknowns \( w_1, w_2, \ldots, w_s \) and there are \( \ell + 1 \) equations. Hence, as long as \( s \geq \ell + 1 \), we have a solution for the \( w_i \)'s. Thus, there exists a weight vector \( w = (w_1, \ldots, w_s, M, \ldots, M) \) such that \( pw^T < 0 \) and \( cw^T \geq 0 \) for all codewords \( c \) in the code. This proves that \( p \) is a bad pseudocodeword.

\[\square\]

**Theorem 5.1.3** \hspace{1em} **Proof:** Let \( M \) be a sufficiently large finite positive integer.

1. Let \( S \) be a stopping set. **Suppose there are no non-zero codewords whose support is contained in \( S \).** The pseudocodeword \( p \) with component value 2 in the positions of \( S \), and 0 elsewhere is then a bad pseudocodeword on the AWGN channel, which may be seen by the weight vector \( w_a = (w_1, \ldots, w_n) \), where for \( i = 1, 2, \ldots, n \),

\[
  w_i = \begin{cases} 
  -1 & \text{if } i \in S, \\ 
  M & \text{if } i \notin S
  \end{cases}
\]

In addition, since all nonzero components have the same value, the weight of \( p \) on the BSC and AWGN channel is \( |S| \).

Suppose now that \( p \) is a non-codeword pseudocodeword with support \( S \). Then the weight vector \( w_a \) again shows that \( p \) is bad, i.e., \( pw_a^T < 0 \) and \( c' w_a^T \geq 0 \) for all \( c' \in C \).
2. Suppose there is at least one non-zero codeword $c$ whose support is in $S$.

(a) Assume $S$ is a minimal stopping set. Then this means that $c$ is the only non-zero codeword whose support is in $S$ and $\text{supp}(c) = S$.

(i) Suppose $S$ has property $\Theta$, then we can divide the variable nodes in $S$ into disjoint equivalence classes such that the nodes belonging to each class are connected pairwise by a path traversing only via degree two check nodes in $G_{|s|}$. Consider the pseudocodeword $p$ having component value 3 in the positions corresponding to all nodes of one equivalence class, component value 1 for the remaining positions of $S$, and component value 0 elsewhere. Let $r = p - c$, and let $\hat{i}$ denote the index of the first non-zero component of $r$ and $i^*$ denote the index of the first non-zero component in $\text{supp}(p) - \text{supp}(r)$. The weight vector $w^b_i = (w_1, \ldots, w_n)$, where for $i = 1, 2, \ldots, n$,

$$w_i = \begin{cases} 
-1 & \text{if } i = \hat{i}, \\
+1 & \text{if } i = i^* \\
M & \text{if } i \notin S \\
0 & \text{otherwise}
\end{cases}$$

ensures that $p$ is bad as in Definition 2.5.2, and it is easy to show that the weight of $p$ on the AWGN channel is strictly less than $|S|$.

Conversely, suppose $S$ does not have property $\Theta$. Then every pair of variable nodes in $S$ is connected by a path in $G_{|s|}$ that contains only degree two check nodes. This means that any pseudocodeword
p with support S must have all its components in S of the same value. Therefore, the only pseudocodewords with support S that arise have the form \( p = kc \), for some positive integer \( k \). (By Theorem 5.1.1, these are good pseudocodewords.) Hence, there exists no bad pseudocodewords with support S.

(ii) Let \( p \) be a non-codeword pseudocodeword with support S. If S contains a codeword c in its support, then since S is minimal \( \text{supp}(c) = S \). Let \( k \) denote the number of times c occurs in the decomposition (as in Lemma 5.1.2) of \( p \). That is, \( p = kc + r \). Note that \( r \) is non-zero since \( p \) is a non-codeword pseudocodeword. Let \( \hat{i} \) denote an index of the maximal component of \( r \), and let \( i^* \) denote the index of the first nonzero component in \( \text{supp}(p) - \text{supp}(r) \). The weight vector \( w_b \), defined above, again ensures that \( p \) is bad.

(b) Suppose S is not a minimal stopping set and there is at least one non-zero codeword c whose support is in S.

(i) Suppose S contains a problematic node v. By definition of problematic node, let us assume that S is the only stopping set among all stopping sets in S that contains v. Then define a set \( S_v \) as

\[
S_v := \{ u \in S \mid \text{there is a path from } u \text{ to } v \text{ containing only degree two check nodes in } G|_{[s]} \}
\]

Then, the pseudocodeword \( p \) that has component value 4 on all nodes in \( S_v \), component value 2 on all nodes in \( S - S_v \) and component value 0 everywhere else is a valid pseudocodeword. Let
let $i^*$ be the index of the variable node $v$ in $G$, and let $i'$ be the index of some variable node in $S - S_v$. Then, the weight vector $w_c = (w_1, \ldots, w_n)$, where for $i = 1, 2, \ldots, n$,

$$w_i = \begin{cases} 
-1 & \text{if } i = i^*, \\
+1 & i = i' \\
M & i \notin S \\
0 & \text{otherwise}
\end{cases}$$

ensures that $pw_c^T < 0$ and $c'w_c^T \geq 0$ for all non-zero codewords $c'$ in $C$. Hence, $p$ is a bad pseudocodeword with support $S$. This shows the existence of a bad pseudocodeword on $S$.

Suppose now that $p'$ is some non-codeword irreducible pseudocodeword with support $S$. If there is a non-zero codeword $c$ such that $c$ has support in $S$ and contains $v$ in its support, then since $v$ is a problematic node in $S$, $v$ cannot lie in a smaller stopping set in $S$. This means that support of $c$ is equal to $S$. We will show that $p'$ is a bad pseudocodeword by constructing a suitable weight vector $w_d$. Let $i'$ be the index of some variable node in $S - S_v$. Then we can define a weight vector $w_d = (w_1, \ldots, w_n)$, where for $i = 1, 2, \ldots, n$,

$$w_i = \begin{cases} 
-1 & \text{if } i = i^*, \\
+1 & i = i' \\
M & i \notin S \\
0 & \text{otherwise}
\end{cases}$$

(Note that since $p'$ is a non-codeword irreducible pseudocodeword
and contains $v$ in its support, $p'_i > p'_j$. This weight vector ensures that $p'w_d^T < 0$ and $c'w_d^T \geq 0$ for all non-zero codewords $c' \in \mathcal{C}$. Thus, $p'$ is a bad pseudocodeword.

If there is no non-zero codeword $c$ such that $c$ has support in $S$ and also contains $v$ in its support. Then, the weight vector $w_e = (w_1, \ldots, w_n)$, where for $i = 1, 2, \ldots, n$,

$$w_i = \begin{cases} 
-1 & \text{if } i = i^*, \\
M & \text{if } i \notin S \\
0 & \text{otherwise}
\end{cases}$$

ensures that $p'w_e^T < 0$ and $c'w_e^T \geq 0$ for all non-zero codewords $c' \in \mathcal{C}$. Thus, $p'$ is a bad pseudocodeword.

This proves that any non-codeword irreducible pseudocodeword with support $S$ is bad.

(ii) Suppose $S$ is not a minimal stopping set and suppose $S$ does not contain any problematic nodes. Then, any node in $S$ belongs to a smaller stopping set within $S$. We claim that each node within $S$ belongs to a minimal stopping set within $S$. Otherwise, a node belongs to a non-minimal stopping set $S'_0 \subset S$ and is not contained in any smaller stopping set within $S'_0$ – thereby, implying that the node is a problematic node. Therefore, all nodes in $S$ are contained in minimal stopping sets within $S$.

To prove the last part of the theorem, suppose one of these minimal stopping sets, say $S_j$, is not the support of any non-zero codeword in $\mathcal{C}$. Then, there exists a bad non-codeword pseudocodeword $p = \ldots$
(\(p_1, \ldots, p_n\)) with support \(S\), where \(p_i = x\), for an appropriately chosen positive even integer \(x\) that is at least 4, for \(i \in S_j\) and \(p_i = 2\) for \(i \in S - S_j\), and \(p_i = 0\) for \(i \notin S\). Let \(i^*\) be the index of a variable node \(v^*\) in \(S_j\). If there are distinct codewords \(c^{(1)}, c^{(2)}, \ldots, c^{(t)}\) whose supports contain \(v^*\) and whose supports are contained in \(S\), then let \(i_1, i_2, \ldots, i_t\) be the indices of variable nodes in the supports of these codewords outside of \(S_j\). (Note that we choose the smallest number \(t' \leq t\) of indices such that each codeword contains one of the variable nodes \(v_{i_1}, \ldots, v_{i_t}\) in its support.) The following weight vector \(w = (w_1, \ldots, w_n)\) ensures that \(p\) is a bad pseudocodeword.

\[
    w_i = \begin{cases} 
    -1 & \text{if } i = i^* \\
    +1 & \text{if } i = i_1, i_2, \ldots, i_t \\
    M & \text{if } i \notin S \\
    0 & \text{otherwise}
    \end{cases}
\]

(Note that \(x\) can be chosen so that \(pw^T < 0\) and it is clear that \(cw^T \geq 0\) for all codewords in the code.)

Now suppose \(S_j\) contains the support of a codeword \(c'\) and has property \(\Theta\), then we can construct a bad pseudocodeword on \(S_j\) using the previous argument and that in part 2(a)(i) above (since \(S_j\) is minimal) and allow the remaining components of \(p\) in \(S - S_j\) to have a component value of 2. It is easy to verify that such a pseudocodeword is bad.

Conversely, suppose every minimal stopping set \(S_j\) within \(S\) does
not have property $\Theta$ and contains the support of some non-zero codeword $c^{(j)}$ within it. Then, this means that $S_j = \text{supp}(c^{(j)})$ and that between every pair of nodes within $S_j$ there is a path that contains only degree two check nodes in $G_{\mid S_j}$. Then, for any pseudocodeword $p$ with support $S$, there is a decomposition of $p$, as in Lemma 5.1.2, such that $p$ can be expressed as a linear combination of codewords $c^{(j)}$'s. Therefore, by Theorem 5.1.1, there are no bad pseudocodewords with support $S$.

\[\]

**Theorem 5.3.1**  \textit{Proof:} Let the LDPC code $C$ represented by the LDPC constraint graph $G$ have blocklength $n$. Note that the weight vector $w = (w_1, \ldots, w_n)$ has only $+1$ or $-1$ components on the BSC, whereas it has every component $w_i$ in the interval $[-L, +L]$ on the truncated AWGN channel $TAWGN(L)$, and has every component $w_i$ in the interval $(-\infty, +\infty)$ on the AWGN channel. That is,

\[
P_{BSC}^B(G) = \{p | \exists w \in \{+1, -1\}^n \text{ s.t. } pw^T < 0, cw^T \geq 0, \forall 0 \neq c \in C\}.
\]

\[
P_{TAWGN(L)}^B(G) = \{p | \exists w \in [+L, -L]^n \text{ s.t. } pw^T < 0, cw^T \geq 0, \forall 0 \neq c \in C\}.
\]

\[
P_{AWGN}^B(G) = \{p | \exists w \in [+\infty, -\infty]^n \text{ s.t. } pw^T < 0, cw^T \geq 0, \forall 0 \neq c \in C\}.
\]

It is clear from the above that, for $L \geq 1$, $P_{BSC}^B(G) \subseteq P_{TAWGN(L)}^B(G) \subseteq P_{AWGN}^B(G)$.

\[\]
CHAPTER 6

GRAPH REPRESENTATIONS AND PSEUDOCODEWORD WEIGHT DISTRIBUTIONS

In this chapter, we examine different representations of individual LDPC codes and analyze the weight distribution of lift-realizable pseudocodewords in each representation. Specifically, we address how the representation affects the performance of the min-sum iterative decoder. The classical [7, 4, 3] and the [15, 11, 3] Hamming codes as used as examples.

6.1 Case Study I: The [7, 4, 3] Hamming Code

Figure 6.1 shows three different graph representations of the [7, 4, 3] Hamming code. We will call the representations A, B, and C, and moreover, for convenience, also refer to the graphs in the three respective representations as A, B,

![Graph Representations](image)

Figure 6.1. Three different representations of the [7,4,3] Hamming code.
and $C$. The graph $A$ is based on the systematic parity check matrix representation of the $[7, 4, 3]$ Hamming code and hence, contains three degree one variable nodes, whereas the graph $B$ has no degree one nodes and is more structured (it results in a circulant parity check matrix) and contains 4 redundant check equations compared to $A$, which has none, and $C$, which has one. In particular, $A$ and $C$ are subgraphs of $B$, with the same set of variable nodes. Thus, the set of lift-realizable pseudocodewords of $B$ is contained in the set of lift-realizable pseudocodewords of $A$ and $C$, individually. Hence, $B$ has fewer number of lift-realizable pseudocodewords than $A$ or $C$. In particular, we state the following result:

**Theorem 6.1.1** The number of lift-realizable pseudocodewords in an LDPC graph $G$ can only reduce with the addition of redundant check nodes to $G$.

The proof is obvious since with the introduction of new check nodes in the graph, some previously valid pseudocodewords may not satisfy the new set of inequality constraints imposed by the new check nodes. (Recall that at a check node $c$ having variable node neighbors $v_{i_1}, \ldots, v_{i_k}$, a pseudocodeword $p = (p_1, \ldots, p_n)$, must satisfy the following inequalities $p_{ij} \leq \sum_{h \neq j, h=1, \ldots, k} p_{ih}$, for $j = 1, \ldots, k$ [28].)

However, the set of valid codewords in the graph remains the same, since we are introducing only redundant (or, linearly dependent) check nodes. Thus, a graph with more check nodes can only have fewer number of lift-realizable pseudocodewords and possibly a better pseudocodeword-weight distribution.

If we add all possible redundant check nodes to the graph, which, we note, is an exponential number in the number of linearly independent rows of the parity check matrix of the code, then the resulting graph would have the smallest number of lift-realizable pseudocodewords among all possible representations of the code.
If this graph does not have any bad nc-pseudocodewords (both lift-realizable ones and those arising on the computation tree) then the performance obtained with iterative decoding is the same as the optimal ML performance.

**Remark 6.1.1** Theorem 6.1.1 considers only the set of lift-realizable pseudocodewords of a Tanner graph. On adding redundant check nodes to a Tanner graph, the shape of the computation tree is altered and thus, it is possible that some new pseudocodewords arise in the altered computation tree, which can possibly have an adverse effect on iterative decoding. The [4, 1, 4] repetition code example from chapter 2 illustrates this. Iterative decoding is optimal on the single cycle representation of this code. However, on adding a degree four redundant check node, the iterative decoding performance deteriorates due to the introduction of bad pseudocodewords to the altered computation tree. (See Figure 2.10.) (The set of lift-realizable pseudocodewords however remains the same for the new graph with redundant check nodes as for the original graph.)

Returning to the Hamming code example, graph $B$ can be obtained by adding edges to either $A$ or $C$, and thus, $B$ has more cycles than $A$ or $C$. The distribution of the weights of the irreducible lift-realizable pseudocodewords for the three graphs $A$, $B$, and $C$ is shown\(^1\) in Figure 6.2. (The distribution considers all irreducible pseudocodewords in the graph, since irreducible pseudocodewords may potentially prevent the min-sum decoder to converge to any valid codeword [17].) Although, all three graphs have a pseudocodeword of weight three\(^2\), Fig-

\(^1\)The plots considered all pseudocodewords in the three graphs that had a maximum component value of at most 3. Hence, for each codeword $c$, $2c$ and $3c$ are also counted in the histogram, and each has weight at least $d_{\text{min}}$. However, each irreducible nc-pseudocodeword $p$ is counted only once, as $p$ contains at least one entry greater than 1, and any nonzero multiple of $p$ would have a component greater than 3. The $t$-value (see chapter 7) is 3 for the graphs $A$, $B$, and $C$ of the [7, 4, 3] Hamming code.

\(^2\)Note that this pseudocodeword is a valid codeword in the graph and is thus a good pseudocodeword for iterative decoding.
Figure 6.2. Pseudocodeword-weight (AWGN) distribution of representations A, B, C of the [7,4,3] Hamming code.

Figure 6.2 shows that B has most of its lift-realizable pseudocodewords of high weight, whereas C, and more particularly, A, have more low-weight lift-realizable pseudocodewords. The corresponding weight distributions over the BEC and the BSC channels are shown in Figure 6.3. B has a better weight distribution than A and C over these channels as well.

The performance of min-sum iterative decoding of A, B, and C on the binary input (BPSK modulated) additive white Gaussian noise (AWGN) channel with signal to noise ratio (SNR) $E_b/N_o$ is shown in Figures 6.4, 6.5, and 6.6, respectively. (The maximum number of decoding iterations was fixed at 100.) The performance plots show both the bit error rate and the frame error rate, and further, they also distinguish between undetected decoding errors, that are caused
Figure 6.3. Pseudocodeword-weight distribution (BEC and BSC channels) of representations $A$, $B$, $C$ of the $[7, 4, 3]$ Hamming code.

due to the decoder converging to an incorrect but valid codeword, and detected errors, that are caused due to the decoder failing to converge to any valid codeword within the maximum specified number of decoding iterations, 100 in this case. The detected errors can be attributed to the existence of non-codeword pseudocodewords and the decoder trying to converge to one of them rather than to any valid codeword.

Representation $A$ has a significant detected error rate, whereas representation $B$ shows no presence of detected errors at all. All errors in decoding $B$ were due to the decoder converging to a wrong codeword. (We note that an optimal ML decoder would yield a performance closest to that of the iterative decoder on representation $B$.) This is interesting since the graph $B$ is obtained by adding 4 redundant check nodes to the graph $A$. The addition of these 4 redundant check nodes to the graph removes most of the low-weight nc-pseudocodewords that were present in $A$. (We note here that representation $B$ includes all possible redundant parity-check equations there are for the $[7,4,3]$ Hamming code.) Representation $C$ has fewer number of pseudocodewords compared to $A$. However, the set of
irreducible pseudocodewords of $C$ is not a subset of the set of irreducible pseudocodewords of $A$. The performance of iterative decoding on representation $C$ indicates a small fraction of detected errors.

Figure 6.7 compares the performance of min-sum decoding on the three representations. Clearly, $B$, having the best pseudocodeword weight distribution among the three representations, yields the best performance with min-sum iterative decoding, with performance almost matching that of the optimal ML decoder. (Figure 6.7 shows the bit error rates as solid lines and the frame error rates as dotted lines.)

6.2 Case Study II: The [15, 11, 3] Hamming Code

Similarly, we also analyzed three different representations of the [15, 11, 3] Hamming code. Representation $A$ has its parity check matrix in the standard systematic form and thus, the corresponding Tanner graph has 4 variable nodes of degree one. Representation $B$ includes all possible redundant parity check equations of representation $A$, meaning the parity check rows of representation $B$ include all linear combinations of the parity check rows of representation $A$, and has the best pseudocodeword-weight distribution. Representation $C$ includes up to order-two redundant parity check equations from the parity check matrix of representation $A$, meaning, the parity check matrix of representation $C$ contained all linear combinations of every pair of rows in the parity check matrix of representation $A$. Thus, its (lift-realizable) pseudocodeword-weight distribution is superior to that of $A$ but inferior to that of $B$. (We have not been able to
plot a pseudocodeword-weight distribution for the different representations of the [15, 11, 3] Hamming code since the task of enumerating all irreducible lift-realizable pseudocodewords for these representations proved to be computationally intensive.

The analogous performance of min-sum iterative decoding of representations $A$, $B$, and $C$ of the [15, 11, 3] Hamming code on the binary input (BPSK modulated) additive white Gaussian noise (AWGN) channel with signal to noise ratio (SNR) $E_b/N_0$ is shown in Figures 6.8, 6.9, and 6.10, respectively. (The maximum number of decoding iterations was fixed at 100.) As in the previous example, here also we observe similar trends in the performance curves. $A$ shows a prominent detected error rate, whereas $B$ and $C$ show no presence of detected errors at all. The results indicate that merely adding order two redundant check nodes to the graph of $A$ is sufficient to remove most of the low-weight pseudocodewords.

Figure 6.11 compares the performance of min-sum decoding on the three representations. Here again, representation $B$, having the best pseudocodeword-weight distribution among the three representations, yields the best performance with min-sum iterative decoding and is closest in performance to that of the ML decoder.

6.3 Conclusions

Inferring from the empirical results of this section, we comment that LDPC codes that have structure and redundant check nodes, for example, the class of LDPC codes obtained from finite geometries [29], are likely to have fewer number of low-weight pseudocodewords in comparison to other randomly constructed
LDPC graphs of comparable parameters. Despite the presence of a large number of short cycles (i.e., 4-cycles and 6-cycles), the class of LDPC codes in [29] perform very well with iterative decoding. It is worth investigating how the set of pseudocodewords among existing LDPC constructions can be improved, either by adding redundancy or modifying the Tanner graphs, so that the number of (bad) pseudocodewords, both lift-realizable ones as well as those occurring on the computation tree, is lowered.
Performance of the [7,4,3] Hamming code with min-sum iterative decoding over the BIAWGNC.
Performance of the [7,4,3] Hamming code with min-sum iterative decoding over the BIAWGNC.
Performance of the $[15,11,3]$ Hamming code with min-sum iterative decoding over the BIAWGNC.
Figure 6.10. Representation C.

Figure 6.11. Comparison between representations.

Performance of the [15,11,3] Hamming code with min-sum iterative decoding over the BIAWGNC.
CHAPTER 7

LIFT DEGREES

Since it is computationally infeasible to enumerate all pseudocodewords arising from all covering graphs of a given LDPC constraint graph, one interesting question is the smallest degree cover needed to obtain an irreducible pseudocodeword of minimum weight. If this lift degree is small enough, then calculating the exact minimum pseudocodeword weight is potentially possible, since any reducible pseudocodeword has weight at least the minimum weight of its constituent pseudocodewords. This chapter relates the components of a pseudocodeword with the smallest degree needed to realize it, and gives a few examples of cases where the maximum component an irreducible pseudocodeword can have is known.

7.1 Realizing Pseudocodewords

Recall that any pseudocodeword can be expressed as a sum of irreducible pseudocodewords, and further, that the weight of any pseudocodeword is lower bounded by the smallest weight of its constituent pseudocodewords. Therefore, given a graph $G$, it is useful to find the smallest lift degree needed to realize all irreducible lift-realizable pseudocodewords (and hence, also all minimum weight pseudocodewords).

One parameter of interest is the maximum component $t$ which can occur in any irreducible lift-realizable pseudocodeword of a given graph $G$, i.e., if a pseudocode-
word \( p \) has a component larger than \( t \), then \( p \) is reducible (see Example 3.4.1).
Recall Definition 3.4.1 in Chapter 3:

**Definition 7.1.1** Let \( G \) be a Tanner graph. Then the maximum component value an irreducible pseudocodeword of \( G \) can have is called the \( t \)-value of \( G \), and will be denoted by \( t \).

**Theorem 7.1.1** Let \( G \) be an LDPC constraint graph with largest right degree \( d_r^+ \) and \( t \)-value \( t \). That is, any irreducible lift-realizable pseudocodeword \( p = (p_1, \ldots, p_n) \) of \( G \) has \( 0 \leq p_i \leq t \), for \( i = 1, \ldots, n \). Then the smallest lift degree \( m_{\min} \) needed to realize all irreducible pseudocodewords of \( G \) satisfies

\[
m_{\min} \leq \max_{h_j} \frac{\sum_{v_i \in N(h_j)} p_i}{2} \leq \frac{td_r^+}{2},
\]

where the maximum is over all check nodes \( h_j \) in the graph and \( N(h_j) \) denotes the variable node neighbors of \( h_j \).

If such a \( t \) is known, then Theorem 7.1.1 may be used to obtain the smallest lift degree needed to realize all irreducible lift-realizable pseudocodewords. This has great practical implications, for an upper bound on the lift degree needed to obtain a pseudocodeword of minimum weight would significantly lower the complexity of determining the minimum pseudocodeword weight \( w_{\min} \). Moreover, Theorem 3.4.1 shows how the weight of a pseudocodeword \( p \) is lower bounded in terms of \( t \) and the support size of \( p \).

**Corollary 7.1.2** If \( p \) is any lift-realizable pseudocodeword and \( b \) is the maximum component, then the smallest lift degree needed to realize \( p \) is at most \( \frac{bd_r^+}{2} \).
Example 7.1.1 Some graphs with known $t$-values are: $t \leq 2$ for cycle codes [22, 51], $t = 1$ for LDPC codes whose Tanner graphs are trees, and $t \leq 2$ for LDPC graphs having a single cycle, and $t \leq s$ for tail-biting trellis codes represented on Tanner-Wiberg-Loeliger graphs [51] with state-space sizes $s_1, \ldots, s_m$ and $s = \max\{s_1, \ldots, s_m\}$.

For any finite bipartite graph, the following holds:

**Theorem 7.1.3** Every finite bipartite graph $G$ representing a finite length LDPC code has a finite $t$.

In [51], Wiberg shows that there are infinitely many irreducible pseudocodewords, or deviation sets, on the computation tree when the tree grows with increasing decoding iterations. Since the graph is finite, this means that some components could assume arbitrarily large values. However, in the graph covers setting, the structure of the graph ensures that at some large value, $t$, which may be exponentially large in the code length and the degree distribution of the graph, all the irreducible pseudocodewords are realizable.

**Lemma 7.1.1** Let $S$ be a stopping set in $G$. Let $t_S$ denote the largest component an irreducible pseudocodeword with support $S$ may have in $G$. If $S$ is a minimal stopping set and does not have property $\Theta$, then a pseudocodeword with support $S$ has maximal component 1 or 2. That is, $t_S = 1$ or 2.
7.2 Proofs

**Theorem 7.1.1**  **Proof:** Let $m$ be the minimum degree lift needed to realize the given pseudocodeword $p$. Then, in a degree $m$ lift graph $\hat{G}$ that realizes $p$, the maximum number of active check nodes in any check cloud is at most $m$. A check cloud $c$ is connected to $\sum_{i \in N(c)} p_i$ active variable nodes from the variable clouds adjoining check cloud $c$. (Note that $N(c)$ represents all the variable clouds adjoining $c$.) Since every active check node in any check cloud has at least two (an even number) active variable nodes connected to it, we have that $2m \leq \max_c \sum_{i \in N(c)} p_i$. This quantity can be upper-bounded by $td_r^+$ since $p_i \leq t$, for all $i$, and $|N(c)| \leq d_r^+$, for all $c$.

**Theorem 7.1.3**  **Proof:** In the polytope representation introduced by Koetter and Vontobel in [28], irreducible pseudocodewords correspond to the edges of the fundamental cone, which can be described by $\{x \mid x_i \geq 0, i = 1, \ldots, n\}$ and $\{x \mid x_i \leq \sum_{j \in N(i) \setminus \{i\}} x_{j'}\}$ for all check nodes $j$ and all variable nodes $i$ in $N(j)$. Consider the polytope that is the intersection of the fundamental cone with the hyperplane $x_1 + \cdots + x_n = 1$. The vertices of this polytope have a one-to-one correspondence with the edges of the fundamental cone. Let $v$ be a fixed vertex of this new polytope. Then $v$ satisfies at least $n - 1$ of the above inequalities with equality. Together with the hyperplane equality, $v$ meets at least $n$ inequalities with equality. The resulting system of linear equations contains only integers as coefficients. By Cramer's rule, the vertex $v$ must have rational coordinates. Taking the least common multiple of all denominators of all coordinates of all vertices gives an upper bound on $t$. Therefore, $t$ is finite.
Lemma 7.1.1 Proof: Suppose $S$ is minimal and does not have property $\Theta$. Then each pair of nodes in $S$ are connected by a path via degree two check nodes, and hence all components of a pseudocodeword $p$ on $S$ are equal. If $S$ is the support of a codeword $c$, then any pseudocodeword is a multiple of $c$, and so $t_S = 1$. If $S$ is not the support of a codeword, then the all-two’s vector is the only irreducible pseudocodeword, and $t_S = 2$. 

\[ \blacksquare \]
CHAPTER 8

TREE-BASED CONSTRUCTION OF LDPC CODES

In this chapter, we present a combinatorial construction of LDPC codes that have relatively few low-weight pseudocodewords and perform well with iterative decoding. The construction involves enumerating a $d$-regular tree for a fixed number of layers and employing a connection algorithm based on permutations or mutually orthogonal Latin squares to close the tree. Methods are presented for degrees $d = p^s$ and $d = p^s + 1$, for $p$ a prime. One class corresponds to the well-known finite-geometry and finite generalized quadrangle LDPC codes; the other codes presented are new. Treating these codes as $p$-ary LDPC codes rather than binary LDPC codes improves their rates, minimum distances, and pseudocodeword weights.

The Type I-A construction and certain cases of the Type II construction presented in this chapter are designed so that the resulting codes have minimum pseudocodeword weight equal to or almost equal to the minimum distance of the code, and consequently, these problematic low-weight pseudocodewords are avoided. Some of the resulting codes have minimum distance which meets the lower tree bound originally presented in [48]. Since $w_{\text{min}}$ shares the same lower bound [25, 27], and is upper bounded by $d_{\text{min}}$, these constructions have $w_{\text{min}} = d_{\text{min}}$. It is worth noting that this property is also a characteristic of some of the FG-LDPC codes [45], and indeed, the projective-geometry-based codes of [29] arise as special cases...
of our Type II construction. Furthermore, the Type I-B construction presented herein yields a family of codes with a wide range of rates and blocklengths that are comparable to those obtained from finite geometries. This new family of codes has $w_{\text{min}} = d_{\text{min}} \geq$ tree bound in most cases.

**Definition 8.0.1** The *tree bound* of a $d$ left (variable node) regular bipartite LDPC constraint graph with girth $g$ is defined as

$$T(d, g) := \begin{cases} 
1 + d + d(d - 1) + d(d - 1)^2 + \ldots + d(d - 1)^{\frac{g-3}{2}}, & \frac{g}{2} \text{ odd,} \\
1 + d + d(d - 1) + \ldots + d(d - 1)^{\frac{g-4}{2}} + (d - 1)^{\frac{g-4}{2}}, & \frac{g}{2} \text{ even.}
\end{cases}$$

(8.1)

We restate the tree-bound of Theorem 3.2.1 from chapter 3 here:

**Theorem 8.0.1** Let $G$ be a bipartite LDPC constraint graph with smallest left (variable node) degree $d$ and girth $g$. Then the minimum pseudocodeword weight $w_{\text{min}}$ (for the AWGN/BSC channels) is lower bounded by

$$w_{\text{min}} \geq T(d, g).$$

8.1 Preliminaries

8.1.1 Permutations

A permutation on set of integers modulo $m$, $\{0, 1, \ldots, m - 1\}$ is a bijective map of the form

$$\pi : \{0, 1, \ldots, m - 1\} \to \{0, 1, \ldots, m - 1\}$$
A permutation is commonly denoted either as

\[
\begin{pmatrix}
0 & 1 & 2 & \ldots & m-1 \\
\pi(0) & \pi(1) & \pi(2) & \ldots & \pi(m-1)
\end{pmatrix}
\]

or as \((a_{11}a_{12} \ldots a_{1s_1})(a_{21}a_{22} \ldots a_{2s_2})\ldots\), where \(a_{i2} = \pi(a_{i1}), a_{i3} = \pi(a_{i2}), \ldots, a_{is_i} = \pi(a_{i,s_i-1}), a_{i1} = \pi(a_{i,s_i})\) for all \(i\).

**Example 8.1.1** Suppose \(\pi\) is a permutation on the set \(\{0, 1, 2, 3\}\) given by \(\pi(0) = 0, \pi(1) = 2, \pi(2) = 3, \pi(3) = 1\). Then \(\pi\) is denoted as

\[
\begin{pmatrix}
0 & 1 & 2 & 3 \\
0 & 2 & 3 & 1
\end{pmatrix}
\]

in the former representation, and as \((0)(123)\) in the latter representation.

**8.1.2 Mutually orthogonal Latin squares (MOLS)**

Let \(\mathbb{F} := GF(q)\) be a finite field of order \(q\) and let \(\mathbb{F}^*\) denote the corresponding multiplicative group – i.e., \(\mathbb{F}^* := \mathbb{F}\setminus\{0\}\). For every \(a \in \mathbb{F}^*\), we define a \(q \times q\) array having entries in \(\mathbb{F}\) by the following linear map

\[\phi_a : \mathbb{F} \times \mathbb{F} \to \mathbb{F}\]

\[(x, y) \mapsto x + a \cdot y\]

where ‘+’ and ‘\cdot’ are the corresponding field operations. The above set of maps define \(q - 1\) mutually orthogonal Latin squares (MOLS) [pg. 182 – 199, [49]]. The map \(\phi_a\) can be written as a matrix \(M_a\) where the rows and columns of the matrix
are indexed by the elements of $\mathbb{F}$ and the $(x, y)^{th}$ entry of the matrix is $\phi_a(x, y)$.

By introducing another map $\phi_0$ defined in the following manner

$$\phi_0 : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$$

$$(x, y) \mapsto x$$

we obtain an additional array $M_0$ which is orthogonal to the above family of $q-1$ MOLS. However, note that $M_0$ is not a Latin square. We use this set of $q$ arrays in the subsequent tree-based constructions.

**Example 8.1.2** let $\mathbb{F} = \{0, 1, \alpha, \alpha^2\}$ be the finite field with four elements, where $\alpha$ represents the primitive element. Then, from the above set of maps we obtain the following four orthogonal squares

$$M_0 := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ \alpha & \alpha & \alpha & \alpha \\ \alpha^2 & \alpha^2 & \alpha^2 & \alpha^2 \end{bmatrix}, \quad M_1 := \begin{bmatrix} 0 & 1 & \alpha & \alpha^2 \\ 1 & 0 & \alpha^2 & \alpha \\ \alpha & \alpha^2 & 0 & 1 \\ \alpha^2 & \alpha & 1 & 0 \end{bmatrix},$$

$$M_2 := \begin{bmatrix} 0 & \alpha & \alpha^2 & 1 \\ 1 & \alpha^2 & \alpha & 0 \\ \alpha & 0 & 1 & \alpha^2 \\ \alpha^2 & 1 & 0 & \alpha \end{bmatrix}, \quad M_{a,2} := \begin{bmatrix} 0 & \alpha^2 & 1 & \alpha \\ 1 & \alpha & 0 & \alpha^2 \\ \alpha & 1 & \alpha^2 & 0 \\ \alpha^2 & 0 & \alpha & 1 \end{bmatrix}.$$

8.2 Tree-based Construction: Type 1

The Type-I codes have graphs that are obtained by connecting two copies of a tree using permutations in the Type-IA case, and mutually orthogonal Latin squares in the Type-IB case.
8.2.1 Type 1-A

For \( d = 3 \), the Type I-A construction yields a \( d \)-regular LDPC constraint graph having \( 1 + d + d(d-1) + \ldots + d(d-1)^{\ell-2} \) variable and constraint nodes, and girth \( g \). The tree \( T \) has \( \ell \) layers. To connect the nodes in \( L_{\ell-1} \) to \( L'_{\ell-1} \), first label the variable (respectively, constraint) nodes in \( L_{\ell-1} \) (respectively, \( L'_{\ell-1} \)) when \( \ell \) is odd (and vice versa when \( \ell \) is even), as \( v_0, v_1, \ldots, v_{2^{\ell-2}-1}, v_{2^{\ell-2}}, \ldots, v_{2^{\ell-2}-1}, v_{2^{\ell-2}}, \ldots, v_{3.2^{\ell-2}-1} \) (resp., \( c_0, c_1, \ldots, c_{3.2^{\ell-2}-1} \)). The nodes \( v_0, v_1, \ldots, v_{2^{\ell-2}-1} \) form the 0th class \( S_0 \), the nodes \( v_{2^{\ell-2}}, \ldots, v_{2^{\ell-2}-1} \) form the 1st class \( S_1 \), and the nodes \( v_{2^{\ell-2}}, \ldots, v_{3.2^{\ell-2}-1} \) form the 2nd class \( S_2 \); classify the constraint nodes into \( S'_0, S'_1 \), and \( S'_2 \) in a similar manner. In addition, define four permutations \( \pi(\cdot), \tau(\cdot), \tau'(\cdot), \tau''(\cdot) \) of the set \( \{0, 1, \ldots, 2^{\ell-2} - 1\} \) and connect the nodes in \( L_{\ell-1} \) to \( L'_{\ell-1} \) as follows: For \( j = 0, 1, \ldots, 2^{\ell-2} - 1, \)

1. The variable node \( v_j \) is connected to nodes \( c_{\pi(j)} \) and \( c_{\tau(j)+2^{\ell-2}} \).

2. The variable node \( v_{j+2^{\ell-2}} \) is connected to nodes \( c_{\pi(j)+2^{\ell-2}} \) and \( c_{\tau'(j)+2^{\ell-2}} \).

3. The variable node \( v_{j+2.2^{\ell-2}} \) is connected to nodes \( c_{\pi(j)+2.2^{\ell-2}} \) and \( c_{\tau''(j)} \).

The permutations for the cases \( g = 6, 8, 10, 12 \) are given in Table 8.1. For \( \ell = 3, 4, 5, 6 \), these permutations yield girths \( g = 6, 8, 10, 12 \), respectively, i.e., \( g = 2\ell \). It is clear that the girth of these graphs is upper bounded by \( 2\ell \). What is interesting is that there exist permutations \( \pi, \tau, \tau', \tau'' \) that achieve this upper bound when \( \ell \leq 6 \). However, when extending this particular construction to \( \ell = 7 \) layers, there are no permutations \( \pi, \tau, \tau', \tau'' \) that yield a girth \( g = 14 \) graph. (This was verified by an exhaustive computer search and computing the girths of the resulting graphs using Magma [1].) The above algorithm to connect the nodes in layers \( L_{\ell-1} \) and \( L'_{\ell-1} \) is rather restrictive, and we need to examine other
TABLE 8.1
PERMUTATIONS FOR TYPE I-A CONSTRUCTION.

<table>
<thead>
<tr>
<th>Girth</th>
<th>( g = 6 )</th>
<th>( g = 8 )</th>
<th>( g = 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi )</td>
<td>(0)(1)</td>
<td>(0)(2)(1,3)</td>
<td>(0)(2)(4)(6)(1,5)(3,7)</td>
</tr>
<tr>
<td>( \tau )</td>
<td>= ( \pi )</td>
<td>= ( \pi )</td>
<td>= ( \pi )</td>
</tr>
<tr>
<td>( \tau' )</td>
<td>= ( \pi )</td>
<td>= ( \pi )</td>
<td>= ( \tau )</td>
</tr>
<tr>
<td>( \tau'' )</td>
<td>= ( \pi )</td>
<td>(0,2)(1)(3)</td>
<td>(0,4)(2,6)(1,3)(5,7)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Girth</th>
<th>( g = 12 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi )</td>
<td>(0)(4)(8)(12)(2,6)(10,14)(1,9)(3,15)(5,13)(7,11)</td>
</tr>
<tr>
<td>( \tau )</td>
<td>(0)(4)(8)(12)(2,6)(10,14)(1,13)(3,11)(5,9)(7,15)</td>
</tr>
<tr>
<td>( \tau' )</td>
<td>(0,8)(4,12)(2,14)(6,10)(1,5)(3)(7)(9,13)(11)(15)</td>
</tr>
<tr>
<td>( \tau'' )</td>
<td>(0,2,4,6)(8,10,12,14)(1,15,5,11)(3,9,7,13)</td>
</tr>
</tbody>
</table>

connection algorithms that may possibly yield a girth 14 bipartite graph. However, the smallest known 3-regular graph with girth 14 has 384 vertices [7]. For \( \ell = 7 \), the graph of the Type I-A construction has a total of 380 nodes (i.e., 190 variable nodes and 190 constraint nodes), and there are permutations \( \pi, \tau, \tau', \) and \( \tau'' \), that only result in a girth 12 (bipartite) graph.

When \( \ell = 3, 5 \), the minimum distance of the resulting code meets the tree bound, and hence, \( d_{\text{min}} = w_{\text{min}} \). When \( \ell = 4, 6 \), the minimum distance \( d_{\text{min}} \) is strictly larger than the tree bound; in fact, \( d_{\text{min}} \) is more than the tree-bound by 2. However, \( w_{\text{min}} = d_{\text{min}} \) for \( \ell = 4, 6 \) as well. Figure 8.1 illustrates the general construction procedure and Figure 8.2 shows a 3-regular girth 10 Type I-A LDPC constraint graph.
Remark 8.2.1 The Type I-A LDPC codes have $d_{\min} = w_{\min} = T(d, 2\ell)$, for \( \ell = 3, 5 \), and $d_{\min} = w_{\min} = 2 + T(d, 2\ell)$, for \( \ell = 4, 6 \).

8.2.2 Type I-B

For $d = p^s$, a prime power, the Type I-B construction yields a $d$-regular LDPC constraint graph having $1 + d + d(d - 1)$ variable and constraint nodes, and girth at least 6. The tree $T$ has 3 layers $L_0, L_1,$ and $L_2$. The tree is reflected to yield another tree $T'$ and the variable and constraint nodes in $T'$ are interchanged. Let $\alpha$ be a primitive element in the field $GF(p^s)$. (Note that $GF(p^s)$ is the set \{0, 1, $\alpha$, $\alpha^2$, ..., $\alpha^{p^s-2}$\}.) The layer $L_1$ (respectively, $L'_1$) contains $p^s$ constraint nodes labeled $(0)_c$, $(1)_c$, $(\alpha)_c$, ..., $(\alpha^{p^s-2})_c$ (respectively, variable nodes labeled $(0)'_c$, $(1)'_c$, $(\alpha)'_c$, ..., $(\alpha^{p^s-2})'_c$). The layer $L_2$ (respectively, $L'_2$) is composed of $p^s$ sets $\{S_i\}_{i=0,1,\alpha,\alpha^2,...,\alpha^{p^s-2}}$ of $p^s-1$ variable (respectively, constraint) nodes in each set. Note that we index the sets by an element of the field $GF(p^s)$. Each set $S_i$ corresponds to the children of one of the branches of the root node. (The "c" in the labeling refers to nodes in the tree $T'$ and the subscript 'c' refers to constraint nodes.) Let $S_i$ (respectively, $S'_i$) contain the variable nodes $(i, 1), (i, \alpha), \ldots, (i, \alpha^{p^s-2})$ (respectively, constraint nodes $(i, 1)'_c, (i, \alpha)'_c, \ldots, (i, \alpha^{p^s-2})'_c$). To use MOLS of order $p^s$ in the connection algorithm, an imaginary node, variable node $(i, 0)$ (respectively, constraint node $(i, 0)'_c$) is temporarily introduced into each set $S_i$ (resp, $S'_i$).
Figure 8.1. Tree construction of Type I-A LDPC code.

Figure 8.2. Type I-A LDPC constraint graph having degree $d = 3$ and girth $g = 10$. 
The connection algorithm proceeds as follows:

1. For $i = 0, 1, \alpha, \ldots, \alpha^{p^s-2}$ and $j = 0, 1, \alpha, \ldots, \alpha^{p^s-2}$, connect the variable node $(i, j)$ in layer $L_2$ to the constraint nodes

\[(0, j + i \cdot 0)'_c, (1, j + i \cdot 1)'_c, \ldots, (\alpha^{p^s-2}, j + i \cdot \alpha^{p^s-2})'_c\]

in layer $L'_2$. (Observe that in these connections, every variable node in the set $S_i$ is mapped to exactly one constraint node in each set $S'_k$, for $k = 0, 1, \alpha, \ldots, \alpha^{p^s-2}$, using the array $M_i$ defined in section 8.1.2.)

2. Delete all imaginary nodes $\{(i, 0), (i, 0)'_c\}_{i=0,1,\ldots,\alpha^{p^s-2}}$ and the edges incident on them.

3. For $i = 1, \ldots, \alpha^{p^s-2}$, delete the edge connecting variable node $(0, i)$ to constraint node $(0, i)'_c$.

The resulting $d$-regular constraint graph represents the Type I-B LDPC code.

**Example 8.2.1** Figure 8.3 illustrates the construction procedure and Figure 8.4 provides a specific example of a Type I-B LDPC constraint graph with $d = 4 = 2^2$ and which uses the squares from Example 8.1.2.

The Type I-B algorithm yields LDPC codes having a wide range of rates and blocklengths that are comparable to, but different from, the two-dimensional LDPC codes from finite Euclidean geometries [29, 45]. The Type I-B LDPC codes are $p^s$-regular with girth at least six, blocklength $N = p^{2s} + 1$, and distance $d_{\text{min}} \geq p^s + 1$. For degrees of the form $d = 2^s$, the resulting binary Type I-B LDPC codes have very good rates, above 0.5, and perform well with iterative decoding.
Figure 8.3. Tree construction of Type I-B LDPC code. (Shaded nodes are imaginary nodes and dotted lines are imaginary lines.)

Figure 8.4. Type I-B LDPC constraint graph having degree $d = 4$ and girth $g = 6$. 
Theorem 8.2.1 The Type I-B LDPC constraint graphs have a girth of at least six.

Theorem 8.2.2 The Type I-B LDPC constraint graphs with degree \( d = p^s \) and girth \( g \geq 6 \) have
\[
2(p^s - 1) \geq d_{\text{min}} \geq w_{\text{min}} \geq T(p^s, 6) = 1 + p^s \text{ for } p > 2 \text{ and,}
\]
\[
2(p^s) + 1 \geq d_{\text{min}} \geq w_{\text{min}} \geq T(p^s, 6) = 1 + p^s \text{ for } p = 2.
\]

When \( p > 2 \), the upper bound \( 2(p^s - 1) \) on minimum distance \( d_{\text{min}} \) (and possibly also \( w_{\text{min}} \)) was met among all the cases of the Type I-B construction we examined. We conjecture that in fact \( d_{\text{min}} = 2(p^s - 1) \) for the Type I-B LDPC codes of degree \( d = p^s \) when \( p > 2 \). Since \( w_{\text{min}} \) is lower bounded by \( 1 + p^s \), we have that \( w_{\text{min}} \) is close, if not equal, to \( d_{\text{min}} \).

8.3 Tree-based Construction: Type II

In the Type II construction, first a \( d \)-regular tree \( T \) of alternating variable and constraint node layers is enumerated from a root variable node (layer \( L_0 \)) for \( \ell \) layers \( L_0, L_1, \ldots, L_{\ell-1} \), as in Type I. The tree \( T \) is not reflected; rather, a single layer of \( (d-1)^{\ell-1} \) nodes is added to form layer \( L_\ell \). If \( \ell \) is odd (respectively, even), this layer is composed of constraint (respectively, variable) nodes. The union of \( T \) and \( L_\ell \), along with edges connecting the nodes in layers \( L_{\ell-1} \) and \( L_\ell \) according to a connection algorithm that is described next, comprise the graph representing a Type II LDPC code. We present the connection scheme that is used for this Type II model, and discuss the resulting codes. First, we state this rather simple observation without proof:

Remark 8.3.1 The girth \( g \) of a Type II LDPC graph for \( \ell \) layers is at most \( 2\ell \).
The connection algorithm for \( \ell = 3 \) and \( \ell = 4 \), wherein this upper bound on girth is in fact achieved, is as follows.

8.3.1 Type II, \( \ell = 3 \)

For \( d = p^s + 1 \), where \( p \) is prime and \( s \) a positive integer, a \( d \)-regular tree is enumerated from a root (variable) node for \( \ell = 3 \) layers \( L_0, L_1, L_2 \). Let \( \alpha \) be a primitive element in the field \( GF(p^s) \). The \( d \) constraint nodes in \( L_1 \) are labeled \((x)_c, (0)_c, (1)_c, (\alpha)_c, \ldots, (\alpha^{p^s-2})_c\) to represent the \( d \) branches stemming from the root node. Note that the first constraint node is denoted as \((x)_c\) and the remaining constraint nodes are indexed by the elements of the field \( GF(p^s) \).

The \( d(d - 1) \) variable nodes in the third layer \( L_2 \) are labeled as follows: the variable nodes descending from constraint node \((x)_c\) form the class \( S_x \) and are labeled \((x, 0), (x, 1), \ldots, (x, \alpha^{p^s-2})\), and the variable nodes descending from constraint node \((i)_c\), for \( i = 0, 1, \alpha, \ldots, \alpha^{p^s-2} \), form the class \( S_i \) and are labeled \((i, 0), (i, 1), \ldots, (i, \alpha^{p^s-2})\).

A final layer \( L_\ell = L_3 \) of \( (d - 1)^{\ell - 1} = p^{2s} \) constraint nodes is added. The \( p^{2s} \) constraint nodes in \( L_3 \) are labeled \((0, 0)_c, (0, 1)_c, \ldots, (0, \alpha^{p^s-2})'_c, (1, 0)_c, (1, 1)_c, \ldots, (1, \alpha^{p^s-2})'_c, \ldots, (\alpha^{p^s-2}, 0)_c, (\alpha^{p^s-2}, 1)_c, \ldots, (\alpha^{p^s-2}, \alpha^{p^s-2})'_c\). (Note that the ‘\( '' \) in the labeling refers to nodes in that are not in the tree \( T \) and the subscript ‘\( c \)’ refers to constraint nodes.)

1. By this labeling, the constraint nodes in \( L_3 \) are grouped into \( d - 1 = p^s \) classes of \( d - 1 = p^s \) nodes in each class. Similarly, the variable nodes in \( L_2 \) are grouped into \( d = p^s + 1 \) classes of \( d - 1 = p^s \) nodes in each class. (That is, the \( i^{th} \) class of constraint nodes is \( S_i' = \{(i, 0)'_c, (i, 1)'_c, \ldots, (i, \alpha^{p^s-2})'_c\}.\)
2. The variable nodes descending from constraint node \((x)_c\) are connected to the constraint nodes in \(L_3\) as follows. Connect the variable node \((x, i)\), for \(i = 0, 1, \ldots, \alpha^{p^s-2}\), to the constraint nodes

\[(i, 0)'_c, (i, 1)'_c, \ldots, (i, \alpha^{p^s-2})'_c.

3. The remaining variable nodes in layer \(L_2\) are connected to the nodes in \(L_3\) as follows: Connect the variable node \((i, j)\), for \(i = 0, 1, \ldots, \alpha^{p^s-2}\), \(j = 0, 1, \ldots, \alpha^{p^s-2}\), to the constraint nodes

\[(0, j + i \cdot 0)'_c, (1, j + i \cdot 1)'_c, (\alpha, j + i \cdot \alpha)'_c, \ldots, (\alpha^{p^s-2}, j + i \cdot \alpha^{p^s-2})'_c.

Observe that in these connections, each variable node \((i, j)\) is connected to exactly one constraint node within each class, using the array \(M_i\) defined in section 8.1.2.

**Example 8.3.1** Figure 8.5 illustrates the construction procedure and Figure 8.6 provides an example of a Type II LDPC constraint graph with degree \(d = 4 = 3+1\) and girth \(g = 6\); the arrays used for constructing this example are\(^1\)

\[
M_0 = \begin{bmatrix}
0 & 0 & 0 \\
1 & 1 & 1 \\
2 & 2 & 2
\end{bmatrix},
M_1 = \begin{bmatrix}
0 & 1 & 2 \\
1 & 2 & 0 \\
2 & 0 & 1
\end{bmatrix},
M_2 = \begin{bmatrix}
0 & 2 & 1 \\
1 & 0 & 2 \\
2 & 1 & 0
\end{bmatrix}.
\]

The ratio of minimum distance to blocklength of the resulting codes is at least \(\frac{2+p^s}{1+p^s+p^{2s}}\), and the girth is six. For degrees \(d\) of the form \(d = 2^s + 1\), the tree bound of Theorem 8.0.1 on minimum distance and minimum pseudocodeword weight

---

\(^1\)Note that in this example, \(GF(3) = \{0, 1, 2\}\), ‘2’ being the primitive element of the field.
[27, 48] is met, i.e., $d_{\text{min}} = w_{\text{min}} = 2 + 2^s$, for the Type II, $\ell = 3$, LDPC codes. For $p > 2$, the resulting binary LDPC codes are repetition codes of the form $[n, 1, n]$, i.e., $d_{\text{min}} = n = 1 + p^s + p^{2s}$ and the rate is $\frac{1}{n}$. However, if we interpret the Type II $\ell = 3$ graphs, that have degree $d = p^s + 1$, as the LDPC constraint graph of a $p$-ary LDPC code, then the rates of the resulting codes are very good and the minimum distances come close to (but are not equal to) the tree bound in Lemma 4.1.1. (See also [41].) We also believe that the minimum pseudocodeword weights (on the $p$-ary symmetric channel) are much closer to the minimum distances for these $p$-ary LDPC codes. This is discussed in the next chapter.

8.3.2 Relation to finite geometry codes

The codes that result from this $\ell = 3$ construction correspond to the two-dimensional projective-geometry-based LDPC (PG LDPC) codes of [45]. We state the equivalence of the tree construction and the finite projective geometry based LDPC codes in the following.

**Theorem 8.3.1** The LDPC constraint graph obtained from the Type II $\ell = 3$ tree construction for degree $d = p^s + 1$ is equivalent to the incidence graph of the finite projective plane over the field $GF(p^s)$.

It has been proved by Bose [8] that a finite projective plane (in other words, a two dimensional finite projective geometry) of order $m$ exists if and only if a complete family of orthogonal $m \times m$ Latin squares exists. The proof of this result, as presented in [39], gives a constructive algorithm to design a finite projective plane of order $m$ from a complete family of $m \times m$ mutually orthogonal Latin squares (MOLS). It is well known that a complete family of mutually orthogonal
Figure 8.5. Tree construction of girth 6 Type II ($\ell = 3$) LDPC code.

Figure 8.6. Type II LDPC constraint graph having degree $d = 4$ and girth $g = 6$. 

121
Latin squares exist when $m = p^s$, a power of a prime, and we have described one such family in section 8.1.2. Hence, the constructive algorithm in [39] generates the incidence graph of the projective plane $PG(2, p^s)$ from the set of $p^s - 1$ MOLS of order $p^s$. The only remaining step is to verify that the incidence matrix of points over lines of this projective plane is the same as the parity check matrix of variable nodes over constraint nodes of the tree-based LDPC constraint graph of the tree construction. This step is easy to verify as the constructive algorithm in [39] is analogous to the tree construction presented in this chapter.

The Type II $\ell = 3$ graphs therefore correspond to the two-dimensional projective-geometry-based LDPC codes of [29]. With a little modification of the Type II construction, we can also obtain the two-dimensional Euclidean-geometry-based LDPC codes of [29, 45]. Since a two-dimensional Euclidean geometry may be obtained by deleting certain points and line(s) of a two-dimensional projective geometry, the graph of a two-dimensional EG-LDPC code [45] may be obtained by performing the following operations on the Type II, $\ell = 3$, graph:

1. In the tree $T$, the root node along with its neighbors, i.e., the constraint nodes in layer $L_1$, are deleted.

2. Consequently, the edges from the constraint nodes $(x)_c, (0)_c, (1)_c, \ldots, (\alpha^{p^s - 2})_c$ to layer $L_2$ are also deleted.

3. At this stage, the remaining variable nodes have degree $p^s$, and the remaining constraint nodes have degree $p^s + 1$. Now, a constraint node from layer $L_3$ is chosen, say, constraint node $(0, 0)_c$. This node and its neighboring variable nodes and the edges incident on them are deleted. Doing so removes exactly one variable node from each class of $L_2$, and the degrees of the remaining constraint nodes in $L_3$ are lessened by one. Thus, the resulting graph is now
$p^s$-regular with a girth of six, has $p^{2s} - 1$ constraint and variable nodes, and corresponds to the two-dimensional Euclidean-geometry-based LDPC code $EG(2, p^s)$ of [45].

**Theorem 8.3.2** The Type II $\ell = 3$ LDPC constraint graphs have girth $g = 6$ and diameter $\delta = 3$.

**Theorem 8.3.3** For degrees $d = 2^s + 1$, the resulting Type II $\ell = 3$ LDPC constraint graphs have $d_{\min} = w_{\min} = T(d, 6) = 2 + 2^s$.

### 8.3.3 Type II, $\ell = 4$

For $d = p^s + 1$, $p$ a prime and $s$ a positive integer, a $d$ regular tree $T$ is enumerated from a root (variable) node for $\ell = 4$ layers $L_0, L_1, L_2, L_3$.

1. The nodes in $L_0, L_1$, and $L_2$ labeled as in the $\ell = 3$ case. The constraint nodes in $L_3$ are labeled as follows: The constraint nodes descending from variable node $(x, j)$, for $j = 0, 1, \alpha, \ldots, \alpha^{p^s-2}$, are labeled $(x, j, 0)_c, (x, j, 1)_c, \ldots, (x, j, \alpha^{p^s-2})_c$, the constraint nodes descending from variable node $(i, j)$, for $i, j = 0, 1, \alpha, \ldots, \alpha^{p^s-2}$, are labeled $(i, j, 0)_c, (i, j, 1)_c, \ldots, (i, j, \alpha^{p^s-2})_c$.

2. A final layer $L_\ell = L_4$ of $(d - 1)^{\ell - 1} = p^{3s}$ variable nodes is introduced. The $p^{3s}$ variable nodes in $L_4$ are labeled as $(0, 0, 0)', (0, 0, 1)', \ldots, (0, 0, \alpha^{p^s-2})', (0, 1, 0)', (0, 1, 1)', \ldots, (0, 1, \alpha^{p^s-2})', \ldots, (\alpha^{p^s-2}, 0, 0)', (\alpha^{p^s-2}, 0, 1)', \ldots, (\alpha^{p^s-2}, 0, \alpha^{p^s-2})', \ldots, (\alpha^{p^s-2}, \alpha^{p^s-2}, 0)', (\alpha^{p^s-2}, \alpha^{p^s-2}, 1)'$, $\ldots, (\alpha^{p^s-2}, \alpha^{p^s-2}, \alpha^{p^s-2})'$. (Note that the ‘$\prime$’ in the labeling refers to nodes that are not in the tree $T$ and the subscript ‘$c$’ refers to constraint nodes.)
3. For $0 \leq i \leq p^s - 1$, $0 \leq j \leq p^s - 1$, connect the constraint node $(x, i, j)_c$ to the variable nodes

$((i, j, 0)^t, (i, j, 1)^t, (i, j, \alpha)^t, \ldots, (i, j, \alpha^{p^s-2})^t)$.

4. To connect the remaining constraint nodes in $L_3$ to the variable nodes in $L_4$, we first define a function $f$. For $i, j, k, t = 0, 1, \ldots, \alpha^{p^s-2}$ let

$$f : \mathbb{F} \times \mathbb{F} \times \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$$

$$(i, j, k, t) \mapsto y,$$

be an appropriately chosen function, that we will define later for some specific cases of the Type II $\ell = 4$ construction. Then, for $i, j, k = 0, 1, \ldots, \alpha^{p^s-2}$, connect the constraint node $(i, j, k)_c$ in $L_3$ to the following variable nodes in $L_4$

$$(0, k + i \cdot 0, f(i, j, k, 0))^t, (1, k + i \cdot 1, f(i, j, k, 1))^t, (\alpha, k + i \cdot \alpha, f(i, j, k, \alpha))^t, \ldots, (\alpha^{p^s-2}, k + i \cdot \alpha^{p^s-2}, f(i, j, k, \alpha^{p^s-2}))^t.$$ 

(Observe that the second index corresponds to the linear map defined by the array $M_i$ defined in section 8.1.2. Further, note that if $f(i, j, k, t) = j + i \cdot t$, then the resulting graphs obtained from the above set of connections have girth at least six. However, there are other functions $f(i, j, k, t)$ for which the resulting graphs have girth exactly eight, which is the best possible when $\ell = 4$ in this construction. At this point, we do not have a closed form expression for the function $f$ and we only provide details for specific cases below. (These cases were verified using the MAGMA software [1].)
The Type II, $\ell = 4$, LDPC codes have girth eight, minimum distance $d_{\min} \geq 2(p^s + 1)$, and blocklength $N = 1 + p^s + p^{2s} + p^{3s}$. (We believe that the tree bound on the minimum distance is met for most of the Type II, $\ell = 4$, codes, i.e. $d_{\min} = w_{\min} = 2(p^s + 1)$.) Figure 8.7 illustrates the general construction procedure.

**Example 8.3.2** For $d = 3$, the Type II, $\ell = 4$, LDPC constraint graph as shown in Figure 8.8 corresponds to the (2,2)-Finite-Generalized-Quadrangles-based LDPC (FGQ LDPC) code of [50]; the function $f$ used in constructing this example is defined by $f(i, j, k, t) = j + (i+1) \cdot t$, i.e., the map defined by the array $M_{i+1}$. The orthogonal arrays used for constructing this code are

$$M_0 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$ 

We now state some results concerning the choice of the function $f$.

1. The Type II $\ell = 4$ construction results in incidence graphs of finite generalized quadrangles for appropriately chosen functions $f$. These graphs have girth 8 and diameter 4.

2. For some specific cases, examples of the function $f(i, j, k, t)$ that resulted in a girth 8 graph is given in Table 8.2. (Note that for the second entry in the table, the function $g : GF(4) \rightarrow GF(4)$ is defined by the following maps: $0 \mapsto 1$, $1 \mapsto \alpha$, $\alpha \mapsto \alpha^2$, and $\alpha^2 \mapsto 0$.)

3. For the above set of functions, the resulting Type II $\ell = 4$ LDPC constraint graphs have minimum distance meeting the tree bound, when $p = 2$, i.e.,

$$d_{\min} = w_{\min} = 2(2^s + 1).$$ 

We conjecture that, in general, for degrees $d =
TABLE 8.2
THE FUNCTION $f$ FOR THE TYPE II $\ell = 4$ CONSTRUCTION.

<table>
<thead>
<tr>
<th>$p$</th>
<th>$s$</th>
<th>elements of $GF(p^s)$</th>
<th>degree</th>
<th>$f(i,j,k,t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>${0, 1}$</td>
<td>3</td>
<td>$j + (i + 1) \cdot t$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>${0, 1, \alpha, \alpha^2}$</td>
<td>5</td>
<td>$j + g(i) \cdot t$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>${0, 1, 2}$</td>
<td>4</td>
<td>$i \cdot (k + 2 \cdot i \cdot t) + j$</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>${0, 1, \alpha, ..., \alpha^7}$</td>
<td>10</td>
<td>$i \cdot (k + \alpha \cdot i \cdot t) + j$</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>${0, 1, 2, 3, 4}$</td>
<td>6</td>
<td>$i \cdot (k + 3 \cdot i \cdot t) + j$</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>${0, 1, 2, ..., 6}$</td>
<td>8</td>
<td>$i \cdot (k + 4 \cdot i \cdot t) + j$</td>
</tr>
</tbody>
</table>

$2^s + 1$, the Type II $\ell = 4$ girth eight LDPC constraint graphs have $d_{\text{min}} = w_{\text{min}} = T(2^s + 1, 8) = 2(2^s + 1)$.

The above results were verified using MAGMA and computer simulations.

8.3.4 Remarks

It is well known in the literature that finite generalized polygons (or, $N$-gons) of order $p^s$ exist [34]. A finite generalized $N$-gon is a non-empty point-line geometry, and consists of a set $\mathcal{P}$ of points and a set $\mathcal{L}$ of lines such that the incidence graph of this geometry is a bipartite graph of diameter $N$ and girth $2N$. Moreover, when each point is incident on $t + 1$ lines and each line contains $t + 1$ points, the order of the $N$-gon is said be to $t$. The Type II $\ell = 3$ and $\ell = 4$ constructions yield finite generalized 3-gons and 4-gons, respectively, of order $p^s$. 

126
Figure 8.7. Tree construction of girth 8 Type II ($\ell = 4$) LDPC code.

Figure 8.8. Type II LDPC constraint graph having degree $d = 3$ and girth $g = 8$. (Shaded nodes highlight a minimum weight codeword.)
These are essentially finite projective planes and finite generalized quadrangles. The Type II construction can be similarly extended to larger \( \ell \). We believe that finding the right connections for connecting the nodes between the last layer in \( T \) and the final layer will yield incidence graphs of these other finite generalized polygons. For instance, for \( \ell = 6 \) and \( \ell = 8 \), the construction can yield finite generalized hexagons and finite generalized octagons, respectively. We conjecture that the incidence graphs of generalized \( N \)-gons yield LDPC codes with minimum pseudocodeword weight \( w_{\text{min}} \) very close to the corresponding minimum distance \( d_{\text{min}} \) and particularly, for generalized \( N \)-gons of order \( 2^s \), the LDPC codes have \( d_{\text{min}} = w_{\text{min}} = T(2^s + 1, 2N) \).

8.4 Simulation Results

8.4.1 Performance with min-sum iterative decoding

Figures 8.9, 8.10, 8.11, 8.12 show the bit-error-rate performance of Type I-A, Type I-B, Type II \( \ell = 3 \) (girth six), and Type II \( \ell = 4 \) (girth eight) LDPC codes, respectively, over the binary input additive white Gaussian noise channel (BIAWGNC) with min-sum iterative decoding. The performance of regular or semi-regular randomly constructed LDPC codes of comparable rates and block-lengths are also shown. (All of the random LDPC codes compared in this chapter have a variable node degree of three and are constructed from the online LDPC software available at \text{http://www.cs.toronto.edu/~radford/ldpc.software.html}.)

Figure 8.9 shows that the Type I-A LDPC codes perform substantially better than their random counterparts. Figure 8.10 reveals that the Type I-B LDPC codes perform better than comparable random LDPC codes at short blocklengths; but as the blocklengths increase, the random LDPC codes tend to perform better.
in the waterfall region. Eventually however, as the SNR increases, the Type I-B LDPC codes outperform the random ones and, unlike the random codes, they do not have a prominent error floor. Figure 8.11 reveals that the performance of Type II $\ell = 3$ (girth-six) LDPC codes is also significantly better than comparable random codes; these codes correspond to the two dimensional PG-LDPC codes of [45]. Figure 8.12 indicates the performance of Type II $\ell = 4$ (girth-eight) LDPC codes; these codes perform comparably to random codes at short blocklengths, but at large blocklengths, the random codes perform better as they have larger relative minimum distances compared to the Type II $\ell = 4$ (girth-eight) LDPC codes.

As a general observation, min-sum iterative decoding of most of the tree-based LDPC codes (particularly, Type I-A, Type II, and some Type I-B) presented here did not typically reveal detected errors, i.e., errors caused due to the decoder failing to converge to any valid codeword within the maximum specified number of iterations, which was set to 200 in these simulations. Detected errors are caused primarily due to the presence of pseudocodewords, especially those of minimum weight. We think that the relatively low occurrence of detected errors with iterative decoding of these LDPC codes is mostly due to their good minimum pseudocodeword weight $w_{\text{min}}$. (For updated simulation results including longer blocklength tree-based LDPC codes, see [26].)

8.4.2 Performance of Type I-B and Type II LDPC codes with sum-product iterative decoding

Figures 8.13, 8.14, and 8.15 show the performance of the Type I-B, Type II $\ell = 3$ and Type II $\ell = 4$, respectively, LDPC codes with sum-product iterative decoding.
decoding for the BIAWGNC. The performance is shown only for a few codes from each construction. The main observation from these performance curves is that the tree-based LDPC codes perform relatively much better than random LDPC codes of comparable parameters when the decoding is sum-product instead of the min-sum algorithm. Although the Type I-B LDPC codes perform a little inferior to their random counterparts in the waterfall region, the gap between the performances of the random and the Type I-B LDPC codes is much smaller with sum-product decoding than with min-sum decoding. (Compare Figures 8.10 and 8.13.) Similarly, comparing Figures 8.14 and 8.11, we see that the Type II $\ell = 3$ LDPC codes perform relatively much better than their random counterparts with sum-product decoding than with min-sum decoding. Figure 8.15, in comparison with Figure 8.12, shows a similar trend in performance of Type II $\ell = 4$ (girth 8) LDPC codes with sum-product iterative decoding.

Note that the simulation results for the min-sum and sum-product decoding correspond to the case when the LDPC codes resulting from constructions Type I and Type II were treated as binary LDPC codes for all choices of degree $d = p^s$ or $d = p^s + 1$. In the next chapter, we will examine the performance of the tree-based constructions when the LDPC constraint graphs of degree $p^s$ (or, $p^s + 1$) are considered to represent $p$-ary LDPC codes.

8.5 Summary of Constructions

The Type I construction yields a family of LDPC codes that, to the best of our knowledge, do not correspond to any of the LDPC codes obtained from finite geometries or other geometrical objects. It would be interesting to extend the Type II construction to more layers as described at the end of Section 8.2.4, and
Figure 8.9. Performance of Type I-A versus Random LDPC codes on the BIAWGNC with min-sum iterative decoding.

to extend the Type I-A construction by relaxing the girth condition. In addition, these codes may be amenable to efficient tree-based encoding procedures. The tree-based constructions presented in this paper yield a wide range of codes that perform well when decoded iteratively, largely due to the maximized minimum pseudocodeword weight. However, the overall minimum distance of the code is relatively small. Constructing codes with larger minimum distance, while still maintaining $d_{\text{min}} = w_{\text{min}}$, remains a challenging problem.
Performance of Type I-B versus Random LDPCs with min-sum iterative decoding

Figure 8.10. Performance of Type I-B versus Random LDPC codes on the BIAWGNC with min-sum iterative decoding.

Performance of Type II (L=3) versus Random LDPCs with min-sum iterative decoding

Figure 8.11. Performance of Type II $\ell = 3$ versus Random LDPC codes on the BIAWGNC with min-sum iterative decoding.
Figure 8.12. Performance of Type II $\ell = 4$ versus Random LDPC codes on the BIAWGNC with min-sum iterative decoding.

Figure 8.13. Performance of Type I-B versus Random LDPC codes on the BIAWGNC with sum-product iterative decoding.
Figure 8.14. Performance of Type II $\ell = 3$ versus Random LDPC codes on the BIAWGNC with sum-product iterative decoding.

Figure 8.15. Performance of Type II $\ell = 4$ versus Random LDPC codes on the BIAWGNC with sum-product iterative decoding.
8.6 Proofs

**Theorem 8.2.1** **Proof:** We need to show that there are no 4-cycles in the graph. By construction, it is clear that there are no 4-cycles that involve the nodes in layers $L_0$, $L'_0$, $L_1$, and $L'_1$. This is because no two nodes, say, variable nodes $(i, j)$ and $(i, k)$ in a particular class $S_i$ are connected to the same node $(s, t)'_c$ in some class $S'_i$; otherwise, it would mean that $t = j + i \cdot s = k + i \cdot s$. But this is only true for $j = k$. Therefore, suppose there is a 4-cycle in the graph, then let us assume that variable nodes $(i, j)$ and $(s, t)$, for $s \neq i$, are each connected to constraint nodes $(a, b)'_c$ and $(e, f)'_c$. By construction, this means that $b = j + i \cdot a = t + s \cdot a$ and $f = j + i \cdot e = t + s \cdot e$. However then $j - t = (s - i) \cdot a = (s - a) \cdot e$, thereby implying that $a = e$. When $a = e$, we also have $b = j + i \cdot a = j + i \cdot e = f$. Thus, $(a, b)'_c = (e, f)'_c$. Therefore, there are no 4-cycles in the Type I-B LDPC graphs.

**Theorem 8.2.2** **Proof:** When $p$ is an odd prime, the assertion follows immediately. Consider the following active variable nodes to be part of a codeword: variable nodes $(0, 1), (0, \alpha), \ldots, (0, \alpha^{p^s-2})$ in $S_0$, and all but the first variable node in the middle layer $L'_1$ of the reflected tree $T'$: i.e., variable nodes $(1)', (\alpha)', (\alpha^2)', \ldots, (\alpha^{p^s-2})'$ in $L'_1$. Clearly all the constraints in $L'_2$ are either connected to none or exactly two of these active variable nodes. The root node in $T'$ is connected to $p^s - 1$ (an even number) active variable nodes and the first constraint node in $L_1$ of $T$ is also connected to $p^s - 1$ active variable nodes. Hence, these $2(p^s - 1)$ active variable nodes form a codeword. This fact along with Theorems 8.0.1 and 8.2.1 prove that $2(p^s - 1) \geq d_{\min} \geq w_{\min} \geq T(p^s, 6) = 1 + p^s$.

When $p = 2$, consider the following active variable nodes to be part of a codeword: the root node, variable nodes $(0, 1), (0, \alpha), \ldots, (0, \alpha^{p^s-2})$ in $S_0$, variable node
$(\alpha^i, \alpha^i)$ from $S_{\alpha^i}$, for $i = 0, 1, 2, \ldots, p^s - 2$, and the first two variable nodes in the middle layer of $T'$ (i.e., variable nodes $(0)',(1)'$). Since $p = 2$, $p^s - 1$ is odd. We need to show that all the constraints are satisfied for this choice of active variable nodes. Each constraint node in the layer $L_1$ of $T$ has an even number of active variable node neighbors: $(0)_c$ has $p^s$ active neighbors, and $(i)_c$, for $i = 1, \alpha, \ldots, \alpha^{p^s-2}$, has two, the root node and variable node $(i,i)$. It remains to check the constraint nodes in $T'$.

In order to examine the constraints in layer $L_2'$ of $T'$, observe that the variable node $(0,\alpha^j)$, for $j = 1, 2, \ldots, p^s - 2$, is connected to constraint nodes $(1,\alpha^j)'_c, (\alpha,\alpha^j)'_c, \ldots, (\alpha^{p^s-2},\alpha^j)'_c$,

and the variable node $(\alpha^i, \alpha^i)$, for $i = 0, 1, 2, \ldots, p^s - 2$, is connected to constraint nodes $(0,\alpha^i)_c, (1,\alpha^i + \alpha^i)'_c, (\alpha,\alpha^i + \alpha^{i+1})'_c, \ldots, (\alpha^k,\alpha^{i+k})'_c, \ldots, (\alpha^{p^s-2},\alpha^{i+p^s-2})'_c$

Therefore, the constraint nodes $(0,\alpha^j)'_c$, for $j = 1, 2, \ldots, p^s - 2$, in $S_0'$ of $L_2'$ are connected to exactly one active variable node from layer $L_2$, i.e., variable node $(\alpha^j, \alpha^j)$; the other active variable node neighbor is variable node $(0)'$ in the middle layer of $T'$. Thus, all constraints in $S_0'$ are satisfied.

The constraint nodes $(1,\alpha^j)'_c$, for $j = 1, 2, \ldots, p^s - 2$, in $S_1'$ are each connected to exactly one active variable node from $L_2$, i.e., variable node $(0,\alpha^j)$ from $S_0$. This is because, all the remaining active variable nodes in $L_2$, $(\alpha^i, \alpha^i)$ connect to the imaginary node $(1,0)'_c$ in $S_1'$ (since $(1,\alpha^i + \alpha^i)'_c = (1,0)'_c$ when the characteristic of the field $GF(p^s)$ is $p = 2$). Thus, all constraint nodes in $S_1'$ have two active
variable node neighbors, the other active neighbor being the variable node \((1)\)' in the middle layer of \(T'\).

Now, let us consider the constraint nodes in \(S'_\alpha\), for \(k = 1, 2, \ldots, p^s - 2\). The active variable nodes \((\alpha^i, \alpha^t)\), for \(i = 0, 1, 2, \ldots, p^s - 2\), are connected to the following constraint nodes

\[
(\alpha^k, \alpha^+1)'_c, (\alpha^k, \alpha + \alpha^+1)'_c, \ldots, (\alpha^k, \alpha^{p^s-2} + \alpha^{p^s-2+k})'_c,
\]

respectively, in class \(S'_\alpha\). Since \(\alpha^r + \alpha^{k+r} \neq \alpha^t + \alpha^{k+t}\) for \(r \neq t\), the variable nodes \((\alpha^i, \alpha^t)\), for \(i = 0, 1, 2, \ldots, p^s - 2\), connect to distinct nodes in \(S'_\alpha\). Hence, each constraint node in \(S'_\alpha\) has exactly two active variable node neighbors – one from \(S_0\) and the other from the set \(\{(\alpha^i, \alpha^t)\mid i = 0, 1, 2, \ldots, p^s - 2\}\).

Last, we note that the root (constraint) node in \(T'\) is connected to two active variable nodes, \(0)'\) and \((1)\)'\). The total number of active variable nodes is \(1 + (p^s - 1) + (p^s - 1) + 2 = 2p^s + 1\). This proves that the set of \(2p^s + 1\) active variable nodes forms a codeword, thereby proving the desired bound.

**Theorem 8.3.2**  **Proof:** We need to show is that there are no 4-cycles in the graph. As in the proof of Theorem 8.2.1, by construction, there are no 4-cycles that involve the nodes in layers \(L_0\) and \(L_1\). This is because, first, no two variable nodes in the first class \(S_x = \{(x, 0), (x, 1), \ldots, (x, \alpha^{p^s-2})\}\) are connected to the same constraint node. Next, if two variable nodes, say, \((i, j)\) and \((i, k)\) in the \(i^{th}\) class \(S_i\), for some \(i \neq x\), are connected to a constraint node \((s, t)'_c\), then it would mean that \(t = j + i \cdot s = k + i \cdot s\). But this is only true for \(j = k\). Hence, there is no 4-cycle of the form \((i)'_c \rightarrow (i, j) \rightarrow (s, t)'_c \rightarrow (i, k) \rightarrow (i)'\). Therefore, suppose there is a 4-cycle in the graph, then let us consider two cases as follows.

Case 1) Assume that variable nodes \((i, j)\) and \((s, t)\), for \(i \neq s\) and \(i \neq x \neq s\),
are each connected to constraint nodes \((a, b)_c\) and \((e, f)_c\). By construction, this means that \(b = j + i \cdot a = t + s \cdot a\) and \(f = j + i \cdot e = t + s \cdot e\). This implies that \(j - t = (s - i) \cdot a = (s - i) \cdot e\), thereby implying that \(a = e\). Consequently, we also have \(b = j + i \cdot a = j + i \cdot e = f\). Thus, \((a, b)_c = (e, f)_c\). Case 2) Assume that two variable nodes, one in \(S_x\), say, \((x, j)\), and the other in \(S_i\), (for \(i \neq x\)), say, \((i, k)\), are connected to constraint nodes \((a, b)_c\) and \((e, f)_c\). Then this would mean that \(a = e = j\). But since \((i, k)\) connects to exactly one constraint node whose first index is \(j\), this case is not possible. Thus, there are no 4-cycles in the Type II-\(\ell = 3\) LDPC graphs.

To show that the girth is exactly six, we see that the following nodes form a six-cycle in the graph: the root-node, the first two constraint nodes \((x)_c\) and \((1)_c\) in layer \(L_1\), variable nodes \((x, 0)\) and \((0, 0)\) in layer \(L_2\), and the constraint node \((0, 0)_c\) in layer \(L_3\).

To prove the diameter, we first observe that the root node is at distance of at most three from any other node. Similarly, it is also clear that the nodes in layer \(L_1\) are at a distance of at most three from any other node. Therefore, it is only necessary to show that any two nodes in layer \(L_2\) are at most distance two apart and similarly show that any two nodes in \(L_3\) are at most distance two apart. Consider two nodes \((i, j)\) and \((s, t)\) in \(L_2\). If \(s = i\), then clearly, there is a path of length two via the parent node \((i)_c\). If \(s \neq i\) and \(s \neq x \neq i\), then by the property of a complete family of orthogonal Latin squares there is a node \((a, b)_c\) in \(L_3\) such that \(b = j + i \cdot a = t + s \cdot a\). This implies that \((i, j)\) and \((s, t)\) are connected by a distance two path via \((a, b)_c\). We can similarly show that if \(s \neq i\) and \(i = x\), then the node \((j, t + s \cdot j)_c\) in \(L_3\) connects to both \((x, j)\) and \((s, t)\). A similar argument shows that any two nodes in \(L_3\) are distance two apart, completing the proof. ■
The constraint nodes in the first class of active variable nodes in the Type II $\ell = 3$ LDPC constraint graph form a minimum-weight codeword: the root (variable) node, variable nodes $(x, 0), (0, 0), (1, \alpha^{p^s-2}), (\alpha, \alpha^{p^s-3}), (\alpha^2, \alpha^{p^s-4}), \ldots, (\alpha^i, \alpha^{p^s-2-i}), \ldots, (\alpha^{p^s-2}, 1)$ in layer $L_2$.

It is clear from this choice that there is exactly one active variable node from each class in layer $L_2$. Therefore, all the constraint nodes at layer $L_1$ are satisfied. The constraint nodes in the first class $S'_0$ of $L_3$ are $(0,0)'_c, (0,1)'_c, \ldots, (0,\alpha^{p^s-2})'_c$.

The constraint node $(0,0)'_c$ connects to $(x,0)$ and $(0,0)$, and the constraint node $(0,\alpha^i)'_c$, for $i = 0, 1, 2, \ldots, p^s - 2$, is connected to variable nodes $(x,0)$ and $(\alpha^i, \alpha^{p^s-2-i})$. Thus, all constraint nodes in $S'_0$ are satisfied. Let us consider the constraint nodes in class $S'_i$, for $i \in \{0, 1, 2, \ldots, p^s - 2\}$. The variable node $(0,0)$ connects to the constraint node $(\alpha^i,0)'_c$ in $S'_i$. The variable node $(1,\alpha^{p^s-2})$ connects to the constraint node $(\alpha^i, \alpha^{p^s-2} + \alpha^i)'_c$ in $S'_i$, and in general, for $j = 0, 1, 2, \ldots, p^s - 2$, the variable node $(\alpha^j, \alpha^{p^s-2-j})$ connects to the constraint node $(\alpha^i, \alpha^{p^s-2-j} + \alpha^{i+j})'_c$ in $S'_i$. So enumerating all the constraint nodes in $S'_i$, with multiplicities, that are connected to an active variable node in $L_2$, we obtain

$$(\alpha^i,0)'_c, (\alpha^i, \alpha^{p^s-2} + \alpha^i)'_c, (\alpha^i, \alpha^{p^s-3} + \alpha^{i+1})'_c, \ldots, (\alpha^i, \alpha^{p^s-(p^s-i-1)} + \alpha^{p^s-3})'_c, (\alpha^i, \alpha^{p^s-(p^s-i)} + \alpha^{p^s-2})'_c, \ldots.$$ 

Simplifying the exponents and rewriting this list, we see that, when $i$ is odd, the constraint nodes are

$$(\alpha^i,0)'_c, (\alpha^i, \alpha^{p^s-2} + \alpha^i)'_c, (\alpha^i, \alpha^{p^s-3} + \alpha^{i+1})'_c, \ldots, (\alpha^i, \alpha^{i+1} + \alpha^{p^s-3})'_c, (\alpha^i, \alpha^i + \alpha^{p^s-2})'_c, (\alpha^i, \alpha^{i-1} + 1)'_c, (\alpha^i, \alpha^{i-2} + \alpha)'_c, \ldots, (\alpha^i, \alpha^{(i-1)/2} + \alpha^{(i-1)/2})'_c, \ldots, (\alpha^i, \alpha + \alpha^{i-2})'_c, (\alpha^i, 1 + \alpha^{i-1})'_c.$$ 

(When $i$ is even, the constraint nodes are $$(\alpha^i,0)'_c, (\alpha^i, \alpha^{p^s-2} + \alpha^i)'_c, (\alpha^i, \alpha^{p^s-3} + \alpha^{i+1})'_c, \ldots, (\alpha^i, \alpha^{(p^s-2-i)/2} + \alpha^{(p^s-2-i)/2})'_c, \ldots, (\alpha^i, \alpha^{i+1} + \alpha^{p^s-3})'_c, (\alpha^i, \alpha^{i} + \alpha^{p^s-2})'_c,$$ 

Theorem 8.3.3 Proof: Let $p = 2$. We will show that the following set of active variable nodes in the Type II $\ell = 3$ LDPC constraint graph form a minimum-weight codeword: the root (variable) node, variable nodes $(x, 0), (0, 0), (1, \alpha^{p^s-2}), (\alpha, \alpha^{p^s-3}), (\alpha^2, \alpha^{p^s-4}), \ldots, (\alpha^i, \alpha^{p^s-2-i}), \ldots, (\alpha^{p^s-2}, 1)$ in layer $L_2$.
\((\alpha^i, \alpha^{i-1} + 1)'_c, (\alpha^i, \alpha^{i-2} + \alpha)'_c, \ldots, (\alpha^i, \alpha^i + \alpha^{i-1})'_c, (\alpha^i, \alpha^{i-1} + \alpha^i)'_c, \ldots, (\alpha^i, 1 + \alpha^{i-1})'_c\). (Note that for \(\beta \in GF(p^s)\), \(\beta + \beta = 0\) when the characteristic of the field \(GF(p^s)\) is two (i.e., \(p = 2\)).)

Observe that each of the constraint nodes in the above list appears exactly twice. Therefore, each constraint node in the list is connected to two active variable nodes in \(L_2\), and hence, all the constraints in \(S'_\alpha\) are satisfied. So we have that the set of \(1 + 2^s + 1\) active variable nodes forms a codeword. Furthermore, they must form a minimum-weight codeword since \(d_{\text{min}} \geq 2 + 2^s = T(2^s + 1, 6)\) by the tree bound of Theorem 8.0.1. This also proves that \(d_{\text{min}} = w_{\text{min}} = T(d, 6)\) for \(d = 2^s + 1\).
In this chapter we discuss the parameters and performance of the tree-based LDPC codes and finite-geometry LDPC codes when treated as $p$-ary codes with binary parity-check matrices and simulated over the $p$-ary symmetric channel, for $p$ a prime number. The bounds and analysis of chapter 4 are used here.

We observe that if the codes resulting from the Type I-B construction and the Type II constructions are treated as $p$-ary codes rather than binary codes when the corresponding degree in the LDPC graph is $d = p^s$ or $d = p^s + 1$, then the rates obtained are much superior. From the cases we examine and the results presented in this chapter, we show that the minimum pseudocodeword weights (on the $p$-ary symmetric channel) are much closer to the minimum distances for these $p$-ary LDPC codes.

9.1 $p$-ary Type I-B LDPC codes

**Theorem 9.1.1** For degree $d = p^s$, the resulting Type I-B LDPC constraint graphs of girth $g$ that represent $p$-ary LDPC codes have minimum distance and minimum pseudocodeword weight $2p^s + 1 \geq d_{\min} \geq w_{\min} \geq T(d, g)$. 
9.2 $p$-ary Type II LDPC Codes

**Theorem 9.2.1** For degrees $d = 2^s + 1$, the resulting Type II, $\ell = 3$ LDPC constraint graphs have $d_{\min} = w_{\min} = T(d, 6) = 2 + 2^s$. For degrees $d = p^s + 1$, $p > 2$, when the resulting Type II, $\ell = 3$ LDPC constraint graphs represent $p$-ary linear codes, the corresponding minimum distance and minimum pseudocodeword weight satisfy

$$T(p^s + 1, 6) \leq w_{\min} \leq d_{\min} \leq 2p^s.$$ 

For degrees $d = p^s + 1$, $p > 2$, when the Type II, $\ell = 3$ LDPC constraint graphs are treated as we believe that the distance $d_{\min} \geq p^s + 3$, and that this bound is in fact tight.

For degrees $d = p^s + 1$, $p > 2$, when the Type II, $\ell = 4$ construction are treated as $p$-ary LDPC codes, we believe the minimum distance $d_{\min}$ is either equal to or very close to the tree bound. Hence, we also expect the corresponding minimum pseudocodeword weight $w_{\min}$ to be close to $d_{\min}$ for these codes.

9.2.1 $p$-ary finite geometry codes

Recall from chapter 8 that the Type II, $\ell = 3$ LDPC codes are equivalent to the two-dimensional projective geometry codes originally used for LDPC coding in [29]. These codes were later extended to higher dimensions in [45].

It is worth noting that these papers do not comment on the rate of the binary finite geometry LDPC codes when $p \neq 2$, and as their rates are low, the examination of these codes as $p$-ary codes is further justified.

In summary, we state the following results:
The rate of a $p$-ary Type II, $\ell = 3$ LDPC code is
$$r = \frac{p^2 + p^s - s^* (p+1)^s}{p^s + p^{s+1}}$$ [21].

The rate of a binary Type II, $\ell = 3$ LDPC code is $\frac{1}{n}$ for $p > 2$.

Note that binary codes with $p = 2$ are a special case of $p$-ary LDPC codes. Moreover, the rate expression for $p$-ary LDPC codes is meaningful for a wide variety of $p$'s and $s$'s. The rate expression for binary codes with $p > 2$ can be seen by observing that any $t$ rows of the corresponding parity-check matrix $H$ is linearly independent if $t < n$. Since the parity-check matrix is equivalent to one obtainable from cyclic difference sets, this can be proven by showing that for any $t < n$, there exists a set of $t$ consecutive positions in the first row of $H$ that has an odd number of ones.

9.3 Performance of $p$-ary Type I-B and Type II LDPC Codes over the $p$-ary Symmetric Channel

We will now examine the performance when the codes are treated as $p$-ary codes if the corresponding degree in the LDPC constraint graph is $d = p^s$ (for Type I-B) or $d = p^s + 1$ (for Type II). (Note that this will affect only the performances of those codes for which $p$ is not equal to two.)

The codes are simulated on the $p$-ary symmetric channel instead of the AWGN channel. The $p$-ary symmetric channel is shown in Figure 4.1. An error occurs with probability $\epsilon$, the channel transition probability. Figures 9.1, 9.2, and 9.3 show the performance of Type I-B, Type II $\ell = 3$ and Type II $\ell = 4$, 3-ary LDPC codes, respectively, on the 3-ary symmetric channel with sum-product iterative decoding. The parity check matrices resulting the the Type I-B and Type II constructions...
are considered to be matrices over the field $GF(3)$ and sum-product iterative decoding is implemented as outlined in [10]. The corresponding plots show the information symbol error rate as a function of the channel transition probability $\epsilon$. In Figure 9.1, the performance of 3-ary Type I-B LDPC codes obtained for degrees $d = 3$, $d = 3^2$, and $d = 3^3$, is shown and compared with the performance of random 3-ary LDPC codes of comparable rates and block lengths. (To make a fair comparison, the random LDPC codes also have only zeros and ones as entries in their parity check matrices. It has been observed in [10] that choosing the non-zero entries in the parity check matrices of non-binary codes cleverly can yield some performance gain, but this avenue was not explored in these simulations.) In Figure 9.2, the performance of 3-ary Type II $\ell = 3$ (girth six) LDPC codes obtained for degrees $d = 3 + 1$, $d = 3^2 + 1$, and $d = 3^3 + 1$, is shown and compared with random 3-ary LDPC codes. Figure 9.3 shows the analogous performance of 3-ary Type II $\ell = 4$ (girth eight) LDPC codes obtained for degrees $d = 3 + 1$ and $d = 3^2 + 1$. In all of these plots, the tree-based constructions perform comparably or better than random LDPC codes of similar rates and block lengths. (In some cases (for example, Figure 9.3), the performance of the tree-based constructions is significantly better than that of random LDPC codes.

The simulation results show that the tree-based constructions yield a wide range of LDPC codes that perform very well with iterative decoding.

9.3.1 Tables of code parameters

The code parameters resulting from the tree-based constructions are summarized in Tables 9.1, 9.2, 9.3, and 9.4. Note that * indicates an upper bound instead of the exact minimum distance (or minimum pseudocodeword weight) since it was
Figure 9.1. Performance of Type I-B versus Random 3-ary LDPC codes on the 3-ary symmetric channel with sum–product iterative decoding.

Computationally hard to find the distance (or pseudocodeword weight) for those cases. Similarly, for cases where it was computationally hard to get any reasonable bound the minimum pseudocodeword weight, the corresponding entry in the table is left empty. The lower bound on $w_{min}$ seen in the tables corresponds to the tree bound (Theorem 8.0.1). It is observed that when the codes resulting from the construction are treated as $p$-ary codes rather than binary codes when the corresponding degree in the LDPC graph is $d = p^s$ (for Type I-B) or $d = p^s + 1$ (for Type II), the resulting rates obtained are much superior; we also believe that the minimum pseudocodeword weights (on the $p$-ary symmetric channel) are much closer to the minimum distances for these $p$-ary LDPC codes.
Figure 9.2. Performance of Type II $\ell = 3$ versus Random 3-ary LDPC codes on the 3-ary symmetric channel with sum-product iterative decoding.

TABLE 9.1

<table>
<thead>
<tr>
<th>No. of layers in $T$: $\ell$</th>
<th>length $n$</th>
<th>deg. $d = 3$</th>
<th>dim.</th>
<th>rate</th>
<th>$d_{\min}$</th>
<th>$w_{\min}$</th>
<th>tree $g$</th>
<th>girth $g$</th>
<th>dia. $\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>0.4000</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>3</td>
<td>4</td>
<td>0.1818</td>
<td>8</td>
<td>8*</td>
<td>6</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>46</td>
<td>3</td>
<td>10</td>
<td>0.2173</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>94</td>
<td>3</td>
<td>14</td>
<td>0.1489</td>
<td>20</td>
<td>20*</td>
<td>18</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>190</td>
<td>3</td>
<td>25</td>
<td>0.1315</td>
<td>$\geq 18$</td>
<td>18</td>
<td>18</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>
Figure 9.3. Performance of Type II $\ell = 4$ versus Random 3-ary LDPC codes on the 3-ary symmetric channel with sum-product iterative decoding.
TABLE 9.2
SUMMARY OF TYPE I-B CODE PARAMETERS.

| \(p\) | \(s\) | length \(p^{2s+1}\) | deg. \(\frac{d}{s} = p^s\) | dim. | rate | \(d_{\min}\) | \(w_{\min}\) | tree LB | gth. \(g\) | dia. \(\delta\) | code alphabet |
|-------|-------|----------------|----------------|------|------|----------------|--------------|---------|----------|----------|-------------|----------------|
| 2     | 1     | 5              | 2              | 1    | 0.2000 | 5              | 5            | 4       | 8        | 4        | 5           | binary         |
| 3     | 1     | 10             | 3              | 3    | 0.2000 (2) | 4              | 4            | 4       | 6        | 5        | binary 3-ary |
| 2     | 2     | 17             | 4              | 5    | 0.2941 | 6              | 6*           | 5       | 6        | 5        | binary      |
| 5     | 1     | 26             | 5              | 7    | 0.2692 (7) | 8              | 8*           | 6       | 6        | 5        | binary 5-ary |
| 7     | 1     | 50             | 7              | 11   | 0.2200 (16) | 12             | 12*          | 8       | 6        | 5        | binary 7-ary |
| 2     | 3     | 65             | 8              | 31   | 0.4769 | 10             | 10*          | 9       | 6        | 5        | binary      |
| 3     | 2     | 82             | 9              | 15 (38) | 0.1829 (46) | 16             | \(\geq 10\) | 10      | 6        | 5        | binary 3-ary |
| 11    | 1     | 122            | 11             | 19 (46) | 0.1557 (377) | 20*           | \(\geq 12\) | 12      | 6        | 5        | binary 11-ary |
| 2     | 4     | 257            | 16             | 161   | 0.6264 | 20*           | \(\geq 17\) | 17      | 6        | 5        | binary      |
| 5     | 2     | 626            | 25             | 47 (377) | 0.075 (6022) | 48*           | \(\geq 26\) | 26      | 6        | 5        | binary 5-ary |
| 3     | 3     | 730            | 27             | 51 (488) | 0.0698 (6684) | 52*           | \(\geq 28\) | 28      | 6        | 5        | binary 3-ary |
| 2     | 5     | 1025           | 32             | 751   | 0.7326 | 40*           | \(\geq 33\) | 33      | 6        | 5        | binary      |
| 7     | 2     | 2404           | 49             | 95 (1572) | 0.0395 (6536) | 96*           | \(\geq 50\) | 50      | 6        | 5        | binary 7-ary |
### TABLE 9.3
**SUMMARY OF TYPE II, $\ell = 3$ CODE PARAMETERS.**

<table>
<thead>
<tr>
<th>$p$</th>
<th>$s$</th>
<th>$n_1+p^2+p^3$</th>
<th>$n_2+p^2+p^3$</th>
<th>$n_3+p^2+p^3$</th>
<th>$d_{min}$</th>
<th>$w_{min}$</th>
<th>$g_{th}$</th>
<th>$g$</th>
<th>$\delta$</th>
<th>code alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>0.4285</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>binary</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>13</td>
<td>4</td>
<td>1</td>
<td>0.0769</td>
<td>13</td>
<td>6°</td>
<td>5</td>
<td>6</td>
<td>3-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.4615)</td>
<td>(6)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>21</td>
<td>5</td>
<td>11</td>
<td>0.5238</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>binary</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>31</td>
<td>6</td>
<td>1</td>
<td>0.0322</td>
<td>31</td>
<td>10°</td>
<td>7</td>
<td>6</td>
<td>3-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.4838)</td>
<td>(10°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>57</td>
<td>8</td>
<td>1</td>
<td>0.017</td>
<td>57</td>
<td>10°</td>
<td>9</td>
<td>6</td>
<td>5-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.4912)</td>
<td>(14°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>73</td>
<td>9</td>
<td>45</td>
<td>0.6164</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>6</td>
<td>binary</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>91</td>
<td>10</td>
<td>1</td>
<td>0.0109</td>
<td>91</td>
<td>$\geq$ 11</td>
<td>11</td>
<td>6</td>
<td>3-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.5934)</td>
<td>(15°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>273</td>
<td>17</td>
<td>191</td>
<td>0.6996</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>6</td>
<td>3-ary</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>651</td>
<td>26</td>
<td>1</td>
<td>0.0015</td>
<td>651</td>
<td>$\geq$ 27</td>
<td>27</td>
<td>6</td>
<td>3-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.6528)</td>
<td>(56°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TABLE 9.4
**SUMMARY OF TYPE II, $\ell = 4$ CODE PARAMETERS.**

<table>
<thead>
<tr>
<th>$p$</th>
<th>$s$</th>
<th>$n_1+p^2+p^3$</th>
<th>$n_2+p^2+p^3$</th>
<th>$n_3+p^2+p^3$</th>
<th>$d_{min}$</th>
<th>$w_{min}$</th>
<th>$g_{th}$</th>
<th>$g$</th>
<th>$\delta$</th>
<th>code alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>15</td>
<td>3</td>
<td>5</td>
<td>0.3315</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>binary</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>40</td>
<td>4</td>
<td>15</td>
<td>0.3700</td>
<td>10</td>
<td>$\geq$ 8</td>
<td>8</td>
<td>4</td>
<td>3-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.3700)</td>
<td>(10°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>85</td>
<td>5</td>
<td>35</td>
<td>0.4117</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>binary</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>156</td>
<td>6</td>
<td>65</td>
<td>0.4167</td>
<td>20°</td>
<td>$\geq$ 12</td>
<td>12</td>
<td>4</td>
<td>5-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.4167)</td>
<td>(26°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>490</td>
<td>8</td>
<td>175</td>
<td>0.4375</td>
<td>66°</td>
<td>$\geq$ 16</td>
<td>16</td>
<td>4</td>
<td>binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.4375)</td>
<td>(119°)</td>
<td>( )</td>
<td></td>
<td></td>
<td>7-ary</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>585</td>
<td>9</td>
<td>287</td>
<td>0.4905</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>4</td>
<td>binary</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>820</td>
<td>10</td>
<td>369</td>
<td>0.4500</td>
<td>112°</td>
<td>$\geq$ 20</td>
<td>20</td>
<td>8</td>
<td>3-ary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.4817)</td>
<td>(96°)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9.4 Proofs

**Theorem 9.1.1**  
**Proof:** Consider as active variable nodes the root node, all the variable nodes in $S_0$, the variable nodes $(\alpha^i, \alpha^i)$, for $i = 0, 1, 2, \ldots, p^s - 2$, the first variable node $(0)'$ in the middle layer of $T'$ and one other variable node $(y)'$, that we will ascertain later, in the middle layer of $T'$.

Since the code is $p$-ary (and $p > 2$), assign the value 1 to the root variable node and to all the active variable nodes in $S_0$. Assign the value $p - 1$ to the remaining active variable nodes in $L_2$, (i.e., nodes $(\alpha^i, \alpha^i)$, $i = 0, 1, 2, \ldots, p^s - 2$). Assign the value 1 for the variable node $(0)'$ in the middle layer of $T'$ and assign the value $p - 1$ for the variable node $(y)'$ in the middle layer of $T'$. We choose $y$ in the following manner:

The variable nodes $(\alpha^i, \alpha^i)$, for $i = 0, 1, 2, \ldots, p^s - 2$, are connected to the following constraint nodes

$$(\alpha^k, 1 + \alpha^k)'_c, (\alpha^k, \alpha + \alpha^{k+1})'_c, \ldots, (\alpha^k, \alpha^i + \alpha^{i+j})'_c, \ldots, (\alpha^k, \alpha^{p^s-2} + \alpha^{k+p^s-2})'_c,$$

respectively, in class $S_{\alpha^k}$. Either the above set of constraint nodes are all distinct, or they are all equal to $(\alpha^k, 0)'_c$. This is because $\alpha^r + \alpha^{r+k} = \alpha^t + \alpha^{t+k}$ if and only if either $r = t$, or $1 + \alpha^k = 0$. So there is only one $k \in \{0, 1, 2, \ldots, p^s - 2\}$, for which $1 + \alpha^k = 0$, and for that value of $k$, we set $y = \alpha^k$.

From the proof of Theorem 8.2.2 and the above assignment, it is easily verified that each constraint node has value zero when the sum of the incoming active nodes is taken modulo $p$. Thus, the set of $2p^s + 1$ active variable nodes forms a codeword, and therefore $d_{\min} \leq 2p^s + 1$. Hence, from Lemmas 4.1.1 and 4.3.1, we have $T(d, 6) \leq w_{\min} \leq d_{\min} \leq 2p^s + 1$. 

\[\blacksquare\]
Theorem 9.2.1 Proof: Let \( p > 2 \). The resulting codes are treated as \( p \)-ary codes. Consider the following set of active variable nodes: the root node, all but one of the nodes \((x, y)\), for an appropriately chosen \( y \), in class \( S_x \), and the nodes \((\alpha^i, \alpha^i)\), for \( i = 0, 1, 2, \ldots, p^s - 2 \). We have chosen \( 2p^s \) active variable nodes in all.

The variable nodes \((1, 1), (\alpha, \alpha), \ldots, (\alpha^{p^s-2}, \alpha^{p^s-2})\) are connected to constraint nodes

\[(\alpha^k, 0)_c', (\alpha^k, 1 + \alpha^k)_c', (\alpha^k, \alpha + \alpha^{k+1})_c', \ldots, (\alpha^k, \alpha^j + \alpha^{k+j})_c', \ldots, (\alpha^k, \alpha^i + \alpha^{k+p^s-2})_c', \]

respectively, in class \( S'_{\alpha^k} \) of constraint nodes in layer \( L_3 \). These nodes are either all distinct or all equal to \((\alpha^k, 0)_c'\), since \( \alpha^r + \alpha^{k+r} = \alpha^t + \alpha^{k+t} \) if and only if either \( r = t \) or \( 1 + \alpha^k = 0 \). Since \( 1 + \alpha^k \) is zero for exactly one value of \( k \in \{0, 1, 2, \ldots, p^s - 2\} \), we have that the variable nodes \((\alpha^i, \alpha^i)\), for \( i = 0, 1, 2, \ldots, p^s - 2 \), are connected to distinct constraint nodes in all but one class \( S_\alpha \) and that, in \( S_\alpha \), they are all connected to the constraint node \((\alpha^{k*}, 0)_c'\). (Note that \( k* \) satisfies \( 1 + \alpha^{k*} = 0 \).)

We let \( y = \alpha^{k*} \). Therefore, the set of active variable nodes includes all nodes of the form \((x, t)\), for \( t = 0, 1, \alpha, \alpha^2, \ldots, \) excluding node \((x, y)\).

Since the code is \( p \)-ary, assign the following values to the chosen set of active variable nodes: assign the value \( 1 \) to the root variable node and to all of the active variable nodes in class \( S_x \), and assign the value \( p - 1 \) to the active variable nodes \((\alpha^i, \alpha^i)\), for \( i = 0, 1, 2, \ldots, p^s - 2 \). It is now easy to verify that all of the constraints are satisfied. Thus, \( d_{\min} \leq 2p^s \). From Theorem 8.3.2 and Lemmas 4.1.1 and 4.3.1, we have \( T(p^s + 1, 6) \leq w_{\min} \leq d_{\min} \leq 2p^s \). \( \blacksquare \)
The design of good graphs becomes a key criterion for the design of good codes. It has been shown in [43] that graphs having good expansion properties are good candidates for LDPC code designs. Among recent developments, Reingold et. al. [38] show that by using small component graphs that are known to be expanders, it is possible to design larger graphs that are also expanders. Their technique is the so called zig-zag product of the two component graphs. In this chapter, we examine the expansion properties of the zig-zag product in relation to the design of LDPC codes. We also examine other graph products and generalize the zig-zag product to unbalanced expander graph components. These resulting product graphs are then analyzed for their potential in LDPC coding.

In our code construction, the vertices of the product graph are interpreted as sub-code constraints of a suitable linear block code and the edges are interpreted as the code bits of the LDPC code, as originally suggested by Tanner in [48]. By choosing component graphs with relatively small degree, we obtain product graphs that are relatively sparse. Our preliminary findings in this direction indicate that LDPC codes based on zig-zag product graphs perform comparably to random LDPC codes for short block lengths, and to obtain a good LDPC code, the vertices of the zig-zag product graph must be fortified with strong (i.e., good
minimum distance) sub-code constraints. This is necessary also to achieve good performance with graph based message passing decoders.

10.1 Expander Graphs

**Definition 10.1.1** A $d$-regular graph $G$ on $N$ vertices is said to be a $(N, d, \lambda)$-graph if the second largest eigenvalue of the normalized adjacency matrix $\tilde{A}$ representing $G$ is $\lambda$.

It is now almost common knowledge that for a graph to be a good expander [43], the second largest eigenvalue of the adjacency matrix $A$ must be as small as possible compared to the index [46]. For a $d$-regular graph $G$, the index of the adjacency matrix $A$ is $d$. Note that for a Tanner graph representing an LDPC code with parity-check matrix $H$, the adjacency matrix is

$$
A = \begin{bmatrix}
0 & H \\
H^T & 0
\end{bmatrix}.
$$

Hence, by normalizing the entries of $A$ by the factor $d$, the normalized matrix $\tilde{A}$ has an index of 1. In this chapter, we will follow the definition provided in [3, 36] for a graph to be an expander. A sequence of graphs is said to be an expander family if for every graph $G$ in the family, the second largest eigenvalue $\lambda(G)$ of the normalized adjacency matrix $\tilde{A}$ is bounded below some constant $\lambda_q < 1$. Or in other words, there is an $\epsilon > 0$ such that for every graph $G$ in the family, $\lambda(G) < 1 - \epsilon$. In particular, a graph belonging to an expander family is called an expander graph. For regular graphs, the best possible expansion based on the eigenvalue bound is achieved by Ramanujan graphs that have $\lambda(G) \leq \frac{2\sqrt{d-1}}{d}$ [33]. Alon and Boppana have shown that for a $d$-regular graph $G$, as the number of
vertices \( n \) in \( G \) tends to infinity, \( \lambda(G) \geq \frac{2\sqrt{d-1}}{d} \) \[2\]; therefore, Ramanujan graphs are optimal in terms of the eigenvalue gap \( 1 - \lambda(G) \).

10.2 Graph Products

This section summarizes several common graph products. In each case, the expansion of the product graph with respect to the expansion of the component graphs is examined.

10.2.1 Tensor product

Let \( G_1 \) be a \((N_1, d_1, \lambda_1)\)-graph and let \( G_2 \) be a \((N_2, d_2, \lambda_2)\)-graph. Then the tensor product of \( G_1 \) and \( G_2 \) is a graph \( G \) on \( N_1 \cdot N_2 \) vertices with the vertex set and edge set defined as follows: the vertices of \( G \) are represented as ordered two tuples \((v, k)\), for \( v \in \{1, 2, \ldots, N_1\} \) and \( k \in \{1, 2, \ldots, N_2\} \), and there is an edge between \((v, k)\) and \((w, \ell)\) if: (i) there is an edge between the vertices \( v \) and \( w \) in \( G_1 \) and, (ii) there is an edge between vertices \( k \) and \( \ell \) in \( G_2 \). The tensor product \( G = G_1 \otimes G_2 \) is a \((N_1 \cdot N_2, d_1 \cdot d_2, \lambda)\)-graph, where \( \lambda = \max\{\lambda_1, \lambda_2\} \) \[38\].

Other standard graph products include the Cartesian product, strong product, and lexicographic product \[23\]. Let \( G_1 \) and \( G_2 \) be the component graphs. Then each of these products yields a graph with \( V(G_1) \times V(G_2) \) as the vertices, and edge relations based on edges in the components. However, since these products yield graphs which are not sparse, we omit further detail based on their lack of potential for LDPC codes (see section 10.3).
10.2.2 Replacement product

Let $G_1$ be a $(N_1, d_1, \lambda_1)$-graph and let $G_2$ be a $(d_1, d_2, \lambda_2)$-graph. (Observe that the number of vertices in $G_2$ is chosen to be equal to the degree of each vertex in $G_1$.) Then the replacement product of $G_1$ and $G_2$ is a graph $G$ with the vertex set and edge set defined as follows: the vertices of $G$ are represented as ordered two tuples $(v, k)$, for $v \in \{1, 2, \ldots, N_1\}$ and $k \in \{1, 2, \ldots, d_1\}$. There is an edge between $(v, k)$ and $(v, \ell)$ if there is an edge between $k$ and $\ell$ in $G_2$; there is also an edge between $(v, k)$ and $(w, \ell)$ if the $k^{th}$ edge incident on vertex $v$ in $G_1$ is connected to vertex $w$ and this edge is the $\ell^{th}$ edge incident on $w$ in $G_1$. The replacement product graph $G = G_1 \otimes G_2$ is a $(N_1 \cdot d_1, d_2 + 1, \lambda)$-graph with

Figure 10.1. Tensor product of two graphs.
\( \lambda \leq (p+(1-p)f(\lambda_1, \lambda_2))^{1/3} \) for \( p = d_2^2/(d_2+1)^3 \), where \( f(\lambda_1, \lambda_2) = \lambda_1 + \lambda_2 + \lambda_2^2 \) [38, Theorem 6.4]. Note that the degree of the replacement product graph depends only on the degree of the smaller component graph \( G_2 \).

![Figure 10.2. Replacement product of two graphs.](image)

We now consider the case when the two component graphs are Cayley graphs [40]. Suppose \( G_1 = C(G_a, S_a) \) is the Cayley graph formed from the group \( G_a \) with \( S_a \) as its generating set. This means that \( G_1 \) has the elements of \( G_a \) as vertices and there is an edge from the vertex representing \( g \in G_a \) to the vertex representing
$h \in G_a$ if for some $s \in S_a$, $g \ast s = h$, where `$\ast$' denotes the group operation. If the generating set $S_a$ is symmetric, i.e., if $a \in S_a$ implies $a^{-1} \in S_a$, then the Cayley graph is undirected.

Let the two components of our replacement product graph be Cayley graphs of the type $G_1 = C(G_a, S_a)$ and $G_2 = C(G_b, S_b)$ and further, let us assume that there is a well-defined group action by the group $G_b$ on the elements of the group $G_a$.

If $S_a$ is the union of $k$ orbits, i.e., the orbits of $a_1, a_2, \ldots, a_k \in G_a$ under the action of $G_b$, then the replacement product graph is the Cayley graph of the semi-direct product group $G_a \rtimes G_b$ and has $(1_{G_a}, S_b) \cup \{(a_1, 1_{G_a}), \ldots, (a_k, 1_{G_a})\}$ as the generating set. The degree of this Cayley graph is $|S_b| + k$ and the size of its vertex set is $|G_a||G_b|$ [32].

10.2.3 Zig-zag product

Let $G_1$ be a $(N_1, d_1, \lambda_1)$-graph and let $G_2$ be a $(d_1, d_2, \lambda_2)$ graph. Then the zig-zag product of $G_1$ and $G_2$, as introduced in [32, 38], is a graph $G$ defined as follows:

- the vertices of $G$ are represented as ordered pairs $(v, k)$, where $v \in \{1, 2, \ldots, N_1\}$ and $k \in \{1, 2, \ldots, d_1\}$. That is, every vertex in $G_1$ is replaced by a cloud of vertices of $G_2$.

- the edges of $G$ are formed by making two steps on the small graph and one step on the big graph as follows:

  - a step "zig" on the small graph $G_2$ is made from vertex $(v, k)$ to vertex $(v, k[i])$, where $k[i]$ denotes the $i^{th}$ neighbor of $k$ in $G_2$, for $i \in \{1, 2, \ldots, d_2\}$. 

157
a deterministic step on the large graph $G_1$ is made from vertex $(v, k[i])$ to vertex $(v[k[i]], k[i])$, where $v[k[i]]$ is the $k[i]^{th}$ neighbor of $v$ in $G_1$ and correspondingly, $v$ is the $k[i]^{th}$ neighbor of $v[k[i]]$ in $G_1$.

- a final step “zag” on the small graph $G_2$ is made from vertex $(v[k[i]], k[i])$ to vertex $(v[k[i]], k[i][j])$, where $k[i][j]$ is the $j^{th}$ neighbor of $k[i]$ in $G_2$, for $j \in \{1, 2, \ldots, d_2\}$.

Therefore, there is an edge between vertices $(v, k)$ and $(v[k[i]], k[i][j])$ for $i, j \in \{1, \ldots, d_2\}$.

It is shown in [38] that the zig-zag product graph $G = G_1 \Box G_2$ is a $(N_1 \cdot d_1, d_2^2, \lambda)$-graph with $\lambda < \lambda_1 + \lambda_2 + \lambda_2^2$, and further, that $\lambda < 1$ if $\lambda_1 < 1$ and $\lambda_2 < 1$.

Therefore, the degree of the zig-zag product graph depends only on the smaller component graph whereas the expansion property depends on the expansion of both the component graphs, i.e., it is a good expander if the two component graphs are good expanders.

As earlier, if we use Cayley graphs as the components for the product graph, then the product graph is again a Cayley graph. More specifically, if $G_1 = C(G_a, S_a)$ and $G_2 = C(G_b, S_b)$, and if $S_a$ is the orbit of $k$ elements $a_1, a_2, \ldots, a_k \in G_a$ under the action of $G_b$, then the generating set $S$ for the Cayley (zig-zag product) graph is

$$ S = \{(1_{G_a}, \beta)(a_i, 1_{G_b})(1_{G_a}, \beta')| \beta, \beta' \in S_b, i \in 1, \ldots, k\}. $$

It is easily verified that when $k = 1$, the Cayley graph $C(G_a \rtimes G_b, S)$ is the zig-zag product originally defined in [38]. The degree of this Cayley graph is at most $k|S_b|^2$ if we disallow multiple edges between vertices. When the group sizes
Figure 10.3. Zig-Zag product of two graphs.

$G_a$ and $G_b$ are large and the $k$ distinct elements $a_1, a_2, \ldots, a_k \in G_a$ are chosen randomly, then the degree of the product graph is almost always $k|S_b|^2$.

10.2.4 Generalized zig-zag product

Extending the original zig-zag construction in a straightforward manner, we are now able to define the zig-zag product construction for the case when the two component graphs are unbalanced bipartite graphs, i.e., the two sets of vertices have different degrees. Let $G_1$ be a $(c_1, d_1)$-regular graph on the vertex sets $V_1, W_1$, where $|V_1| = N$ and $|W_1| = M$. Let $G_2$ be a $(c_2, d_2)$-regular graph on the vertex sets $V_2, W_2$, where $|V_2| = d_1$ and $|W_2| = c_1$. Let $\lambda_1$ and $\lambda_2$ denote the second largest
eigenvalues of the normalized adjacency matrices of $G_1$ and $G_2$, respectively. Then the zig-zag product graph, which we also will denote by $G = G_1 \Box G_2$, is a $(c_2^2, d_2^2)$-regular bipartite graph on the vertex sets $V, W$ with $|V| = N \cdot d_1$, $|W| = M \cdot c_1$, formed in the following manner:

Figure 10.4. Zig-Zag product of two unbalanced bipartite graphs.

- Every vertex $v \in V_1$ and $w \in W_1$ of $G_1$ is replaced by a copy of $G_2$. The cloud at a vertex $v \in V_1$ has vertices $V_2$ on the left and vertices $W_2$ on the right, with each vertex from $W_2$ corresponding to an edge from $v$ in $G_1$. The cloud at a vertex $w \in W_1$ is similarly structured with each vertex in $V_2$ in the cloud corresponding to an edge of $w$ in $G_1$. (See Figure 10.4.) Then the vertices from $V$ are represented as ordered pairs $(v, k)$, for $v \in \{1, \ldots, N\}$.
and $k \in \{1, \ldots, d_1\}$, and the vertices from $W$ are represented as ordered pairs $(w, \ell)$, for $w \in \{1, \ldots, M\}$ and $\ell \in \{1, \ldots, c_1\}$.

- A vertex $(v, k) \in V$ is connected to a vertex in $W$ by making three steps in the product graph:
  
  - A small step “zig” from left to right in the local copy of $G_2$. This is a step $(v, k) \rightarrow (v, k[i])$, for $i \in \{1, \ldots, c_2\}$.
  
  - A deterministic step from left to right on $G_1$ $(v, k[i]) \rightarrow (v[k[i]], \ell)$, where $v[k[i]]$ is the $k[i]^{th}$ neighbor of $v$ in $G_1$ and $v$ is the $\ell^{th}$ neighbor of $v[k[i]]$ in $G_1$.
  
  - A small step “zag” from left to right in the local copy of $G_2$. This is a step $(v[k[i]], \ell) \rightarrow (v[k[i]], \ell[j])$, where the final vertex is in $W$, for $j \in \{1, \ldots, c_2\}$.

Therefore, there is an edge between $(v, k)$ and $(v[k[i]], \ell[j])$.

We will show that $\lambda(G_1 \boxtimes G_2) \leq \lambda_1 + \lambda_2 + \lambda_2^2$. The expansion of the unbalanced zig-zag product graph is similar to that of the original zig-zag product graph [38]. (The case of balanced bipartite graphs has been dealt in [32] and is a special case of this generalized variation.) Note that unlike in the original zig-zag product construction [38], the vertex set of $G$ does not include vertices from the set $W_2$ in any cloud of vertices from $V_1$, nor vertices from $V_2$ in any cloud of vertices from $W_1$. 
10.3 Properties of the Product Graphs

10.3.1 Degree

A small degree is desirable for designing LDPC codes over graphs. We have seen that the replacement product has a vertex degree that is of the same order as that of the smaller component graph, while the zig-zag product graph has a degree that is the square of the vertex degree of the smaller component graph. These parameters help determine our choice of graphs in the design of LDPC codes.

10.3.2 Expansion

The expansion coefficients in the previous section show that the replacement product graphs and zig-zag product graphs will be expanders if both of the component graphs are. The results in [38] show that the replacement product graph is not as good an expander as the zig-zag product graph. The following result that shows the zig-zag product of two unbalanced bipartite component graphs to be an expander if the component graphs are.

**Theorem 10.3.1** Let $G_1$ be a $(c_1, d_1)$-regular bipartite graph on $(N, M)$ vertices with $\lambda(G_1) = \lambda_1$, and let $G_2$ be a $(c_2, d_2)$-regular bipartite graph on $(d_1, c_1)$ vertices with $\lambda(G_2) = \lambda_2$. Then, the zig-zag product graph $G_1 \boxtimes G_2$ is a $(c_2^2, d_2^2)$-regular bipartite on $(N \cdot d_1, M \cdot c_1)$ vertices with $\lambda = \lambda(G_1 \boxtimes G_2) \leq \lambda_1 + \lambda_2 + \lambda_2^2$. Moreover, if $\lambda_1 < 1$ and $\lambda_2 < 1$, then $\lambda = \lambda(G_1 \boxtimes G_2) < 1$.

10.3.3 Diameter and girth

We now look at the diameter and girth of the graph products considered in section 10.2. Let us assume that the component graphs $G_1$ and $G_2$ have girths $g_1$
and \( g_2 \), respectively and diameters \( t_1 \) and \( t_2 \), respectively. We state the following results without proof as they are easy to verify:

**Lemma 10.3.1** The girth and diameter of the replacement product graph \( G = G_1 \otimes G_2 \) are given by: (a) girth \( \min\{g_2, g_1\} \leq g \leq \min\{g_2, g_1 + t_2\} \), and (b) diameter \( t \leq t_1 + t_2 \).

**Lemma 10.3.2** The girth and diameter of the zig-zag product graph \( G = G_1 \odot G_2 \) are given by: (a) girth \( g = 4 \), and (b) diameter \( t \leq t_1 + 2t_2 \).

10.3.4 Remarks

For designing codes over graphs, a graph with good expansion, relatively small degree, small diameter, and large girth, is desired. Despite their expansion, the standard graph products mentioned earlier are not good candidates for LDPC code construction. The resulting graphs have relatively large degree and therefore are not sparse – a drawback for graph based message passing decoding. In addition, each has girth \( g = 4 \). The replacement product graph, while having a relatively small degree and promising girth, has relatively inferior expansion. The zig-zag product graph falls somewhere in between – its degree is not too large and its expansion is not too bad. However, the girth of the zig-zag product graph is always four. This means that the codes based on the zig-zag product fare poorly when the size of the graph increases. (For any randomly designed graph, the girth grows almost logarithmically with the size of the graph.)

10.4 LDPC codes from Zig-zag Product Graphs

In this section, we use the zig-zag product graphs as building blocks for designing LDPC codes. The zig-zag product of regular graphs yields a regular graph
which may or may not be bipartite, depending on the choice of the component graphs. Therefore, to translate the zig-zag product graph into a LDPC code, the vertices of the zig-zag product are interpreted as sub-code constraints of a suitable linear block code and the edges are interpreted as code bits of the LDPC code. This is akin to the procedure described in [48] and [31].

We further restrict the choice of the component graphs for our zig-zag product to be appropriate Cayley graphs so that we can work directly with the group structure of the Cayley graphs. The following examples illustrate the code construction technique:

**Example 10.4.1** [3] Let $A = \mathbb{F}_2^p$ be the Galois field of $2^p$ elements for a prime $p$, where the elements of $A$ are represented as vectors of a $p$-dimensional vector space over $\mathbb{F}_2$. Let $B = \mathbb{Z}_p$ be the group of integers modulo $p$. (Further, let $p$ be chosen such that the element 2 generates the multiplicative group $\mathbb{Z}_p^* = \mathbb{Z}_p - \{0\}$.) The group $B$ acts on an element $x = (x_0, x_1, \ldots, x_{p-1}) \in A$ by cyclically shifting its coordinates, i.e. $\phi_b(x) = (x_b, x_{b+1}, \ldots, x_{b-1})$, $\forall b \in B$. Let us now choose $k$ elements $a_1, a_2, \ldots, a_k$ randomly from $A$. The result in [3, Theorem 3.6] says that for a random choice of elements $a_1, a_2, \ldots, a_k$, the Cayley graph $C(A, \{a_1^B, a_2^B, \ldots, a_k^B\})$ is an expander with high probability. (Here, $a_i^B$ is the orbit of $a_i$ under the action of $B$.) The Cayley graph for the group $B$ with the generators $\{\pm 1\}$ is the cyclic graph on $p$ vertices, $C(B, \{\pm 1\})$. The zig-zag product of the two Cayley graphs is the Cayley graph $C(A \rtimes B, S = \{(0, \beta)(a_i, 0), (0, \beta')| \beta, \beta' = \pm 1, i = 1, 2, \ldots, k\})$ on $N = 2^p \cdot p$ vertices, where $A \rtimes B$ is the semi-direct product group and the group operation is $(a, b)(c, d) = (a + \phi_b(c), b + d)$, for $a, c \in A$, $b, d \in B$. This is a regular graph with degree\(^1\ $d = k|S_B| = 4k$. If we interpret

\(^1\)Depending on the choice of the $a_i$'s, the number of distinct elements in $S$ may be fewer than $k|S_B|^2$.  

164
the vertices of the graph as sub-code constraints of a \([d_g, k_g, d_m]\) linear block code and the edges of the graph as code bits of the LDPC code, then the block length \(N_{LD}\) of the LDPC code is \(2^p \cdot p \cdot d_g/2\) and the rate of the LDPC code is \(r \geq \frac{N_{LD} - N(d_g - k_g)}{N_{LD}} = 1 - \frac{2(d_g - k_g)}{d_g} = \frac{2k_g}{d_g} - 1\). (Observe that \(r \geq 2r_1 - 1\), where \(r_1\) is the rate of the sub-code.)

In some cases, to achieve a certain desired rate, we may have to use a mixture of sub-code constraints from two or more linear block codes. For example, to design a rate 1/2 LDPC code when \(d_g\) is odd, we may have to impose a combination of \([d_g, k_g, d_m1]\) and \([d_g, k_g + 1, d_m2]\) block code constraints, for an appropriate \(k_g\), on the vertices of the graph.

Another entirely different approach for the design of asymptotically good codes is suggested in [3]. If the elements \(a_1, \ldots, a_k\) are appropriately chosen and the group \(B\) acts on each of them producing an orbit of elements for each \(a_i\), then these vectors could be arranged as rows of a circulant matrix (For the example above, the set of elements obtained when \(B\) acts on \(a_i\) is all cyclic shifts of \(a_i\)). Hence, a block of \(k\) circulant matrices \([C_1, C_2, \ldots, C_k]\) is obtained from the action of \(B\) on \(a_1, a_2, \ldots, a_k\). The authors in [3] establish that the result of multiplying any non-zero vector \(x \in \mathbb{F}_2^p\) with \([C_1, \ldots, C_k]\) is a vector \(c\) that contains at least \(\delta\) fraction of ones and \(\delta\) fraction of zeros for some \(\delta > 0\). This is a probabilistic result that holds with high probability when the choice of \(a_i\)'s is completely random. Suppose we consider the block of circulants as the generator matrix \(G_{en}\) of a binary linear block code, then the result in [3] says for any non-zero message vector \(x, c = xG_{en}\) has a relative Hamming weight of at least \(\delta\), meaning the relative minimum distance of the code described by \(G_{en}\) is at least \(\delta > 0\). Therefore, by selecting a large prime \(p\), we can get a long block length binary linear code of
rate $1/k$ with minimum distance linear in the block length of the code. However, it is not clear at this point whether it is possible to obtain a sparse parity check matrix representation for such a code. A sparse parity check representation would then guarantee efficient graph based iterative decoding with the benefit of good minimum distance imposed by the construction.

Example 10.4.2 [3] Let $B = SL_2(\mathbb{F}_p)$ be the group of all $2 \times 2$ matrices over $\mathbb{F}_p$ with determinant one. Let $S_B = \left\{ \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \right\}$ be the generating set for the Cayley graph $C(B, S_B)$. Further, let $\mathbb{P}_1 = \mathbb{F}_p \cup \{\infty\}$ be the projective line. The Möbius action of $B$ on $\mathbb{P}_1$ is given by $\begin{pmatrix} a & b \\ c & d \end{pmatrix} (x) = \frac{ax + b}{cx + d}$. Let $A = \mathbb{F}_2^{p+1}$ and let the action of $B$ on the elements of $A$ be the Möbius permutation of the coordinates as above. If we now choose $k$ elements $a_1, a_2, \ldots, a_k$ randomly from $A$ as in the previous example, then with high probability the Cayley graph $C(A, \{a_1^B, \ldots, a_k^B\})$ is an expander. Further, the zig-zag product of the two Cayley graphs is the Cayley graph $C(A \times B, S)$ (as in Example 10.4.1) on $|A||B| = 2^{p+1}(p^3 - p)$ vertices. However, this Cayley graph will be a directed Cayley graph since the generating set $S$ is not symmetric. Hence, we modify our graph construction a little by taking two copies of the vertex set $A \times B$. A vertex $v$ from one copy is connected to vertex $w$ in the other copy if there is a $s \in S$ such that $v * s = w$. The new product graph obtained has $2|A||B|$ vertices and every vertex has degree $d_g = |S|$; moreover, it is a balanced bipartite graph. An LDPC code of block length $|A||B|d_g$ is obtained by interpreting the vertices of the graph as sub-code constraints of a $[d_g, k_g, d_m]$ linear block code, and the edges as code bits of the LDPC code. The rate of this code is $r \geq 1 - \frac{2(d_g - k_g)}{d_g} = \frac{2k_g}{d_g} - 1$. 

166
Example 10.4.3 Codes from unbalanced bipartite zig-zag product graphs.

Using a random construction, we design a \((c_1, d_1)\)-regular bipartite graph \(G_1\) on \((N, M)\) vertices. Similarly, we design a \((c_2, d_2)\)-regular bipartite graph \(G_2\) on \((d_1, c_1)\) vertices. The zig-zag product of \(G_1\) and \(G_2\) is a \((c_2^2, d_2^2)\)-regular graph on \((N \cdot d_1, M \cdot c_1)\) vertices. An LDPC code is obtained as before by interpreting the degree \(c_2^2\) vertices [resp. degree \(d_2^2\) vertices] as sub-code constraints of a \(C_{S1} = [c_2^2, k_1, d_m]\) [resp. a \(C_{S2} = [d_2^2, k_2, d_m]\)] linear block code and the edges of the product graph as code bits of the LDPC code. The block length of the LDPC code thus obtained is \(N_{LD} = Nd_1 c_2^2\) and the rate is \(r \geq \frac{Nd_1 c_2^2 - (Nd_1 (c_2^2 - k_1) + Mc_1 (d_2^2 - k_2))}{Nd_1 c_2^2} = \frac{k_1}{c_2^2} + \frac{k_2}{d_2^2} - 1\) (since \(Nd_1 c_2^2 = Mc_1 d_2^2\) is the number of edges in the graph). Observe that \(r \geq r_1 + r_2 - 1\), where \(r_1\) and \(r_2\) are the rates of the two sub-codes \(C_{S1}\) and \(C_{S2}\), respectively.

10.5 Existing Minimum Distance Bounds

Sipser and Spielman in [43] and Janwa and Lal [24] lower bound the minimum distance of codes based on expander graphs. Let us assume that the vertices of a \(d\)-regular graph are interpreted as sub-code constraints of a \([d, k, \epsilon d]\) linear block code and the edges are interpreted as the code bits of the LDPC code as above. If the graph is an expander with eigenvalue \(\lambda(G) = \lambda\) and if \(\epsilon > \lambda\), then the relative minimum distance\(^2\) is at least \(\epsilon \frac{(\epsilon - \lambda)}{(1 - \lambda)}\). This is a decreasing function in \(\lambda\) for \(0 \leq \lambda \leq \epsilon\). Therefore, for a LDPC code based on the \((N \cdot d_1, d_2^2, \lambda < f(\lambda_1, \lambda_2))\)-zig-zag product graph, if the sub-code constraint used is of a \([d_2^2, k, \epsilon d_2^2]\) linear block

\(^2\)i.e., relative to the block length of the LDPC code.
code, then the relative minimum distance of the LDPC code is lower bounded by

\[ d_{\text{min}} \geq \epsilon \frac{(\epsilon - f(\lambda_1, \lambda_2))}{(1 - f(\lambda_1, \lambda_2))}. \]

Janwa and Lal extend this argument to the unbalanced bipartite case. Let the two sets of vertices of a \((c, d)\)-regular bipartite graph on the vertex sets \(V_1 (|V_1| = N)\) and \(W_1 (|W_1| = M)\) be interpreted as sub-code constraints of a \([c, k_1, \epsilon_1 c]\) and a \([d, k_2, \epsilon_2 d]\) linear block codes, respectively, and the edges be interpreted as code bits of a LDPC code. If the bipartite graph \(G\) has \(\lambda(G) = \lambda\) and if \(d\epsilon_2 \geq c\epsilon_1 > \lambda\sqrt{cd}/2\), then the relative minimum distance of the LDPC code is at least

\[ d_{\text{min}} \geq \{\epsilon_1 \epsilon_2 - \frac{\lambda}{2}(\epsilon_1 \sqrt{\frac{c}{d}} + \epsilon_2 \sqrt{\frac{d}{c}})\}. \]

Using Lemma 4.1 and the above result, the relative minimum distance of LDPC codes based on Example 10.4.3 can be lower bounded by

\[ d_{\text{min}} \geq \{\epsilon_1 \epsilon_2 - \frac{\lambda(G_1 \bar{\otimes} G_2)}{2}(\epsilon_1 \frac{c_2}{d_2} + \epsilon_2 \frac{d_2}{c_2})\}. \]

(Here, we use sub-codes \([c_2^2, k_1, \epsilon_1 c_2^2]\), \([d_2^2, k_2, \epsilon_2 d_2^2]\) and assume that \(d_2^2 \epsilon_2 \geq c_2^2 \epsilon_1 > \frac{\lambda(G_1 \bar{\otimes} G_2)}{2}\).)

10.6 Performance of Zig-zag LDPC Codes

The performance of the LDPC code designs based on zig-zag product graphs is examined for use over the additive white Gaussian noise (AWGN) channel. (Binary modulation is simulated and the bit error performance with respect to signal to noise ratio (SNR) \(E_b/N_o\) is determined.) The LDPC codes are decoded using
the graph based iterative belief propagation (BP) algorithm. Since the LDPC codes based on the zig-zag product graph use sub-code constraints, the decoding at the constraint nodes is accomplished using the BCJR algorithm on a trellis representation of the appropriate sub-code. (A simple procedure to obtain the trellis representation of the sub-code based on its parity check matrix representation is discussed in [52].) It must be noted that as the number of states in the trellis representation and the block length of the sub-code increases, the decoding complexity correspondingly increases.

Figure 10.5 shows the performance with BP decoding of a LDPC code that is designed based on Example 10.4.1. For the parameters \( p = 5 \) and \( k = 5 \), five elements in \( A = \mathbb{F}_2^p \) are chosen (randomly) to yield a set of generators for the Cayley graph of the semi-direct product group. The Cayley graph has 160 vertices, each of degree 20. The sub-code used in this design is a \([20, 15, 4]\) code and the resulting LDPC code has rate 1/2 and block length 1600. The figure also shows the performance of a LDPC code based on a randomly designed degree 20 regular graph on 160 vertices which also uses the same sub-code constraints as the former code. The two codes perform comparably, indicating that the expansion of the zig-zag product code compares well with that of a random graph of similar size and degree. Also shown in the figure is the performance of a \((3,6)\) regular LDPC code, that uses no special sub-code constraints other than simple parity check constraints, having the same block length and rate. Clearly, using strong sub-code constraints improves the performance significantly, albeit at the cost of higher decoding complexity. The figure also shows another set of curves for a longer block length design. Choosing \( p = 11 \) and \( k = 5 \) and the \([20, 15, 4]\) sub-code constraints yields a rate 1/2 and block length 225,280 LDPC code. At
Figure 10.5. LDPC codes from zig-zag product graphs based on Example 10.4.1.

This block length also, the LDPC based on the zig-zag product graph is found to perform comparably, if not, better than the LDPC code based on a random degree 20 graph. The zig-zag product graph has a poor girth and this causes the performance of the zigzag LDPC code to be inferior to that of the random LDPC codes at high signal to noise ratios.

Figure 10.6 shows the performance of LDPC codes that are designed based on Example 10.4.2. Once again, this performance is compared with the analogous performance of a LDPC code based on a random graph using identical sub-code constraints and having the same block length and rate. These results are also compared with a (3, 6) regular LDPC code that uses simple parity check constraints. For the parameters $p = 3$ and $k = 5$ in Example 10.4.2, a bipartite graph, based on the zig-zag product graph, on 768 vertices with degree 20 is obtained. Using

\[ \text{Note that there is no growth in the girth of the zig-zag product graph as opposed to that for a randomly chosen graph, with increasing graph size.} \]
the [20, 15, 4] sub-code constraints as earlier, a block length 7680 rate 1/2 LDPC code is obtained. This code performs comparably with the random LDPC code that is based on a degree 20 randomly designed graph. Using the parameters \( p = 5 \) and \( k = 4 \) and a \([16, 12, 2]\) sub-code, a longer block length 122880 LDPC code is obtained. As in the previous case, this code also performs comparably, if not, better than its random counterpart for low to medium signal-to-noise ratios (SNRs). Once again, we attribute its slightly inferior performance at high SNRs to the poor girth of the zig-zag product graph.

Figure 10.7 shows the performance of LDPC codes designed based on the zig-zag product of two unbalanced bipartite graphs as in Example 10.4.3. A \((6, 10)\)-regular bipartite graph on \((20, 12)\) vertices is chosen as one of the component graphs and a \((3, 5)\)-regular bipartite graph on \((10, 6)\) vertices is chosen as the other component. Their zig-zag product is a \((9, 25)\)-regular bipartite graph on \((200, 72)\) vertices. Using sub-code constraints of two codes – a \([9, 6, 2]\) and a
Figure 10.7. Performance of LDPC codes from zig-zag product graphs based on Example 10.4.3.

[25, 21, 2] linear block code – a block length 1800 LDPC code of rate 0.5066 is obtained. The performance of this code is compared with a LDPC code based on a random (9, 25)-regular bipartite graph using the same sub-code constraints, and also with a block length 1800 random (3, 6) regular LDPC code. All three codes perform comparably, with the random (3, 6) showing a small improvement over others at high SNRs. Given that the zigzag product graph is composed of two very small graphs, this result highlights the fact that good graphs may be designed using just simple component graphs.

10.7 Conclusions

In this chapter we discussed the properties of the replacement product and the zig-zag product, and investigated their potential for LDPC code constructions. Included in this was a straightforward generalization of the zig-zag product for the case of unbalanced bipartite graphs. Our preliminary results indicate that
LDPC codes based on the zig-zag product of two good component graphs yields decent performance compared to random LDPC codes. However, the performance of the zigzag-product-based LDPC codes are slightly inferior to that of random LDPC codes at high SNRs due to the poor girth and cycle distribution of the zig-zag product graph. A modification in the zig-zag product steps may be necessary to alleviate this problem. The code design may be improved by a more judicious choice of component graphs. These construction techniques may also be applied to the replacement product graphs. Given that the replacement product graph has better girth than the zig-zag product, we suspect that LDPC codes based on the replacement product would perform better.

10.8 Proofs

*Theorem 10.3.1*  
*Proof:* Let $M_G$ denote the adjacency matrix of $G$. For convenience, we also let $G_2$ denote the $d_1 \times c_1$ matrix that describes the connections between the nodes in $V_2$ to the nodes in $W_2$ for the graph $G_2$, and $G_1$ denote the $N \times M$ matrix that describes the connections between the nodes in $V_1$ and the nodes in $W_1$ for the graph $G_1$. This means that the adjacency matrix for the graph $G_2$ is given by $M_2 = \begin{bmatrix} 0 & G_2 \\ G_2^T & 0 \end{bmatrix}$, and the adjacency matrix for the graph $G_1$ is given by $M_1 = \begin{bmatrix} 0 & G_1 \\ G_1^T & 0 \end{bmatrix}$.

The adjacency matrix for the zig-zag product graph $G$ is given by

$$M_G = \begin{bmatrix} 0 & (G_2 \otimes I_n)\tilde{A}_2(G_2^T \otimes I_m) \\ (G_2^T \otimes I_m)\tilde{A}_2(G_2 \otimes I_n) & 0 \end{bmatrix},$$

173
where $\tilde{A}_2$ is a permutation matrix of size $Nc_1 \times Nc_1$ that describes the zig-zag product connections.

The largest eigenvalue of $M_G$ is $c_2d_2$ and the corresponding eigenvector is $v_0 = [1 \cdots 1 \ r \cdots r]^T$, where the first $Nd_1$ components are equal to 1 and the remaining $Mc_1$ components are equal to $r = \frac{d_2}{c_2} = \frac{d_1}{c_1}$.

Let $1_x$ denote a column vector of length $x$ with all entries equal to 1. Then, the largest eigenvalue of $M_1$ is $\sqrt{c_1d_1}$ and the corresponding eigenvector is $w_0 = \begin{bmatrix} 1_N \\ r_11_M \end{bmatrix}$, where $r_1 = \sqrt{\frac{d_1}{c_1}} = \sqrt{r}$. Similarly, the largest eigenvalue of $M_2$ is $\sqrt{c_2d_2}$ and the corresponding eigenvector is $u_0 = \begin{bmatrix} 1_{d_1} \\ r_21_{c_1} \end{bmatrix}$, where $r_2 = \sqrt{\frac{d_2}{c_2}} = r_1 = \sqrt{r}$.

Let $\lambda_2 = \max_{u \perp u_0} \frac{\langle M_2u, u \rangle}{\langle u, u \rangle} = \max_{u \perp u_0} \frac{2(u^T G_2 u_0)}{\|u_0\|^2 + \|u\|^2}$, where $u = \begin{bmatrix} u_a \\ u_b \end{bmatrix}$. That is $u_a$ is a column vector of length $d_1$ corresponding to the vertices in $V_2$ and $u_b$ is a column vector of length $c_1$ corresponding to the vertices in $W_2$. We choose $u$ such that $u \perp u_0$. Furthermore, $u$ can be written as two vectors $u^\parallel$ and $u^\perp$ where $u^\parallel$ is a vector that is parallel to the constant (non-zero) vector and $u^\perp$ is a vector that is perpendicular (or, orthogonal) to the constant (non-zero) vector. That is $u = u^\parallel + u^\perp$.

(Note that, by definition, $\lambda_2$ corresponds to the second largest eigenvalue of $M_2$ and the corresponding eigenvector $u$ that maximizes $\lambda_2$ in the above is orthogonal to the $u_0$, the eigenvector corresponding to the eigenvalue $\sqrt{c_2d_2}$.)

Similarly, let $\lambda_1 = \max_{w \perp w_0} \frac{\langle M_1w, w \rangle}{\langle w, w \rangle} = \max_{w \perp w_0} \frac{2(w^T G_1 w_0)}{\|w_0\|^2 + \|w\|^2}$, where $w = \begin{bmatrix} w_a \\ w_b \end{bmatrix}$. (Here, $w_a$ is a column vector of length $N$ and $w_b$ is a column vector of length $M$. $w$ can also be broken down as $w = w^\parallel + w^\perp$ as above. By definition,
\( \lambda_1 \) is the second largest eigenvalue of \( M_1 \).

The eigenvector of \( M_G \) corresponding to the largest eigenvalue \( c_2 d_2 \) is \( v_0 = \begin{bmatrix} 1_{Nd_1} \\ r1_{Mc_1} \end{bmatrix} \). Let \( \alpha = \begin{bmatrix} \alpha_a \\ \alpha_b \end{bmatrix} \) be an eigenvector of \( M_G \), where \( \alpha_a \) has length \( Nd_1 \) and \( \alpha_2 \) has length \( Mc_1 \). Let \( e_m \) be a basis vector with component value 1 at the \( m \)th entry and component value 0 elsewhere. Then, the vectors \( \alpha_a \) and \( \alpha_b \) can be written as \( \alpha_a = \sum_{n \in [N]} \alpha_{an} \otimes e_n \), and \( \alpha_b = \sum_{m \in [M]} \alpha_{bm} \otimes e_m \), where \( \otimes \) denotes the Kronecker product, \( \alpha_{an} \) is the vector \( \alpha_a \) restricted to the components corresponding to the vertices in the \( n \)th vertex cloud, \( n \in [N] \), of the graph \( G \), and \( \alpha_{bm} \), for \( m \in [M] \), is defined similarly.

Then the second largest eigenvalue of \( M_G \) is \( \lambda = \max_{v \perp v_0} \frac{2 \alpha_a^T (K) \alpha_b}{\langle \alpha, \alpha \rangle} \), where \( K = (G_2 \otimes I_N) \tilde{A}_2 (G_2 \otimes I_M) \).

Let \( s = \left| 2 (\alpha_a^T K \alpha_b) \right| \). Splitting \( \alpha_{an} \) (and, \( \alpha_{bm} \)) into parallel and perpendicular parts, \( \alpha_{an} = \alpha_{an}^\parallel + \alpha_{an}^\perp \), we can write

\[
\begin{align*}
\quad s &= 2 \left( \sum_{n \in [N]} (\alpha_{an}^\parallel^T G_2 \otimes e_n) + \sum_{n \in [N]} (\alpha_{an}^\perp^T G_2 \otimes e_m) \right) \tilde{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{bm}^\parallel \otimes e_m + \sum_{m \in [M]} g_2 \alpha_{bm}^\perp \otimes e_m \right)
\end{align*}
\]

We want to show that \( s \leq f(\lambda_1, \lambda_2)(\| \alpha_a \|^2 + \| \alpha_b \|^2) \), where \( f(\lambda_1, \lambda_2) \) is some positive-valued function such that \( f(\lambda_1, \lambda_2) \leq \lambda_1 + \lambda_2 + \lambda_2^2 \). Note that \( \alpha_a \in \mathbb{R}^{Nd_1} \), and \( \alpha_b \in \mathbb{R}^{Mc_1} \).

Observe the following:

1. \( \lambda_2 = \max_{u \perp u} \frac{2 u_a^T g_2 u_b}{\| u_a \|^2 + \| u_b \|^2} \), where \( u = \begin{bmatrix} u_a \\ u_b \end{bmatrix} \).
2. $\lambda_1 = \max_{u \perp 1_N} \|\mathbf{w}_a\|_2^2 + \|\mathbf{w}_b\|_2^2$, where $w = \begin{bmatrix} \mathbf{w}_a \\ \mathbf{w}_b \end{bmatrix}$.

3. $\alpha_a = \begin{bmatrix} \alpha_{a1} \\ \vdots \\ \alpha_{aN} \end{bmatrix}$, where $\alpha_{ai} \in \mathbb{R}^{d_i} i \in [N]$, $\alpha_b = \begin{bmatrix} \alpha_{b1} \\ \vdots \\ \alpha_{bM} \end{bmatrix}$, where $\alpha_{bi} \in \mathbb{R}^{c_i}$, $i \in [M]$, and $\alpha = \begin{bmatrix} \alpha_a \\ \alpha_b \end{bmatrix} \in \mathbb{R}^{N_{d_1} + M_{c_1}}$.

4. From the definition of $\lambda_2$, we have

$$\| \begin{bmatrix} 0 & G_2 \\ G_2^T & 0 \end{bmatrix} \begin{bmatrix} \alpha_{a_n} \\ 0 \end{bmatrix} \| \leq \lambda_2 \| \begin{bmatrix} \alpha_{a_n} \\ 0 \end{bmatrix} \|.$$ 

$$\Rightarrow \| G_2^T \alpha_{a_n} \| \leq \lambda_2 \| \alpha_{a_n} \| \text{ and } \| G_2^T \alpha_{a_n} \| \leq \lambda_2 \| \alpha_{a_n} \|.$$ 

This implies that $\begin{bmatrix} \alpha_{a_n} \\ 0 \end{bmatrix} \perp \begin{bmatrix} 1_{d_1} \\ \sqrt{T_{c_1}} \end{bmatrix}$.

5. Since $\alpha_{a_n}^\perp$ is orthogonal to the constant vector, we have $\alpha_{a_n}^\perp \perp 1_{d_1}$.

Rewriting, we have that

$$s = 2 \left( \sum_{n \in [N]} (\alpha_{a_n}^\perp)^T G_2 \otimes e_n \right) \hat{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{b_m}^\perp \otimes e_m \right) + 2 \left( \sum_{n \in [N]} (\alpha_{a_n}^\perp)^T G_2 \otimes e_n \right) \hat{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{b_m}^\perp \otimes e_m \right) + 2 \left( \sum_{n \in [N]} (\alpha_{a_n}^\perp)^T G_2 \otimes e_n \right) \hat{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{b_m}^\perp \otimes e_m \right).$$
So $s = s_1 + s_2 + s_3 + s_4$, where

$$s_1 = 2 \left( \sum_{n \in [N]} (\alpha_{\alpha_n}^\|)^T G_2 \otimes e_n \right) \tilde{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{\beta_m}^\| \otimes e_m \right)$$

$$s_2 = 2 \left( \sum_{n \in [N]} (\alpha_{\alpha_n}^\|)^T G_2 \otimes e_n \right) \tilde{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{\beta_m}^\perp \otimes e_m \right)$$

$$s_3 = 2 \left( \sum_{n \in [N]} (\alpha_{\alpha_n}^\perp)^T G_2 \otimes e_n \right) \tilde{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{\beta_m}^\| \otimes e_m \right), \text{ and}$$

$$s_4 = 2 \left( \sum_{n \in [N]} (\alpha_{\alpha_n}^\perp)^T G_2 \otimes e_n \right) \tilde{A}_2 \left( \sum_{m \in [M]} G_2 \alpha_{\beta_m}^\perp \otimes e_m \right).$$

We will bound each part of $s$ separately:

1. Since $\tilde{A}_2$ is a permutation matrix, we have

$$s_4 \leq 2 \left\| \sum_{n \in [N]} (\alpha_{\alpha_n}^\perp)^T G_2 \otimes e_n \right\| \left\| \sum_{m \in [M]} G_2 \alpha_{\beta_m}^\perp \otimes e_m \right\|$$

Since $\| (\alpha_{\alpha_n}^\perp)^T G_2 \| \leq \lambda_2 \| \alpha_{\alpha_n}^\perp \|$ by definition of $\lambda_2$ (and similarly, $\| G_2 \alpha_{\beta_m}^\perp \| \leq \lambda_2 \| \alpha_{\alpha_n}^\perp \|$), we have

$$s_4 \leq 2 \left\| \sum_{n \in [N]} \lambda_2 \alpha_{\alpha_n}^\perp \otimes e_n \right\| \lambda_2 \left\| \sum_{m \in [M]} \alpha_{\beta_m}^\perp \otimes e_m \right\|$$

$$= 2\lambda_2^2 \| \alpha_{\alpha_n}^\perp \| \| \alpha_{\beta_m}^\perp \| \leq \lambda_2^2 (\| \alpha_{\alpha_n}^\perp \|^2 + \| \alpha_{\beta_m}^\perp \|^2) = \lambda_2^2 (\| \alpha^\perp \|^2)$$

2. Since $\tilde{A}_2$ is a permutation matrix, we have

$$s_3 \leq 2 \left\| \sum_{n \in [N]} (\alpha_{\alpha_n}^\perp)^T G_2 \otimes e_n \right\| \left\| \sum_{m \in [M]} G_2 \alpha_{\beta_m}^\| \otimes e_m \right\|$$
Using the argument from the previous step and since $\| G_2 \alpha_{b_m} \| \leq \| \alpha_{b_m} \|$, we have

$$s_3 \leq 2\lambda_2 \| \alpha_{a'}^1 \| \| \alpha_{b}^1 \|$$

3. Similarly, we can show that

$$s_2 \leq 2\lambda_2 \| \alpha_{a'}^2 \| \| \alpha_{b}^1 \|$$

4. To upper bound $s_1$, define a new vector $C'(\alpha'_a)$ for every vector $\alpha'_a$ such that its $m$th component is

$$(C'(\alpha'_a)_m \ := \frac{1}{d_1} \sum_{a=1}^{d_1} \alpha'_{a_m}, \text{ for } m \in [M]$$

This implies that $\alpha' = C(\alpha'_a) \otimes \frac{d_1}{d_4}$.

Similarly, define a new vector $C'(\alpha'_b)$ for every $\alpha'_b$ as

$$(C'(\alpha'_b)_n \ := \frac{1}{c_1} \sum_{b=1}^{c_1} \alpha'_{b_n}, \text{ for } n \in [N]$$

This implies that $\alpha' = C(\alpha'_b) \otimes \frac{c_1}{c_4}$.

That is, the functions $C(\cdot)$ and $C'(\cdot)$ computes the average value of the components in each vertex cloud of the zig-zag product graph $G$.

Therefore, we have $C' \tilde{A}_2 (e_m \otimes \frac{1d_1}{d_4}) = G_1 e_m$. Note that $\alpha_{b_m}^1 G_2 = \alpha_{a_m}^1$, where $\tilde{a}_m$ in the subscript refers to the left vertices on the right of the zig-zag product graph that are used for the construction but do not belong to the vertex set of the zig-zag product graph. Rewriting $s_1$, we have
\[ s_1 = 2(\sum_{n \in [N]} (\alpha_{an}^\parallel)^T G_2 \otimes e_n)^T \tilde{A}_2(\sum_{m \in [M]} \alpha_{bm}^\parallel G_2 \otimes e_m) = 2(\sum_{n \in [N]} \alpha_{bn}^\parallel \otimes e_n)^T \tilde{A}_2(\sum_{m \in [M]} \alpha_{am}^\parallel \otimes e_m). \]

This is because, \((\alpha_{an}^\parallel)^T G_2 = \alpha_{bn}^\parallel\), the components of the right vertices of the \(G_2\) clouds on the left of \(G\), and \(G_2(\alpha_{bn}^\parallel) = \alpha_{an}^\parallel\), the components corresponding to the left vertices of the \(G_2\) clouds on the right of \(G\). (That is, since \(G_2\) denotes the connections between the left vertices and right vertices, multiplying with \(G_2\) takes \((\alpha_{an}^\parallel)^T\) to \((\alpha_{bn}^\parallel)\) and \((\alpha_{bn}^\parallel)\) to \((\alpha_{an}^\parallel)\).

But \(\alpha_a = (C(\alpha_a)) \otimes \frac{1d_1}{c_1}\), \(\alpha_b = (C'(\alpha_b)) \otimes \frac{1c_1}{d_1}\). Hence,

\[ s_1 = 2(C'(\alpha_b) \otimes \frac{1c_1}{c_1})^T \tilde{A}_2(C(\alpha_a) \otimes \frac{1d_1}{d_1}) \]

\[ \Rightarrow s_1 = 2(C(\alpha_b))^T G_1(C(\alpha_a))/(c_1d_1) = \frac{2(\sqrt{c_1} C'(\alpha_b))^T G_1(\sqrt{d_1} C(\alpha_a))}{\sqrt{c_1d_1}} \]

But observe that \[
\begin{bmatrix}
\frac{1}{\sqrt{c_1}} C'(\alpha_b) \\
\frac{1}{\sqrt{d_1}} C(\alpha_a)
\end{bmatrix}
\] is orthogonal to the vector \[
\begin{bmatrix}
1_N \\
\sqrt{r} 1_M
\end{bmatrix}
\].

This is because

\[
< \begin{bmatrix}
\frac{1}{\sqrt{c_1}} C'(\alpha_b) \\
\frac{1}{\sqrt{d_1}} C(\alpha_a)
\end{bmatrix} ; \begin{bmatrix}
1_N \\
\sqrt{r} 1_M
\end{bmatrix} > = \frac{1}{\sqrt{c_1}} \sum_{b=1}^{c_1} \sum_{n=1}^{N} \alpha_{bn} + \frac{1}{\sqrt{d_1}} \sqrt{r} 1_M \sum_{a=1}^{M} \sum_{m=1}^{d_1} \alpha_{am} = (**) \]

However, \(\sum_{b=1}^{c_1} \sum_{n=1}^{N} \alpha_{bn} = c_2 \sum_{a=1}^{d_1} \sum_{n=1}^{N} \alpha_{an}\) and

\(\sum_{a=1}^{d_1} \sum_{m=1}^{M} \alpha_{am} = d_2 \sum_{b=1}^{c_1} \sum_{m=1}^{M} \alpha_{bn}\).

Since \[
\begin{bmatrix}
\alpha_a \\
\alpha_b
\end{bmatrix}
\] was chosen to be or-
thogonal to \( \begin{bmatrix} 1_{Nd_1} \\ r 1_{Mc_1} \end{bmatrix} \), it is easy to verify that the sum in \( (*) \) is zero.

Hence, from the definition of \( \lambda_1 \), we have

\[
s_1 = 2 \left( \frac{1}{\sqrt{c_1}} C'(\alpha_b) \right)^T G_1 \left( \frac{1}{\sqrt{d_1}} C(\alpha_a) \right) \leq \frac{\lambda_1}{c_1 d_1} \left( \frac{1}{c_1} \| C'(\alpha_b) \|^2 + \frac{1}{d_1} \| C(\alpha_a) \|^2 \right) \quad (*)
\]

It is easy to verify that the RHS in \( (*) \) can be upper bounded as

\[
s_1 \leq RHS(*) \leq \lambda_1 (\| \alpha_a \|^2 + \| \alpha_b \|^2) = \lambda_1 (\| \alpha \|^2)
\]

Combining the upper bounds on \( s_1, s_2, s_3, s_4 \), we have

\[
s = s_1 + s_2 + s_3 + s_4
\]

Thus, we have

\[
s \leq \lambda_1 (\| \alpha \|^2) + 2 \lambda_2 (\| \alpha_a \|^2 + \| \alpha_b \|^2) + \lambda_2^2 (\| \alpha \|^2)
\]

However, observe that

\[
2 \lambda_2 (\| \alpha_a \|^2 + \| \alpha_b \|^2) \leq \lambda_2 (\| \alpha \|^2)
\]

Further, \( \| \alpha \|^2 \leq \| \alpha_a \|^2 \) and \( \| \alpha \|^2 \leq \| \alpha_b \|^2 \).

Thus, we have

\[
s \leq (\lambda_1 + \lambda_2 + \lambda_2^2)(\| \alpha \|^2)
\]

The second largest eigenvalue of \( M_G \) is defined as \( \lambda = \max_{\alpha \perp w} \frac{\alpha^T}{\langle \alpha, \alpha \rangle} \), where \( s = 2 \alpha_a^T (G_2 \otimes I_N) \tilde{A}_2 (G_2 \otimes I_M) \alpha_b \). Using the upper bound on \( s \), we get \( \lambda \leq \)
\[ \frac{(\lambda_1 + \lambda_2 + \lambda_2^2)(\|\alpha\|^2)}{\|\alpha\|^2} = \lambda_1 + \lambda_2 + \lambda_2^2. \]

The only remaining step is to show that if \( \lambda_1 < 1 \) and \( \lambda_2 < 1 \), then \( \lambda < 1 \). Suppose \( \lambda_1 < 1, \lambda_2 < 1 \) and suppose \( \| \alpha^\perp \| \leq \frac{1-\lambda_1}{3\lambda_2} \| \alpha \| \). Then, we can upper bound \( s \) as follows

\[
s \leq \lambda_1 \| \alpha^\parallel \|^2 + 2\lambda_2 \| \alpha^\parallel \| \| \alpha^\perp \| + \lambda^2_2 \| \alpha^\perp \|^2
\]

\[
\leq \| \alpha \|^2 + \frac{2(1-\lambda_1)}{3} \| \alpha \|^2 + \frac{(1-\lambda_1)^2}{9} \| \alpha \|^2 = (1 - \frac{1-\lambda_1}{3})^2 \| \alpha \|^2 \leq \| \alpha \|^2
\]

Suppose \( \| \alpha^\perp \| > \frac{1-\lambda_1}{3\lambda_2} \| \alpha \| \).

Then, notice that

\[
s = 2(\alpha^\parallel_b + \alpha^\perp_b)(\sum_n G^T_2 \otimes e_n)A_2(\sum_m G_2 \otimes e_m)(\alpha^\parallel_b + \alpha^\perp_b).
\]

The RHS can be written as

\[
2(\alpha^\parallel_b + \sum_n \alpha^\perp_{a_n}(G^T_2) \otimes e_n)A_2(\alpha^\parallel_b + \sum_m G_2\alpha^\perp_{b_m} \otimes e_m).
\]

However, \( \sum_n \alpha^\perp_{a_n}(G^T_2) \otimes e_n \) is orthogonal to \( \alpha^\parallel_b \) and \( \sum_m G_2\alpha^\perp_{b_m} \otimes e_m \) is orthogonal to \( \alpha^\perp_b \). Thus,

\[
s = 2(\alpha^\parallel_b)A_2(\alpha^\parallel_b) + 2(\sum_n \alpha^\perp_{a_n}(G^T_2) \otimes e_n)A_2(\sum_m G_2\alpha^\perp_{b_m} \otimes e_m)
\]

From the previous arguments, we have

\[
s \leq \lambda_1(\| \alpha^\parallel \|^2) + \lambda^2(\| \alpha^\perp \|^2)
\]

\[
= \lambda_1(\| \alpha \|^2 - \| \alpha^\perp \|^2) + \lambda^2_2 \| \alpha^\perp \|^2
\]

\[
\leq (\| \alpha \|^2 - \| \alpha^\perp \|^2) + \lambda^2_2 \| \alpha^\perp \|^2 = (1 - \frac{(1-\lambda_1)^2(1-\lambda_2^2)}{9}) \| \alpha \|^2 \leq \| \alpha \|^2
\]

This completes the proof. \[\blacksquare\]
CHAPTER 11

EIGENVALUE BOUNDS ON PSEUDOCODEWORD WEIGHTS FOR EXPANDER CODES

Lower bounds on the minimum stopping set size and the minimum pseudocodeword weight of expander (LDPC) codes are derived. These bounds are compared with the known expander-type lower bounds on the minimum distance of expander codes. Furthermore, Tanner’s parity-oriented eigenvalue lower bound for the minimum distance is extended to yield a new lower bound on the minimum pseudocodeword weight.

11.1 Distance Bounds For Expander Codes

Definition 11.1.1 A graph $G = (X, Y; E)$ is \((c, d)\)-regular bipartite if the set of vertices in $G$ can be partitioned into two disjoint independent sets $X$ and $Y$ such that all vertices in $X$ have degree $c$ and all vertices in $Y$ have degree $d$ and each edge $e \in E$ of $G$ is incident with one vertex in $X$ and one vertex in $Y$, i.e., $e = (x, y), x \in X, y \in Y$.

We will refer to the vertices of degree $c$ as the left vertices, and to vertices of degree $d$ as the right vertices.
Definition 11.1.2 Let $0 < \alpha, \delta < 1$. A $(c,d)$-regular bipartite graph $G$ with $n$ degree $c$ nodes on the left and $m$ degree $d$ nodes on the right is an $(\alpha n, \delta c)$ expander if for every subset $U$ of degree $c$ nodes such that $|U| \leq \alpha n$, the size of the set of neighbors of $U$, $|\Gamma(U)|$ is at least $\delta c|U|$.

Definition 11.1.3 A connected, simple, graph $G$ is said to be a $(n, d, \mu)$ expander if $G$ has $n$ vertices, is $d$-regular, and the second largest eigenvalue of $G$ (in absolute value) is $\mu$.

The adjacency matrix of a $d$-regular connected graph has $d$ as its largest eigenvalue. A graph is a good expander if the gap between the first and the second largest eigenvalues is as big as possible.

Definition 11.1.4 A $(c,d)$-regular bipartite graph $G$ on $n$ left vertices and $m$ right vertices is a $(c,d,n,m,\mu)$ expander if the second largest eigenvalue of $G$ (in absolute value) is $\mu$.

Note that the largest eigenvalue of a $(c,d)$-regular bipartite graph is $\sqrt{cd}$.

We consider four different cases of expander graphs and obtain LDPC codes from these graphs.

- (A) Let a $(c,d)$-regular bipartite graph $G$ with $n$ left vertices and $m$ right vertices be a $(\alpha n, \delta c)$ expander. An LDPC code is obtained from $G$ by interpreting the degree $c$ vertices in $G$ as variable nodes and the degree $d$ vertices as simple parity-check nodes.

Lemma 11.1.1 [43] If $\delta > 1/2$, the LDPC code obtained from the $(\alpha n, \delta c)$ expander graph $G$ as above has minimum distance $d_{\text{min}} \geq \alpha n$. 

183
(B) Let a \((c, d)\)-regular bipartite graph \(G\) with \(n\) left vertices and \(m\) right vertices be a \((\alpha n, \delta c)\) expander. An LDPC code is obtained from \(G\) by interpreting the degree \(c\) vertices in \(G\) as variable nodes and the degree \(d\) vertices as sub-code constraints imposed by a \([d, rd; d]\) linear block code\(^1\). Such an LDPC code is a generalized LDPC code (see Definition 1.2.2).

\[\text{Lemma 11.1.2} \quad [43] \quad \text{If } \delta > 1/(ed), \text{ the LDPC code obtained from the } (\alpha n, \delta c) \text{ expander graph } G \text{ as above has minimum distance } d_{\text{min}} \geq \alpha n.\]

(C) Let a \(d\)-regular graph \(G\) be an \((n, d, \mu)\) expander. An LDPC code is obtained from \(G\) by interpreting the edges in \(G\) as variable nodes and the degree \(d\) vertices as constraint nodes imposing constraints of an \([d, rd, \epsilon d]\) linear block code. The resulting LDPC code has block length \(N = nd/2\) and rate \(R \geq 2r - 1.\)

\(^1\)The parameters of an \([n, k, d]\) binary linear block code are the block length \(n\), the dimension \(k\), and the minimum distance \(d\).
Degree $c$ vertices: variable nodes, degree $d$ vertices: sub-code constraints of a $[d, rd, cd]$ code.

Figure 11.2. Expander code: Case B

- Generalized constraint
  \[ \langle d, rd, cd \rangle \] constraint

- $|\Gamma(x)| \geq \delta c |x|

- $|x| \leq \alpha n$

Figure 11.3. Expander code: Case C
Lemma 11.1.3 \([43]\) The LDPC code obtained from an \((n, d, \mu)\) expander graph \(G\) as above has minimum distance \(d_{\min} \geq N \frac{(e-\frac{\mu}{2})^2}{(1-\frac{\mu}{2})^2} \).

- (D) Let a \((c, d)\)-regular bipartite graph \(G\) be an \((c, d, n, m, \mu)\) expander. An LDPC code is obtained from \(G\) by interpreting the edges in \(G\) as variable nodes, the degree \(c\) left vertices as sub-code constraints imposed by an \([c, r_1c, \epsilon_c]\) linear block code, and the degree \(d\) vertices as constraint nodes imposing constraints of an \([d, r_2d, \epsilon_d]\) linear block code. The resulting LDPC code has block length \(N = nc = md\) and rate \(R \geq r_1 + r_2 - 1\).

Lemma 11.1.4 \([24]\) If \(\epsilon_2d \geq \epsilon_1c > \mu/2\), the LDPC code obtained from the \((c, d, n, m, \mu)\) expander graph \(G\) as above has minimum distance

\[
d_{\min} \geq N(\epsilon_1\epsilon_2 - \frac{\mu}{2\sqrt{cd}}(\epsilon_1\sqrt{\frac{c}{d}} + \epsilon_2\sqrt{\frac{d}{c}})).
\]
11.2 Extension to Stopping Number

We extend the bounds on the minimum distance from the previous section to lower bound the minimum stopping set size. The proofs are simple extensions of the proofs for the minimum distance bounds from the previous section, and are hence omitted here.

- (A) Let a \((c, d)\)-regular bipartite graph \(G\) with \(n\) left vertices and \(m\) right vertices be a \((\alpha n, \delta c)\) expander and consider the LDPC code represented by \(G\) (Case A).

**Definition 11.2.1** [12] A stopping set is a set \(S\) of variable nodes such that every node that is a neighbor of some node \(s \in S\) is connected to \(S\) at least twice.

**Lemma 11.2.1** If \(\delta > 1/2\), the LDPC code obtained from the \((\alpha n, \delta c)\) expander graph \(G\) as above has a minimum stopping set size \(s_{\text{min}} \geq \alpha n\).

- (B) Let a \((c, d)\)-regular bipartite graph \(G\) with \(n\) left vertices and \(m\) right vertices be a \((\alpha n, \delta c)\) expander. Consider the LDPC code represented by \(G\) as in Case B. We provide the following definition of stopping set under the assumption that the \([d, rd, \epsilon d]\) subcode has no idle components, meaning that there are no components that are zero in all of the codewords of the subcode.

**Definition 11.2.2** A stopping set in a generalized LDPC code is a set of variable nodes such that every node that is a neighbor of some node \(s \in S\) is connected to \(S\) at least \(\epsilon d\) times.
Lemma 11.2.2 If $\delta > 1/(ed)$, the LDPC code obtained from the $(\alpha n, \delta c)$ expander graph $G$ as above has a minimum stopping set size $s_{\text{min}} \geq \alpha n$.

- (C) Let a $d$-regular graph $G$ be an $(n, d, \mu)$ expander. Consider the LDPC code represented by $G$ as in Case C.

Lemma 11.2.3 The LDPC code obtained from an $(n, d, \mu)$ expander graph $G$ has a minimum stopping set size $s_{\text{min}} \geq N\left(\frac{\epsilon - \frac{\mu}{2}}{1 - \frac{\mu}{2}}\right)$.

- (D) Let a $(c, d)$-regular bipartite graph $G$ be an $(c, d, n, m, \mu)$ expander. Consider the LDPC code represented by $G$ as in Case D.

Definition 11.2.3 A stopping set in a generalized LDPC code as in Case D is a set of variable nodes such that every node that is a degree $c$ neighbor of some node $s \in S$ is connected to $S$ at least $\epsilon_1 c$ times and every node that is a degree $d$ neighbor of some node $s \in S$ is connected to $S$ at least $\epsilon_2 d$ times.

Lemma 11.2.4 If $\epsilon_2 d \geq \epsilon_1 c > \mu/2$, the LDPC code obtained from the $(c, d, n, m, \mu)$ expander graph $G$ has a minimum stopping set size $s_{\text{min}} \geq N(\epsilon_1 \epsilon_2 - \frac{\mu}{2\sqrt{cd}}(\epsilon_1 \sqrt{\frac{c}{d}} + \epsilon_2 \sqrt{\frac{d}{c}}))$. 

188
11.3 Extension to Minimum Pseudocodeword Weight

In this section, we derive lower bounds on the minimum pseudocodeword weight \( w_{\text{min}} \) on the binary symmetric channel for the LDPC constraint graphs considered in the previous sections.

**Definition 11.3.1** A pseudocodeword \( p \) of the LDPC constraint graph \( G \) having \( N \) variable nodes is obtained from a codeword \( \hat{c} \) in a lift of \( G \). The pseudocodeword corresponding to \( \hat{c} \) is a vector \( p = (p_1, p_2, \ldots, p_N) \) over the reals where \( 0 \leq p_i \leq 1, \ 1 \leq i \leq N \), denotes the fraction of variable nodes that are 1 in \( \hat{c} \) in the \( i^{th} \) variable cloud of the finite-degree lift graph.

Recall from chapter 3 that the weight of a pseudocodeword \( p \) on the binary symmetric channel (BSC) is defined as follows.

**Definition 11.3.2** Let \( e \) be the smallest number such that the sum of the \( e \) largest components of \( p \) is at least the sum of the remaining components of \( p \). Then, the weight of \( p \) is

\[
w_{\text{BSC}}(p) = \begin{cases} 
2e, & \text{if } \sum_{e \text{ largest}} p_i = \sum_{\text{remaining}} p_i \\
2e - 1, & \text{if } \sum_{e \text{ largest}} p_i > \sum_{\text{remaining}} p_i
\end{cases}
\]

**Definition 11.3.3** The minimum pseudocodeword weight of an LDPC constraint graph \( G \) on the BSC is the minimum weight among all pseudocodewords obtained from all finite-degree lifts of \( G \). This parameter is denoted by \( w_{\text{min}}^{\text{BSC}} \).
We now consider the four cases of LDPC codes as in the previous sections.

- (A) Let a \((c, d)\)-regular bipartite graph \(G\) with \(n\) left vertices and \(m\) right vertices be a \((\alpha n, \delta c)\) expander. An LDPC code is obtained from \(G\) by setting the degree \(c\) vertices in \(G\) as variable nodes and degree \(d\) vertices as simple parity-check nodes.

**Theorem 11.3.1** If \(\delta > 2/3 + 1/3c\), the LDPC code obtained from the \((\alpha n, \delta c)\) expander graph \(G\) as above has a minimum pseudocodeword weight

\[
W_{\min}^{BSC} > \frac{2(\alpha n - 1)(3\delta - 2)}{(2\delta - 1)}.
\]

- (B) Let a \((c, d)\)-regular bipartite graph \(G\) with \(n\) left vertices and \(m\) right vertices be a \((\alpha n, \delta c)\) expander. An LDPC code is obtained from \(G\) by setting the degree \(c\) vertices in \(G\) as variable nodes and degree \(d\) vertices as sub-code constraints imposed by a \([d, rd, \epsilon d]\) linear block code.

**Theorem 11.3.2** If \(\delta > 2/(\epsilon d + 1) + 1/(c(\epsilon d + 1))\), the LDPC code obtained from the \((\alpha n, \delta c)\) expander graph \(G\) has a minimum pseudocodeword weight

\[
W_{\min}^{BSC} \geq \frac{2(\alpha n - 1)(3\delta - 2)}{(2\delta - 1)}.
\]
• (C) Let a $d$-regular graph $G$ be an $(n, d, \mu)$ expander. An LDPC code is obtained from $G$ by setting the edges in $G$ as variable nodes and degree $d$ vertices as constraint nodes imposing constraints of an $[d, rd, \epsilon d]$ linear block code.

**Theorem 11.3.3** The LDPC code obtained from an $(n, d, \mu)$ expander graph $G$ has a minimum pseudocodeword weight

$$w_{\text{min}}^{BSC} \geq N \epsilon \left(\frac{\epsilon - \frac{5\mu}{4d}}{1 - \frac{\mu}{d}}\right) f(d\epsilon),$$

where $f(d\epsilon) = \frac{48}{25} \left(\frac{1 - \frac{1}{3d} (d\epsilon + 1)}{1 - \frac{1}{3d} (d\epsilon + 1)}\right)$ is a constant less than 1 depending only on the distance of the sub-code.

• (D) Let a $(c, d)$-regular bipartite graph $G$ be an $(c, d, n, m, \mu)$ expander. An LDPC code is obtained from $G$ by setting the edges in $G$ as variable nodes, the degree $c$ left vertices as sub-code constraints imposed by an $[c, r_1c, \epsilon_1c]$ linear block code, and the degree $d$ vertices as constraint nodes imposing constraints of an $[d, r_2d, \epsilon_2d]$ linear block code. The resulting LDPC code has block length $N = nc = md$ and rate $R \geq r_1 + r_2 - 1$.

**Theorem 11.3.4** If $\epsilon_2d \geq \epsilon_1c > \mu/2$, the LDPC code obtained from the $(c, d, n, m, \mu)$ expander graph $G$ has a minimum pseudocodeword weight

$$w_{\text{min}}^{BSC} \geq N \frac{k}{\sqrt{cd}} \left(\frac{k}{\sqrt{cd}} - \frac{\mu}{\sqrt{cd}}\right),$$

where $k$ is a constant depending only on $c\epsilon_1$ and $d\epsilon_2$. 191
11.4 A Parity-oriented Bound on Pseudocodeword Weight

**Definition 11.4.1** The weight of a pseudocodeword \( q = (q_1, q_2, \ldots, q_n) \) of an LDPC constraint graph \( G \) on the AWGN channel is defined as

\[
w_{AWGN}(q) = \frac{(\sum_{i=1}^{n} q_i)^2}{(\sum_{i=1}^{n} q_i^2)}.
\]

The following bound on the minimum pseudocodeword weight on the AWGN channel is an adaptation of Tanner’s parity-oriented lower bound on the minimum distance [47].

**Theorem 11.4.1** Let \( G \) be the graph representing an LDPC code with parity check matrix \( H \). Suppose \( H \) is \((j, m)\)-regular with \( r \) rows and \( n \) columns. Then the minimum pseudocodeword weight on the AWGN channel is lower bounded as

\[
w_{min}^{AWGN} \geq \frac{(m - \mu_2 m + j - 1)n}{(\mu_1 - \mu_2)m}
\]

where \( \mu_1 \) and \( \mu_2 \) are the largest and second-largest eigenvalues (in absolute value) of \( H^T H \).

11.5 Pseudocodeword Properties of Zig-Zag LDPC Codes

In this section, we apply the lower bounds from the previous section to obtain lower bounds for the minimum pseudocodeword weight of the LDPC codes constructed from zig-zag product graphs.

First, a graph obtained from the zig-zag graph product is a \((N_1 d_1, d_2^2, \lambda)\) graph which can be used to represent a generalized LDPC code as in Case C. Taking
linear block codes as subcodes for the constraint nodes and applying the lower bound in Theorem 11.3.3, we obtain a graph representing a generalized LDPC code with

\[ w_{\text{min}}^{\text{BSC}} \geq \frac{N_1 d_1 d_2^2}{2} (\frac{\epsilon - \frac{5\lambda}{4d_2^2}}{1 - \frac{\lambda}{d_2^2}}) f(d_2^2 \epsilon) \geq \frac{N_1 d_1 d_2^2}{2} \left(\frac{\epsilon - \frac{5\lambda}{4d_2^2}}{1 - \frac{\lambda}{d_2^2}}\right)^2 f(d_2^2 \epsilon). \]

Further, note that a graph obtained from the generalized zig-zag graph product is a \((c_2^2, d_2^2)\)-regular graph on \((Nc_1, N_1d_1)\) vertices. Taking \([c_2^2, r_1c_2^2, \epsilon c_2^2]\) and \([d_2^2, r_1d_2^2, \epsilon d_2^2]\) linear codes as subcodes for the left and right constraints, respectively, and applying the lower bound in Theorem 11.3.4, we obtain a graph representing a generalized LDPC code with

\[ w_{\text{min}}^{\text{BSC}} \geq \frac{N_1 c_2^2 k}{2} \left(\frac{k - \frac{c_2 d_2^2 c_2^2}{c_2^2 d_2^2}}{\sqrt{c_2^2 d_2^2}}\right) = \frac{N_1 c_2^2 k}{2} \left(\frac{k - \frac{\mu}{c_2 d_2^2}}{c_2 d_2^2}\right) = \frac{N_1 k d_2^2}{2d_2^2} (k - \mu), \]

where \(k = \frac{c_2^2 d_2^2 c_2^2}{c_2^2 c_2^2 + d_2^2 c_2^2}\).

11.6 Proofs

Theorem 11.3.1 Proof: Let \(p = (p_1, \ldots, p_n)\) be a pseudocodeword in \(G\). Without loss of generality, let \(p_1 \geq p_2 \geq \ldots \geq p_n\). Let \(U = \{v_1, \ldots, v_e\}\) be a set of \(e\) variable nodes corresponding to the first \(e\) largest components of \(p\). Let \(\lambda = 2(1 - \delta) + \frac{1}{e}\). Let \(\hat{U} = \{v_i \in V|v_i \notin U, |\Gamma(v_i) \cap \Gamma(U)| \geq (1 - \lambda)c + 1\}\), where \(\Gamma(X)\) is the set of neighbors of the vertices in \(X\). Let \(U' = U \cup \hat{U}\). Suppose \(e = |U| = \frac{(an - 1)}{(1 + \beta)}\), where \(\beta = \frac{(1 - \delta)}{1 + \beta}\). Then by Lemma 6 in [13], \(|\hat{U}| \leq \beta|U|\). This implies that \(|U'| \leq (1 + \beta)|U| \leq (an - 1)\). By Proposition 4 in [13], \(U\) has a \(\delta\)-matching. This means that there is a set of edges \(M\) such that every check node in \(\Gamma(U')\) is incident to at most one edge in \(M\), and every vertex in \(U\) is incident
to at least $\delta c$ edges in $M$, and every vertex in $\hat{U}$ is incident to at least $\lambda c$ edges in $M$. Consider all of the check nodes in $\Gamma(U)$ that are incident with edges from $M$ that are also incident with the vertices in $U$. The number of ones that these vertices in $U$ contribute to these check nodes is at least $\delta c(p_1 + \cdots + p_e)$. These ones must be balanced by the remaining ones coming into these checks from the other nodes. This means at most $(1 - \delta)c$ edges from each vertex in $U$ are incident with these check nodes. Moreover, at most $(1 - \lambda)c$ edges from each vertex in $\hat{U}$ are incident with these check nodes. This gives the following inequality

$$\delta c(p_1 + \cdots + p_e) \leq (1 - \delta)c(p_1 + \cdots + p_e)$$

$$+ (1 - \lambda)c\left( \sum_{v_i \in \hat{U}} p_i \right) + (1 - \lambda)c\left( \sum_{v_i \in V \setminus U'} p_i \right)$$

The above inequality implies that

$$p_1 + \cdots + p_e \leq \frac{(1 - \lambda)}{(2\delta - 1)}(p_{e+1} + \cdots + p_n) < p_{e+1} + \cdots + p_n$$

from the choice of $\lambda$. From the definition of pseudocodeword weight on the BSC, we have $w_{BSC}(p) \geq 2e' - 1$, where $e' > e$. Thus, $w_{BSC}(p) > 2e = \frac{(an-1)(3\delta-2)}{(2\delta-1)}$. 

**Theorem 11.3.2**  
Proof: Suppose $G$ is an $(\alpha n, \delta c)$-expander, where $\delta > \frac{2}{(de+1)} + \frac{1}{(de+1)c}$. Then $w_{min}^{BSC} > 2e = \frac{(an-1)(3\delta-2)}{(2\delta-1)}$. by using a strong subcode, the $\delta$ required is less than in Case A, thereby allowing $\alpha$ to be larger [13] and yielding a larger bound overall. The argument is the same as in the proof of Case A, where now we set $\lambda = 2 - de\delta + \frac{1}{c}$. Suppose the pseudocodeword components involved in a constraint node are $p_1, p_2, \ldots, p_r$, then the components satisfy the following

194
inequality

\[(d_\epsilon - 1)p_i \leq \sum_{j \neq i} p_j\]

Hence, the first inequality in the proof of Case A now becomes

\[(d_\epsilon - 1)(p_1 + \cdots + p_e) \leq (1 - \delta)c(p_1 + \cdots + p_e)\]

\[+ (1 - \lambda)c\left(\frac{1}{n} \right) + (1 - \lambda)c\left(\frac{1}{n} \right)\]

This yields

\[p_1 + \cdots + p_e \leq \frac{(1 - \lambda)}{(d_\epsilon - 1)}(p_{e+1} + \cdots + p_n) < p_{e+1} + \cdots + p_n\]

from the choice of \(\lambda\). Thus, the weight of \(p\) is \(w(p) \geq 2e\).

\[\square\]

Theorem 11.3.3 \hspace{1cm} Proof: The \(d\)-regular graph \(G\) can be transformed to a \((2, d)\)-regular bipartite graph \(G'\) by representing every edge in \(G\) by a vertex in \(G'\) and every vertex in \(G\) by a vertex in \(G'\) and connecting the edge nodes to the vertex nodes in \(G'\) in a natural way. The edge nodes have degree two and they represent variable nodes of the LDPC code \(C\), whereas the vertex nodes have degree \(d\) and each represents a \([d, rd, \epsilon d]\)-subcode constraints. We now find the \(\alpha\) and \(\delta\) parameters of \(G'\) and the lower bound follows from the proof of Case B.

By the Alon-Chung Lemma [43], any set \(X\) of \(\gamma n\) vertices in \(G\) can contain at most \(\frac{nd}{2}(\gamma^2 + \frac{\mu}{d}(\gamma - \gamma^2))\) edges in the subgraph induced by \(X\) in \(G\). Thus, we have \(\delta \geq \frac{\gamma n}{2d(\gamma^2 + \frac{\mu}{d}(\gamma - \gamma^2))}\) in \(G'\). Following the proof of Case B, we impose the condition
that \( \delta > \frac{2}{de+1} + \frac{1}{2(de+1)} = \frac{5}{2(de+1)} \). This implies that in the worst case,

\[
\frac{\gamma n}{\frac{nd}{2}(\gamma^2 + \frac{\mu}{d}(\gamma - \gamma^2))} \geq \frac{5}{2(de+1)}.
\]

This implies that

\[
\gamma < \frac{\frac{4}{5}(de + 1) - \mu}{d - \mu} = \gamma_c
\]

Hence, we obtain a bound on \( \alpha \). Setting \( \alpha = (\gamma_c^2 + \frac{\mu}{d}(\gamma_c - \gamma_c^2)) \) and applying the lower bound from Case B yields

\[
w(p) \geq \frac{2(\alpha N - 1)(3\delta - 2)}{2\delta - 1}
\]

\[
\geq 2 \frac{nd}{2} \left( \frac{\frac{4}{5}(de + 1)(\frac{4}{5}(de + 1) - \mu)}{d - \mu} \right) \left( \frac{15 - 4(de + 1)}{10 - 2(de + 1)} \right)
\]

\[
\geq N \left( \frac{48 \epsilon}{25} \right) \left( \frac{1 - \frac{5\mu}{4d}}{1 - \frac{\mu}{d}} \right) \left( 1 - \frac{\frac{4}{15}(de + 1)}{1 - \frac{\mu}{10}(de + 1)} \right)
\]

Theorem 11.3.4  \hspace{1cm} \textbf{Proof:}

Let \( p = (p_1, p_2, \ldots, p_N) \) be a pseudocodeword. Without loss of generality, let us assume that \( p_1 \geq p_2 \geq \ldots p_N \). Let \( e \) be the smallest number such that \( p_1 + p_2 + \cdots + p_e \geq p_{e+1} + \cdots + p_N \). Let \( X_e \) be the set of edges in \( G \) that correspond to the support of the \( e \) largest components of \( p \). Now we define a set \( S \) as the set of left neighbors (degree \( c \) neighbors) to the edges in \( X_e \) and similarly define a set \( T \) as the set of right neighbors to \( X_e \).

Furthermore, let \( m_1 \) nodes in \( S \) be connected to \( X_e \) exactly once, \( m_2 \) nodes in \( S \) be connected to \( X_e \) exactly twice, and so on until, \( m_c \) nodes in \( S \) be connected to \( X_e \) exactly \( c \) times. Similarly, let \( n_i \) nodes in \( T \), for \( i = 1, 2, \ldots, d \), be connected
to $X_e$ exactly $i$ times.

Then, we have $|S| = m_1 + m_2 + \cdots + m_c$, $|T| = n_1 + n_2 + \cdots + n_d$, and $e = m_1 + 2m_2 + \cdots + cm_c = n_1 + 2n_2 + \cdots + dn_d$.

Suppose $|S| + |T| \leq 2e(d_1 + d_2)/(d_1d_2) = \frac{2e}{d_1} + \frac{2e}{d_2}$, suppose in the worst case $k = 1$. Then, by applying the edge-expansion inequality of [24], we get

$$e = |X_e| \leq |E(S,T)| \leq \frac{d}{m} |S||T| + \frac{\mu}{2} (|S| + |T|)$$

Since $|S||T| \leq (|S| + |T|)^2/2$ and $|S| + |T| \leq 2e/k$, where $k = \frac{d_1d_2}{d_1 + d_2}$, we get

$$e \leq \frac{d}{m} \frac{2e^2}{k^2} + \frac{e\mu}{k}$$

yielding

$$2e \geq \frac{mk}{d}(k - \mu) = N \frac{k}{\sqrt{cd}} \left( \frac{1}{\sqrt{cd}} - \frac{\mu}{\sqrt{cd}} \right)$$

This yields the desired bound on $w_{\min}$.

Suppose $|S| + |T| > \frac{2e}{d_1} + \frac{2e}{d_2}$. Let $X'_e \subset X_e$ be the subset of edges among the $e$ dominant components of $p$ such that every edge in $X'_e$ is either connected to a node on the left set of vertices such that the node is connected to at most $d_1/2$ edges in $X_e$ or is connected to a node on the right set of vertices of the graph such that that node is connected to at most $d_2/2$ edges in $X_e$. The pseudocodeword components at every such node on the left that is incident with at most $d_1/2$ edges in $X_e$ or every such node on the right that is incident with at most $d_2/2$ edges in $X_e$ satisfy the inequality

$$\sum_{i \in X'_e} p_i \leq \sum_{i \notin X_e} p_i$$

197
However, \( e = |X_e| \) is chosen such that \( \sum_{i \in X_e} p_i \geq \sum_{i \notin X_e} p_i \). This means that \( e' = |X_e \setminus X'_e| \) must be sufficiently large and the components \( p_i \) for \( i \in X_e \setminus X'_e \) must be sufficiently large. In addition, note that \( X'_e \neq \emptyset \), for otherwise every neighbor of \( X_e \) in the left (resp., right) vertex set would have at least \( \frac{d_1}{2} \) (resp., \( \frac{d_2}{2} \)) connections to \( X_e \), giving

\[
|S| \leq \frac{|X_e|}{d_1/2} \text{ and } |T| \leq \frac{|X_e|}{d_2/2}.
\]

This means \( |S| + |T| \leq \frac{2e}{d_1} + \frac{2e}{d_2} \), which is a contradiction. Continuing on, every edge in \( X_e \setminus X'_e \) is incident on a node on the left that is connected to \( X_e \setminus X'_e \) at least \( d_1/2 + 1 \) times and is incident on a node on the right that is connected to \( X_e \setminus X'_e \) at least \( d_2/2 + 1 \) times. If \( S' \) is the set of nodes on the left that are incident with at least \( d_1/2 + 1 \) edges in \( X_e \setminus X'_e \) and \( T' \) is the set of nodes on the right that are incident with at least \( d_2/2 + 1 \) edges in \( X_e \setminus X'_e \), then we have that \( |S'| \leq \frac{e'}{d_1/2+1} \) and \( |T'| \leq \frac{e'}{d_2/2+1} \). Thus, we have \( \frac{2e'}{d_1} + \frac{2e'}{d_2} \geq |S'| + |T'| \). Applying the edge-expansion inequality of [24]

\[
e' = |X_e \setminus X'_e| \leq |E(S', T')| \leq \frac{d}{m} |S'||T'| + \frac{\mu}{2}(|S'| + |T'|)
\]

Since \( |S'||T'| \leq (|S'| + |T'|)^2/2 \) and \( |S'| + |T'| \leq 2e/k \), where \( k = \frac{d_1d_2}{d_1+d_2} \), we get

\[
e' \leq \frac{d}{m} \frac{2e'^2}{k^2} + \frac{\mu e'}{k}
\]

yielding

\[
2e' \geq \frac{mk}{d}(k-\mu) = N\frac{k}{\sqrt{cd}} \left( \frac{k}{\sqrt{cd}} - \frac{\mu}{\sqrt{cd}} \right).
\]

However, since \( e \geq e' \), we have \( 2e \geq 2e' \geq N\frac{k}{\sqrt{cd}} \left( \frac{k}{\sqrt{cd}} - \frac{\mu}{\sqrt{cd}} \right) \).
Theorem 11.4.1  Proof: Let \( \mathbf{q} = (q_1, \ldots, q_n) \) be a pseudocodeword of \( G \), and let \( \mathbf{p} = H \mathbf{q} \) be a real-valued vector of length \( r \). The first eigenvector of \( H H^T \) is \( \mathbf{e}_1 = (1, 1, \ldots, 1)^T / \sqrt{r} \). Let \( \mathbf{p}_i \) be the projection of \( \mathbf{p} \) onto the \( i \)th eigenspace.

We will now upper bound \( \| H^T \mathbf{p} \|^2 \). Converting \( \| H^T \mathbf{p} \|^2 \) into eigenspace representation, we get

\[
\| H^T \mathbf{p} \|^2 = \sum_{i=1}^{s} \mu_i \| \mathbf{p}_i \|^2 = \mu_1 \| \mathbf{p}_1 \|^2 + \sum_{i=2}^{s} \mu_i \| \mathbf{p}_i \|^2 \\
\leq \mu_1 \| \mathbf{p}_1 \|^2 + \mu_2 (\| \mathbf{p} \|^2 - \| \mathbf{p}_1 \|^2)
\]

Note that

\[
\| \mathbf{p}_1 \|^2 = \frac{j^2}{r} (\sum_{i=1}^{n} q_i)^2, \quad \text{and}
\]

\[
\| \mathbf{p} \|^2 \leq mj (\sum_{i=1}^{n} q_i)^2
\]

The first equality follows from the choice of \( \mathbf{p} \) and the regularity of the parity check matrix \( H \). The second inequality follows by applying the identity \( (q_1 + q_2 + \cdots + q_t)^2 \leq t(q_1^2 + q_2^2 + \cdots + q_t^2) \) to the terms in the expansion of \( \| \mathbf{p} \|^2 \).

The above set of equations yield

\[
\| H^T \mathbf{p} \|^2 \leq (\mu_1 - \mu_2) \frac{j^2}{r} (\sum_{i=1}^{n} q_i)^2 + \mu_2 mj (\sum_{i=1}^{n} q_i^2)
\]

We now lower bound \( \| H^T \mathbf{p} \|^2 \) as follows

\[
\| H^T \mathbf{p} \|^2 = \sum_{t=1}^{n} (\sum_{i=1}^{r} \sum_{\ell=1}^{n} h_{i,\ell} q_{i,\ell})^2 \geq ((m - 1)j + j^2)(\sum_{t=1}^{n} q_t^2)
\]

This bound follows by observing that for each \( t \) in the outer summation, the inner sums over the indices \( i \) and \( \ell \) contribute \( j q_t \) terms and \( (m - 1)j \) terms.
involving other $q_k$’s. Thus for each $t$, the square of the inner sums over $i$ and $\ell$ can be lower bounded by $j^2 q_t^2 + q_{t_1}^2 + q_{t_2}^2 + \cdots + q_{j(m-1)}^2$. Now using the regularity of $H$, we observe that in the overall sum for $t = 1, 2, \ldots, n$, each $q_t^2$ appears $j^2 + (m-1)j$ times.

Combining the upper and lower bounds, we get

$$\frac{(m-1)j + j^2 - \mu_2 m j}{(\mu_1 - \mu_2) j^2} \leq \frac{(\sum_{i=1}^{n} q_i)^2}{(\sum_{i=1}^{n} q_i^2)} = w^{AWGN}(q)$$

Since $nj = rm$ we obtain the desired lower bound.
CHAPTER 12

CONCLUSIONS

This dissertation considered the finite-length analysis of low-density parity-check codes in relation to the min-sum iterative decoding algorithm and the design of finite-length LDPC codes suitable for iterative decoding. The main contributions of this dissertation are the following.

- New lower bounds on the minimum pseudocodeword weight of Tanner graphs for the BSC and AWGN channels. A tree-based lower bound obtained here provided a basis for an algebraic construction of LDPC codes.

- The analysis of pseudocodewords for non-binary LDPC codes on the $p$-ary symmetric channel. A weight definition for pseudocodewords of $p$-ary LDPC codes on the $p$-ary symmetric channel was derived. The tree-based lower bound for binary LDPC codes was also shown to hold for $p$-ary LDPC codes in this context.

- An illustration of the fundamental differences between the graph-covers polytope characterization of pseudocodewords and the characterization of pseudocodewords arising on the iterative decoder’s computation tree.

- The relation between the max-fractional weight parameter of linear-programming decoders and the pseudocodeword weight parameter of iterative decoders.
• A concrete analysis of the structure of good and bad pseudocodewords in Tanner graphs for min-sum iterative decoding.

• An upper bound on the minimum lift degree needed to realize all irreducible pseudocodewords for a given Tanner graph. This upper bound on the lift degree was then used to obtain a lower bound for the minimum pseudocodeword weight of the graph as a fraction of its minimum stopping set size.

• A case study of different graph representations of Hamming codes and the pseudocodewords arising in each representation. It was shown how each representation affects the pseudocodeword weight distribution.

• A generalized version of the bipartite zig-zag graph product that operates on unbalanced bipartite graph components. This included the original bipartite zig-zag graph product as a special case. The resulting zig-zag graph product was shown to be a good expander if the component graphs were good expanders.

• New constructions of LDPC codes with good minimum pseudocodeword weight and/or good expansion.

• Lower bounds on the minimum pseudocodeword weight for expander codes. The lower bounds derived for the minimum pseudocodeword weight are valid for different levels of expansion (or, δ’s). Thus, they are more general than the lower bound on the minimum distance presented in [43] which is valid for a particular δ.
We conclude by outlining some directions for future work in this area.

- How to choose an optimal graph representation with respect to pseudocode-word weight distribution.

- Efficient encoding techniques for the new codes presented herein.

- Construction of LDPC codes where $d_{\text{min}} = w_{\text{min}} \gg$ Tree-bound.

- Closed-form expression for the $t$-value of Tanner graphs in terms of graph parameters.

- A mathematical characterization of pseudocodewords arising on the iterative decoder’s computation tree.

- Pseudocodeword analysis for other graph-based codes such as turbo codes, repeat-accumulate codes, tail-biting convolutional codes, as well as for more sophisticated communication channels.
BIBLIOGRAPHY


