PROTOGRAPH-BASED GENERALIZED LDPC CODES: ENUMERATORS, DESIGN, AND APPLICATIONS

by

Shadi Ali Abu-Surra

Copyright © Shadi Ali Abu-Surra 2009

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA

2009
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Shadi Ali Abu-Surra entitled “Protograph-Based Generalized LDPC Codes: Enumerators, Design, and Applications” and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Date: November 16, 2009
William E. Ryan, Ph.D.

Date: November 16, 2009
Bane Vasic, Ph.D.

Date: November 16, 2009
Ivan Djordjevic, Ph.D.

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Date: November 16, 2009
Dissertation Director: William E. Ryan, Ph.D.
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the copyright holder.

SIGNED: ___________________________  Shadi Ali Abu-Surra
ACKNOWLEDGEMENTS

First, I would like to thank my advisor Professor William Ryan for his continuing support and advice. I have learned so much through him and his experience in channel coding. I am extremely appreciative to him for all his help and for his abundant proofreadings that greatly improved this work. Without his guidance this dissertation would not have been possible.

I would like to give a special thanks to Dr. Dariush Divsalar for teaching me how to have a handle on protograph-based LDPC codes. His numerous insights and strong experience on this topic motivated me to dig deeper and look at the bigger picture, which allowed me to solve many of the challenging problems presented in this work.

I would also like to express my deep gratitude to Dr. Gianluigi Liva for the valuable insight and the interesting discussions, specially, at the start of my work on generalized LDPC codes. Also, I would like to acknowledge my lab partners Fei Peng, Yang Han, Yifei Zhang, and Matthew Viens for helping me keep my sanity and being a great team to work with.

I also appreciate the financial support of the NASA Jet Propulsion Lab (JPL) and NASA Goddard Space Flight Center during my doctoral program. I like to thank Wai Fong of NASA-Goddard for his help and advice.

I also like to thank the staff of the electrical and computer engineering department for their help. I mainly like to thank Tami Whelan for her advice and handling all necessary paperwork.

Finally, I like to give special thanks to my parents for their love, help, and encouragement throughout the years. Whatever I say, words will not be enough to thank them for all they have done for me. I also like to thank my wife Suad for making the past two years exceptionally joyful and exciting.
DEDICATION

To my parents and my wife Suad.
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................. 8

LIST OF TABLES ........................................................................ 12

ABSTRACT ................................................................................. 13

CHAPTER 1 Introduction .......................................................... 15
  1.1 Dissertation Theme ........................................................... 15
  1.2 Brief introduction on low-density parity-check codes ................. 16
  1.3 Dissertation contributions and motivation ............................. 18
  1.4 Dissertation outline .......................................................... 19

CHAPTER 2 Protograph-Based G-LDPC Codes ............................ 21
  2.1 Introduction ................................................................. 21
  2.2 Design of protograph-based G-LDPC codes via Hamming-node or
       RSRC-node doping .......................................................... 22
      2.2.1 Overview of the Design Technique .............................. 22
      2.2.2 Example Doped LDPC Code Designs .......................... 26
  2.3 Protograph-based G-LDPC decoders .................................... 37
      2.3.1 Rate-($\kappa - 1)/\kappa$ RSC code SISO decoder ............... 41
  2.4 Conclusion .................................................................... 45

CHAPTER 3 Protograph-Based G-LDPC Codes enumerators ............ 46
  3.1 Introduction ................................................................. 46
  3.2 Ensemble weight enumerators for protograph-based G-LDPC codes 49
      3.2.1 Finite-length ensemble weight enumerators .................. 50
      3.2.2 Asymptotic ensemble weight enumerators .................... 56
      3.2.3 On the complexity of computing asymptotic ensemble enu-
           merators ................................................................. 63
  3.3 Ensemble stopping set enumerators .................................... 68
  3.4 Ensemble Trapping Set Enumerators ................................... 79
      3.4.1 Finite-size trapping set enumerators ............................ 79
      3.4.2 Asymptotic trapping set enumerators ........................... 80
      3.4.3 Elementary Trapping set enumerators .......................... 83
  3.5 Ensemble pseudocodeword weight enumerators ..................... 91
      3.5.1 Finite-length ensemble pseudocodeword weight enumerators 93
TABLE OF CONTENTS – Continued

3.5.2 Asymptotic ensemble pseudocodeword weight enumerator . . 97
3.6 Conclusion ........................................................................ 102

CHAPTER 4 Trapping sets enumerators for specific LDPC codes .... 105
4.1 Introduction ....................................................................... 105
4.2 Algorithm description ....................................................... 106
4.2.1 Graph transformation ..................................................... 106
4.2.2 Customized impulse algorithm ....................................... 107
4.3 Intuition and complexity of the customized impulse algorithm . . 110
4.4 Trapping set enumerators for selected LDPC codes ............ 111
4.5 Conclusion ....................................................................... 117

CHAPTER 5 LDPC Codes for Free-Space Optical Communications .. 118
5.1 Introduction ....................................................................... 118
5.2 System model .................................................................... 120
5.2.1 Bit-to-PPM modulator .................................................. 121
5.2.2 Channel model ............................................................... 121
5.3 LDPC-PPM coded modulation decoding ............................ 121
5.3.1 LDPC-PPM non-iterative demodulator/decoder ............... 122
5.3.2 LDPC-PPM iterative demodulator/decoder ..................... 123
5.3.3 LDPC-APPM iterative decoder ....................................... 127
5.4 Density evolution for the LDPC-PPM iterative decoding ....... 127
5.5 Numerical results ............................................................... 135
5.6 Conclusion ....................................................................... 136

CHAPTER 6 Conclusion .......................................................... 141

APPENDIX A ......................................................................... 145

APPENDIX B ......................................................................... 147

REFERENCES ..................................................................... 151
LIST OF FIGURES

2.1 Vectorized protograph. .............................. 22
2.2 Rate-1/6 RSC-LDPC protograph. .................... 27
2.3 Frame error rate comparison between (600, 100) HD-LDPC code and RSC-LDPC codes at different $T$, $I_{\text{max}} = 50$. .................. 29
2.4 Rate-1/6 RSC-LDPC encoder. .......................... 31
2.5 Rate-1/4 RSC-LDPC protograph. ..................... 32
2.6 Adjacency matrix of the (16352, 4088) RSC-LDPC code, and a magnified view of the first 2336 columns. .................... 33
2.7 Performance of (16352, 4088) RSC-LDPC code compared to that of (16384, 4096) QCRA code. $I_{\text{max}} = 20$. ...................... 34
2.8 Protograph of rate-1/2 G-LDPC with two rate-2/3 RSC constraints. . 35
2.9 Performance of the (8160, 4080) G-LDPC code, compared with that of (8192, 4096) AR4JA code. .......................... 36
2.10 Protograph of rate-1/2 G-LDPC with rate-5/6 RSC constraints. . 37
2.11 Performance of the (576, 288) G-LDPC code. ................ 38
2.12 Comparison between the performance of BCJR decoder, and the other sub-optimal decoders. The (2048, 1024) HD-LDPC components are (32, 26) extended Hamming codes. .................. 40
3.1 Vectorized protograph. .............................. 50
3.2 Check split method to compute enumerators for checks nodes with higher degrees ................................. 55
3.3 Rate-1/7, $n_v = 7$, (or, rate-7/15, $n_v = 15$) G-LDPC protograph. ... 60
3.4 Asymptotic codeword weight enumerator for the rate-1/7 and rate-1/6 G-LDPC code. .......................... 60
3.5 Rate-1/6 HD-LDPC protograph. ........................ 62
3.6 Asymptotic codeword weight enumerator for the rate-1/6 and rate-1/5 HD-LDPC code ensembles. ......................... 62
3.7 Asymptotic codeword weight enumerator for the rate-7/15 and rate-1/2 G-LDPC code. .......................... 65
3.8 Rate-1/3 G-LDPC protograph. .......................... 66
3.9 Asymptotic codeword weight enumerator for the rate-1/3 G-LDPC code. .............................. 67
LIST OF FIGURES – Continued

3.10 Protographs for several rate-1/2 LDPC codes: (a) (6, 3) regular LDPC code. (b) Precoded (6, 3) regular LDPC code. (c) RJA LDPC code. (d) ARJA LDPC code. (Note: The shaded VNs represent punctured nodes.) ................................................. 68

3.11 Asymptotic codeword weight enumerators for the protograph-based LDPC code ensembles in Fig. 3.10. ................................................................. 69

3.12 Asymptotic stopping set enumerators for the protograph-based LDPC code ensembles in Fig. 3.10. ................................................................. 73

3.13 Check-split method results and asymptotic stopping set enumerators for the RJA code ensemble in Fig. 3.10(c). .................................................. 74

3.14 Asymptotic stopping set enumerator for the rate-1/7 and the rate-7/15 G-LDPC code. ................................................................. 76

3.15 Asymptotic stopping set enumerator for the rate-1/6 G-LDPC code. 77

3.16 Asymptotic stopping set enumerator for the rate-1/3 G-LDPC code. 78

3.17 Modified vectorized protograph with auxiliary VNs along the top and the original VNs along the bottom. ................................................................. 80

3.18 The zero-contour curves of the asymptotic trapping set enumerators for two different ensembles. The zero-contour curves correspond to Ensemble 1 and Ensemble 2 are the solid and the dashed curves, respectively ................................................................. 82

3.19 Asymptotic trapping set enumerators for regular (6, 3) LDPC code ensemble. ................................................................. 85

3.20 Asymptotic trapping set enumerators as a function of $\alpha$ parametrized by $\beta$ for the regular (6, 3) LDPC code ensemble. .................. 86

3.21 Asymptotic elementary trapping set enumerators for the (6, 3) regular LDPC code ensemble. ................................................................. 87

3.22 Asymptotic trapping set enumerators for different $\Sigma_\Delta$ in the regular (6, 3) LDPC ensemble. ................................................................. 88

3.23 The zero-contour curves of the asymptotic trapping set enumerators for different protograph-based LDPC code ensembles. ............... 89

3.24 The zero-contour curves of the asymptotic trapping set enumerators for the RJA and the ARJA code ensembles. ................................. 90

3.25 Pseudocodeword weight enumerators over AWGN channel for the degree-3 cover of the (7, 4) Hamming code. ................................. 94

3.26 Asymptotic pseudocodeword weight enumerators over AWGN channel for the rate-1/7 G-LDPC code ensembles with degree-2 and degree-3 covers. ................................................................. 100

3.27 Asymptotic pseudocodeword weight enumerators over the BEC for the rate-1/7 G-LDPC code ensemble with degree-2 and degree-3 covers.101
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.28</td>
<td>Asymptotic pseudocodeword weight enumerators over AWGN for the (6, 3) LDPC code ensemble with degree-2 and degree-3 covers</td>
<td>102</td>
</tr>
<tr>
<td>4.1</td>
<td>LDPC graph.</td>
<td>107</td>
</tr>
<tr>
<td>4.2</td>
<td>Modified LDPC graph.</td>
<td>108</td>
</tr>
<tr>
<td>4.3</td>
<td>Parity check matrix for the (32, 16) LDPC code.</td>
<td>112</td>
</tr>
<tr>
<td>5.1</td>
<td>System block diagram.</td>
<td>120</td>
</tr>
<tr>
<td>5.2</td>
<td>LDPC-PPM decoder.</td>
<td>124</td>
</tr>
<tr>
<td>5.3</td>
<td>PPM demodulator output LLR pdfs for 0/1 transmissions at LDPC decoder feedback SNR=-10 dB. (Poisson channel parameters: $n_b=0.2$, $T_s=32$ns, $M=64$, $n_s/(MT_s)=-30$ dB photon/ns. Feedback input (from LDPC): Symmetric Gaussian, SNR=-10 dB.)</td>
<td>128</td>
</tr>
<tr>
<td>5.4</td>
<td>PPM demodulator output LLR pdfs for 0 transmission at LDPC decoder feedback SNR = -10, 0, and 5 dB. (Poisson channel parameters: $n_b=0.2$, $T_s=32$ns, $M=64$, $n_s/(MT_s)=-30$ dB photon/ns. Feedback input (from LDPC): Symmetric Gaussian, various SNRs. Txd symbol is zero.)</td>
<td>128</td>
</tr>
<tr>
<td>5.5</td>
<td>PPM demodulator output LLR pdfs for 0 transmission at LDPC decoder feedback SNR = 5 and 10 dB. (Poisson channel parameters: $n_b=0.2$, $T_s=32$ns, $M=64$, $n_s/(MT_s)=-30$ dB photon/ns. Feedback input (from LDPC): Symmetric Gaussian. Txd symbol is zero. Note, as the feedback SNR from the LDPC increases, the PPM output tends to be independent from the feedback input.)</td>
<td>129</td>
</tr>
<tr>
<td>5.6</td>
<td>The pdf evolution of the messages send from the DM to the VN. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-29$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$.</td>
<td>129</td>
</tr>
<tr>
<td>5.7</td>
<td>The pdf evolution of the messages send from the DM to the VN. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-30$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$.</td>
<td>130</td>
</tr>
<tr>
<td>5.8</td>
<td>The pdf evolution of the messages send from the VN to the DM. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-29$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$.</td>
<td>131</td>
</tr>
<tr>
<td>5.9</td>
<td>The pdf evolution of the messages send from the VN to the DM. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-30$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$.</td>
<td>132</td>
</tr>
<tr>
<td>5.10</td>
<td>Density evolution block diagram for the LDPC-PPM iterative decoder.</td>
<td>133</td>
</tr>
<tr>
<td>5.11</td>
<td>Block diagram of the DM evolution estimation procedure.</td>
<td>133</td>
</tr>
</tbody>
</table>
LIST OF FIGURES – Continued

5.12 Bit error rate of the (4096, 2048) U of Arizona LDPC code with LDPC-PPM iterative and non-iterative demodulator/decoder (labeled LDPC↔PPM and LDPC→PPM, respectively). Performance also compared to the (4096, 2048) serial turbo code of JPL (labeled SCPPM), and to the (4085, 2047) Reed-Solomon code (labeled RSPPM). 137

5.13 Performance comparison between the LDPC-PPM iterative demodulator/decoder and the LDPC-APPM decoder for several LDPC codes with different VNs degree distribution. 138

5.14 Performance comparison between two (4096, 2048) LDPC code designed by Shu Lin, the (4096, 2048) U of Arizona code, and a (4096, 2048) LDPC code with degree distributions \( \lambda(z) = 0.33898z + 0.22373z^2 + 0.34124z^3 + 0.096045z^9 \) and \( \rho(z) = 0.59322z^4 + 0.40678z^7 \) (labeled DE LDPC). 139

5.15 Performance of the (4096, 2048) U of Arizona code with LDPC-PPM iterative demodulator/decoder for different mismatch between the real \( n_s/n_b \) and the estimated one. 140


LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>$(a, b)$-trapping set enumerators for the $(2640, 1320)$ Margulis code</td>
<td>112</td>
</tr>
<tr>
<td>4.2</td>
<td>$(a, b)$-trapping set enumerators for the $(32, 16)$ LDPC code in Example 22</td>
<td>113</td>
</tr>
<tr>
<td>4.3</td>
<td>$(a, b)$-trapping set enumerators for the $(128, 64)$ LDPC code in Example 23</td>
<td>113</td>
</tr>
<tr>
<td>4.4</td>
<td>$(a, b)$-trapping set enumerators for the $(256, 128)$ LDPC code in Example 23 (only those with $b \leq 4$ are listed below)</td>
<td>114</td>
</tr>
<tr>
<td>4.5</td>
<td>$(a, b)$-trapping set enumerators for the $(648, 324)$ LDPC code in 802.11n</td>
<td>115</td>
</tr>
<tr>
<td>4.6</td>
<td>$(a, b)$-trapping set enumerators for the $(1296, 648)$ LDPC code in 802.11n</td>
<td>116</td>
</tr>
<tr>
<td>4.7</td>
<td>$(a, b)$-trapping set enumerators for the $(1944, 972)$ LDPC code in 802.11n</td>
<td>116</td>
</tr>
<tr>
<td>4.8</td>
<td>$(a, b)$-trapping set enumerators for the $(2048, 1723)$ LDPC code in 802.3an</td>
<td>117</td>
</tr>
</tbody>
</table>
Among the recent advances in the area of low-density parity-check (LDPC) codes, protograph-based LDPC codes have the advantages of a simple design procedure and highly structured encoders and decoders. These advantages can also be exploited in the design of protograph-based generalized LDPC (G-LDPC) codes. In this dissertation we provide analytical tools which aid the design of protograph-based LDPC and G-LDPC codes. Specifically, we propose a method for computing the codeword-weight enumerators for finite-length protograph-based G-LDPC code ensembles, and then we consider the asymptotic case when the block-length goes to infinity. These results help the designer identify good ensembles of protograph-based G-LDPC codes in the minimum distance sense (i.e., ensembles which have minimum distances grow linearly with code length). Furthermore, good code ensembles can be characterized by good stopping set, trapping set, or pseudocodeword properties, which assist in the design of G-LDPC codes with low floors. We leverage our method for computing codeword-weight enumerators to compute stopping-set, and pseudocodeword enumerators for the finite-length and the asymptotic ensembles of protograph-based G-LDPC codes. Moreover, we introduce a method for computing trapping set enumerators for finite-length (and asymptotic) protograph-based LDPC code ensembles. Trapping set enumerators for G-LDPC codes represents a more complex problem which we do not consider here. Inspired by our method for computing trapping set enumerators for protograph-based LDPC code ensembles, we developed an algorithm for estimating the trapping set enumerators for a specific LDPC code given its parity-check matrix. We used this algorithm to enumerate trapping sets for several LDPC codes from communication standards. Finally, we study coded-modulation schemes with LDPC codes and pulse position modulation (LDPC-PPM) over the free-space optical channel. We present three different de-
coding schemes and compare their performances. In addition, we developed a new density evolution tool for use in the design of LDPC codes with good performances over this channel.
1.1 Dissertation Theme

It has been known for over a decade that very long turbo code and low-density parity-check (LDPC) codes are capable of providing reliable communication near the Shannon limit for many channels, a feat made possible by iterative decoders [1–4]. Research in more recent years have targeted the deficiencies of these codes that surface when faced with practical requirements such as low decoding complexity, low decoding latency, and a low error-rate floor.

While it is possible to design LDPC codes and companion decoders with lower floors, perhaps at the expense of coding loss in the waterfall region, we do not believe it is possible to design an LDPC code and decoder with a floor that is five orders of magnitude lower as required by the data storage applications, among others. For this reason, we propose to search for a solution among a broader class of codes that includes turbo and LDPC codes as special cases. This broader class of codes is the class of generalized LDPC (G-LDPC) codes. The G-LDPC codes that we study are based on protographs, which are small bipartite graphs from which a code’s Tanner graph can be derived: multiple copies of the protograph are made and the edges are re-connected per some deterministic strategy among the replicated nodes.

Generalized LDPC codes were suggested by Tanner [5] and are graphical codes for which subsets of the set of code bits obey a more complex constraint than a single parity check (SPC) constraint, such as a Hamming code constraint. The generalized-constraint nodes are called super-constraint nodes, or simply constraint nodes (CNs). There are several advantages to employing CNs. First, CNs tend to lead to larger minimum distances. Second, because a complex constraint node can encapsulate multiple SPC constraints, the resulting Tanner graph will contain fewer
edges so that deleterious graphical properties are more easily avoided. Third, the belief propagation decoder tends to converge more quickly because the CN processors now correspond to stronger codes. The first two advantages lead to a lower error-rate floor. The third advantage leads to lower decoder complexity and/or higher decoding speed.

The overarching theme of this dissertation then is the enumeration, analysis, and design of protograph-based G-LDPC codes in regions where LDPC and turbo codes fail, in particular, in the low-rate, short-length, and low-floor regions. We propose to resolve some of the deficiencies of LDPC codes and turbo codes by exploring the broader class of G-LDPC codes, in particular, protograph-based G-LDPC codes.

1.2 Brief introduction on low-density parity-check codes

In a race toward achieving near-capacity-limit error-correcting codes, LDPC codes outperform other candidates. LDPC codes were first proposed by Gallager [6] in 1962. But it was not of practical use until the nineties after the work by MacKay and Neal [7], and the work by Wiberg, Loeliger, and Koetter [8] on codes on graphs and message-passing iterative decoding. Currently, there is a large body of research on LDPC codes.

An LDPC code is a linear block code which can be described by a sparse parity check matrix (i.e., the density of non-zero elements in the parity check matrix is very small). Alternatively, LDPC codes can be described using a graphical structure called Tanner graph. Tanner [5] was the first to describe linear codes using a bipartite graph (graph with two types of nodes, namely, variable nodes and single parity-check nodes, where nodes of the same type cannot be connected to each other). This simple representation of LDPC codes helps the study of LDPC codes in the following two regards: First, studying the behavior of the iterative decoder on the code’s graph provides an accurate estimate of the performance of the code in the asymptotic sense [9–12]. Moreover, it helps the description of several sub-optimal reduced-complexity iterative decoders (e.g., min-sum and scaled min-sum).
Second, identifying certain graphical structures in the code’s graph (such as, cycles, stopping sets [13,14], trapping sets [15]) and understanding their effect on the code’s performance results in the design of LDPC codes with good performance in the floor region.

The floor region of the code is identified by a sudden change in the slope of the error-rate curve, which is usually seen in the high signal-to-noise (SNR) region. Originally, it was thought that this is caused by low-weight codewords in the code. However, in LDPC codes Frey and Koetter [16] notice that the iterative decoder converged to a non-codeword pattern called a pseudocodeword. Later, the authors in [17, 18] presented more comprehensive study of pseudocodewords and how they contribute to the error-floor in iterative decoding. At about the same time, the authors of [13,14] found that the error-floor of LDPC codes over the binary erasure channel (BEC) is caused by graphical structures called stopping sets. Later, Richardson [15] related the error-floor of LDPC codes on the AWG channel to another graphical structure called trapping sets, and in his research successfully predicted the error-floor for the Margulis codes and other LDPC codes.

J. Thorpe in [19] introduces a new genre of LDPC codes, in which he uses a Tanner graph with relatively small number of nodes, called a protograph, as a template to construct the code Tanner graph: (as discribed earlier) multiple copies of the protograph are made and the edges are re-connected per some deterministic strategy among the replicated nodes. This idea rapidly caught the attention of other researchers, not only because it provides a compact way for describing the code, but also as the performance of the protograph-based code can be predicted from its protograph, and it has a modular decoder. Many examples of protograph-based codes can be found in literature, among them, the accumulate repeat accumulate (ARA) codes proposed in [20], and the quasi-cyclic repeat-accumulate codes in [21].

Generalized LDPC codes (G-LDPC) codes [5] also can be represented by a Tanner graph. The difference from LDPC code’s graph is that the single parity-check nodes are replaced by a general constraint nodes (CNs), which can be any code, such as a Hamming code, a recursive systematic convolutional code, or even an LDPC
code. Tanner’s G-LDPC codes were investigated by several researchers in recent years. In [22, 23] Hamming component codes were used in regular Tanner graphs. In [24] codes were designed by using BCH or Reed-Solomon code constraints and in [25] constraints based on recursive systematic convolutional (RSC) codes were used. Hybrid and irregular G-LDPC codes were investigated in [26–28]. In each of these works Hamming constraints were used. In [27, 28], G-LDPC codes, called Hamming-doped LDPC codes (HD-LDPC), which allow more than one CNs type in the graph and permit irregular node degrees, were proposed.

1.3 Dissertation contributions and motivation

The design of protograph-based LDPC and G-LDPC codes relies on what is effectively a computer-based search. As such, following Gallager, it is prudent to restrict the search to a “good ensemble”, for example, an ensemble whose minimum distance grows linearly with code length. A good ensemble can also mean one with good stopping set, trapping set, or pseudocodeword properties. In this dissertation, we derive ensemble codeword weight enumerators for finite-length G-LDPC codes based on protographs, and then we consider the asymptotic case. The asymptotic results allow us to determine whether or not the typical minimum distance in the ensemble grows linearly with codeword length. We then show how the codeword weight enumerator technique may be simply adapted to yield stopping set enumerators for such protograph G-LDPC codes. In this case, the asymptotic results allow us to determine whether or not the typical smallest stopping set size grows linearly with codeword length. We then again leverage our method for finding weight enumerators to compute finite-length ensemble trapping set enumerators for protograph-based LDPC codes. The asymptotic results are also derived which allow us to determine whether or not the typical smallest trapping set size grows linearly with codeword length. We also developed an algorithm for enumerating small size trapping sets for specific LDPC code. Trapping set enumerators for G-LDPC codes represents a more complex problem which we do not consider here. Moreover, we
derive the finite-length and the asymptotic ensemble pseudocodeword enumerators for protograph-based G-LDPC codes. The asymptotic result indicates whether or not the typical pseudocodeword weight grows linearly with codeword length.

Finally, based on interest from NASA Goddard space flight, in this dissertation we study coded modulation schemes with LDPC codes and pulse position modulation (LDPC-PPM) over the free-space optical channel. We present three different decoding schemes and compare their performances. These schemes are non-iterative LDPC-PPM, iterative LDPC-PPM, and iterative-accumulated PPM decoding schemes. We also developed a new density evolution tool for use in the design of LDPC codes with good performances over this channel.

1.4 Dissertation outline

The dissertation outline is as follows:

- Chapter 2: This chapter explains in detail the concept of protograph-based G-LDPC codes. It also shows how to design protograph-based G-LDPC codes by doping good LDPC protographs with Hamming nodes or recursive systematic convolutional (RSC) nodes. It then presents several protograph-based G-LDPC design examples. Moreover, it discusses several iterative decoders which can be used in the decoding of G-LDPC codes, and it analyzes the complexity of these decoders.

- Chapter 3: This chapter derives ensemble codeword weight enumerators for protograph-based LDPC and G-LDPC codes for both finite-length and infinite-length code ensembles. It then presents a method for computing stopping set enumerators for protograph-based LDPC and G-LDPC codes. It also derives trapping set enumerators for protograph-based LDPC codes ensemble. Furthermore, it derives ensemble pseudocodeword G-LDPC enumerators. The chapter presents several illustrative examples.

- Chapter 4: This chapter proposes a method to find trapping set enumerators
for a specific LDPC code given its parity-check matrix. It also discusses the
complexity of the proposed algorithm. It then finds trapping set enumerators
for several LDPC codes from the communication standards.

• Chapter 5: This chapter presents three decoding schemes for decoding the
LDPC-PPM coded modulation scheme over the free space optical channel. It
also proposes a density evolution tool for use in the design of LDPC codes
with good performances over this channel.

• Chapter 6: This chapter concludes the dissertation.
CHAPTER 2

Protograph-Based G-LDPC Codes

2.1 Introduction

This chapter focuses on the design and decoding of protograph-based G-LPDC codes. Let us start with the definition of a protograph. A protograph [19, 29] is a relatively small bipartite graph, containing variable nodes (VNs) and generalized constraint nodes (CNs). Each VN in the protograph represents a different “type”. Similar for each CN in the protograph. From the protograph a larger graph can be obtained by the following copy-and-permute procedure. Each edge in the protograph is assigned a different type and then the protograph is copied \( N \) times, after which the edges of the same type among the replicas are permut ed and reconnected to nodes of the same type to which they were connected to obtain a single, large graph. Parallel edges are allowed in the protograph, but not in the derived graph.

Note that the copy-and-permute procedure described in the definition can be simply represented by replacing each node in the protograph with a vector of nodes of the same type and replacing each edge in the protograph with a bundle of (permuted) edges of the same type. This “vectorized” protograph is depicted in Fig. 2.1, where \( \vec{v}_i \) represents an \( N \)-vector of type-\( v_i \) VNs and similarly for \( \vec{c}_j \). The boxes \( \pi_e \) along each \( N \)-edge in Fig. 2.1 represents a permutation or adjacency matrix. For LDPC codes, the constraint nodes are simple SPC constraints, for G-LDPC codes the constraints may be more complex, such as Hamming constraints. It should be obvious from Fig. 2.1 that if the permutation matrices \( \pi_e \) are circulant, then the \( H \) matrix for the corresponding LDPC code is an array of circulant permutation matrices, so the code would be quasi-cyclic. Conversely, if an LDPC code is quasi-cyclic and has an \( H \) matrix that is an array of circulant permutation matrices, then the code is a protograph-based LDPC code. A similar comment holds for G-LDPC
codes [30], but it is not as readily seen.

Figure 2.1: Vectorized protograph.

After getting familiar with the general concept of protograph-based G-LDPC codes, in the following sections we study these codes in more detail. In Section 2.2, we show how to design protograph-based G-LDPC codes by doping good LDPC protographs with Hamming nodes or recursive systematic convolutional (RSC) nodes. Also in this section, we present several protograph-based G-LDPC design examples. In Section 2.3, we discuss several iterative decoders which can be used in the decoding of the G-LDPC codes, and analyze the complexity of the these decoders. In Section 2.4 we conclude this chapter.

2.2 Design of protograph-based G-LDPC codes via Hamming-node or RSCC-node doping

2.2.1 Overview of the Design Technique

Consider a G-LDPC protograph, $G_p = (V_p, C_p, E_p)$, where $V_p = \{v_1, v_2, \ldots, v_{n_v}\}$ is the set of $n_v$ VNs, $C_p = \{c_1, c_2, \ldots, c_{n_c}\}$ is the set of $n_c$ CNs, and $E$ is the set of edges. Denote by $q_{v_i}$ the degree of variable node $v_i$ and by $q_{c_j}$ the degree of constraint node $c_j$. Now consider the G-LDPC code constructed from a protograph $G_p$ by making $N$ replicas of $G_p$ and using circulant permutations, each of size $N$, to permute the edges among the replicas of the protograph. The graph of this G-LDPC code, $G = (V, C, E)$, has $n = N \cdot n_v$ variable nodes and $m_c = N \cdot n_c$ constraint nodes. The
connections between the set of variable nodes $V$ and constraint nodes $C$ is given by an $m_c \times n$ adjacency matrix $\Gamma$. For an LDPC code, the adjacency matrix $\Gamma$ and the parity-check matrix $H$ are identical. For a G-LDPC code, knowledge of the parity-check matrices of the component codes is also required. To describe how $H$ is constructed for the case of G-LDPC codes, it is convenient to choose an adjacency matrix $\Gamma$ in block-circulant form,

$$
\Gamma = \begin{bmatrix}
\Gamma^{(0)} \\
\Gamma^{(1)} \\
\vdots \\
\Gamma^{(n_c-1)}
\end{bmatrix},
$$

where $\Gamma^{(i)} = [\pi_{i,0} \; \pi_{i,1} \; \ldots \; \pi_{i,n_c-1}]$ and $\pi_{i,j}$ is either a $N \times N$ circulant permutation matrix or a $N \times N$ zero matrix. (If the protograph has parallel edges, then $\pi_{i,j}$ will be a sum of circulant permutation matrices.) We will call each row of permutation matrices a block row which we observe has $N$ rows and $n = N \cdot n_v$ columns. We note that there is one block row for each protograph constraint node. We note also that the number of nonzero permutation matrices in a block row is simultaneously equal to the degree of its corresponding protograph constraint node and the length of the node’s component code.

Since there is one matrix $H_i$ for each block row, we need only discuss the $i$-th block row. Let $H_i$ be $m_i \times n$. Then for each row in the $i$-th block row, replace the $n_i$ ones in the row by the corresponding $n_i$ columns of $H_i$. This expands the $i$-th block row from $N \times n$ to $(N \cdot m_i) \times n$. (For the special case of an SPC constraint node, $m_i = 1$ and the row block is not expanded.) Once this process has been applied to each block row, the resulting parity-check matrix $H$ for the Tanner code will be $(\sum_i N \cdot m_i) \times n$.

A process analogous to this one can be followed for the case when $\Gamma$ is not block circulant. However, when it is block circulant, the resulting matrix $H$ can also be put in a block-circulant form (thus, the G-LDPC code will be quasi-cyclic). To do
this, first observe that \( \mathbf{H} \) so constructed will have the form

\[
\mathbf{H} = \begin{bmatrix}
H^{(0)} \\
H^{(1)} \\
\vdots \\
H^{(m_c-1)}
\end{bmatrix}
\]

where \( H^{(i)} \) is the \( i \)-th expanded block row. To obtain the block-circulant form of \( \mathbf{H} \), we re-order the rows within each \( H^{(i)} \) by taking row 0, then row \( m_i \), then row \( 2m_i \), ..., then row \( (N-1)m_i \); then take row 1, then row \( m_i + 1 \), then row \( 2m_i + 1 \), ..., then row \( (N-1)m_i + 1 \); and so on. Symbolically, if we let \( s = 0, 1, ..., N \cdot m_i - 1 \) be a row index for \( H^{(i)} \), then the index for the re-ordered version of \( H^{(i)} \) is

\[
l = N(s \mod m_i) + \lfloor s/m_i \rfloor.
\]

Theoretically, in a G-LDPC code, a constraint node can represent any code and a G-LDPC code can include different codes for each constraint nodes. However, in this dissertation, we consider only Hamming or RSC constraint nodes in addition to the more common SPC constraint nodes. Because the parity-check matrices for SPC and Hamming codes are straightforward, we will instead discuss the parity-check matrices for (possibly punctured) rate-\((\kappa-1)/\kappa\) finite-length RSC codes. For a memory-\(\nu\), rate-\((\kappa-1)/\kappa\) RSC code with generator polynomials \( g_1(D) = g_{10} + g_{11}D + \ldots + g_{1\nu}D^\nu \), \( g_2(D) = g_{20} + g_{21}D + \ldots + g_{2\nu}D^\nu \), and \( g_\kappa(D) = g_{\kappa0} + g_{\kappa1}D + \ldots + g_{\kappa\nu}D^\nu \), the rational-form generator matrix is

\[
G_{\text{RSC}}(D) = \begin{bmatrix}
1 & 0 & \ldots & 0 & \frac{g_2(D)}{g_1(D)} \\
0 & 1 & \ldots & 0 & \frac{g_3(D)}{g_1(D)} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & 1 & \frac{g_\kappa(D)}{g_1(D)}
\end{bmatrix},
\]

the corresponding parity-check matrix is

\[
H(D) = \begin{bmatrix}
g_2(D) & \ldots & g_\kappa(D) & g_1(D)
\end{bmatrix}.
\]
Because we consider finite block lengths, the binary parity-check matrix for such a code is given by

\[
H = \begin{bmatrix}
g_{20} & \cdots & g_{k_0} & g_{10} & 0 & \cdots & 0 & 0 & \cdots \\
g_{21} & \cdots & g_{k_0} & g_{11} & g_{20} & \cdots & g_{k_0} & g_{10} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
g_{2\nu} & \cdots & g_{k_{\nu}} & g_{1\nu} & \vdots & \vdots & \vdots & g_{2\nu} & \vdots \\
0 & 0 & 0 & 0 & g_{2\nu} & \vdots & g_{1\nu} & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix},
\]

which is in contrast to what one might find in a textbook, where infinite block lengths are assumed. As an example, for the \((5, 7)_{8}\) RSC (in octal representation) with \(\bar{g}_1 = [101]\), and \(\bar{g}_2 = [111]\), the parity-check matrix for a block length of 10 is

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

To find the rate of a doped graph (defined later) with \(n\) variable nodes and \(m_c\) constraint nodes, note that each component-code contributes \((1 - R_i)n_i\) redundant bits, where \(n_i\) and \(R_i\) are the length and rate of the \(i^{th}\) component-code, respectively. Consequently, the total number of redundant bits in the code cannot exceed \(m = \sum_{i=1}^{m_c}(1 - R_i)n_i\), and so the number of information bits in the code will be at least \(n - m\). This implies that the code rate satisfies \(R_c \geq 1 - \frac{m}{n}\), with equality when the check equations are independent.

The parameters in standard LDPC code design which most affect code performance are the degree distributions of the node types, the topology of the graph (e.g., to maximize girth), and the minimum distance, \(d_{\text{min}}\). For the design of G-LDPC codes, decisions must also be made on the types and multiplicities of constraint nodes to be used. The choice of constraint node types and their multiplicities is dictated
by the code rate and complexity requirements. Regarding complexity, we consider only short Hamming codes for which the number of parity bits is \((1 - R_i)n_i \leq 4\) and only RSC codes for which the number of trellis states is at most eight. Note that this constraint on the Hamming code family limits the number of states in the time-varying BCJR trellis [31] to be at most 16.

As for LDPC codes, the topology of the graph for a Tanner code should be free of short cycles. Obtaining optimal or near-optimal degree distributions for the graphs of G-LDPC codes can proceed as for LDPC codes, using EXIT charts [12] [32], for example. Here, we instead follow the pragmatic design approach introduced in [27], [28], which starts with a protograph that is known to have a good decoding threshold and replaces selected SPC nodes with either Hamming or RSC nodes (in this sense we call it doping and the resulting graph is called doped graph). Although we provide no proof, the substitution of these more complex nodes tends to increase minimum distance as shown by simulations. Further, it leads to a smaller adjacency matrix since multiple SPC nodes are replaced by a single component code node. The implication of a smaller adjacency matrix is that short cycles and other deleterious graphical properties are more easily avoided.

2.2.2 Example Doped LDPC Code Designs

In this subsection, we present several protograph-based G-LDPC codes whose design relies on doping protographs with Hamming constraints and RSC constraints, and we present selected simulation results for these codes on the AWGN channel.

Example 1: Rate-1/6 RSC-LDPC Code (G-LDPC code with RSC CNs): In [30], the authors generated a rate-1/6 LDPC code by doping a rate-1/4 ARA protograph [20], with a \((6, 3)\) shortened Hamming code. The Hamming node had the effect of amplifying the minimum distance of the designed code [33]. The idea of adding a complex constraint node to amplify weight (hence, \(d_{\text{min}}\)) led us to consider RSC nodes, particularly since RSC codes produce large weight for low-weight inputs. Since a rate-1/2 RSC code can have any even length, we must consider in the design of an RSC-doped protograph what this length should be. Fig. 2.2 depicts
Figure 2.2: Rate-1/6 RSC-LDPC protograph.

a protograph with an unterminated \((6T, 3T)\) RSC component code, where \(T\) is a design parameter, so that the overall protograph has \(T\) inputs and \(6T\) outputs. The \(6T\) outputs are represented by all of the circles in Fig. 2.2, some of which are obscured; the RSC node in Fig. 2.2 has \(3T\) inputs and \(3T\) outputs. Notice that this figure contains \(T\) equivalent sub-protographs. In the copy-and-permute procedure, we ignore the fact that these were formerly protographs, and apply the copy-and-permute rules only to the overall protograph.

We point out that codes based on this protograph are turbo-like [34] in the sense that copies of the information bits are permuted and distributed over \(T\) accumulators, and then part of their outputs together with the remaining information bits are permuted and fed to the RSC code encoder. One major difference, however, is that the present code uses multiple short RSC code blocks rather than one or two long RSC code blocks.

We designed a \((600, 100)\) G-LDPC code based on the rate-1/6 HD-LDPC protograph in [30, 33] and three \((600, 100)\) RSC-LDPC codes based on the protograph in Fig. 2.2. All four codes have (pseudo-)random adjacency matrices. The three RSC-LDPC codes correspond to three different values of the parameter \(T\): \(T = 2\),
4, and 8. All of these codes were constructed using random permutations on the edges of their protographs, but two constraints are taken into consideration: the protograph structure and the girth of the graph. The progressive edge growth construction in [35] is used to give the required girth, which is eight for all loops that have only SPC nodes. On the other hand, loops that include Hamming or RSC nodes can be of length less than eight.

A comparison between the frame error rate (FER) curves of these codes and the random coding bound (RCB) is presented in Fig. 2.3. The iterative decoder described above was used, where BCJR decoders are used to decode the Hamming and RSC component codes. The maximum number of iterations is $I_{\text{max}} = 50$ and 20 error events were collected at each $E_b/N_0$ value on each curve, except for the point at 4.5 dB of the $T = 8$ RSC-LDPC curve where only three error events occurred during the decoding of $7.26 \times 10^8$ codewords. Note that the floor for almost every code is quite low, even though the code length is 600. Note also the lowest floor occurs for the $T = 8$ RSC-LDPC code, which shows no evidence of a floor down to FER $\approx 10^{-9}$. This code is about 1.3dB from the random coding bound at FER=$10^{-4}$.

Finally, a code designed on this protograph can be encoded using the encoder in Fig. 2.4. This encoder takes $k$ information bits as input, then repeat each bit 6 times. So there are $6k$ bits of which $2k$ bits are permuted then enters the parallel-to-serial converter (P/S) to enter the RSC encoder after that. The remaining $4k$ are divided into two sets each has $2k$ bits. After that each set is permuted, then each two consecutive bits are added together (So each set has $k$ bits at this stage). The two sets of bits enters a P/S, then accumulated using an accumulator, labeled $1/(1 + D)$, to produce $2k$ parity bits (these parities correspond to the degree-3 and degree-2 VNs directly after the SPC nodes in the protograph). Next, half of these $2k$ parity bits are permuted then passed to the S/P to enter the RSC encoder. The selection of half the $2k$ bits is done by the ‘0x’ puncturing (e.i., of each two bits pass the first and delete the second), labeled ‘PUN 0x’. Finally, the $3k$ bits at the output of the P/S enter the RSC encoder, which output only the RSC parity bits. Also the RSC encoder must be initialized to the all-zero state every $3T$ bits.
Figure 2.3: Frame error rate comparison between (600, 100) HD-LDPC code and RSC-LDPC codes at different $T$, $I_{max} = 50$. 
The encoder in Fig. 2.4 is simple in the sense that the RSC encoder and the accumulator are implemented using shift registers and few xor operations. So this encoder has very low computational complexity and very low power consumption. This with having very low error-floor make this code suitable for use in deep space applications.
Figure 2.4: Rate-1/6 RSC-LDPC encoder.
**Example 2:** Rate 1/4 RSC-LDPC Code: As depicted in Fig. 2.5, we can obtain a rate-1/4 RSC-LDPC protograph which resembles the rate-1/2 HD-LDPC protograph in [30, 33], with two different rate-1/2 RSC nodes (of length 48) used in place of the Hamming nodes. Note that the two RSC component-codes form 48 parity check equations, which necessitate the existence of 64 variable nodes in the protograph in order to achieve a rate-1/4 code. Moreover, the number of information bits among these 64 bits is 16 and each 64-bit word must satisfy these 48 check equations. In Fig. 2.5, we divided the variable nodes into eight similar groups (enclosed in the dash boxes), with six connections to each RSC code. Each group contains two information bits, $u_0$ and $u_1$, and six parity bits, $p_0$ to $p_5$, which are ordered in a sequence relevant to the decoder.

The rate-1/2 RSC component codes have two different polynomial sets; one has polynomials $(17, 15)_8$ and the other has polynomials $(3, 2)_8$. Assuming that both have unterminated trellises; the resultant code has rate 1/4. However, we have to terminate one of the two component codes to obtain good performance. (Terminating both of them also works, but at the cost of code rate.) In this code, the $(17, 15)_8$ RSC code trellis has been terminated. Since $\nu = 3$ and the rate is 1/2 for this component code, 6 code bits are related to these termination bits.

From Fig. 2.5, the last six bits of each of the RSC component codes include two information bits. Consequently, trellis termination process reduces the rate from
16/64 to 14/64. In order to obtain rate 1/4, we puncture eight of the 64 bits, four degree-one variable nodes from each RSC code.

Finally, we constructed a \((16352,4088)\) RSC-LDPC code by making 292 copies of the above protograph. A block-circulant adjacency matrix was used in our simulations and so this code is QC. In summary, \(n = 18688\), \(m_c = 584\), \(n_c = 2\), and \(N = 292\). The adjacency matrix \(\Gamma\) is shown in Fig. 2.6.

Figure 2.6: Adjacency matrix of the \((16352,4088)\) RSC-LDPC code, and a magnified view of the first 2336 columns.

The performance of this rate-1/4 RSC-LDPC code \((I_{max} = 20)\) is presented in Fig. 2.7. Its performance is compared to that of the quasi-cyclic repeat-accumulate code (QCRA) in [21] as well as the random coding bound. The curves show that our code is superior to the QCRA code at low \(E_b/N_0\) values. But at higher \(E_b/N_0\) values, the QCRA code has a slightly better FER than the RSC-LDPC. We noticed that by increasing \(I_{max}\) from 20 to 50 in RSC-LDPC code, the FER at \(E_b/N_0 = 0.8\) dB reduced to around \(2 \times 10^{-6}\).

□

Example 3: Consider the rate-1/2 G-LDPC protograph with RSC constraints in Fig. 2.8. It consists of two rate-2/3 RSC component codes. In Fig. 2.8 we divided the variable nodes into \(T\) identical groups (enclosed in the dashed boxes). Each group contains two information bits \(i_1, i_2\), and two parity bits, \(p_1\) (for RSC on left) and \(p_2\) (for RSC on right). Note that the RSC component codes each have
Figure 2.7: Performance of (16352, 4088) RSC-LDPC code compared to that of (16384, 4096) QCRA code. $I_{\text{max}} = 20$. 
blocklength $3T$. This code is a turbo-like code, and in fact a rate-1/2 turbo code can be obtained from this protograph using a large $T$ and only one copy of the protograph. However, in our codes we used a comparatively small value for $T$ and a large number of protograph copies $N$. This allows the use of a modular decoder with reasonable complexity and hardware requirements.

![Protograph of rate-1/2 G-LDPC with two rate-2/3 RSC constraints.](image)

A rate-1/2 $(8160, 4080)$ G-LDPC code can be constructed from the protograph in Fig. 2.8 as follows. First, we use tail-biting RSC component codes with memory $v = 4$, blocklength $3T = 60$, and generator polynomials $(32, 36, 31)_8$. It follows that $T = 20$ and each protograph has 40 information bits and 40 parity bits. Next we make $N = 102$ replicas of the protograph. By choosing the adjacency matrix $\Gamma$ of the code to be in block-circulant form, the code will be quasi-cyclic because the component codes are tail-biting codes.

The performance of the $(8160, 4080)$ G-LDPC code based on the protograph in Fig. 2.8 is shown in Fig. 2.9. The maximum number of decoding iterations used was $I_{\text{max}} = 50$. The performance is compared to the random coding bound (RCB) [36] for $(8160, 4080)$ codes and it is seen that it is about 0.3 dB from the RCB at a frame error rate (FER) of $7 \times 10^{-4}$. The FER curve has an error floor near $10^{-4}$. However, its bit error rate (BER) curve shows that it is very good in both the waterfall and floor regions. Compared to the BER of the $(8192, 4096)$ AR4JA code [37], the BER of this code has a gain of about 0.3 dB down to $\text{BER} = 10^{-7}$.

□
Example 4: Another rate-1/2 G-LDPC code has the protograph in Fig. 2.10. We started with the rate-1/2 AR4JA protograph in [37] and we replaced each rate-5/6 SPC node in the AR4JA protograph with a rate-5/6 RSC component code of blocklength $6T$. Note that the protograph in Fig. 2.10 contains $T$ equivalent AR4JA sub-protographs, where each sub-protograph contains two information bits, $i_1$ and $i_2$, and three parity bits, $p_1$, $p_2$, and $p_3$. The overall protograph corresponds to a rate-2/5 G-LDPC code. In order to achieve rate-1/2, we puncture $i_1$. The RSC component code has the generator polynomials $(25, 37, 27, 31, 23, 35)_8$ and memory $\nu = 4$.

We designed a rate-1/2 (576, 288) G-LDPC code based on the protograph in Fig. 2.10 with $T = 8$. The number of information bits in the protograph is 16. To obtain $k = 288$, we made $N = 18$ copies of the protograph.
Figure 2.10: Protograph of rate-1/2 G-LDPC with rate-5/6 RSC constraints.

In Fig. 2.11 we present the performance (with $I_{\text{max}} = 10$) of the (576, 288) G-LDPC code based on the protograph in Fig. 2.10. The FER curve is about 1.8 dB from the RCB and has no floor down to $10^{-4}$, which is excellent considering its short blocklength.

In Fig. 2.11, we also show that unequal error protection can be facilitated by the generalized constraints of the G-LDPC codes. Note that there is a factor of five difference in the bit error rates of the information bits of type $i_1$ and those of type $i_2$.

We will counter few more examples on protograph-based G-LDPC codes in the following chapters. Now, let us move to the study of some G-LDPC decoding algorithms in the following section.

2.3 Protograph-based G-LDPC decoders

In general, for LDPC codes in this dissertation, we used the standard sum-product algorithm (SPA). For the G-LDPC codes which have more complex constraint nodes,
a soft-input soft-output (SISO) decoder is used to compute the soft-output messages. The choice of the SISO decoder for non-SPC constraint codes depends on the code type. For RSC codes we use the BCJR decoder [38].

In HD-LDPC codes, the Hamming constraints can be replaced by their equivalent SPC equations. However, except for the (6,3) shortened Hamming code, the large number of 4-cycles the resultant graph degrades the performance of the SPA decoder. Alternatively, for the Hamming nodes, we can use the BCJR decoder applied to the BCJR trellis [31]. We also consider the modified Chase algorithm [39] and the cyclic-2 pseudo-maximum likelihood (PML) decoder [40].

The modified Chase and cyclic-2 PML decoders are both SISO list-based decoders. Cyclic-2 has an advantage over modified Chase in term of complexity as it uses a list that refers to nearby codewords, which are independent of its input,
resulting in fewer addition operations. The complexity reduction factor from using either of these decoders instead of the BCJR decoder depends on the number of the states in the code’s trellis. As an example, in the decoding of $10^7$ codewords of the (32, 26) extended Hamming code, we observed that the cyclic-2 decoder was 9 times faster than BCJR decoder, and the modified Chase decoder was 4.5 times faster than BCJR.

To gain insight on the decoding complexity of the HD-LDPC and RSC-LDPC codes compared with that of standard regular LDPC we consider the following rate 1/6 codes. The first is an HD-LDPC code constructed from $W$ copies of the rate-1/6 HD-LDPC protograph in [30,33]. The second is an RSC-LDPC code based on one copy of the protograph in Fig. 2.2, using the RSC code polynomials $(5,7)_8$. The last code is an LDPC code derived from the previous HD-LDPC code, where the Hamming constraint is replaced by its SPC constraints.

The number of additions per iteration are $131W$, $50W$, and $20W$ for HD-LDPC, RSC-LDPC, and LDPC codes, respectively. This calculation is based on the following ($\eta$ is the relevant block length, $N_{s,total}$ is the total number of trellis states in the finite-length trellis): (1) For a standard LDPC codes, the number of additions equals to the number of ones in its parity-check matrix. (2) The number of additions in the HD-LDPC BCJR is given by $2N_{s,total} + 4\eta$. (3) For the RSC-LDPC BCJR, the number of additions is $2N_{s,total} + 5\eta/2$ because the number of stages in a rate-1/2 RSC trellis is half the block length, but it has to compute two values at each stage; hence, 5 instead of 4.

Lastly, we examined the performance of a (2048, 1024) HD-LDPC code, constructed from the (32,26) extended Hamming code, using the BCJR decoder, the Chase decoder (radius 6), and the cyclic-2 PML decoder. Note in Fig. 2.12 that the performance curves of the modified Chase and the BCJR decoders are almost the same, and about 0.5dB better than that of the cyclic-2 PML decoder. On the other hand, cyclic-2 PML decoder is about twice as fast as the Chase decoder and about nine times as fast as the BCJR decoder.
Figure 2.12: Comparison between the performance of BCJR decoder, and the other sub-optimal decoders. The (2048, 1024) HD-LDPC components are (32, 26) extended Hamming codes.
2.3.1 Rate-$(\kappa - 1)/\kappa$ RSC code SISO decoder

One can use the BCJR decoder on the trellis of the rate-$(\kappa - 1)/\kappa$ code as a SISO decoder. However, the complexity of the BCJR decoder [38] for high-rate RSC nodes is prohibitive. Consequently, in this subsection we present a variation of Riedel’s decoder [41] (described in equation (20) in [41]) which uses the trellis of the reciprocal-dual code. As an example of a reciprocal-dual code, consider the rate-4/5 RSC code with (octal) generator polynomials $(7, 2, 6, 5, 7)$ and memory $\nu = 2$. Its parity check matrix is given by
\[ H(D) = [1 + D + D^2 D 1 + D 1 + D^2 1 + D + D^2]. \]
The reciprocal-dual code is generated by
\[ G_{rd}(D) = D^2H(D^{-1}) = [1 + D + D^2 D D + D^2 1 + D^2 1 + D + D^2]. \]
In other words, the trellis of the reciprocal-dual code in this example is generated by the feedforward convolutional code generators $(7, 2, 3, 5, 7)$.

The decoder in this paper is a log-domain (additive) version of Riedel’s decoder. Before describing it, we introduce our adopted notation. Let us consider a rate-$(\kappa - 1)/\kappa$ RSC code, $C$, with block length $N_{\text{RSC}}$, and memory $\nu$. Its reciprocal-dual code has the same blocklength and memory, but its rate is $1/\kappa$. The trellises of both the original and the reciprocal-dual code have $\Lambda = N_{\text{RSC}}/\kappa$ trellis sections, each with $2^\nu$ left-states and $2^\nu$ right-states. The original code has $2^{\kappa - 1}$ branches leaving/entering each state and the reciprocal-dual code has 2 branches leaving/entering each state.

We use $s_l$ and $s_r$ to refer to a left-state and a right-state, respectively. The reciprocal-dual code encoder output associated with the transition from $s_l$ to $s_r$ is denoted by the $\kappa$-tuple $\bar{b}(s_l, s_r) = (b_0(s_l, s_r), \ldots, b_{\kappa - 1}(s_l, s_r))$. Also, define the two sets $S_L(s_r)$ and $S_R(s_l)$ as follows: $S_L(s_r) = \{ s_l : s_l \rightarrow s_r \text{ exists} \}$ and similarly, $S_R(s_l) = \{ s_r : s_l \rightarrow s_r \text{ exists} \}$. Denote the transmitted codeword $\bar{c} \in C$ by $\bar{c} = (c_0, c_1, \ldots, c_{N_{\text{RSC}}-1})$ and the corresponding channel output by $\bar{y} = (y_0, y_1, \ldots, y_{N_{\text{RSC}}-1})$.

Riedel’s decoder can be summarized by the log-likelihood ratio $L(c_i)$ of the $i$-th
bit in $\bar{c}$ given by

$$L(c_i) = L(y_i | c_i)$$

$$+ \ln \left| \sum_{s_l} \sum_{s_r} A_t(s_l) \Theta_t(i, s_l, s_r) B_{t+1}(s_r) \right|$$

$$- \ln \left| \sum_{s_l} \sum_{s_r} A_t(s_l) \Theta_t(i, s_l, s_r) B_{t+1}(s_r) \right|$$

$$(2.4)$$

where the $s_l$ summations are over all $s_l \in \{0, ..., 2^v - 1\}$ and the $s_r$ summations are over all $s_r \in S_R(s_l)$. The soft output from the channel $L(y_i | c_i) = 2y_i/\sigma^2$, where $\sigma^2$ is the AWGN variance. The forward, backward, and branch metrics $A_t(s_l)$, $B_{t+1}(s_r)$, and $\Theta_t(i, s_l, s_r)$, respectively, are defined in [41], where $t$ designates a trellis section and is related to $i$ by $i = [i/\kappa]$.

We use the convention in [42] to derive the additive version of the above decoder. Let $X = \chi_s e^{\chi_m}$ and $Z = \zeta_s e^{\zeta_m}$ be real numbers where $\chi_s \in \{\pm 1\}$ and $\zeta_s \in \{\pm 1\}$ represent signs and $\chi_m \in \mathbb{R}$ and $\zeta_m \in \mathbb{R}$ represent logarithms of the magnitudes. We define the one-to-one mapping

$$X \Rightarrow \chi = (\chi_m, \chi_s) \triangleq (\ln |X|, \text{sign}(X)),$$

and similarly for $Z$. From this definition, the product $XZ$ maps to

$$\chi + \zeta \triangleq (\ln |XZ|, \text{sign}(XZ)) = (\chi_m + \zeta_m, \chi_s \zeta_s)$$

and the sum $X + Z$ maps to

$$\max^*(\chi, \zeta) \triangleq (\ln |X + Z|, \text{sign}(X + Z)) = \left( \max(\chi_m, \zeta_m) \right.$$ 

$$+ \ln \left| 1 + \chi_s \zeta_s e^{-|\chi_m - \zeta_m|} \right|, \max^m(\chi_s, \zeta_s) \Big),$$

where $\max^m(\chi_s, \zeta_s)$ equals $\chi_s$ if $\chi_m > \zeta_m$, and equals $\zeta_s$, otherwise. Now we can rewrite (2.4) as follows

$$L(c_i) = L(y_i | c_i) + \gamma_m^{(1)}(y_i, c_i) + \gamma_m^{(2)}(y_i, c_i).$$

$$(2.5)$$
where $\gamma^{(1)}(y_i, c_i)$, and $\gamma^{(2)}(y_i, c_i)$ are given in the following two equations:

$$
\gamma^{(1)}(y_i, c_i) = \max_{s_l} \left\{ \max_{s_r} \{ \alpha_t(s_l) + \vartheta_t(i, s_l, s_r) + \beta_{t+1}(s_r) \} \right\},
$$

$$
\gamma^{(2)}(y_i, c_i) = \max_{s_l} \left\{ \max_{s_r} \{ \alpha_t(s_l) + \vartheta_t(i, s_l, s_r) + \beta_{t+1}(s_r) + (0, (-1)^{b_{t-\tau}(s_l, s_r)}) \} \right\},
$$

where $s_l$ is taken over all $s_l \in \{0, ..., 2^\nu - 1\}$ and $s_r$ is taken over all $s_r \in S_R(s_l)$. Also, $\alpha_t(s_l)$, $\beta_t(s_r)$, and $\vartheta_t(i, s_l, s_r)$ are given by

$$
\alpha_t(s_r) = \max_{s_l \in S_L(s_r)} \{ \alpha_{t-1}(s_l) + \vartheta_{t-1}(s_l, s_r) \},
$$

$$
\beta_t(s_l) = \max_{s_r \in S_R(s_l)} \{ \beta_{t+1}(s_r) + \vartheta_{t+1}(s_l, s_r) \},
$$

and

$$
\vartheta_t(i, s_l, s_r) = \left( \sum_{j=0}^\kappa b_j(s_l, s_r) \ln \left| \tanh \left( \frac{L(y_{\kappa+j} | c_{\kappa+j})}{2} \right) \right| \right) \cdot \prod_{j=0, j \neq i-\tau}^\kappa \text{sign} \left( \frac{L(y_{\kappa+j} | c_{\kappa+j})}{2} \right).
$$

Finally, $\vartheta_t(s_l, s_r)$ in (2.8), and (2.9) is given by

$$
\vartheta_t(s_l, s_r) = \left( \sum_{j=0}^\kappa b_j(s_l, s_r) \ln \left| \tanh \left( \frac{L(y_{\kappa+j} | c_{\kappa+j})}{2} \right) \right| \right) \cdot \prod_{j=0, j \neq i-\tau}^\kappa \text{sign} \left( \frac{L(y_{\kappa+j} | c_{\kappa+j})}{2} \right).
$$

The decoder requires initialization of the forward and backward recursions. The initializations depend on the type of the RSC codes employed, as follows. For truncated RSC codes, set $\alpha_0(s)$ to $(0, 1)$ for all states $s$, and set $\beta_\Lambda(s)$ to $(0, 1)$ for the zero state, and $(-\infty, 1)$ for the other states. On the other hand, for terminated RSC codes set both $\alpha_0(s)$ and $\beta_\Lambda(s)$ to $(0, 1)$ for all states. The idea behind
this initialization is that the code and its dual are related by a Fourier transform relation [43].

Similar to the above procedure, one can easily derive the optimal additive version of Riedel’s tail-biting decoder. However, the complexity of this optimal decoder is $2^\nu$ times the complexity of the decoder in (2.5). Instead, we used a suboptimal decoder analogous to that in [44]. The idea is to find “correct” initializations, $\alpha_0(s)$ and $\beta_\Lambda(s)$, then run the decoder in (2.5). Making use of the circular form of the tail-biting trellis, Anderson [44] showed that the forward and backward recursions define an eigenvector problem whose solution is the desired initialization. Moreover, he showed that starting with any random forward initialization, then iterating the forward recursion in (2.8) enough times with a proper normalization, the forward initialization converges to the “correct” initialization. In a similar way, the backward initialization can be found. The approximate additive tail-biting decoder is summarized as follows:

- For $s = 0, 1, \ldots, 2^\nu - 1$, set $\alpha'_0(s) = (0, 1)$, if $s = 0$, and $(-\infty, 1)$, if $s \neq 0$, where $\alpha'_t(s)$ refers to a normalized $\alpha_t(s)$.

- Find a set of normalized vectors $\alpha'_1, \alpha'_2, \ldots, \alpha'_\Lambda$, where $\alpha'_t = (\alpha'_t(0), \alpha'_t(1), \ldots, \alpha'_t(2^\nu - 1))$, as follows: First use (2.8) to find $\alpha_t$. But $\alpha_{t-1}(s_l)$ on the right side of (2.8) is replaced by $\alpha'_{t-1}(s_l)$. Then, $\alpha'_t = \alpha_t - \alpha_t(0)$. Continue this recursion to find $\alpha'_t$, $t = \Lambda + 1, \Lambda + 2, \ldots$ with $\vartheta_t(s_l, s_r) = \vartheta_\tau(s_l, s_r)$ and $\tau = (t \mod \Lambda)$. Stop, when $\|\alpha'_t - \alpha'_{t-\Lambda}\|$ is sufficiently small, or after a preset number of rounds (we used three rounds). Now, the forward recursion initialization is given by $\alpha_0 = \alpha'_t$.

- Use a similar procedure to find the backward recursion initialization $\beta_\Lambda$.

- Run the decoder in (2.5).
2.4 Conclusion

In this chapter, we studied the design of structured G-LDPC codes with low error-rate floors on the AWGN channel. The design technique involves the doping of standard LDPC protographs with Hamming or RSC code constraints. We showed that the doping of a good graph with Hamming or RSC codes is a pragmatic approach that frequently results in a code with a good threshold and very low error-rate floor. Moreover, we proposed several decoders which can be used in the decoding of the complex constraint nodes of G-LPDC codes. These decoders compromise between complexity and performance.
CHAPTER 3

Protograph-Based G-LDPC Codes enumerators

3.1 Introduction

Codeword weight enumerators for specific codes are useful for bounding or estimating the performance of maximum-likelihood (ML) decoders of channel codes. As noted by Gallager in his seminal monograph on LDPC codes [2], it is generally impractical to calculate the weight enumerator for a specific code. Given this, Gallager and others have calculated the average enumerators for ensembles of codes, from which the average performance of ML decoders may be estimated. In many cases, it is often easier to calculate the asymptotic weight enumerators, that is, the weight enumerators for code ensembles with code length tending to infinity. The asymptotic results allow us to make statistical statements about the distance properties of code ensembles (particularly, the minimum distance) and on the minimum operable signal-to-noise-ratio (i.e., SNR threshold) for maximum-likelihood decoders. For example, a “good ensemble” is one for which the minimum distance grows linearly with code length. In the design of codes, it is useful to know beforehand whether or not the design space resides within a good ensemble of codes.

Gallager derived asymptotic codeword weight enumerators for the “Gallager ensembles” of regular LDPC codes in [2]. This result was extended to the irregular LDPC ensembles in [45–48]. In [34, 49], ensemble weight enumerators for serially concatenated codes and turbo-like codes were derived. Asymptotic weight enumerators for ensembles of protograph-based LDPC codes were computed in [50,51]. In the latter paper, Divsalar first derived the ensemble weight enumerators for finite-length protograph-based LDPC codes [19] and then obtained the asymptotic results by letting the code length go to infinity. Divsalar also extended the enumerator technique in [51] to stopping set enumerators in [52]. Analogous to the role of ensemble weight
enumerators in determining average code performance on a Gaussian channel with
ML decoding, stopping set enumerators determine the performance of LDPC codes
under iterative decoding on the binary erasure channel (BEC) \[13, 14, 53\] and the
burst-erasure channel. A generalization of the notion of a stopping set, called a gen-
eralized stopping set, was introduced in [24] and used to evaluate the performance
of G-LDPC codes with a single constraint type.

Whereas stopping sets generate error-rate floors for iterative decoders on the
BEC, trapping sets do likewise for the AWGN channel [15, 54–57]. In [15], Richard-
son established the notion of trapping sets and suggested a two-step technique for
predicting the performance of LDPC codes in the error-floor region. The authors
of [54, 55] used a similar technique to predict the floor of several LDPC codes on the
binary symmetric channel (BSC). In [56], the authors derived the average asymptotic behavior of trapping set enumerators over random, regular and irregular, LDPC
code ensembles. After the dominant trapping sets are found [15], the trapping set
enumerators allow one to estimate the average performance of LDPC code ensembles
assuming iterative decoders.

In an attempt to explain the deviation of iterative decoders from optimal de-
coders for LDPC codes, Frey et al. [16] introduced the notion of a pseudocodeword.
Then Forney et al. in [17] suggested that an iterative decoder’s performance can
be predicted using the pseudocodeword weight enumerators of the code rather than
codeword weight enumerators. They also computed the effective weight of pseu-
docodewords for the AWGN, BEC, and BSC channels. Moreover, Koetter et al. [18]
used the idea of a graph cover to show how pseudocodewords determine iterative
decoder convergence to non-codeword patterns, and how an iterative decoder can-
not distinguish between the graph and its covers. More recently, Kelley provided
in [58] a detailed study of pseudocodewords of Tanner graphs. Kelley also derived
upper bounds on the minimum pseudocodeword weight of a specific LDPC code
for the AWGN, BEC, and BSC channels. Also, Chertkov and Stepanov [59] pro-
posed a pseudocodeword search algorithm to estimate the pseudocodeword weight
enumerator of a specific LDPC code.
Generalized LDPC (G-LDPC) codes, for which more complex constraints than single parity-check (SPC) constraints are permissible, were first proposed by Tanner [5]. G-LDPC codes based on protographs were studied in [27, 28, 30, 33, 60] where certain advantages are demonstrated, such as a simpler graphical structure that is less prone to cycles and their impact on iterative decoding. In this paper, we first present a straightforward technique for obtaining codeword weight enumerators for protograph-based G-LDPC codes for both finite-length and infinite-length ensembles. The asymptotic results allow us to determine whether or not the typical minimum distance in the ensemble grows linearly with codeword length. We then show that it is a simple matter to adapt the codeword weight enumerator technique to yield stopping set enumerators for protograph-based G-LDPC codes. In this case, the asymptotic results allow us to determine whether or not the typical smallest stopping set size grows linearly with codeword length. Next, we show that one may again leverage the weight enumerator technique to obtain finite-length ensemble trapping set enumerators for protograph-based LDPC codes. The asymptotic results are also derived, which allow us to determine whether or not the typical smallest trapping set size grows linearly with codeword length. We do not provide trapping set enumerators for G-LDPC codes as this is a more complex problem which does not fit well with the rest of the paper. Lastly, we derived the finite-length and the asymptotic ensemble pseudocodeword weight enumerators for protograph-based G-LDPC code ensembles. The asymptotic result indicates whether or not the typical pseudocodeword weight grows linearly with codeword length. The results in this paper allow us to identify good G-LDPC and LDPC code ensembles, particularly for erasure and AWGN channels, as demonstrated by several examples.

This chapter proceeds as follows: In Section 3.2, the notation used in the paper is defined, and ensemble codeword weight enumerators for protograph-based LDPC and G-LDPC codes are derived for both finite-length and infinite-length code ensembles. In Section 3.3, the stopping set enumerator technique for protograph-based LDPC and G-LDPC codes is presented. In Section 3.4, trapping set enumerators for protograph-based LDPC codes ensemble is derived. In Section 3.5, ensemble pseu-
docodeword G-LDPC enumerators are derived. Illustrative examples are included in each of these sections. Finally, a summary and conclusion is given in Section 3.6.

3.2 Ensemble weight enumerators for protograph-based G-LDPC codes

A protograph [19, 29] is a relatively small bipartite graph, containing variable nodes (VNs) and generalized constraint nodes (CNs), from which a larger graph can be obtained by the following copy-and-permute procedure. Each edge in the protograph is assigned a different “type” and then the protograph is copied \( N \) times, after which the edges of the same type among the replicas are permuted and reconnected to obtain a single, large graph. Parallel edges are allowed in the protograph, but not in the derived graph.

Note that the copy-and-permute procedure described in the definition can be simply represented by replacing each node in the protograph with a vector of nodes of the same type and replacing each edge in the protograph with a bundle of (permuted) edges of the same type. This “vectorized” protograph is depicted in Fig. 3.1, where \( \bar{v}_i \) represents an \( N \)-vector of type-\( v_i \) VNs and similarly for \( \bar{c}_j \). The boxes \( \pi_e \) along each \( N \)-edge in Fig. 3.1 represents a permutation or adjacency matrix. For LDPC codes, the constraint nodes are simple SPC constraints, for G-LDPC codes the constraints may be more complex, such as Hamming constraints. It should be obvious from Fig. 3.1 that if the permutation matrices \( \pi_e \) are circulant, then the \( H \) matrix for the corresponding LDPC code is an array of circulant permutation matrices, so the code is quasi-cyclic. Conversely, if an LDPC code is quasi-cyclic and has an \( H \) matrix that is an array of circulant permutation matrices, then the code is a protograph-based LDPC code. A similar comment holds for G-LDPC codes [30], but it is not as readily seen.

In this section, a method for finding finite-length weight enumerators for protograph-based G-LDPC code ensembles is first derived. From these results, the asymptotic results are then obtained. Because G-LDPC codes contain LDPC codes as a special case, these results are directly applicable to protograph-based LDPC
Figure 3.1: Vectorized protograph.

codes and we provide examples of both LDPC and G-LDPC codes.

3.2.1 Finite-length ensemble weight enumerators

Consider a G-LDPC protograph, $G = (V, C, E)$, where $V = \{v_1, v_2, \ldots, v_{n_v}\}$ is the set of $n_v$ VNs, $C = \{c_1, c_2, \ldots, c_{n_c}\}$ is the set of $n_c$ CNs, and $E$ is the set of edges. Denote by $q_v_i$ the degree of variable node $v_i$ and by $q_c_j$ the degree of constraint node $c_j$. Now consider the G-LDPC code constructed from a protograph $G$ by making $N$ replicas of $G$ and using uniform interleavers [34, 49], each of size $N$, to permute the edges among the replicas of the protograph. Recall that a length-$L$ uniform interleaver [49] is a probabilistic device that maps each weight-$w$ input into the $L \choose w$ distinct permutations of it with equal probability, $1/\binom{L}{w}$. This approach allows ensemble average weight enumerators to be derived for various types of concatenated codes [34, 49]. As shown in [51], by exploiting the uniform interleaver concept, the average weight enumerator for a protograph-based LDPC ensemble can be obtained. To do so, the VNs and CNs are treated as constituent codes in a serial concatenated code (SCC) scheme as explained further below.

In the case of two serially concatenated constituent codes, $C_1$ and $C_2$, separated by a uniform interleaver, the average number of weight-$d$ codewords created by weight-$f$ inputs in the SCC ensemble is given [49] by

$$A_{f,d}^{SCC} = \sum_w A_{f,w}^{C_1} A_{w,d}^{C_2} \binom{N}{w}$$  (3.1)
where $A^C_{f,w}$ is the number of weight-$w$ codewords in $C_1$ corresponding to weight-$f$ inputs and $A^C_{w,d}$ is the number of weight-$d$ codewords in $C_2$ corresponding to weight-$w$ inputs. To apply this result to the G-LDPC code case, the group of $N$ VNs of type $v_i$ is considered to be a constituent (repetition) code with a weight-$d_i$ input of length $N$ and $q_{vi}$ length-$N$ outputs. Further, the group of $N$ CNs of type $c_j$ is considered to be a constituent code with $q_{cj}$ inputs, each of length $N$, and a fictitious output of weight zero. Now let $A(d)$ be the average (over the ensemble) number of codewords having weight-vector $d = [d_1, d_2, \ldots, d_{nv}]$ corresponding to the $n_v$ length-$N$ inputs satisfying the protograph constraints. $A(d)$ is the weight-vector enumerator for the ensemble of codes of length $N \cdot n_v$ described by the protograph. Let us further define

- $A^{v_i}(w_i) = (N_{d_i})^{\delta_{d_i,w_i,1}} \cdot \delta_{d_i,w_i,q_{vi}}$ is the weight-vector enumerator for the type-$v_i$ (VN) constituent code for a weight-$d_i$ input, where $w_i = [w_{i,1}, w_{i,2}, \ldots, w_{i,q_{vi}}]$ is a weight vector describing the constituent code’s output, and

- $A^{c_j}(z_j) = \pi_{l=1}^{n_c} A^{v_i}(w_i) \prod_{r=1}^{q_{vu}} A^{c_j}(z_j)$ is the weight-vector enumerator for the type-$c_j$ (CN) constituent code and $z_j = [z_{j,1}, z_{j,2}, \ldots, z_{j,q_{cj}}]$, where $z_{j,l} = w_{i,k}$ if the $l^{th}$ edge of CN $c_j$ is the $k^{th}$ edge of VN $v_i$.

The SCC result (3.1) is applied to each of the multiple concatenations that exist in the G-LDPC protograph to obtain the average protograph weight-vector enumerator as follows (see also [34])

$$A(d) = \sum_{w_{m,u}} \frac{\prod_{i=1}^{n_v} A^{v_i}(w_i) \prod_{j=1}^{n_c} A^{c_j}(z_j)}{\prod_{s=1}^{n_v} (N_{w_{s,u}})}$$

$$= \prod_{j=1}^{n_c} A^{c_j}(d_j) \prod_{i=1}^{n_v} (N_{d_i})^{q_{vi} - 1},$$

where the summation in the first line is over all weights $w_{m,u}$, where $w_{m,u}$ is the weight along the $u^{th}$ edge of VN $v_m$, where $m = 1, \ldots, n_v$ and $u = 1, \ldots, q_{vu}$. The vector $d_j = [d_{j_1}, d_{j_2}, \ldots, d_{j_{nc_j}}]$ is a weight vector which describes the weights of the $N$-bit words on the edges connected to CN $c_j$, produced by the VNs neighboring $c_j$. The elements of $d_j$ comprise a subset of the elements of $d$. 
Let \( S_t \) be the set of transmitted VNs and let \( S_p \) be the set of punctured VNs. Then the average number of codewords of weight \( d \) in the ensemble, denoted by \( A_d \), equals the sum of \( A(d) \) over all \( d \) for which \( \sum_{i \in S_t} d_i = d \). Notationally,
\[
A_d = \sum_{\{d_i \in S_t\}} \sum_{\{d_k \in S_p\}} A(d)
\]
(3.4)
under the constraint \( \sum_{i \in S_t} d_i = d \). To evaluate \( A_d \) in (3.4), one first needs to compute the weight-vector enumerators, \( A^c(d_j) \), for the constraint nodes \( c_j \), as seen in (3.3). This was done for SPC nodes in [51]. In this section, we extend this computation to constraint nodes which correspond to any linear block code.

Consider the constituent \((\mu, \kappa)\) linear block code \( C \). We need to find its weight-vector enumerator \( A^c(\mathbf{w}) \), where \( \mathbf{w} = [w_1, w_2, \ldots, w_\mu] \) is the weight vector at the input to the constituent code. Following [51], the \( \{A^c(\mathbf{w})\} \) may be easily found as the coefficients of the multi-dimensional z-transform of \( \{A^c(\mathbf{w})\} \). Exploiting the uniform interleaver property and the fact that the multi-dimensional z-transform of a single constraint node is \( \sum_{\mathbf{x} \in C} W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu} \), the multi-dimensional z-transform for \( N \) copies of the protograph is
\[
A^c(W_1, W_2, \ldots, W_\mu) = \left( \sum_{\mathbf{x} \in C} W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu} \right)^N,
\]
(3.5)
where the \( W_i \)'s are indeterminate bookkeeping variables and \( \mathbf{x} = [x_1, x_2, \ldots, x_\mu] \), \( x_i \in \{0, 1\} \), is a codeword in \( C \). Expanding the righthand side of (3.5) will yield the form
\[
A^c(W_1, W_2, \ldots, W_\mu) = \sum_{\mathbf{w}} A^c(\mathbf{w}) W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu},
\]
(3.6)
from which we may obtain \( A^c(\mathbf{w}) \). The direct application of the multinomial theorem on the righthand side of (3.5) gives
\[
A^c(W_1, W_2, \ldots, W_\mu) = \sum_{n_1, n_2, \ldots, n_\mu \geq 0} C (N; n_1, n_2, \ldots, n_\mu) \times \prod_{\mathbf{x} \in C} (W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu})^{n_\kappa}
\]
(3.7)
where \( K = 2^\kappa \) is the number of codewords in \( C \), \( n_k \) is the number of occurrences of the \( k^\text{th} \) codeword, and \( C (N; n_1, n_2, \ldots, n_K) \) is the multinomial coefficient, given by

\[
C (N; n_1, n_2, \ldots, n_K) = \frac{N!}{n_1! n_2! \ldots n_K!}.
\]

(3.8)

Let \( M^C \) be the \( K \times \mu \) matrix with the codewords of \( C \) as its rows and \( n = [n_1, n_2, \ldots, n_K] \). Then (3.7) can be written as

\[
A^C(W_1, W_2, \ldots, W_\mu) = \sum_\mathbf{n} \sum \{\mathbf{n}\} C (N; n_1, n_2, \ldots, n_K) \times W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu},
\]

(3.9)

where \( \{\mathbf{n}\} \) is the set of integer solutions to \( \mathbf{w} = \mathbf{n} \cdot M^C \), under the constraints \( n_1, n_2, \ldots, n_K \geq 0 \) and \( \sum_{k=1}^K n_k = N \). To see the last step, note that the product in (3.7) can be manipulated as follows

\[
\prod_{x \in C} (W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu})^{n_k} = W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu},
\]

(3.10)

where \( w_j = \sum_{x \in C} x_j n_k \), \( j = 1, 2, \ldots, \mu \). Also, if \( \mathbf{w} = \mathbf{n} \cdot M^C \) has more than one solution for \( \mathbf{n} \), the term \( W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu} \) will appear as a common factor in all of the terms that are associated with these solutions. This explains the presence of the second summation in (3.9). Finally, comparing (3.6) and (3.9) leads to the expression of the weight-vector enumerator,

\[
A^C(\mathbf{w}) = \sum_{\{\mathbf{n}\}} C (N; n_1, n_2, \ldots, n_K),
\]

(3.11)

where \( \{\mathbf{n}\} \) is the set of integer solutions to \( \mathbf{w} = \mathbf{n} \cdot M^C \), with \( n_1, n_2, \ldots, n_K \geq 0 \) and \( \sum_{k=1}^K n_k = N \). Appendix A presents a method for solving this system of equations.

**Example 5:** Consider the degree-4 SPC constraint node. The codeword set is

\[
C = \{0000, 1001, 0101, 1100, 0011, 1010, 0110, 1111\},
\]

and so \( K = 8 \).

(a) Consider \( N = 3 \) protograph copies and the weight-vector \( \mathbf{w} = [2, 2, 2, 2] \). From \( \mathbf{w} = \mathbf{n} \cdot M^C \) and the associated constraints on \( \mathbf{n} \), it is easy to see that \( \{\mathbf{n}\} = \{[0, 0, 0, 1, 1, 0, 0, 1], [0, 0, 1, 0, 0, 1, 0, 1], [0, 1, 0, 0, 0, 1, 0, 1], [1, 0, 0, 0, 0, 0, 0, 2]\}$. 

From this, \( A_C^{c}(w) = 21 \) (via (3.11)).

(b) When \( N = 4 \) and \( w = [4, 2, 2, 2] \), \( \{n\} = \{[0, 1, 0, 1, 0, 1, 0, 1]\} \) and so \( A_C^{c}(w) = 24 \).

(c) When \( N = 4 \), and \( w = [3, 2, 2, 2] \), \( \{n\} \) is empty and so \( A_C^{c}(w) = 0 \). \( \square \)

**Example 6:** Consider the protograph with a single \((7, 4)\) Hamming constraint node and 7 degree-1 VNs, all transmitted. Noting that the denominator in (3.3) is unity (since \( q_i = 1 \) for all \( i \)) and the numerator is \( A^{c1}(d_1) \) (since \( n_c = 1 \)), we will compute \( A_d \) for \( d = 0, 1, 2, 3 \), assuming \( N = 4 \) copies of the protograph. The Hamming code is generated by

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

from which the matrix \( M^c \) may be obtained; we assume the codewords are listed in the natural binary order with respect to the 16 input words. From (3.3) and (3.4), with \( n_v = 7 \) and \( n_c = 1 \), \( A_d = \sum_d A^{Ham}(d) \) under the constraint \( \sum d_j = d \). Thus, \( A_0 = A_{[0,0,0,0,0,0]} = C(4; 4, 0, \ldots, 0) = 1 \). \( A_1 = \sum_d A^{Ham}(d) \) under \( \sum d_j = 1 \), but any such \( d \) must result in \( \{n\} \) empty. Consequently, \( A_1 = 0 \). Similarly, one finds \( A_2 = 0 \). With \( A_3 = \sum_d A^{Ham}(d) \) such that \( \sum d_j = 3 \), \( d = \{[1,0,0,0,1,1], [0,1,0,0,1,1,0], [1,0,1,0,1,0,0], [0,1,1,0,1,0,0], [0,0,0,1,1,0,1], [1,1,0,1,0,0,0], [0,0,1,0,1,0,0]\} \). Each \( d \) yields only one solution \( n \) to the equation \( d = n \cdot M^c \). Each solution has \( n_1 = 3 \) together with \( n_k = 1 \), where \( k \) corresponds to a row in \( M^c \) containing one of the 7 weight-3 codewords. Note, the other \( d \)'s that achieve \( \sum d_j = 3 \) result in \( \{n\} \) empty. Using (3.3), (3.4), and (3.11), \( A_3 = \sum_d A^{Ham}(d) = \sum_{\{n\}} C(4; 3, 0, \ldots, 1, \ldots, 0) = 28 \). \( \square \)

Now we obtain weight-vector enumerators for protograph-based LDPC codes as a special case of G-LDPC codes.

First we compute \( A^c(w) \) for a check \( c \) (SPC) with degree 3. Define \( u = \frac{1}{2}(w_1 + w_2 + w_3) \). If \( w_1 + w_2 + w_3 \) is even and \( \max\{w_1, w_2, w_3\} \leq u \leq N \), then using (3.11)
with \( K = 4 \) we obtain

\[
A_c(w_1, w_2, w_3) = \frac{N!}{(N - u)!(u - w_1)!(u - w_2)!(u - w_3)!}; \quad (3.12)
\]

otherwise \( A_c(w_1, w_2, w_3) = 0 \). It is interesting to note that the inequality

\[
\max\{w_1, w_2, w_3\} \leq u
\]

is equivalent to a set of inequalities \( w_i < \sum_{j \neq i} w_j \). This set relates to cocircuit inequalities in matroid theory [18]. Computation of \( A_c(w) \) for a check \( c \) with degree 4 can be obtained from the result for a check with degree 3 by concatenation. For example \( A_c(w_1, w_2, w_3, w_4) \) can be obtained as

\[
A_c(w_1, w_2, w_3, w_4) = \sum_{l=1}^{N} \frac{A(w_1, w_2, l)A(w_3, w_4, l)}{N \choose l}; \quad (3.13)
\]

The weight enumerators for higher degree checks can be obtained in a similar way. This method of weight enumerator computation for check nodes suggests that, for a check \( c \) with degree \( q_c \), one can split the check into \( q_c - 2 \) checks each with degree 3 that are connected by \( q_c - 3 \) degree 2 untransmitted (punctured) variable nodes, as shown in Fig. 3.2 for \( q_c = 6 \).

Figure 3.2: Check split method to compute enumerators for checks nodes with higher degrees

Thus, for the purpose of weight enumeration we can modify the original protograph to a protograph with \( n'_c = \sum_{j=1}^{n_e}(q_{c_j} - 2) \) degree 3 check nodes and
\( n'_v = n_v + \sum_{j=1}^{n_c} (q_{c_j} - 3) \) variable nodes; that is, an additional \( \sum_{j=1}^{n_c} (q_{c_j} - 3) \) untransmitted degree 2 variable nodes are added to the number of variable nodes in the original protograph. This, (3.3), and (3.12) suggest the weight-vector enumerator can be presented in the closed form

\[
A(d) = \prod_{\{d_j; j=1,\ldots,n'_v\}} \frac{N!}{\left(N - \frac{d_{j_1} + d_{j_2} + d_{j_3}}{2}\right)! \left(-\frac{d_{j_1} + d_{j_2} + d_{j_3}}{2}\right)! \left(\frac{d_{j_1} - d_{j_2} + d_{j_3}}{2}\right)! \left(\frac{d_{j_1} + d_{j_2} - d_{j_3}}{2}\right)!} \cdot \left(\prod_{i=1}^{n'_v} \left(N \cdot q_{v_i} - 1\right)\right)^{-1}.
\]

(3.14)

3.2.2 Asymptotic ensemble weight enumerators

In the asymptotic case, following Gallager [2] let us define the normalized logarithmic asymptotic weight enumerator (we will simply call it the asymptotic weight enumerator),

\[
r(\delta) = \lim_{n \to \infty} \sup \frac{\ln A_d}{n},
\]

where \( \delta = d/n \) (recall \( n \) is the number of transmitted variable nodes in the code). Following [51], because the formulas in the previous section involve the number of copies, \( N \), instead of \( n \), we define the function

\[
\tilde{r}(\tilde{\delta}) = \lim_{N \to \infty} \sup \frac{\ln A_d}{N},
\]

where \( \tilde{\delta} = d/N \). Note that \( n = |S_t| \cdot N \) and so

\[
r(\delta) = \frac{1}{|S_t|} \tilde{r}(|S_t| \cdot \delta).
\]

(3.17)

We also define \( \max^*(x, y) \triangleq \ln (e^x + e^y) \) and we similarly define \( \max^* \) when more than two variables are involved. When \( x \) and \( y \) are large and distinct (so that \( e^x \) and \( e^y \) are vastly different), then \( \max^*(x, y) \simeq \max(x, y) \). Similar comments apply for more than two variables.
From (3.4), we have

$$\ln A_d = \max_{\{d_i: v_i \in S_t\}} \left\{ \max_{\{d_k: v_k \in S_p\}} \{\ln A(d)\} \right\},$$

$$\approx \max_{\{d_i: v_i \in S_t\}} \left\{ \max_{\{d_k: v_k \in S_p\}} \{\ln A(d)\} \right\},$$

$$= \max_{\{d_i: v_i \in S_t\}} \left\{ \max_{\{d_k: v_k \in S_p\}} \left\{ \sum_{j=1}^{n_c} \ln A^c_j (d_j) \right\} \right\},$$

under the constraint $\sum_{\{d_i: v_i \in S_t\}} d_i = d$. The second line holds when $N$ is large and the third line follows by invoking (3.3). Taking the limit as $N \to \infty$ and applying the result (from Stirling’s formula) $\lim_{N \to \infty} \sup \ln \left( \frac{N}{d_i} \right) = H(\tilde{\delta}_i) = -(1 - \tilde{\delta}_i) \ln(1 - \tilde{\delta}_i) - \tilde{\delta}_i \ln \tilde{\delta}_i$, where $\tilde{\delta}_i = d_i/N$, we obtain

$$\tilde{r}(\tilde{\delta}) = \max_{\{\tilde{\delta}_i: v_i \in S_t\}} \left\{ \max_{\{\tilde{\delta}_k: v_k \in S_p\}} \left\{ \sum_{j=1}^{n_c} a^c_j (\tilde{\delta}_j) \right\} \right\},$$

under the constraint $\sum_{\{\tilde{\delta}_i: v_i \in S_t\}} \tilde{\delta}_i = \tilde{\delta}$. In (3.19), $a^c_j (\tilde{\delta}_j)$ is the asymptotic weight-vector enumerator of the constraint node $c_j$, and $\tilde{\delta}_j = d_j/N$. For a generic constituent CN code $\mathcal{C}$, this enumerator is defined as

$$a^c (\omega) = \lim_{N \to \infty} \sup \ln \frac{A^c(\omega)}{N},$$

where $\omega = w/N$.

We may obtain a simple expression for $a^c (\omega)$ using the method of types [61, Ch. 12]. We define the type $P_\omega$ as the relative proportion of occurrences of each codeword of constituent CN code $\mathcal{C}$ in a sequence of $N$ codewords. In other words, $P_\omega = [p_1, p_2, \ldots, p_K]$ is the empirical probability distribution of the codewords in $\mathcal{C}$ given a sequence of $N$ such codewords, where $p_k = n_k/N$ and $n_k$ is the number of occurrences of the $k^{th}$ codeword. Then the type class of $P_\omega$,
$T(P_\omega)$, is the set of all length-$N$ sequences of codewords in $C$, each containing $n_k$ occurrences of the $k^{th}$ codeword in $C$, for $k = 1, 2, ..., K$. Observe that $|T(P_\omega)| = C(N; n_1, n_2, \ldots, n_K)$. From [61, Thm.12.1.3] $|T(P_\omega)| \rightarrow e^{N \cdot H(P_\omega)}$, as $N \rightarrow \infty$, where $H(P_\omega) = - \sum_{k=1}^{K} p_k \ln p_k$. Consequently, as $N \rightarrow \infty$ we rewrite (3.11) as

$$A^C(\omega) = \sum_{\{n\}} C(N; n_1, n_2, \ldots, n_K)$$

$$= \sum_{\{P_\omega\}} |T(P_\omega)|$$

$$\rightarrow \sum_{\{P_\omega\}} e^{N \cdot H(P_\omega)}.$$  \hspace{1cm} (3.21)

It follows from (3.20) and (3.21) that

$$a^C(\omega) = \max_{\{P_\omega\}} \{H(P_\omega)\},$$  \hspace{1cm} (3.22)

under the constraint that $\{P_\omega\}$ is the set of solutions to $\omega = P_\omega \cdot M^C$, with $p_1, p_2, \ldots, p_K \geq 0$ and $\sum_{k=1}^{K} p_k = 1$. These are the asymptotic equivalents of the constraints mentioned below (3.11).

Let $d_{\min}$ be the minimum distance of a linear block code. In [2] it was shown that

$$\Pr\{d_{\min} \leq n\delta\} \leq \sum_{l=1}^{n\delta} \left(\begin{array}{c} n \\ l \end{array}\right) P(l) = \sum_{l=1}^{n\delta} A_l,$$  \hspace{1cm} (3.23)

where $P(l)$ is the probability that a particular sequence of weight $l$ is a codeword in the ensemble. Let $\delta_{\min}$ be the second zero crossing of $r(\delta)$ (the first crossing is $r(0) = 0$). Assuming $\delta_{\min}$ exists, it is called the typical minimum distance if $r(\delta) < 0$ for all $0 < \delta < \delta_{\min}$ [51]. If $\sum_{l=1}^{n\delta_{\min}} A_l \rightarrow A_{n\delta_{\min}}$ as $n$ becomes large, then $\Pr\{d_{\min} \leq n\delta_{\min} - 1\} \leq \sum_{l=1}^{n\delta_{\min}-1} A_l \rightarrow 0$, so that with probability near one, $d_{\min} \approx n\delta_{\min}$. That is, the minimum distances of virtually all of the members of the code ensemble increase linearly with $n$.

Now the asymptotic ensemble weight-vector enumerator for protograph based LDPC can be obtained using the finite length result for the weight-vector enumerator
\( A(\mathbf{d}) \) in (3.14), resulting in
\[
a(\tilde{\delta}) = \sum_{\{\mathbf{d} : j=1, \ldots, n'\}} H \left( \left[ \left( 1 - \tilde{\delta}_{j_1} + \tilde{\delta}_{j_2} + \tilde{\delta}_{j_3} \right) / 2, \left( -\tilde{\delta}_{j_1} + \tilde{\delta}_{j_2} + \tilde{\delta}_{j_3} \right) / 2 \right], \left( \tilde{\delta}_{j_1} - \tilde{\delta}_{j_2} + \tilde{\delta}_{j_3} \right) / 2 \right) - \sum_{i=1}^{n'} (q_{v_i} - 1) H(\tilde{\delta}_i)
\]
(3.24)

Before presenting some examples, note that the ensemble of all rate-\( R \) (“random”) linear codes has the asymptotic weight enumerator [2]
\[
r(\delta) = H(\delta) - (1 - R) \ln(2),
\]
(3.25)
which has the largest \( \delta_{\min} \) among all rate-\( R \) ensembles. The asymptotic weight enumerators in the following examples will be compared to that given by (3.25).

**Example 7:** Consider the protograph in Fig. 3.3, where the parity-check matrix \( \mathbf{H}_1 \) corresponds to the \((7, 4)\) Hamming code generated by \( \mathbf{G} \) of Example 6. The parity-check matrix \( \mathbf{H}_2 \) corresponds to the following column-permutation of \( \mathbf{H}_1 \): \((6, 7, 1, 2, 3, 4, 5)\). That is, the first column of \( \mathbf{H}_2 \) is the sixth column of \( \mathbf{H}_1 \), ..., and the seventh column of \( \mathbf{H}_2 \) is the fifth column of \( \mathbf{H}_1 \). A code constructed per this protograph has rate 1/7 since each CN represents three redundant bits. The variable nodes \( v_1, v_2, \ldots, v_7 \) have the normalized weights \( \tilde{\delta}_1, \tilde{\delta}_2, \ldots, \tilde{\delta}_7 \). The asymptotic weight enumerator is
\[
r(\delta) = \tilde{r}(7\delta)/7,
\]
where
\[
\tilde{r}(\tilde{\delta}) = \max_{\tilde{\delta}_1, \ldots, \tilde{\delta}_7} \left\{ a^{\mathbf{H}_1}(\tilde{\delta}_1) + a^{\mathbf{H}_2}(\tilde{\delta}_2) - \sum_{j=1}^{7} H(\tilde{\delta}_j) \right\}.
\]
(3.26)
such that \( \sum_{i=1}^{7} \tilde{\delta}_i = \tilde{\delta} \), and where \( \tilde{\delta}_1 = [\tilde{\delta}_1, \tilde{\delta}_2, \ldots, \tilde{\delta}_7] \) and \( \tilde{\delta}_2 = [\tilde{\delta}_6, \tilde{\delta}_7, \tilde{\delta}_1, \tilde{\delta}_2, \tilde{\delta}_3, \tilde{\delta}_4, \tilde{\delta}_5] \).

We also evaluated the asymptotic weight enumerator for the rate-1/6 G-LDPC code which results from puncturing one of the VNs. The result is presented in Fig. 3.4. Note that this ensemble has a relatively large \( \delta_{\min} \). Consequently, a long code based on this protograph has, with probability near one, a large minimum distance.
\( \square \)
Figure 3.3: Rate-1/7, \( n_v = 7 \), (or, rate-7/15, \( n_v = 15 \)) G-LDPC protograph.

Figure 3.4: Asymptotic codeword weight enumerator for the rate-1/7 and rate-1/6 G-LDPC code.
Example 8: We evaluate the asymptotic weight enumerator for the rate-1/6 HD-LDPC code presented in [28, 33, 60, 62], which is based on the protograph in Fig. 3.5 (HD = “Hamming-doped”). The constraint node marked by $H$ is a $(6, 3)$ shortened Hamming code with parity check matrix

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

(3.27)

Associate the vector of normalized weights $[\tilde{\delta}_0, \tilde{\delta}_1, \tilde{\delta}_2, \tilde{\delta}_3, \tilde{\delta}_4, \tilde{\delta}_5]$ with the vector of variable nodes $[u_0, p_0, p_1, p_2, p_3, p_4]$, and label the columns of $H$ by $[u_0, u_0, p_0, p_2, p_3, p_4]$, in accordance with Fig. 3.5. Note that the two SPC nodes have the same neighborhood, and so they have the same weight-vector numerator. Under this setup, because $|S_t| = 6$, the asymptotic weight enumerator is given by $r(\delta) = \tilde{r}(6\delta)/6$, where

$$
\tilde{r}(\delta) = \max_{\delta_0, \ldots, \delta_5} \left\{ 2a^{SPC}(\tilde{\delta}_1) + a^{H}(\tilde{\delta}_2) \right. \\
-5H(\tilde{\delta}_0) - 2H(\tilde{\delta}_1) - H(\tilde{\delta}_2) \right\}
$$

(3.28)

such that $\sum_{i=0}^{5}\tilde{\delta}_i = \delta$, and where $\tilde{\delta}_1 = [\tilde{\delta}_0, \tilde{\delta}_0, \tilde{\delta}_1, \tilde{\delta}_2]$ and $\tilde{\delta}_2 = [\tilde{\delta}_0, \tilde{\delta}_0, \tilde{\delta}_1, \tilde{\delta}_3, \tilde{\delta}_4, \tilde{\delta}_5]$. The asymptotic weight enumerator is shown in Fig. 3.6. The figure shows that there is no typical minimum distance $\delta_{\text{min}}$ for this code (or $\delta_{\text{min}} = 0$) so that $d_{\text{min}}$ for this code ensemble does not grow linearly with $n$. We also evaluated the asymptotic weight enumerator for the case when the variable node $u_0$ is punctured. Equation (3.28) still holds but the constraint becomes $\sum_{i=1}^{5}\tilde{\delta}_i = \delta$. The result is represented by the rate-1/5 HD-LDPC curve in Fig. 3.6, where again we see that $d_{\text{min}}$ for the ensemble does not grow linearly with $n$. (In spite of this, we have found via simulation that short codes designed according to this protograph provide good performance in both the waterfall and the floor region [28, 33, 60, 62].)

□
Figure 3.5: Rate-1/6 HD-LDPC protograph.

Figure 3.6: Asymptotic codeword weight enumerator for the rate-1/6 and rate-1/5 HD-LDPC code ensembles.
3.2.3 On the complexity of computing asymptotic ensemble enumerators

The drawback of this method in evaluating the asymptotic enumerators can be seen from (3.22): the number of the maximization arguments equals the number of the codewords $K$ in the CN code $C$. As an example, for the $(15, 11)$ Hamming code, the maximization is over $K = 2^{11} = 2048$ variables.

To alleviate this issue of having to maximize over a large number of variables, the following steps are considered. First, the protograph’s VNs are partitioned into subsets based on their neighborhoods. That is, two VNs belong to the same subset if and only if they have the same type-neighborhood, meaning, they are connected to an identical distribution of CN types. Given this partitioning of the set of VNs into subsets, the bits for a given CN code can themselves be partitioned into bit subsets in accordance with their membership in the VN subsets. Now, define the *subset-weight vector* (SWV) of a CN codeword as the vector whose components are the weights of bit subsets of the CN codeword. As an example, let $\vec{x} = [1010111]$ be a codeword of a length-7 CN code and let the CN code’s bit subsets be $\{1, 2, 7\}, \{3, 4\}, \{5, 6\}$; then $\text{SWV}(\vec{x}) = [2, 1, 2]$. Also, define the *SWV enumerator* as the number of CN codewords that have the same SWV.

Next, several examples were examined, all of which led to the following conjecture: *In the maximization in (3.22), the optimal point occurs when codewords of equal SWV have the same proportion of occurrence in the constituent CN code.* The implication is that many of the elements of $P_\omega = [p_1, p_2, \ldots, p_K]$ are identical and so the maximization in (3.22) is over a vastly reduced number of distinct variables.

This conjecture is used with some simple linear algebra to rewrite (3.22) as

$$a^C(\omega) = \max_{\{\hat{P}\}} \left\{ H^*(\hat{P}) \right\}, \quad (3.29)$$

under the constraint that $\{\hat{P}\}$ is the set of solutions to $\omega = \hat{P} \cdot \hat{M}^C$, $p^{(1)}, p^{(2)}, \ldots \geq 0$ and $\Psi \cdot \hat{P}^T = 1$, where $\hat{P} = [p^{(1)}, p^{(2)}, \ldots]$ is a vector of the distinct $p_i$’s in $P_\omega$, $\Psi = [\psi_1, \psi_2, \ldots]$ is a vector of the SWV enumerators of $C$, and $\hat{M}^C$ is constructed from $M^C$ by adding the rows of $M^C$ having the same SWV. Note that it is possible
to have identical columns in $\hat{M}^C$. This implies that the corresponding $\omega_i$'s in $\omega = [\omega_1, \omega_2, ..., \omega_\mu]$ are equal. Finally, $H^*(\hat{P})$ is related to $H(P_\omega)$ as follows:

$$
H(P_\omega) = -\sum_{k=1}^{K} p_k \ln p_k
= -\sum (p^{(i)} \ln p^{(i)}) \cdot \psi_i
\triangleq H^*(\hat{P}).
$$

Example 9: Consider again the protograph in Fig. 3.3, but with (15, 11) Hamming codes for the constraints $H_1$ and $H_2$ [28, 33, 60], where

$$
H_1 = [M_1, M_2] = \\
\begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

$$
H_2 = [M_2, M_1].
$$

Note that there are $K = 2048$ codewords in each constituent code, so it would be difficult to evaluate (3.22) for this code. But, applying the conjecture, all VNs in this protograph have the same type-neighborhood. Also, after finding $\hat{M}$, one finds that all of its columns are identical. Consequently, the VNs have the same normalized weight $\delta$. The asymptotic weight enumerator is

$$
r(\delta) = \frac{1}{15} \max_\delta \left\{ 2^{a_{H_1}(\tilde{\delta})} - 15H(\delta) \right\},
$$

where $\tilde{\delta} = [\delta, \delta, \ldots, \delta]$ (15 of them). Now, to find $a_{H_1}(\tilde{\delta})$, define $p^{(\rho)}$ as the proportion of occurrence of a codeword of weight $\rho$ in the constituent CN code, and so $\hat{P} = [p^{(0)}, p^{(3)}, p^{(4)}, p^{(5)}, p^{(6)}, p^{(7)}, p^{(8)}, p^{(9)}, p^{(10)}, p^{(11)}, p^{(12)}, p^{(15)}]$ and $\Psi = [1, 35, 105, 168, 280, 435, 435, 280, 168, 105, 35, 1]$. Consequently,

$$
a_{H_1}(\tilde{\delta}) = \max_{\tilde{\delta}} \left\{ H^*(\hat{P}) \right\},
$$

under the constraint $\tilde{\delta} = \hat{P} \cdot \hat{M}^C$, $p^{(\rho)} \geq 0$, for all $p^{(\rho)}$ in $\hat{P}$, and $\Psi \cdot \hat{P}^T = 1$. Clearly, under these assumptions, the computation of
\( a^{H_1}(\delta) \), hence \( r(\delta) \), is vastly simplified. The \( r(\delta) \) result appears in Fig. 3.7. Also included in the figure is \( r(\delta) \) for a rate-1/2 code obtained by puncturing one bit in the protograph. Note that, for both cases, \( d_{\min} \) for the ensemble increases linearly with \( n \).

\[ \begin{align*}
\text{Rate-7/15 HD-LDPC} & \quad \text{Rate-7/15 Random} \\
\text{Rate-1/2 Random} & \quad \text{Rate-1/2 HD-LDPC}
\end{align*} \]

Figure 3.7: Asymptotic codeword weight enumerator for the rate-7/15 and rate-1/2 G-LDPC code.

\( \Box \)

**Example 10:** Consider the rate-1/3 G-LDPC protograph in Fig. 3.8. The different subsets of VNs, according to their type-neighborhoods, are \( \{v_2, v_3, v_4, v_6, v_7, v_8\}, \{v_1, v_9\}, \{v_5\} \). From the \( \tilde{M}_C \) matrices of the CNs, we noticed that VNs in the same subset have the same normalized weight. So we associate the normalized weights \( \{\tilde{\delta}_0, \tilde{\delta}_1, \tilde{\delta}_2\} \) with the VNs of the sets. The asymptotic weight
enumerator is given by $r(\delta) = \tilde{r}(9\delta)/9$, where

$$
\tilde{r}(\delta) = \max_{\tilde{\delta}_0, \tilde{\delta}_1, \tilde{\delta}_2} \left\{ 2a^{SPC}(\tilde{\delta}_1) + a^H(\tilde{\delta}_2) - 12H(\tilde{\delta}_0) - 4H(\tilde{\delta}_1) - 2H(\tilde{\delta}_2) \right\}.
$$

such that $6\tilde{\delta}_0 + 2\tilde{\delta}_1 + \tilde{\delta}_2 = \tilde{\delta}$, and where $\tilde{\delta}_1$ and $\tilde{\delta}_2$ are the vectors of the associated normalized weights. The result appears in Fig. 3.9, which shows that $\delta_{\text{min}} = 0.055$. Thus, the minimum distance for this code ensemble increases linearly with $n$.

![Rate-1/3 G-LDPC protograph.](image)

**Example 11:** In this example we used our proposed method to evaluate the asymptotic weight enumerators for several related protograph-based LDPC code ensembles. In particular, we first consider the (6, 3) regular LDPC code ensemble, which has the protograph in Fig. 3.10(a) (We remark that the simplest (6, 3) protograph has two degree-3 VNs connected to a single degree-6 CN, but we use the form in Fig.3.10(a) so that we may build upon it in the rest of the figure.) Its asymptotic weight enumerator curve appears in Fig. 3.11, which shows that this ensemble has $\delta_{\text{min}} = 0.023$. Similar results for the (6, 3) LDPC appears in [45] and [51]. We also computed the asymptotic weight enumerator for a precoded version of the (6, 3) ensemble. The protograph of this ensemble is shown in Fig. 3.10(b), and the computation results are given in Fig. 3.11, which shows that precoding increased $\delta_{\text{min}}$ from 0.023 to 0.028. Note that this agrees with the result in [37].
Figure 3.9: Asymptotic codeword weight enumerator for the rate-1/3 G-LDPC code.
Second, we consider the RJA and the ARJA (precoded RJA) LDPC codes presented in [37] which have the protographs in Fig. 3.10(c) and 3.10(d), respectively. We computed the asymptotic weight enumerators for their ensembles, which appear in Fig. 3.11. The figure shows that the RJA ensemble has $\delta_{\text{min}} = 0.013$ and the ARJA has $\delta_{\text{min}} = 0.015$, which agrees with what was obtained using the method in [51].

Figure 3.10: Protographs for several rate-1/2 LDPC codes: (a) (6, 3) regular LDPC code. (b) Precoded (6, 3) regular LDPC code. (c) RJA LDPC code. (d) ARJA LDPC code. (Note: The shaded VNs represent punctured nodes.)

\[ \square \]

3.3 Ensemble stopping set enumerators

In the context of iterative decoding of LDPC codes on the BEC, a stopping set $S$ is a subset of the set of VNs whose neighboring SPC nodes are connected to $S$ at least twice [14]. The implication of this definition is that if all of the bits in a stopping set are erased, the iterative erasure decoder will never resolve the values of those bits. For single-CN-type G-LDPC codes [24], a generalized stopping set, $S^d$, is a subset of the set of VNs, such that all neighbors of $S^d$ are connected to $S^d$ at least $d$ times, where $d - 1$ is the erasure capability of the neighboring CNs.

In this work, in addition to single-CN-type G-LDPC codes, we consider multi-CN-type G-LDPC codes, where a mixture of CN types are permissible. We assume
Figure 3.11: Asymptotic codeword weight enumerators for the protograph-based LDPC code ensembles in Fig. 3.10.
the use of a standard iterative erasure decoding tailored to the Tanner graph of
the code, with bounded-distance algebraic decoders employed at the CNs. That is,
CN $c_j$ can correct up to $d^{(j)}_{\min} - 1$ erasures. A set of erased VNs will fail to decode
exactly when all of the neighboring CNs see more erasures than they are capable of
resolving. Consequently, we introduce a new definition for the generalized stopping
set as follows: A *generalized stopping set*, $S^D$, of a G-LDPC code is a subset of the
set of VNs such that every neighboring CN $c_j$ of $S^D$ is connected to $S^D$ at least $d^{(j)}_{\min}$
times, for all $c_j$ in the neighborhood of $S^D$. Here $D$ is the set of $d^{(j)}_{\min}$’s for the CNs
that are neighbors of $S^D$. Hereafter, we will use the term stopping set to refer to a
generalized stopping set.

The method for finding weight enumerators for protograph-based G-LDPC code
ensembles can be leveraged to obtain stopping set enumerators for these same en-
sembles. To see how, let us consider the mapping $\phi$ from the set of stopping sets
$\{S^D\}$ to $\mathbb{F}_2^n$ defined as $\phi(S^D) = [x_0, x_1, \ldots, x_{n-1}]$, where $x_i = 1$, if and only if the
corresponding VN $v_i$ in the Tanner graph is in the stopping set $S^D$. Note that the
set of all binary words $\{\phi(S^D)\}$ corresponding to a Tanner graph $G$ need not form
a linear code. We call the (nonlinear) code $\{\phi(S^D)\}$ induced by the set of stopping
sets of $G$ under the mapping $\phi$ a *stopping set code* and we denote it by $S_G$. Also note
that the weight of $\phi(S^D)$ equals the size of $S^D$, and so the weight enumerator for the
stopping set code $S_G$ is identical to the stopping set enumerator for $G$. For example,
in a graph $G$ with a single CN $C$, where $C$ is a $(\mu, \kappa)$ linear block code of minimum
distance $d_{\min}$, the stopping set code is $S_C = S_G = \{\bar{x} \in \mathbb{F}_2^\mu : \text{weight}(\bar{x}) \geq d_{\min}\}$. Thus the stopping set enumerator for this simple graph is exactly the weight en-
umerator for $S_G$. This simple example also lets us speak of a stopping set code for a
single CN $c_j$, which is $S_{c_j} = \{\bar{x} \in \mathbb{F}_2^\mu : \text{weight}(\bar{x}) \geq d^{(j)}_{\min}\}$.

In the case of a non-trivial G-LDPC Tanner graph $G$, we first find the stopping
set codes $S_{c_j}$ for each of the CNs $c_j$. We then form a new graph, $G'$, from $G$
by replacing CN $c_j$ in $G$ by $S_{c_j}$, for all $j$. The stopping set enumerator for $G'$ is
then given by the weight enumerator for $S_{G'}$, which is given by $S_{G'} = \{\bar{x} \in \mathbb{F}_2^n : \bar{x} \text{satisfies the constraints of } G'\}$. In summary, the ensemble *stopping set enumerator*
for a protograph-based G-LDPC code with graph $G$ is identically the ensemble weight enumerator for the graph $G'$ formed from $G$ by replacing the CNs in $G$ with their corresponding stopping set code constraint nodes.

In light of the foregoing discussion, we may follow the weight enumerator techniques of Sections 3.2 to derive stopping set enumerators for G-LDPC codes. We will use the superscript $(s)$ for quantities associated with stopping sets to distinguish them from weight enumerator quantities. For example, we use $A_d^{(s)}$ and $r^{(s)}(\delta)$ for finite-length and asymptotic stopping set enumerators, respectively. Moreover, $d_{\min}^{(s)}$ is the stopping set number, i. e., the size of the smallest, non-empty, stopping set of the G-LDPC code’s graph, and $\delta_{\min}^{(s)}$ is the typical stopping set number. Recall that we are interested in codes with a large stopping set number because small stopping sets dominate the iterative decoding performance on the BEC. If $\delta_{\min}^{(s)}$ exists, the average stopping set number over the code ensemble increases linearly with $n$ (or, $d_{\min}^{(s)} \approx n\delta_{\min}^{(s)}$). To complete our notation, we will use $M^S$ to denote the matrix whose rows are the codewords of a CN’s stopping set code. For example, for a CN $C$, corresponding to $(\mu, \kappa)$ linear block code of minimum distance $d_{\min}$, $M^S$ is the $K^{(s)} \times \mu$ matrix, whose rows are the $\mu$-bit words of weight $= 0$ or weight $\geq d_{\min}$ where $K^{(s)} = 1 + \sum_{l=d_{\min}}^{\mu} \binom{\mu}{l}$. The weight-zero word corresponds to the empty set in $\{S^D\}$ and means that the CN is not a neighbor of a stopping set.

Example 12:

Consider the computation of $A^{(s)}(w)$ and $a^{(s)}(\omega)$ for a degree-3 SPC code. The stopping set code $S_{SPC} = \{000, 110, 101, 011, 111\}$. Now use (3.11) to compute $A^{(s)}(w)$ as follows. First, construct the $5 \times 3$ matrix $M^S$ from the words of $S_{SPC}$. Then, find the nonnegative integer solution to the system of equations $w = n \cdot M^S$ with $\sum_{k=1}^{5} n_k = N$. These equations are

$$
\begin{align*}
    n_2 + n_3 + n_5 &= w_1 \\
    n_2 + n_4 + n_5 &= w_2 \\
    n_3 + n_4 + n_5 &= w_3 \\
    n_1 + n_2 + n_3 + n_4 + n_5 &= N
\end{align*}
$$

(3.32)
Solving this set of equations we get $n_1 = N - u + n_5/2$, $n_2 = u - w_3 - n_5/2$, $n_3 = u - w_2 - n_5/2$, and $n_4 = u - w_1 - n_5/2$, where $u = (w_1 + w_2 + w_3)/2$. Since $n_i \geq 0$, this implies that $\max \{0, 2(u - N)\} \leq n_5 \leq 2u - 2 \max \{w_1, w_2, w_3\}$. Let $l = \lfloor n_5/2 \rfloor$

If $w_1 + w_2 + w_3$ is even, then

$$A^{(s)}(w) = \sum_l C(N; (N - u + l), (u - w_3 - l)) , (u - w_2 - l), (u - l - 1/2), (2l))$$

If $w_1 + w_2 + w_3$ is odd, then

$$A^{(s)}(w) = \sum_l C(N; (N - u + l + 1/2), (u - w_3 - l - 1/2), (u - w_2 - l - 1/2), (u - l - 1/2), (2l + 1))$$

In the asymptotic case, we use (3.22) to evaluate $d^{(s)}(\omega)$, which gives

$$d^{(s)}(\omega) = \max_{p_5} \left\{ H \left( \left[ 1 - \sigma + p_5/2 \right], \left( \sigma - \omega_1 - p_5/2 \right) \right), \left( \sigma - \omega_2 - p_5/2 \right), \left( \sigma - \omega_3 - p_5/2 \right), (p_5) \right\}$$

where $\sigma = (\omega_1 + \omega_2 + \omega_3)/2$ such that $\max \{0, 2(\sigma - 1)\} \leq p_5 \leq 2\sigma - 2 \max \{\omega_1, \omega_2, \omega_3\}$.

Unfortunately, the check-split method, described at the end of Section 3.2.1, does not apply to enumerate the stopping sets for SPC codes with higher degree. The check-split method overcounts and can only serve as an upper bound. □

**Example 13:** Consider the different protograph-based LDPC codes in Example 11. We computed the asymptotic stopping set enumerators for their ensembles, which appear in Fig. 3.12. The figure shows that the (6, 3) LDPC ensemble has $\delta^{(s)}_{\text{min}} = 0.018$, the precoded version of the (6, 3) ensemble has $\delta^{(s)}_{\text{min}} = 0.022$, the RJA ensemble has $\delta^{(s)}_{\text{min}} = 0.011$, and the ARJA ensemble has $\delta^{(s)}_{\text{min}} = 0.012$.

In Fig. 3.13, we compared the results computed using the check-split method to the exact asymptotic stopping set enumerators for the RJA code ensemble. The figure demonstrates that the check-split method overcounts.
Figure 3.12: Asymptotic stopping set enumerators for the protograph-based LDPC code ensembles in Fig. 3.10.
Figure 3.13: Check-split method results and asymptotic stopping set enumerators for the RJA code ensemble in Fig. 3.10(c).
Example 14: To find the asymptotic stopping set enumerator for the rate-1/7 protograph-based G-LDPC code ensemble in Example 7, first note that all VNs in this protograph have the same type-neighborhood. Also, after finding $\hat{M}^S$ from $M^S$ as described in Subsection 3.2.3, one finds that all of its columns are identical. Consequently, the protograph’s VNs have the same normalized weight $\delta$. The asymptotic stopping set enumerator is

$$r^{(s)}(\delta) = \frac{1}{7} \max_{\bar{\delta}} \left\{ 2a^{SU_1}(\bar{\delta}) - 7H(\delta) \right\},$$  \hspace{1cm} (3.36)$$

where $\bar{\delta} = [\delta, \delta, \delta, \delta, \delta, \delta, \delta]$. This result is presented in Fig. 3.14. Note that this ensemble has $\delta_{min}^{(s)} = 0.01$, indicating that the stopping number for the ensemble increases linearly with $n$.

We also evaluated the stopping set enumerator for the rate-7/15 G-LDPC in Example 9. As seen in Fig. 3.14, there is no typical stopping set number for this code.

Example 15: We now evaluate the asymptotic stopping set enumerator for the rate-1/6 G-LDPC in Example 8. The asymptotic stopping set enumerator is given by $r^{(s)}(\delta) = \bar{r}(6\delta)/6$, where

$$\bar{r}(\bar{\delta}) = \max_{\bar{\delta}_0, \ldots, \bar{\delta}_5} \left\{ 2a^{SPC}(\bar{\delta}_1) + a^{SU}(\bar{\delta}_2) \right\}$$

$$-5H(\bar{\delta}_0) - 2H(\bar{\delta}_1) - H(\bar{\delta}_2),$$  \hspace{1cm} (3.37)$$

such that $\sum_{i=0}^5 \bar{\delta}_i = \bar{\delta}$. In (3.37), $\bar{\delta}_1 = [\bar{\delta}_0, \bar{\delta}_0, \bar{\delta}_1, \bar{\delta}_2]$ and $\bar{\delta}_2 = [\bar{\delta}_0, \bar{\delta}_0, \bar{\delta}_1, \bar{\delta}_3, \bar{\delta}_4, \bar{\delta}_5]$. Following Subsection 3.2.3, in this protograph each CN has three bit subsets, where the only set with more than one element is $\{p_2, p_3, p_4\}$. Moreover, after constructing $\hat{M}^S$ for the Hamming CN, we found that $\tilde{\delta}_3 = \tilde{\delta}_4 = \tilde{\delta}_5$. The asymptotic stopping set enumerator is shown in Fig. 3.15. The figure shows that there is no typical stopping set number for this code.
Figure 3.14: Asymptotic stopping set enumerator for the rate-1/7 and the rate-7/15 G-LDPC code.
Figure 3.15: Asymptotic stopping set enumerator for the rate-1/6 G-LDPC code.
Example 16: Consider the rate-1/3 G-LDPC ensemble in Example 10. The asymptotic stopping set enumerator is given by \( r^{(s)}(\tilde{\delta}) = \tilde{r}(9\tilde{\delta})/9 \), where

\[
\tilde{r}(\tilde{\delta}) = \max_{\tilde{\delta}_0, \tilde{\delta}_1, \tilde{\delta}_2} \left\{ 2a^{S_{PC}}(\tilde{\delta}_1) + a^{SH}(\tilde{\delta}_2) - 12H(\tilde{\delta}_0) - 4H(\tilde{\delta}_1) - 2H(\tilde{\delta}_2) \right\},
\]

such that \( 6\tilde{\delta}_0 + 2\tilde{\delta}_1 + \tilde{\delta}_2 = \tilde{\delta} \), where \( \tilde{\delta}_1 \) and \( \tilde{\delta}_2 \) are the vectors of the associated normalized weights. This result is plotted in Fig. 3.16 which shows that \( \delta^{(s)}_{\min} = 0.023 \). Thus, the stopping set number for this code ensemble increases linearly with \( n \).

Figure 3.16: Asymptotic stopping set enumerator for the rate-1/3 G-LDPC code.
3.4 Ensemble Trapping Set Enumerators

An \((a, b)\) \textit{general trapping set} \cite{15}, \(T_{a,b}\), is a set of VNs of size \(a\) which induce a subgraph with exactly \(b\) odd-degree check nodes (and an arbitrary number of even-degree check nodes). We use the qualifier “general” to make the distinction from elementary trapping sets to be defined later.

3.4.1 Finite-size trapping set enumerators

Our method for determining trapping set enumerators for an LDPC code ensemble characterized by a given protograph is similar to our strategy used to find stopping set enumerators in Section 3.3. That is, we create a modified protograph from the original protograph and then determine the codeword weight enumerator for the modified protograph, from which the trapping set enumerators may be obtained.

We describe the technique as follows. Assume we are interested in trapping sets of weight \(a\) in the graph \(G\) in Fig. 3.1. The value of the companion parameter \(b\) depends on which \(a\) VNs are of interest, and so we set the values of the \(a\) VNs of interest to ‘1’ and the values of the remaining VNs to ‘0’. With the \(a\) VNs so fixed, we are now interested in which CNs “see” odd weight among its neighboring VNs, for the number of such CNs is the corresponding parameter \(b\). Such odd-weight CNs can be identified by the addition of auxiliary “flag” VNs to each CN (see protograph \(G'\) in Fig. 3.17), where the value of an auxiliary VN equals ‘1’ exactly when its corresponding original CN sees odd weight. The number of auxiliary VNs equal to ‘1’ is precisely the parameter \(b\) for the set of \(a\) VNs that were set to ‘1’. Thus, to obtain the trapping set enumerator for the original protograph \(G\), one may apply the codeword weight enumerator technique to \(G'\), partitioning the set of VNs into the original set \((S_t \cup S_p)\) and auxiliary flag set \((S_f)\), much like we had earlier partitioned the set of VNs into subsets of transmitted and punctured VNs.

Note that the set \(S_t \cup S_p\) (the VNs at the bottom of Fig. 3.17) accounts for the weight \(a\) and the set \(S_f\) (the VNs at the top of Fig. 3.17) accounts for the weight \(b\). Also note that, in counting the number of trapping sets, we do not distinguish
between transmitted and punctured VNs (refer to the trapping set definition). However, in evaluating the failure rate of a trapping set, one should make this distinction, although this is not considered in this paper.

Based on the above discussion and (3.4), the trapping set enumerator $A_{a,b}$ is given by

$$A_{a,b} = \sum_{\left\{d_i: v_i \in S_t \cup S_p\right\}} \sum_{\left\{d_k: v_k \in S_f\right\}} A(d)$$

under the constraint $\sum_{\left\{d_i: v_i \in S_t \cup S_p\right\}} d_i = a$ and $\sum_{\left\{d_i: v_i \in S_f\right\}} d_i = b$, where

$$A(d) = \frac{\prod_{j=1}^{n_c} A_{c_j}(d_j)}{\prod_{i=1}^{n_v} \left(\frac{N}{q_{v_i} - 1}\right)}.$$  

Notice the use of $c_j'$ instead of $c_j$ in (3.40) to indicate that the weight-vector enumerators in (3.40) are the CNs in $G'$. Those weight-vector enumerators can be evaluated using (3.11).

### 3.4.2 Asymptotic trapping set enumerators

As in Section 3.2, define the normalized logarithmic asymptotic trapping set enumerator (we will simply call it the *asymptotic trapping set enumerator*), $r(\alpha, \beta)$,
as
\[ r(\alpha, \beta) = \lim_{n \to \infty} \sup \frac{\ln A_{a,b}}{n}, \]  
(3.41)
where \( \alpha = a/n \) and \( \beta = b/n \) (recall \( n = |S| \cdot N \) is the transmit code length). The derivation of an expression for \((3.41)\) from \((3.39)\) uses the same steps used in deriving \((3.15)\) from \((3.4)\) (see Section 3.2). Thus, we omit the derivation and present the final result:
\[ r(\alpha, \beta) = \frac{1}{|S|} \tilde{r}(\alpha|S|, \beta|S|), \]  
(3.42)
under the constraint \( \sum_{\{\delta_i: v_i \in S_t \cup S_p\}} \tilde{\delta}_i = \tilde{\alpha}, \) and \( \sum_{\{\delta_i: v_i \in S_f\}} \tilde{\delta}_i = \tilde{\beta}. \) The asymptotic weight-vector enumerator, \( a^c(\cdot) \), can be evaluated using \((3.22)\). (To avoid ambiguity, note the use of \( a^c(\cdot) \) to refer to the asymptotic weight-vector enumerator, and the use of \( a \) to refer to the size of the \((a, b)\) trapping set).

To establish an analogy to the typical minimum distance in the weight enumerator problem, let \( \Delta = b/a = \beta/\alpha, \) \( \Delta \in [0, \infty) \). Then classify the trapping sets as
\[ \mathfrak{T}_\Delta = \{ T_{a,b} : b = \Delta \cdot a \}. \]  
(3.44)
Now, for each \( \Delta \), define \( d^{(ts)}_{\min}(\Delta) \) to be the \( \Delta \)-trapping set number, which is the size of the smallest, non-empty, trapping set in \( \mathfrak{T}_\Delta \). Further, let \( \delta^{(ts)}_{\min}(\Delta) \) be the typical \( \Delta \)-trapping set number, which is the value of \( \alpha \) at the second zero-crossing of the \( r(\alpha, \Delta \cdot \alpha) \)-versus-\( \alpha \) curve \((r(0,0) \) is the first zero-crossing). Note that, from this definition, as \( \Delta \) ranges from 0 to \( \infty \), the points \( \left( \delta^{(ts)}_{\min}(\Delta), \Delta \cdot \delta^{(ts)}_{\min}(\Delta) \right) \) trace out what we call the zero-contour curve: \( \{(\alpha, \beta) : r(\alpha, \beta) = 0\}. \) As for the codeword weight enumerator problem, if \( \delta^{(ts)}_{\min}(\Delta) \) exists, then the average \( \Delta \)-trapping set number over the code ensemble increases linearly with \( n \) (or, \( d^{(ts)}_{\min}(\Delta) \approx n\delta^{(ts)}_{\min}(\Delta) \)). Thus, we are interested in codes with large \( \Delta \)-trapping set numbers because small trapping sets dominate iterative decoding performance in the error-floor region.
Fig. 3.18 shows the zero-contour curves for two different hypothetical ensembles. The solid curve and the dashed curve corresponds to Ensemble 1 and Ensemble 2, respectively. We see in Fig. 3.18 the utility of the ordered pair $(\alpha, \beta) = (\alpha, \Delta \cdot \alpha)$. The line containing these points (and the origin) intersect a given zero-contour curve exactly when the abscissa of the intersection point is $\delta_{\text{min}}^{(ts)}(\Delta)$. Note, for any given $\Delta$, the $\Delta$-trapping set number for Ensemble 2 is greater than that for Ensemble 1, a consequence of the fact that the Ensemble 2 zero-contour curve resides above the Ensemble 1 zero-contour curve. Therefore, a code drawn from Ensemble 2 is expected to have a lower floor than a code drawn from Ensemble 1.

Figure 3.18: The zero-contour curves of the asymptotic trapping set enumerators for two different ensembles. The zero-contour curves correspond to Ensemble 1 and Ensemble 2 are the solid and the dashed curves, respectively.
3.4.3 Elementary Trapping set enumerators

An elementary \((a, b)\) trapping set [56], \(T_{(a,b)}^{(e)}\), is a set of VNs which induce a subgraph with only degree-one and degree-two check nodes, and exactly \(b\) of the CNs have degree one. It was observed that, in the error-floor region, after running extensive Monte Carlo simulations most of the error patterns correspond to elementary trapping sets [15, 54, 55, 57]. Some examples of elementary trapping sets are the \((12, 4)\) trapping set in the \((2640, 1320)\) Margulis code, and the \((4, 4)\) and \((5, 3)\) trapping sets in the \((1008, 504)\) and \((816, 408)\) MacKay codes [55]. As a result, it is desired to compute trapping set enumerators for just the elementary trapping sets in the code ensembles.

To find elementary trapping set enumerators, note that the only difference between the elementary and the general trapping set is the constraints on the degrees of the CNs in the induced subgraph. But this can be taken care of by choosing a proper matrix \(\mathbf{M}_C\) in (3.11) and (3.22) when evaluating the weight-vector enumerators in (3.40) and the asymptotic weight-vector enumerators in (3.43). Consequently, the discussion regarding general trapping set enumerators is valid for elementary trapping set enumerators after redefining \(\mathbf{M}_C\).

To find \(\mathbf{M}_C\) for the case of elementary trapping sets, note that when the bits of an elementary trapping set are set to ‘1’ with all other bits in \(G\) set to ‘0’, the CNs in \(G\) can see only certain bit patterns. Specifically, the set of patterns that can be seen by a degree-\(q\) \(CN\) \(c\) in \(G\) are the all-zeros word, the \(\left(\begin{array}{c}q\end{array}\right)\) weight-1 words, and the \(\left(\begin{array}{c}q+1\end{array}\right)\) weight-2 words. This implies that the possible patterns for its corresponding degree-\((q + 1)\) \(CN\) \(c'\) in \(G'\) are the all-zeros pattern and the \(\left(\begin{array}{c}q+1\end{array}\right)\) \(2\) patterns of weight two. Therefore, \(\mathbf{M}_C\) contains the all-zeros pattern and all possible weight-2 patterns.

**Example 17:** We consider trapping set enumerator results for the rate-1/2 LDPC code ensembles in Example 11. In Fig. 3.19 through Fig. 3.21, asymptotic trapping set enumerators for the regular \((6, 3)\) LDPC code are presented. Fig. 3.19 presents a three-dimensional plot of the asymptotic trapping set enumerator...
$r(\alpha, \beta)$. More illuminating is the two-dimensional plot of $r(\alpha, \beta)$ in Fig. 3.20 which is parameterized by $\beta$. Note that, by our set-up earlier in this section, the $\beta = 0$ curve corresponds to the weightumerator for the original protograph (since this implies $b = 0$). Observe that in the figure, with fixed $\alpha$, $r(\alpha, \beta)$ increases with increasing $\beta$. This indicates that, for this ensemble, the number of near codewords of a given weight exceeds that of the true codewords of the same weight.

In Fig. 3.22, $r(\alpha, \Delta \cdot \alpha)$ is evaluated for several values of $\Delta$. Note in Fig. 3.22, for a given $\Delta$, the first zero-crossing is at $r(0, 0)$ and, if there is a second zero-crossing, then a typical $\Delta$-trapping set number exists. Observe that the typical $\Delta$-trapping set numbers decrease as $\Delta$ increases. This demonstrates the benefit of partitioning the trapping sets into the sets $\mathcal{T}_\Delta$.

In Fig. 3.23 we compare the regular (6, 3) protograph and the precoded (6, 3) protograph by drawing the zero-contour curves of their asymptotic trapping set enumerators. The figure shows that the precoded (6, 3) LDPC code ensemble has larger typical $\Delta$-trapping set numbers for the values of $\Delta$ between 0 and 0.1674 (for $\alpha > 0.003$). The zero-contour curve of the RJA protograph also appears in Fig. 3.23 to demonstrate the fact that the (6, 3) protograph outperforms the RJA protograph in the sense of having higher typical $\Delta$-trapping set numbers for all $\Delta$. The zero-contour curves for the RJA and the ARJA ensembles are presented in Fig. 3.24. It is observed that the ARJA code ensemble has larger typical $\Delta$-trapping set numbers for the values of $\Delta$ between 0 and 0.05 (for $\alpha > 0.0043$).

Fig. 3.21 shows the asymptotic elementary trapping set enumerators for the (6, 3) protograph. We found that the zero-contour curves for the asymptotic elementary trapping set enumerators are approximately the same as those for the general trapping set enumerators and so we omit these. The reason they are approximately the same is that, for small values $\alpha$ and $\beta$, the majority of the trapping sets are of the elementary type.

Finally, we reproduced, using our proposed method, the regular (6, 3) trapping set enumerator results in [56]. Note that any regular or irregular LDPC ensemble has a protograph, and so this method can be used to find the trapping set enumerator for
Figure 3.19: Asymptotic trapping set enumerators for regular (6, 3) LDPC code ensemble.
Figure 3.20: Asymptotic trapping set enumerators as a function of $\alpha$ parametrized by $\beta$ for the regular (6, 3) LDPC code ensemble.
Figure 3.21: Asymptotic elementary trapping set enumerators for the (6, 3) regular LDPC code ensemble.
Figure 3.22: Asymptotic trapping set enumerators for different $\Delta$ in the regular $(6, 3)$ LDPC ensemble.
Figure 3.23: The zero-contour curves of the asymptotic trapping set enumerators for different protograph-based LDPC code ensembles.
Figure 3.24: The zero-contour curves of the asymptotic trapping set enumerators for the RJA and the ARJA code ensembles.
any such code. However, unless they are quasi-cyclic or otherwise highly structured, irregular LDPC ensembles have a large number of nodes in their protographs, in which case this method is not practical. □

We would like to make the following remarks: 1) The results for enumerators for \((a, b)\) trapping sets can be used as input-output weight enumerators \(A(a, b)\) for protograph-based low-density generator matrix (LDGM) codes where \(a\) corresponds to the input weight and \(b\) to the output weight (or vice versa). An LDGM code can be used as a precoder to a protograph-based LDPC (a special case is an accumulator that we used as a precoder to lower iterative decoding threshold in ARJA code) or as an outer code with inner accumulator in the irregular repeat accumulate (IRA) protograph codes. 2) Our method (modifying the original protograph for counting check nodes) can also be applied to G-LDPC codes provided we have a definition for trapping sets for G-LDPC codes. For example, we can count the constraint nodes that fail, those that cannot correct more than \((d_{\text{min}} - 1)/2\) errors, noncodewords, or the failure of a certain number of check nodes within the constraint node (in this case we need one output per check node connected to a degree one variable node).

We also know that definition of trapping sets for G-LDPC codes depends on the type of decoder and channel.

3.5 Ensemble pseudocodeword weight enumerators

In this section, we develop a method for computing pseudocodeword weight enumerators for protograph-based G-LDPC code ensembles. Once again, this method leverages off of the method presented in Section 3.2 for computing codeword weight enumerators. We will first require some basic definitions related to pseudocodewords and a description of how pseudocodeword weight enumerators are computed for a specific G-LDPC code.

Consider the Tanner graph \(G\) with \(n\) VNs. Then a degree-\(m\) cover of \(G\) [18] is a Tanner graph \(G^{(m)}\) which is obtained by replicating, \(m\) times, each node of \(G\) and then introducing edges so that the local adjacency is preserved between
the replicated nodes. Now, let \( \mathbf{\hat{c}} = (c_{11}, \ldots, c_{1m}, c_{21}, \ldots, c_{2m}, \ldots, c_{n1}, \ldots, c_{nm}) \) be a codeword of \( G^{(m)} \). Then the vector \( \mathbf{w} = [w_1, w_2, \ldots, w_n] \) is a pseudocodeword of \( G \), where \( w_s = c_{s1} + c_{s2} + \ldots + c_{sm}, s = 1, 2, \ldots, n \).

In [18], it was shown that the set of pseudocodewords for a graph \( G \) can be described by a graph-cover polytope. Here, we instead define the graph pseudocodewords using the notation established in Section 3.2. The key point is to compare the degree-\( m \) cover of the graph \( G \) and a protograph code based on \( m \) copies of \( G \) treated as a protograph: they are identical. (This is true even if \( G \) is itself derived from a protograph, which is the case we consider. Thus, the degree-\( m \) cover of \( G \) is the \( m \)-fold expansion of a graph \( G \) that is the \( N \)-fold expansion of a protograph.) Consequently, a vector \( \mathbf{w} \) is a pseudocodeword of \( G \) if and only if \( A(\mathbf{w}) > 0 \), where \( A(\mathbf{w}) \) is given by (3.3) when it has been applied to \( G \), treated as a protograph. (For distinction from the codeword weight vector \( \mathbf{d} \), we use \( \mathbf{w} \) for the pseudocodeword weight vector.) \( A(\mathbf{w}) \) can be obtained directly from the definition of a pseudocodeword by noticing that \( \mathbf{w} \) is the weight vector of a codeword in the graph cover (of degree \( m \)) and \( A(\mathbf{w}) > 0 \) implies that there is at least one (pseudo)codeword in the graph cover with weight vector \( \mathbf{w} \). It is important to note that \( A(\mathbf{w}) > 0 \) (i.e., \( \mathbf{w} \) is a pseudocodeword of \( G \)) if and only if each subset of \( \mathbf{w} \) “seen” by a given CN is a pseudocodeword of that CN. This can be understood from (3.3) (with \( \mathbf{d} \) replaced by \( \mathbf{w} \)) where \( A(\mathbf{w}) = 0 \) if and only if \( A^{c_j}(\mathbf{w}_j) = 0 \) for at least one CN \( c_j \), where \( A^{c_j}(\mathbf{w}_j) \) is the pseudocodeword enumerator for CN \( c_j \). We denote the set of pseudocodewords for CN \( c_j \) by \( \mathcal{P}_{c_j} \).

Now, let us denote the set of pseudocodewords of \( G \) by \( \mathcal{P}_G \) and the number of pseudocodewords of weight \( d \) by \( A_d^{(pcw)} \). Then

\[
A_d^{(pcw)} = |\{\mathbf{w} : \mathbf{w} \in \mathcal{P}_G, \ PCW(\mathbf{w}) = d\}|, \tag{3.45}
\]

where the function \( PCW(\mathbf{w}) \) gives the pseudocodeword weight of the vector \( \mathbf{w} \), dependent on the channel model as follows [17, 58]:

**AWGN channel:**

\[
PCW(\mathbf{w}) = \left( \sum_{s=1}^{n} w_s \right)^2 / \sum_{s=1}^{n} w_s^2. \tag{3.46}
\]
BEC channel:
\[ \text{PCW}(w) = \text{Number of nonzero elements in } w = |\text{Supp}(w)|. \] 
(3.47)

BSC channel:
\[ \text{PCW}(w) = \begin{cases} 
2e & \text{if } \sum_{s\in E} w_s = \sum_{s=1}^n w_s/2 \\
2e - 1 & \text{if } \sum_{s\in E} w_s > \sum_{s=1}^n w_s/2 
\end{cases}, \] 
(3.48)

where \( e = |E| \) and \( E \) is the smallest set of indices satisfying: \( \sum_{s\in E} w_s \geq \sum_{s=1}^n w_s/2 \).

Example 18: Consider the \((7, 4)\) Hamming code described by
\[ H = \begin{bmatrix} 
1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 
\end{bmatrix}. \]

Using (3.45), we computed the pseudocodeword weight enumerators for the AWGN channel in the degree-3 cover of the code’s graph. The results are presented in the histogram of Fig. 3.25. We also computed these enumerators for the AWGN channel in the degree-20 cover of the code’s graph, where any \( w_s \in w \) takes at most the value 3. The results obtained are identical to those in [58, Fig. 20]. \( \square \)

3.5.1 Finite-length ensemble pseudocodeword weight enumerators

Following the outline of the previous section, in order to leverage the weight enumerator technique of Section 3.2, we consider a degree-\( m \) cover \( G^{(m)} \) of a G-LDPC graph \( G \), where \( G^{(m)} \) is an \( m \)-fold expansion of \( G \), and \( G \) is treated as a protograph. But, because we consider protograph-based G-LDPC codes here, \( G \) is itself an \( N \)-fold expansion of some protograph \( G_p \). Further, just as we considered in Section 3.2 a weight-vector \( d = [d_1, d_2, \ldots, d_n] \) for the graph \( G \) derived from \( N \) copies of the protograph \( G_p \), we consider a weight-vector \( w \) for the graph \( G^{(m)} \) derived from \( m \) copies of \( G \). Moreover, while the elements \( \{d_j\} \) of \( d \) take values from \( \{0, 1, \ldots, N\} \), the elements \( \{w_s\} \) of \( w \) take values from \( \{0, 1, \ldots, m\} \).
Figure 3.25: Pseudocodeword weight enumerators over AWGN channel for the degree-3 cover of the (7, 4) Hamming code.
Now, following the technique for finding codeword weight enumerators, the VNs and the CNs of graph $G$ are treated as constituent codes in a concatenated scheme. Specifically, the group of $N$ VNs of type $v_i$ in $G$ is considered to be a single constituent code with $N$ symbols at its input. And, since each of these $N$ VNs were copied $m$ times, each pseudocodeword symbol belongs to the set $\{0, 1, \ldots, m\}$. Moreover, these $N$ pseudocodeword symbols are characterized by the distribution vector $\partial_i = [d_{i1}d_{i2}\ldots d_{im}]$, where $d_{il}$ is the number of $l$-valued pseudocodeword symbols among the $N$ pseudocodeword symbols at the input of VN $v_i$. In a similar manner, the group of $N$ CNs of type $c_j$ is considered to be a constituent code with $q_{cj}$ $N$-symbol inputs.

Recall that a pseudocodeword on the graph $G$ has to satisfy the constraints of all of the CNs in $G$, which means any CN $c_j$ in $G$ has to see a pseudocodeword from $P_{c_j}$ (note the analogy to a codeword on $G$). Given this and the foregoing discussion, one can mimic the derivation in Section 3.2 to obtain a pseudocodeword enumerator from a code ensemble described by $G$. To do so, one must first transform the nonbinary pseudocodewords to binary words. One then obtains for the ensemble average vector pseudocodeword weight enumerator,

$$A^{(pcw)}(\mathbf{d}) = \frac{\prod_{j=1}^{p_c} A^{P_{c_j}}(\mathbf{d}_j)}{\prod_{i=1}^{n_v} C(N; d_{i0}, d_{i1}, \ldots, d_{im})^{q_{vi} - 1}}, \quad (3.49)$$

where $\mathbf{d} = [\partial_1 \partial_2 \ldots \partial_{n_v}]$, $d_{i0} = N - (d_{i1} + d_{i2} + \ldots + d_{im})$, $\mathbf{d}_j = [\partial_{j1} \partial_{j2} \ldots \partial_{jv_{cj}}]$ is a vector of the distributions which describe the $N$-symbol words on the edges connected to CN $c_j$, and where $A^{P_{c_j}}(\mathbf{d}_j)$ is the pseudocodeword weight-vector enumerator for the type-$c_j$ constituent code. This vector enumerator is given by

$$A^{P_{c_j}}(\mathbf{d}_j) = \sum_{\{n\}} C(N; n_1, n_2, n_K), \quad (3.50)$$

where $\mathbf{n} = [n_1 n_2 \ldots n_K]$, $\{\mathbf{n}\}$ is the set of integer solutions to $\mathbf{d}_j = \mathbf{n} \cdot \mathbf{M}^{{P_{c_j}}}$ with $n_1, n_2, n_K > 0$ and $\sum_{k=1}^{K} n_k = N$, and $\mathbf{M}^{{P_{c_j}}}$ is the binary matrix whose rows are obtained as follows. Each row is related to a given CN pseudocodeword $\mathbf{w}_j \in P_{c_j}$ by the nonbinary-to-binary mapping $\varphi$ defined as $\varphi(\mathbf{w}_j) = \ldots$
\[ x_{11} \ldots x_{1m}, \ x_{21} \ldots x_{2m}, \ldots, \ x_{q_{e1}} \ldots x_{q_{e1}m}, \] where \( x_{il} = 1 \), if \( l = w_i \), otherwise \( x_{il} = 0 \). Note that \( \varphi(w_j) \) acts as a type of incidence vector so that the location of a 1 in \( \varphi(w_j) \) indicates the presence of the corresponding value in \( w_j \). Finally, the average pseudocodeword weight enumerator can be computed as
\[
A^{(pcw)}_d = \sum_{(d)} A^{(pcw)}(d), \quad (3.51)
\]
under the channel-dependent constraint:

**AWGN channel:**

\[
d = \frac{(\sum_{i=1}^{n_v} \sum_{l=1}^{m} l \cdot d_{il})^2}{\sum_{i=1}^{n_v} \sum_{l=1}^{m} l^2 \cdot d_{il}}. \quad (3.52)
\]

**BEC channel:**

\[
d = \sum_{i=1}^{n_v} \sum_{l=1}^{m} d_{il}. \quad (3.53)
\]

**BSC channel:**

\[
d = \begin{cases} 
2e & \text{if } \sum_{l=1}^{m} l \cdot e_l = \sum_{i=1}^{n_v} \sum_{l=1}^{m} l \cdot d_{il}/2 \\
2e - 1 & \text{if } \sum_{l=1}^{m} l \cdot e_l > \sum_{i=1}^{n_v} \sum_{l=1}^{m} l \cdot d_{il}/2 
\end{cases}, \quad (3.54)
\]

where \( e = e_m + e_{m-1} + \ldots + e_1 \) is the smallest number such that \( m \cdot e_m + (m - 1) \cdot e_{m-1} + \ldots + e_1 \geq \sum_{i=1}^{n_v} \sum_{l=1}^{m} l \cdot d_{il}/2 \) and \( e_l \) is an integer between 0 and \( \sum_{i=1}^{n_v} d_{il} \).

The constraint (3.52) is derived from (3.46) as follows: The \( n \) VN s of any realization in the ensemble are divided into \( n_v \) types. Assume that the pseudocodeword components \( w_{s+(i-1)N}, s = 1, 2, \ldots, N, \) are associated with the \( N \) VN s of type \( v_i \). Then one can write \( \sum_{s=1}^{n} w_s \) as \( \sum_{i=1}^{n_v} \sum_{s=1}^{N} w_{s+(i-1)N} \). Now from the definition of \( d_{il} \) above, it is clear that \( \sum_{s=1}^{N} w_{s+(i-1)N} = \sum_{l=1}^{m} l \cdot d_{il} \) and the \( \sum_{s=1}^{N} w_{s+(i-1)N}^2 = \sum_{l=1}^{m} l^2 \cdot d_{il} \). The constraint (3.53) is directly derived from (3.47) by noticing that \( \text{Supp}(w) = |\{w_s > 0, s = 1, \ldots, n\}| = \sum_{i=1}^{n_v} |\{w_{s+(i-1)N} > 0, s = 1, \ldots, N\}| \). But, \( |\{w_{s+(i-1)N} > 0, s = 1, \ldots, N\}| \) is exactly \( \sum_{l=1}^{m} d_{il} \). To derive (3.54) from (3.48), note that a pseudocodeword, which achieves the distribution \( d \), has exactly \( \sum_{i=1}^{n_v} d_{il} \) components with value \( l \) for \( l = 1, 2, \ldots, m \). Consequently, the constraint under (3.54) is equivalent to that under (3.48).
Remark, to find the exact pseudocodeword weight enumerator for an LDPC code ensemble one needs to use the method proposed in this section. Using the check-split method to compute the pseudocodeword weight enumerator for LDPC code ensemble does not give exact results. The check-split method overcounts (see Example 13) and can only be used as an upper bound.

3.5.2 Asymptotic ensemble pseudocodeword weight enumerator

Similar to what was done in Section 3.2 for codeword weight enumerators, one can obtain the asymptotic ensemble pseudocodeword weight enumerator (APCWE). Here asymptotic ensembles means ensembles with infinite code length $n$ and finite cover degree $m$. The expression obtained for a given $m$ is

$$r^{(pcw)}(\delta) = \frac{1}{|S|} \tilde{r}^{(pcw)}(|S| \cdot \delta), \quad (3.55)$$

where

$$\tilde{r}^{(pcw)}(\delta) = \max \left\{ \tilde{\delta} \right\} \left\{ \sum_{j=1}^{n_c} a_{P_{c,j}}(\tilde{\delta}_j) - \sum_{i=1}^{n_v} (q_{e_i} - 1) H \left( \tilde{\delta}_{i0}, \tilde{\delta}_{i1}, \ldots, \tilde{\delta}_{im} \right) \right\}, \quad (3.56)$$

under the channel-dependent constraint (normalized versions of the constraints (3.52)-(3.54)):

**AWGN channel:**

$$\tilde{\delta} = \frac{\left( \sum_{i=1}^{n_v} \sum_{l=1}^{m} l \cdot \tilde{\delta}_{il} \right)^2}{\sum_{i=1}^{n_v} \sum_{l=1}^{m} l^2 \cdot \tilde{\delta}_{il}}. \quad (3.57)$$

**BEC channel:**

$$\tilde{\delta} = \sum_{i=1}^{n_v} \sum_{l=1}^{m} \tilde{\delta}_{il}. \quad (3.58)$$

**BSC channel:**

$$\tilde{\delta} = 2\zeta, \quad (3.59)$$

where $\zeta = \zeta_m + \zeta_{m-1} + \ldots + \zeta_1$ is the smallest number such that $m \cdot \zeta_m + (m-1) \cdot \zeta_{m-1} + \ldots + \zeta_1 = \sum_{i=1}^{n_v} \sum_{l=1}^{m} l \cdot \tilde{\delta}_{il} / 2$ and $0 \leq \zeta \leq \sum_{i=1}^{n_v} \tilde{\delta}_{il}$. In (3.56), $\tilde{\delta} = d / N$, $\tilde{\delta}_j = d_j / N$. 

\( \tilde{d}_i = d_i / N \), \( H(\cdot) \) is the entropy function, and \( a^{P_{c^j}}(\tilde{\delta}_j) \) is the asymptotic vector pseudocodeword weight enumerator of the constraint node \( c^j \). This enumerator is computed as

\[
a^{P_{c^j}}(\tilde{\delta}_j) = \max_{\{P_{\tilde{\delta}_j}\}} \left\{ H(P_{\tilde{\delta}_j}) \right\},
\]

under the constraint that \( \{P_{\tilde{\delta}_j}\} \) is the set of solutions to \( \tilde{\delta}_j = P_{\tilde{\delta}_j} \cdot M^{P_{c^j}} \), with \( P_{\tilde{\delta}_j} = [p_1, p_2, \ldots, p_K] \), \( p_1, p_2, \ldots, p_K \geq 0 \) and \( \sum_{k=1}^{K} p_k = 1 \).

**Example 19:** Consider the rate-1/7 G-LDPC code ensemble in Fig. 3.3, where the parity-check matrix \( H_1 \) corresponds to the (7, 4) Hamming code in Example 18. The parity-check matrix \( H_2 \) corresponds to the following column-permutation of \( H_1 \): (6, 7, 1, 2, 3, 4, 5). That is, the first column of \( H_2 \) is the sixth column of \( H_1 \), ..., and the seventh column of \( H_2 \) is the fifth column of \( H_1 \). Now, let us evaluate its asymptotic pseudocodeword weight enumerator over AWGN channel in the degree-2 cover. First, associate the vector of normalized distributions \( \tilde{\delta} = [D_1, D_2, \ldots, D_7] \) to the vector of VNs \([v_1, v_2, \ldots, v_7]\), where \( D_i = \tilde{d}_i / N = [\tilde{\delta}_{i1}, \tilde{\delta}_{i2}] \). The asymptotic pseudocodeword weight enumerator is given by \( \tilde{r}^{(pcw)}(\tilde{\delta}) = \tilde{r}^{(pcw)}(7\tilde{\delta}) / 7 \), where

\[
\tilde{r}^{(pcw)}(\tilde{\delta}) = \max_{\{\tilde{\delta}\}} \left\{ a^{P_{H_1}}(\tilde{\delta}_1) + a^{P_{H_2}}(\tilde{\delta}_2) - \sum_{i=1}^{7} H \left( [\tilde{\delta}_{i0}, \tilde{\delta}_{i1}, \tilde{\delta}_{i2}] \right) \right\},
\]

such that \( (\sum_{i=1}^{7} \tilde{\delta}_{i1} + 2\tilde{\delta}_{i2})^2 / \sum_{i=1}^{7} (\tilde{\delta}_{i1} + 4\tilde{\delta}_{i2}) = \tilde{\delta} \), where \( \tilde{\delta}_1 = [D_1, D_2, \ldots, D_7] \) and \( \tilde{\delta}_2 = [D_6, D_7, D_1, \ldots, D_5] \). While evaluating \( a^{P_{H_1}}(\tilde{\delta}_1) \), one will see that \( P_{H_1} \) consists of 129 pseudocodewords. For example, \([2, 1, 1, 2, 0, 0, 1] \in P_{H_1} \) and its corresponding pattern in \( M^{P_{H_1}} \) is \( \varphi([2, 1, 1, 2, 0, 0, 1]) = [0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0] \). And so, even for the degree-2 cover, the optimization in (3.61) has a large number of arguments. However, the results of this example showed that the conjecture in Section 3.2 is still valid after defining how to find the SWV of a pattern in \( M^{P_{c^j}} \). To define this, note that each VN in the protograph is associated with \( m \) columns in \( M^{P_{c^j}} \). Accordingly, a row (or, pattern) in \( M^{P_{c^j}} \) can be viewed as \( n_v \) \( m \)-bits symbols. Now, the SWV definition in Section 3.2 is applied to the \( i^{th} \) bit of every symbol. In
this example, the pattern \([0 1, 1 0, 1 0, 0 1, 0 0, 0 0, 1 0]\) can be viewed as \([0 1]\n 1 0\n 0 1\n 0 0\n 0 0\n 1 0\], and so \(\text{SWV}([0 1, 1 0, 1 0, 0 1, 0 0, 0 0, 1 0]) = [3 2]\), or simply \([3 2]\). (Recall that in this example all VNs have the same type.). Applying the conjecture, the number of rows in \(\hat{M}_{PH}^{1}\) is 12 (compared to 129 rows in \(M_{PH}^{1}\)), and from the columns of \(\hat{M}_{PH}^{1}\) one can see that \(D_1 = D_i\), for \(i = 2, ..., 7\).

The asymptotic pseudocodeword weight numerator (APCWE) curve for this code ensemble for degree-2 and degree-3 covers assuming the AWGN channel are shown in Fig. 3.26. Also included in the figure is the asymptotic codeword weight enumerator for the same ensemble. Note that the asymptotic ensemble weight enumerator (AEWE) curve gives a lower bound on the asymptotic pseudocodeword weight numerator curves. This makes sense as any codeword of an LDPC code has a corresponding pseudocodeword in the code’s degree-\(m\) cover, for any value \(m = 2, 3, \ldots\). Further, this pseudocodeword has the same weight as the weight of the corresponding codeword. This means that pseudocodeword weight enumerators of any realization from the code ensemble have to be greater than or equal to the weight enumerators of the realization. Consequently, the average pseudocodeword weight enumerators of the code ensemble have to be greater than or equal to the average weight enumerator of the ensemble. Also, note that both curves have positive zero crossings. Thus, it may be concluded that the typical minimum pseudocodeword weight increases linearly with code length in both the degree-2 and the degree-3 covers. Moreover, the typical minimum weight decreases when moving into a cover of higher degree.

Analogous curves were computed for the BEC and are presented in Fig. 3.27. Similar observations may be made.

Example 20: We evaluate the asymptotic pseudocodeword weight enumerator for the regular LDPC code ensemble with degree-3 VNs and degree-6 CNs. We consider degree-2 and the degree-3 covers on the AWGN channel. The results appear in Fig. 3.28. Note that all three curves intersects the zero axis at the same value, \(\delta = 0.023\). This implies that the typical minimum distance and the typical mini-
Figure 3.26: Asymptotic pseudocodeword weight enumerators over AWGN channel for the rate-1/7 G-LDPC code ensembles with degree-2 and degree-3 covers.
Figure 3.27: Asymptotic pseudocodeword weight enumerators over the BEC for the rate-1/7 G-LDPC code ensemble with degree-2 and degree-3 covers.
minimum pseudocodeword weights for the degree-2 and the degree-3 covers are identical. Consequently, one can say that, asymptotically, with probability close to one all of the ensemble pseudocodewords in the degree-2 and degree-3 covers are good pseudocodewords. A *good pseudocodeword* [58] is a pseudocodeword with weight greater than or equal to the code’s minimum distance.

![Figure 3.28: Asymptotic pseudocodeword weight enumerators over AWGN for the (6, 3) LDPC code ensemble with degree-2 and degree-3 covers.](image)

3.6 Conclusion

In this paper a method for finding codeword weight enumerators for G-LDPC protograph ensembles is derived for finite-length and infinite-length code ensembles. The method is relatively simple for CNs with small dimensionality $k$ and it allowed us to show which G-LDPC ensembles under consideration had the property that $d_{min}$
grows linearly with code length. To reduce the computational complexity for finding the asymptotic codeword weight enumerator, when the CNs in the protograph have large dimensionality $k$, a simplified procedure based on a conjecture is suggested.

We have also presented methods for finding stopping set enumerators, trapping set enumerators, and pseudocodeword enumerators for protograph-based G-LDPC code ensembles for both finite and infinite code lengths (The trapping set case was restricted to LDPC codes.). The methods are conceptually simple and leverage the weight enumeration technique by transforming the stopping set, trapping set, pseudocodeword, enumerator problem into a codeword weight enumerator problem. The asymptotic stopping set enumeration allows us to show which G-LDPC ensembles have a typical stopping set number, that is, have minimum stopping set size that grows linearly with code length. Additionally, to obtain analogous results for the asymptotic trapping set enumerators, the parameter $\Delta$-trapping set number was created and then used to compare various LDPC code ensembles. We believe the trapping set technique can be extended to protograph-based G-LDPC code ensembles, although this is an open problem. The difficulty is that trapping sets for G-LDPC codes never have been defined or studied in literature.

Throughout this paper, several examples for protograph LDPC and G-LDPC code ensembles were presented. Codeword weight enumerators, stopping set enumerators, trapping set enumerators, and pseudocodeword enumerators were evaluated and examined in term of their typical minimum distances, typical stopping set numbers, typical $\Delta$-trapping set numbers, and typical minimum pseudocodeword weight. In general, the asymptotic enumerators for these ensembles indicated that a code drawn from one of these ensembles will has a good performance properties. For most of these examples, instances from the considered protographs were studied in earlier publications [28, 33, 37, 60] by evaluating their error rate performance curves via simulation. The error rate curves showed that these protograph codes are characterized by good performance in both waterfall and error-floor regions, which agrees with the predictions from their enumerators. Also, to show the universality of this method in evaluating weight enumerators for protograph-based LDPC code
ensembles, regular LDPC ensembles, and irregular LDPC ensembles (along with the G-LDPC ensembles), we used this method to successfully regenerate enumerators results in [51, 52, 56].

Future work can utilize these results with other tools (e.g., EXIT chart) in the design of G-LDPC codes with good properties: large $d_{min}$, large $d_{min}^{(s)}$, large $d_{min}^{(ts)}(\Delta)$, large $d_{min}^{(pcw)}$, and good iterative decoding threshold. Also, these results can be extended to include the multi-edge-type LDPC (G-LDPC) codes. In some cases, the asymptotic enumerator results failed to predict whether the ensemble has good properties or not (see the rate-1/6 G-LDPC examples). Consequently, future work may provide an approximation for our results in the finite case, similar to the work in [63], which we expect will enable us to compare a larger number of protographs.
CHAPTER 4

Trapping sets enumerators for specific LDPC codes

4.1 Introduction

Graphical configurations associated with so-called trapping sets in the Tanner graph of an LDPC code are known to create an error-rate floor in the performance curves for iterative decoders [15,54–57]. In [15], Richardson established the notion of trapping sets and suggested a two-phase technique for predicting the performance of LDPC codes in the error-floor region. The first phase searches for trapping sets and their multiplicities (e.g., by simulation) and the second phase evaluates their contribution to the error floor. The authors of [54,55] used a similar technique to predict the floor of several LDPC codes on the binary symmetric channels. In [56], the authors derive the average asymptotic behavior of trapping set enumerators over random, regular and irregular, LDPC code ensembles. In Section 3.4, [64], we proposed a method for computing the trapping set enumerators for protograph-based LPDC codes. We considered both finite-length ensembles and infinite-length ensembles.

In this chapter, we introduce a method to find trapping set enumerators for a specific LDPC code given its parity-check matrix. Together with Richardson’s performance evaluation technique, these results allow one to estimate the performance of an LDPC code in the error-floor region.

In the following section we describe our method which is based on our idea for enumerating protograph-based LDPC codes in Section 3.4, and the improved impulse algorithm in [65]. In Section 4.3, we discuss the complexity of the proposed method. In Section 4.4, we find trapping set enumerators for several LDPC codes that can be found in the standards. In Section 4.5, we conclude this chapter.
4.2 Algorithm description

The proposed method for computing trapping set enumerators for a specific LDPC code consists of two main steps. The first step is to transform the graph of the LDPC code $G$ into a new graph $\tilde{G}$, such that the codewords of the new graph give information about the trapping sets of the original graph. This transformation is similar to the one in Section 3.4. Later in this section we will describe this transformation in detail. The second step is to use a “customized” version of the improved impulse algorithm in [65] to find the enumerators of low-weight codewords for the LDPC code, based on $\tilde{G}$, which can be interpreted as enumerators for small trapping in $G$.

4.2.1 Graph transformation

Recall that, a general $(a, b)$ trapping set, $T_{a,b}$, is a set of VNs of size $a$ which induce a subgraph with exactly $b$ odd-degree check nodes (and an arbitrary number of even-degree check nodes). Now, we describe the first step as follows. Assume we are interested in trapping sets of weight $a$ in the graph $G$ in Fig. 4.1. The value of the companion parameter $b$ depends on which $a$ VNs are of interest, and so we set the values of the $a$ VNs of interest to one and the values of the remaining VNs to zero. With the $a$ VNs so fixed, we are now interested in which CNs “see” odd weight among its neighboring VNs, for the number of such CNs is the corresponding parameter $b$. Such odd-weight CNs can be identified by the addition of auxiliary “flag” VNs to each CN (see protograph $\tilde{G}$ in Fig. 4.2), where the value of an auxiliary VN equals one exactly when its corresponding original CN sees odd weight. The number of auxiliary VNs equal to one is precisely the parameter $b$ for the set of $a$ VNs that were set to one. Thus, to obtain the trapping set enumerator for the original protograph $G$, one may find the weight enumerators of the code described by $\tilde{G}$, partitioning the set of VNs into the original set of transmitted and punctured VNs ($S_t \cup S_p$) and auxiliary flag set ($S_f$). Note that the set $S_t \cup S_p$ (bottom of Fig. 4.2) accounts for the weight $a$ and the set $S_f$ (top of Fig. 4.2) accounts for the weight
b. Also note that, in counting the number of trapping sets, we do not distinguish between transmitted and punctured VNs. However, in evaluating the failure rate of a trapping set, one should make this distinction, but this is not considered in this paper.

From the discussion above, it is important to see that the new graph also corresponds to an LDPC code. Because the value of an auxiliary VN equals one exactly when its corresponding original CN sees odd weight, the inputs at any CN in the new graph, in Fig. 4.2, satisfy the even parity check constraint. The parity check matrix of the new LDPC code is given by

$$\tilde{H} = [H_{m \times n}, I_{m \times m}], \quad (4.1)$$

where $H_{m \times n}$ is the $m \times n$ parity check matrix of the original code, and $I_{m \times m}$ is the $m \times m$ identity matrix.

![Figure 4.1: LDPC graph.](image)

4.2.2 Customized impulse algorithm

With the parity check matrix of the new LDPC code being defined, we need to find a way to enumerate its codewords. For this purpose we use a customized version of the improved impulse algorithm in [65] (customized for finding trapping sets).

It is known that an exhaustive search for the codewords of an LDPC code is an NP-hard problem. Luckily, there exist reliable algorithms, the impulse method
and its modifications, which can approximate the enumerators of low-weight codewords. The impulse method was originally introduced to find the minimum distance of turbo and LDPC codes [66–68]. Later it was modified several times to efficiently approximate the low-weight codeword enumerators of LDPC codes [65, 69–72]. Declercq et al., in [65], improved the impulse method and showed a significant gain in the efficiency of their algorithm. Consequently, we have adopted their method in this work. First, we are going to give a quick overview on this algorithm, and then we will show how to customize this algorithm to search for trapping sets.

Consider a linear code with parity check matrix $H$ of size $m \times n$. Also, observe that the search for low-weight codewords is in the vicinity of the all-zeros codeword. The impulse method in [65] consists of two stages: First, a message passing decoder, initialized with multiple impulses (defined below), is used to update the variable nodes log-likelihood-ratio (LLR) values. The initialization is done as follows: Select one to five VNs randomly and initialize them each to a $-\infty$ LLR called an impulse, then initialize all the other VNs to a small positive LLR (assuming positive LLRs correspond to bit ‘0’). In [65], the min-sum decoder was found to produce the best results. Second, a list decoder is used to construct a list of nearby codewords based on the LLR at each iteration.
The list of nearby codewords is constructed as follows:

- Find the permutation that sorts the LLR values of the bits in an increasing order. Then apply this permutation on $H$ to get $H'$ and perform Gauss elimination on $H'$ to find the most reliable bases of $H$ [73, Chapter 10] conditioned on the all-zero codeword. Next, apply the column permutation employed (if any) during the Gauss elimination on $H'$ to get $H''$. Note, $H''$ has the form of the transpose of a generator matrix (i.e., $H'' = [I|P]$, where $I$ is identity matrix), and so it will be used in the next step to systematically encode a $k$-tuple vector ($k = \text{Rank}(H)$). Moreover, the overall permutation $\pi$, which permutes $H$ to $H''$ can be used to map a codeword $[c_0 \ldots c_{n-1}]$ in the code with $H$ into $[c_{\pi(0)} \ldots c_{\pi(n-1)}]$ in the code with $H''$. Call the $k = n-m$ bits $\{c_{\pi(m)} \ldots c_{\pi(n-1)}\}$ the most reliable bits, and call the other bits the least reliable bits.

- Use order statistics decoder (OSD) [74] to build an initial list of nearby codewords. In an OSD with encoding order $d$ (OSD-$d$), all the non-zero $k$-tuples $[c_{\pi(m)} \ldots c_{\pi(n-1)}]$ with Hamming weight less or equal than $d$ are encoded using the systematic code described by $H''$ to construct the initial list. For practical considerations, namely, memory limitations, keep only codewords with Hamming weight below fixed threshold (fixed to three times the maximum codeword weights for which we would like to enumerate). Also, in our simulations we only used OSD-1.

- Increase the size of the nearby-codeword list by combining the codewords in the list, which share a given number, $n_{cb}$, of common bits in the least reliable part. This step is done cleverly using the box-and-match technique [65, 75].

To enumerate the $(a, b)$-trapping sets in the graph corresponding to $H$, we run the impulse algorithm above on the LDPC code defined by $\tilde{H}$ in (4.1) with the following modifications (this is where the impulse method is customized): 1) Distinguish between the set of the original VNs and the set of the flag VNs, such that the Hamming weight of the original VNs is $a$ and the Hamming weight of
the flag nodes is \( b \). 2) Initialize the LLR values for the set of the flag VNs to a large positive value (in our simulation we set the LLR values for the original variable nodes to 4, and the LLR values for the flag VNs to 150). 3) The maximum codeword weight limit in the algorithm above sets a constraint on the maximum \( a+b \) we are enumerating. 4) Add a constraint on the maximum ratio \( b/a \) returned by the algorithm.

4.3 Intuition and complexity of the customized impulse algorithm

In this section, let us explain the intuition behind choosing these customizations. In particular, why we put a constraint on the ratio \( b/a \), and why we initialize the LLR values of the flag VNs to a large positive value. If the impulse algorithm is run without the proposed customization, the list will become huge very fast. Note, any VN in the graph with degree say \( q_v \) is a \((1, q_v)\)-trapping set and so there are \( n \) trapping set of this kind. Also, any combination of two or more of them results in a new trapping set, and the number of such combination is impractically huge. However, this kind of trapping set, which has a large \( b \) relative to \( a \), is not of great interest to us, as these trapping sets have very small failure rates. Evaluating the failure rate for the trapping sets is not considered here, but according to [15], trapping sets with high failure rates have small values \( a \) and small ratios \( b/a \) (generally, \( b/a \leq 1 \)). Consequently, we have a constraint on the ratio \( b/a \).

Also we force the iterative decoder to work in the the region where trapping sets are more dominate by initializing the LLR values for the flag VNs to a large positive value. Note, if the flag VNs were set to \( +\infty \) LLR values, the iterative decoder will only return codewords of the code described by \( H \) (i.e., \((a, 0)\)-trapping sets)). However, by initializing the LLR values to large positive values instead of infinite values, the decoder will tend to return trapping sets with nominal failure rates (roughly speaking, almost codewords).

The computational complexity of the proposed algorithm depends on the code length \( n \), the number of check nodes \( m \), and the parameter \( n_{cb} \). From the previous
section, the algorithm runs on the code defined by $\tilde{H}$ of size $m \times (n+m)$. In the first stage, the complexity of the iterative decoder is proportional to the number of ones in the parity check matrix. Since we only add VNs of degree one, the complexity of the iterative decoder running on $H$ is approximately the same as when running on $\tilde{H}$. Consequently, the complexity of the iterative decoder depends on the density of the LDPC matrix $H$. Moving to the next stage, the complexity of Gauss elimination for sparse matrices is proportional to the number of ones in the matrix, which depends on the density of the LDPC matrix $H$. Next, the size of the OSD-$d$ list is $\sum_{l=1}^{d} \binom{n}{l}$, and the encoding of each codeword in the list requires $O(n^2)$ operations. Finally, the addition of one codeword to the nearby codeword list in the box-and-match step requires $n + m$ addition operations.

4.4 Trapping set enumerators for selected LDPC codes

In this section we demonstrate the efficiency of the proposed method by estimating the trapping set enumerators for several known LDPC codes. For the simulation in all our examples we used OSD-1, $n_{cb} = 7$, LLR value 4 for the original VNs, and LLR value 150 for the flag VNs.

*Example 21:* Consider the $(2640, 1320)$ Margulis code [76]. This code is a well-known example and often used as a reference when studying trapping sets. We evaluated the enumerators for the $(a, b)$-trapping sets, which have $a + b < 25$ and $b/a \leq 0.5$. The results are shown in Table 4.1.

The enumerators in Table 4.1 represent a lower bound on the actual enumerators. This is because the algorithm found these trappings sets but it may have missed some of them. We claim that these enumerators have a good accuracy. This accuracy is inherited from the high efficiency of the impulse method in [65]. Also, it was reported in [15] that this Margulis code has exactly 1320 $(12, 4)$-trapping sets, and 1320 $(14, 4)$-trapping sets, in agreement with Table 4.1. Finally, it is worth mentioning that we run the program for about one week to get these results, but about 90% of them were obtained in the first 24 hours. □
Table 4.1: \((a, b)\)-trapping set enumerators for the \((2640, 1320)\) Margulis code

<table>
<thead>
<tr>
<th>((a, b))</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>((12, 4))</td>
<td>1320</td>
</tr>
<tr>
<td>((14, 4))</td>
<td>1320</td>
</tr>
<tr>
<td>((11, 5))</td>
<td>9</td>
</tr>
<tr>
<td>((13, 5))</td>
<td>2699</td>
</tr>
<tr>
<td>((15, 5))</td>
<td>7938</td>
</tr>
<tr>
<td>((14, 6))</td>
<td>2703</td>
</tr>
<tr>
<td>((16, 6))</td>
<td>21153</td>
</tr>
<tr>
<td>((18, 6))</td>
<td>2642</td>
</tr>
<tr>
<td>((15, 7))</td>
<td>27766</td>
</tr>
<tr>
<td>((17, 7))</td>
<td>46223</td>
</tr>
</tbody>
</table>

Example 22: Let us consider the \((32, 16)\) LDPC code \cite{17}, which is described by the parity check matrix in Fig. 4.3 (each dot in the figure corresponds to a one in the parity check matrix). We used our algorithm to evaluate the \((a, b)\)-trapping set enumerators of this code, under the constraints \(a + b < 7\) and \(b/a \leq 0.75\). The results appears in Table 4.2.

Figure 4.3: Parity check matrix for the \((32, 16)\) LDPC code.

Note that the \((4, 0)\)-trapping sets and the \((6, 0)\)-trapping sets are codewords (of both the original code and of the transformed code) of weight 4 and 6, respectively.
Table 4.2: $(a, b)$-trapping set enumerators for the $(32, 16)$ LDPC code in Example 22

<table>
<thead>
<tr>
<th>$(a, b)$</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(4, 0)$</td>
<td>6</td>
</tr>
<tr>
<td>$(6, 0)$</td>
<td>16</td>
</tr>
<tr>
<td>$(3, 1)$</td>
<td>8</td>
</tr>
<tr>
<td>$(5, 1)$</td>
<td>12</td>
</tr>
<tr>
<td>$(4, 2)$</td>
<td>200</td>
</tr>
</tbody>
</table>

Table 4.3: $(a, b)$-trapping set enumerators for the $(128, 64)$ LDPC code in Example 23

<table>
<thead>
<tr>
<th>$(a, b)$</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(7, 1)$</td>
<td>48</td>
</tr>
<tr>
<td>$(4, 2)$</td>
<td>96</td>
</tr>
<tr>
<td>$(6, 2)$</td>
<td>224</td>
</tr>
<tr>
<td>$(5, 3)$</td>
<td>2464</td>
</tr>
</tbody>
</table>

One can easily check that this code has exactly 6 codewords of weight 4, and 16 codewords of weight 6. These results agrees with the results returned by the proposed method. The proposed algorithm returns these trapping set enumerators in about one minute. □

Example 23: We also evaluated the trapping set enumerators for another two codes from the [77]. The first is the $(128, 64)$ LDPC code, for which we enumerate $(a, b)$-trapping sets which satisfy the constraints $a + b < 10$ and $b/a \leq 0.6$. The results are shown in Table 4.3. The second is the $(256, 128)$ LDPC code. The outcome enumerators for the $(a, b)$-trapping set, which satisfy the constraints $a + b < 21$ and $b/a \leq 0.6$, are shown in Table 4.4. The algorithm needs about one day to return the results for each code.

Example 24: Consider the following three LDPC codes from the 802.11n standard: the $(648, 324)$ LDPC code, the $(1296, 648)$ LDPC code, and the $(1944, 972)$ LDPC code. We estimated the $(a, b)$-trapping set enumerators for each code. The
Table 4.4: \((a, b)\)-trapping set enumerators for the \((256, 128)\) LDPC code in Example 23 (only those with \(b \leq 4\) are listed below)

<table>
<thead>
<tr>
<th>((a, b))</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>((19, 1))</td>
<td>38</td>
</tr>
<tr>
<td>((8, 2))</td>
<td>32</td>
</tr>
<tr>
<td>((12, 2))</td>
<td>32</td>
</tr>
<tr>
<td>((14, 2))</td>
<td>91</td>
</tr>
<tr>
<td>((16, 2))</td>
<td>131</td>
</tr>
<tr>
<td>((18, 2))</td>
<td>287</td>
</tr>
<tr>
<td>((5, 3))</td>
<td>32</td>
</tr>
<tr>
<td>((7, 3))</td>
<td>32</td>
</tr>
<tr>
<td>((9, 3))</td>
<td>191</td>
</tr>
<tr>
<td>((11, 3))</td>
<td>117</td>
</tr>
<tr>
<td>((13, 3))</td>
<td>457</td>
</tr>
<tr>
<td>((15, 3))</td>
<td>794</td>
</tr>
<tr>
<td>((17, 3))</td>
<td>863</td>
</tr>
<tr>
<td>((8, 4))</td>
<td>540</td>
</tr>
<tr>
<td>((10, 4))</td>
<td>1600</td>
</tr>
<tr>
<td>((12, 4))</td>
<td>1645</td>
</tr>
<tr>
<td>((14, 4))</td>
<td>2466</td>
</tr>
<tr>
<td>((16, 4))</td>
<td>2231</td>
</tr>
</tbody>
</table>
Table 4.5: \((a, b)\)-trapping set enumerators for the \((648, 324)\) LDPC code in 802.11n

<table>
<thead>
<tr>
<th>((a, b))</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>((15, 0))</td>
<td>27</td>
</tr>
<tr>
<td>((12, 1))</td>
<td>54</td>
</tr>
<tr>
<td>((13, 1))</td>
<td>155</td>
</tr>
<tr>
<td>((14, 1))</td>
<td>316</td>
</tr>
<tr>
<td>((4, 2))</td>
<td>216</td>
</tr>
<tr>
<td>((5, 2))</td>
<td>189</td>
</tr>
<tr>
<td>((6, 2))</td>
<td>189</td>
</tr>
<tr>
<td>((7, 2))</td>
<td>297</td>
</tr>
<tr>
<td>((8, 2))</td>
<td>590</td>
</tr>
<tr>
<td>((10, 2))</td>
<td>1810</td>
</tr>
<tr>
<td>((11, 2))</td>
<td>2948</td>
</tr>
<tr>
<td>((12, 2))</td>
<td>4266</td>
</tr>
<tr>
<td>((13, 2))</td>
<td>5569</td>
</tr>
<tr>
<td>((5, 3))</td>
<td>11301</td>
</tr>
<tr>
<td>((6, 3))</td>
<td>14200</td>
</tr>
<tr>
<td>((7, 3))</td>
<td>14324</td>
</tr>
</tbody>
</table>

results are shown in Table 4.5, Table 4.6, and Table 4.7.

Note, these LDPC codes are quasi-cyclic codes. The parity check matrix for the \((648, 324)\), \((1296, 648)\), and \((1944, 972)\) LDPC codes are constructed from circulant-block matrices of size 27, 54, and 81, respectively. Interestingly, the number of \((15, 0)\)-trapping sets in the \((648, 324)\) LDPC code, which are returned by the proposed algorithm is 27 (matches the size of the constructing circulant permutation matrices). The same observation holds for the other two codes: the number of \((12, 1)\)-trapping sets in the \((1296, 648)\) LDPC code is 54, and the number of \((12, 1)\)-trapping sets in the \((1944, 972)\) LDPC code is 81. In general, the number of \((a, b)\)-trapping sets tends to be multiple of the size of the circulant permutation matrices. Lastly, the proposed algorithm returned the results for the \((648, 324)\) LDPC code in about two days. For the other two codes, the simulation took about five days. □

**Example 25:** Consider the \((2048, 1723)\) LDPC code from the IEEE 802.3an...
Table 4.6: \((a, b)\)-trapping set enumerators for the \((1296, 648)\) LDPC code in 802.11n

<table>
<thead>
<tr>
<th>((a, b))</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>(12, 1)</td>
<td>54</td>
</tr>
<tr>
<td>(13, 1)</td>
<td>108</td>
</tr>
<tr>
<td>(14, 1)</td>
<td>161</td>
</tr>
<tr>
<td>(4, 2)</td>
<td>432</td>
</tr>
<tr>
<td>(5, 2)</td>
<td>378</td>
</tr>
<tr>
<td>(6, 2)</td>
<td>378</td>
</tr>
<tr>
<td>(7, 2)</td>
<td>537</td>
</tr>
<tr>
<td>(8, 2)</td>
<td>803</td>
</tr>
<tr>
<td>(10, 2)</td>
<td>1301</td>
</tr>
<tr>
<td>(11, 2)</td>
<td>1406</td>
</tr>
<tr>
<td>(12, 2)</td>
<td>1396</td>
</tr>
<tr>
<td>(13, 2)</td>
<td>1559</td>
</tr>
<tr>
<td>(5, 3)</td>
<td>19623</td>
</tr>
<tr>
<td>(6, 3)</td>
<td>23514</td>
</tr>
<tr>
<td>(7, 3)</td>
<td>21916</td>
</tr>
</tbody>
</table>

Table 4.7: \((a, b)\)-trapping set enumerators for the \((1944, 972)\) LDPC code in 802.11n

<table>
<thead>
<tr>
<th>((a, b))</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>(12, 1)</td>
<td>81</td>
</tr>
<tr>
<td>(13, 1)</td>
<td>162</td>
</tr>
<tr>
<td>(14, 1)</td>
<td>162</td>
</tr>
<tr>
<td>(4, 2)</td>
<td>648</td>
</tr>
<tr>
<td>(5, 2)</td>
<td>567</td>
</tr>
<tr>
<td>(6, 2)</td>
<td>486</td>
</tr>
<tr>
<td>(7, 2)</td>
<td>485</td>
</tr>
<tr>
<td>(8, 2)</td>
<td>637</td>
</tr>
<tr>
<td>(10, 2)</td>
<td>1210</td>
</tr>
<tr>
<td>(11, 2)</td>
<td>1635</td>
</tr>
<tr>
<td>(12, 2)</td>
<td>2166</td>
</tr>
<tr>
<td>(13, 2)</td>
<td>2930</td>
</tr>
<tr>
<td>(5, 3)</td>
<td>27821</td>
</tr>
<tr>
<td>(6, 3)</td>
<td>33378</td>
</tr>
<tr>
<td>(7, 3)</td>
<td>31337</td>
</tr>
</tbody>
</table>
Table 4.8: \((a, b)\)-trapping set enumerators for the (2048,1723) LDPC code in 802.3an

<table>
<thead>
<tr>
<th>((a, b))</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>(14, 0)</td>
<td>1407</td>
</tr>
<tr>
<td>(16, 0)</td>
<td>13359</td>
</tr>
<tr>
<td>(18, 0)</td>
<td>822</td>
</tr>
<tr>
<td>(20, 0)</td>
<td>1809</td>
</tr>
<tr>
<td>(22, 0)</td>
<td>4500</td>
</tr>
<tr>
<td>(24, 0)</td>
<td>43731</td>
</tr>
<tr>
<td>(13, 6)</td>
<td>35</td>
</tr>
<tr>
<td>(15, 6)</td>
<td>441</td>
</tr>
<tr>
<td>(16, 6)</td>
<td>1211</td>
</tr>
<tr>
<td>(16, 8)</td>
<td>23</td>
</tr>
</tbody>
</table>

(10 Gigabit Ethernet) standard. This code is constructed based on Reed-Solomon code [78]. We used the proposed algorithm to enumerate the \((a, b)\)-trapping sets of this code which satisfy \(a + b < 25\) and \(b/a \leq 0.5\). The results are shown in Table 4.8. The algorithm returned these results in about 2 days.

□

4.5 Conclusion

In this chapter we proposed an algorithm for estimating the trapping set enumerators for specific LDPC codes. We demonstrated the efficiency of the proposed algorithm by evaluating the trapping set enumerators for the Margulis code and for several LDPC codes from standards. The algorithm is reliable in term of its speed and accuracy. Note that obtaining these results using an exhaustive search for trapping sets in the code’s graph is practically impossible. However, this algorithm represent a very attractive practical solution for this problem.
5.1 Introduction

The Mars Laser Communications Development (MLCD) project was initiated because of the recognition of certain advantages that laser communications systems have over radio-frequency (RF) communication systems. These advantages include smaller transmitter mass (important for deep-space missions), more concentrated transmit beamwidth (hence, more efficient reception), and larger bandwidth (hence, higher data rates).

Consequently, researchers have been interested in the development of a suitable forward error control (FEC) scheme for the free-space optical channel. In [79–82], the authors proposed a serially concatenated convolutional code (SCCC) scheme in combination with pulse-position modulation (PPM) for this project. In [83], the authors also designed an LDPC code for this project. In their design of the LDPC code, the authors proposed an EXIT chart technique, in which they assume that the interchanged messages in the decoder have a Gaussian distribution. The SCCC scheme out-performs the LDPC code design by about 0.3 dB photons/ns. Moreover, an SCCC encoder has lower complexity than an LDPC encoder. Low-complexity encoding is clearly of great importance for this scenario in which the encoder is space-based and the decoder is earth-based. However, it is not clear if the authors of [79–82] considered quasicycle (QC) LDPC codes or protograph-based LDPC codes [19] for which the encoder is often low-complexity because they are implementable via shift-register circuits [84]. Thus, NASA engineers are still curious about the outcome of the SCCC vs. LDPC comparison in view of the changing LDPC code state-of-the-art.

Although these coding schemes were developed for space applications, visible
light communications (being developed for in-room communications) can use similar coded-modulation schemes to achieve reliable communications. As LDPC codes decoders have lower complexity than SCCC codes and can have parallel decoding architecture for fast decoding speed, LDPC codes are more suitable in such applications where decoding speed and complexity are very important factors.

In this chapter we give a detailed presentation of the decoding of LDPC-PPM coded modulation scheme over the free space optical channel. We present three decoding schemes for LDPC codes: 1) a non-iterative LDPC-PPM decoder, where no iteration is done between the PPM demodulator and the LDPC decoder; 2) an iterative LDPC-PPM decoder; and 3) an iterative LDPC-APPM decoder (APPM refers to accumulate PPM, see [82]), where we use of an accumulator between the PPM demodulator and the LDPC decoder. The authors in [83] did not publish the math used in their decoding scheme and so we derive the math for the first two decoding schemes presented in this chapter (to our knowledge, this math is not available in literature). The third decoding scheme has the same APPM decoder in [82], and so, we refer the reader to [82] for detailed derivations. We also compare the performance of the different decoders and simulated different LDPC codes, which were optimized for the AWGN channel. Moreover, we propose a density evolution algorithm for LDPC codes over the free space optical channel. We show that the Gaussian assumption on the decoder messages is not accurate, and so the proposed density evolution algorithm provides more accurate predictions on the code’s threshold (performance). We provide some degree distributions for LDPC codes designed using the proposed density evolution tool, and showed that these codes have better performances than those optimized for the AWGN channel.

The chapter is organized as follows: In Section 5.2, the system model, which we use in later stages, is described. In this section, the various system blocks are described in general, but with special focus on the channel model. In Section 5.3, we present three decoding schemes for decoding the LDPC-PPM coded modulation scheme over the free space optical channel. In Section 5.4, we present our proposed density evolution algorithm. Numerical results are presented in Section 5.5. Section
5.2 System model

The LDPC-PPM coded modulation scheme is shown in Fig. 5.1. First, the information bits, $U$, are encoded in the LDPC encoder block. Then, the encoded bits, $X$, are mapped into PPM symbols, $C$, in the bit-to-PPM block. After that, the PPM symbols are transmitted over the free-space optical channel. Next, the received data, $Y$, is demodulated using the PPM demodulator block, which produces soft-values for the encoded bits, denoted by $L(\hat{X})$. Then, these LLR values are passed to the LDPC decoder. At this stage, depending on the decoding scheme, either the LDPC iterative decoder is run for some number of iterations followed by decisions on the information bits, $\hat{U}$, or an iterative procedure is performed between the LDPC decoder and the PPM demodulator. In the latter case, new LLR values are generated after one iteration of the LDPC decoder, then sent back to the PPM demodulator, which in turn recalculate new LLR values based on the channel data and the feedback LLR. This process is repeated for a given number of iterations, or until it is stopped by a stopping rule, after which decisions on the information bits are made.

The LDPC-PPM decoder will be studied in the next section. Before that, we define the bit-to-PPM demodulator and the channel model considered in this chapter.
5.2.1 Bit-to-PPM modulator

The bit-to-PPM modulator under consideration maps each block $X$ of size $\log(M)$ of into an $M$-tuple symbol $C$. Denote the $k^{th}$ block at the modulator input by $X_k = [x_{k,0}, x_{k,1}, \ldots, x_{k,\log(M)-1}]$, and the corresponding output by $C_k = [c_{k,0}, c_{k,1}, \ldots, c_{k,M-1}]$. $C_k$ consists of $M - 1$ zeros and a single one whose position depends on $X_k$. The mapping rule is the following: when the binary number $x_{k,0}x_{k,1} \ldots x_{k,\log(M)-1} \equiv j$ (decimal), $c_{k,j} = 1$. Further, let us refer to the set of all possible $C_k$ by $C$ (Note, $|C| = M$).

5.2.2 Channel model

Recall, in a PPM scheme an optical pulse is transmitted in the time slot corresponding to $c_{k,j} = 1$. To detect this at the receiver side, a photon-counting detector is used. The output of the photon-counting detector is $Y$. Denote the output sequence, which correspond to $C_k$, by $Y_k = [y_{k,0}, y_{k,1}, \ldots, y_{k,M-1}]$, where $y_{k,i}$ is an integer between zero and infinity. $y_{k,i}$, call it $y$, can be modeled as a Poisson process with mean $n_b$ in an empty slot (noise slot) and mean $n_b + n_s$ in an occupied slot (signal-plus-noise slot). Consequently, the corresponding probability mass functions (pmf) are

$$p_{Y|C}(y|c = 1) = \frac{(n_s + n_b)y^{-(n_s+n_b)}}{y!}, \quad (5.1)$$

and

$$p_{Y|C}(y|c = 0) = \frac{(n_b)y^{-n_b}}{y!}. \quad (5.2)$$

Note that the channel is assumed to be memoryless channel, that is, $y_{k,i}$ and $y_{k,j}$, $i \neq j$, are independent Poisson random variables.

5.3 LDPC-PPM coded modulation decoding

In this section, the PPM demodulator is described in detail, and its interaction with the LDPC decoder is demonstrated. The LDPC-PPM demodulator/decoder can be iterative or non-iterative as described below.
5.3.1 LDPC-PPM non-iterative demodulator/decoder

In the non-iterative receiver, the PPM demodulator calculates LLR values for the received bits based on the channel output \( y \). Then the LLR values at the output of the PPM demodulator are passed to the LDPC decoder, which runs for some number of iterations then provide decisions on the information bits. No iteration between the LDPC decoder and the PPM demodulator are performed.

The belief propagation decoder is used in the LDPC decoder. The demodulator computes the LLR

\[
L(x_{k,i}) = \log \left( \frac{p(x_{k,i} = 0 | Y_k)}{p(x_{k,i} = 1 | Y_k)} \right). \tag{5.3}
\]

To derive a close-form expression for this LLR, note that

\[
p(x_{k,i} = 1 | Y_k) = \frac{p(x_{k,i} = 1, Y_k)}{p(Y_k)} = \frac{1}{p(Y_k)} \sum_{C_k \in C^1_i} p_{Y|C}(Y_k | C_k)p(C_k), \tag{5.4}
\]

where \( C_k \in C \) and \( C^1_i = \{C_k : C_k \in C, \text{ where corresponding } X_k \text{ has } x_{k,i} = 1\} \). Also, one can assume that \( p(C_k) = 1/M \). Now, because the channel is memoryless,

\[
p_{Y|C}(Y_k | C_k) = \prod_{m=0}^{M-1} p_{Y|C}(y_{k,m} | c_{k,m}),
\]

\[
= p_{Y|C}(y_{k,j} | c_{k,j} = 1) \prod_{m=0, m \neq j}^{M-1} p_{Y|C}(y_{k,m} | c_{k,m} = 0),
\]

\[
= \frac{p_{Y|C}(y_{k,j} | c_{k,j} = 1)}{p_{Y|C}(y_{k,j} | c_{k,j} = 0)} \prod_{m=0}^{M-1} p_{Y|C}(y_{k,m} | c_{k,m} = 0). \tag{5.5}
\]

And so, (5.4) can be rewritten as

\[
p(x_{k,i} = 1 | Y_k) = \frac{\prod_{m=0}^{M-1} p_{Y|C}(y_{k,m} | c_{k,m} = 0)}{M \cdot p(Y_k)} \sum_{C_k \in C^1_i} \frac{p_{Y|C}(y_{k,j} | c_{k,j} = 1)}{p_{Y|C}(y_{k,j} | c_{k,j} = 0)}. \tag{5.6}
\]

Similarly, \( p(x_{k,i} = 0 | Y_k) \) is derived as

\[
p(x_{k,i} = 0 | Y_k) = \frac{\prod_{m=0}^{M-1} p_{Y|C}(y_{k,m} | c_{k,m} = 0)}{M \cdot p(Y_k)} \sum_{C_k \in C^0_i} \frac{p_{Y|C}(y_{k,l} | c_{k,l} = 1)}{p_{Y|C}(y_{k,l} | c_{k,l} = 0)} . \tag{5.7}
\]
where $C_0^i = \{ C_k : C_k \in C, \text{where corresponding} \ X_k \text{has} \ x_{k,i} = 0 \}$. Also, note from (5.1) and (5.2)

\[
p_{Y|C}(y|c = 1) = \left( \frac{n_s + n_b}{n_b} \right)^y e^{-n_s}.
\]

(5.8)

By substituting (5.6), (5.7), and (5.8) in (5.3) one gets

\[
L(x_{k,i}) = \log \left( \sum_{\{l : C_k \in C_0^i, c_{k,l} = 1 \}} \left( 1 + \frac{n_s}{n_b} \right)^{y_{k,l}} \right) - \log \left( \sum_{\{j : C_k \in C_1^i, c_{k,j} = 1 \}} \left( 1 + \frac{n_s}{n_b} \right)^{y_{k,j}} \right).
\]

(5.9)

For sake of implementation, note that $x^y = e^{y \log(x)}$. Consequently,

\[
\log \left( \sum_j \left( 1 + \frac{n_s}{n_b} \right)^{y_j} \right) = \log \left( \sum_j e^{y_j \log \left( 1 + \frac{n_s}{n_b} \right)} \right) = \max_j \{ y_j' \},
\]

(5.10)

where $y_j' = y_j \cdot \log \left( 1 + \frac{n_s}{n_b} \right)$, and $\max^* (x, y) = \log(e^x + e^y)$ (we similarly define $\max^*$ when more than two variables are involved.). And so, $L(x_{k,i})$ can be written as

\[
L(x_{k,i}) = \max_{\{l : C_k \in C_0^i, c_{k,l} = 1 \}} \{ y_{k,l}' \} - \max_{\{j : C_k \in C_1^i, c_{k,j} = 1 \}} \{ y_{k,j}' \}.
\]

(5.11)

5.3.2 LDPC-PPM iterative demodulator/decoder

In contrast to the demodulator/decoder in the previous subsection, this demodulator/decoder iteratively passes messages between the LDPC decoder and the PPM demodulator. In fact, both blocks can be viewed as a single block, which we will call LDPC-PPM decoder. The LDPC-PPM decoder works on a graph, which consists of three types of nodes (constraint nodes (CNs), variable nodes (VNs), and demodulator nodes (DMs)). This graph is depicted in Fig. 5.2.

To decode on this graph, the DMs operations must be derived. Note that the functionalities of the VNs and the CNs are the same as in any LDPC decoder. For example, a VN, computing a message to send on edge $e$, sums all the incoming messages (from all edges) except that from edge $e$. (The edge connecting the DM with the VN is treated as any other edge, though there is a difference in the scheduling
Figure 5.2: LDPC-PPM decoder.

as will be explained later.) Note the DM node needs to compute

\[
L^{(t)}_{\text{DM}\rightarrow\text{VN}}(x_{k,i}) = \log \left( \frac{p(x_{k,i} = 0|Y_k, L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k))}{p(x_{k,i} = 1|Y_k, L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k))} \right),
\]

(5.12)

where \(L^{(t)}_{\text{DM}\rightarrow\text{VN}}(x_{k,i})\) is the message from the DM to the VN which estimates the LLR value for the bit \(x_{k,i}\), \(t\) is the iteration number, and

\[
L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k) = [L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(x_{k,0}), L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(x_{k,1}), \ldots, L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(x_{k,\log(M)-1})]
\]

is a vector of the LLR values send from the VNs to the DMs at the end of the \((t-1)^{th}\) iteration.

Note,

\[
p(x_{k,i} = 1|Y_k, L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k)) = \frac{1}{p(Y_k)} \sum_{C_k \in \mathcal{C}_i^1} p(Y_k|C_k, L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k)) p(C_k|L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k)),
\]

(5.13)

In this equation, \(p_{Y|C}(Y_k|C_k)\) is evaluated in (5.5). To evaluate \(p(C_k|L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k))\), the following trick is used. Recall that there is one-to-one correspondence between \(C_k\) and \(X_k\), then for any \(C_k \in \mathcal{C}_i^1\), write

\[
p(C_k|L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k)) = p(X_k|L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k)) = p(X_k|x_{k,i} = 1, L^{(t-1)}_{\text{VN}\rightarrow\text{DM}}(X_k)),
\]

(5.14)
In the last equality, the independence assumption is considered to hold. Then, we normalize the above expression by 
\[ \prod_{q=0, q \neq i}^{\log(M)-1} p(x_{k,q} = 0|L_{\text{VN-DM}}^{(t-1)}(X_k)) = B, \]
which is a constant independent of \( C_k \). Now,
\[
p(C_k|L_{\text{VN-DM}}^{(t-1)}(X_k)) = \prod_{q=0, q \neq i}^{\log(M)-1} \frac{p(x_{k,q} = 0|L_{\text{VN-DM}}^{(t-1)}(X_k))}{p(x_{k,q} = 0|L_{\text{VN-DM}}^{(t-1)}(X_k))},
\]

\[
= \prod_{q \in Q_i(X_k)} \frac{p(x_{k,q} = 0|L_{\text{VN-DM}}^{(t-1)}(X_k))}{p(x_{k,q} = 0|L_{\text{VN-DM}}^{(t-1)}(X_k))}, \tag{5.15}
\]

where \( Q_i(X_k) \triangleq \{ q \in \{0, 1, \ldots, \log(M) - 1 \} : q \neq i, x_{k,q} = 1 \text{ in } X_k \} \).

But
\[
L_{\text{VN-DM}}^{(t-1)}(x_{k,q}) = \log \left( \frac{p(x_{k,q} = 0|L_{\text{VN-DM}}^{(t-1)}(X_k))}{p(x_{k,q} = 1|L_{\text{VN-DM}}^{(t-1)}(X_k))} \right), \tag{5.16}
\]

then
\[
p(C_k|L_{\text{VN-DM}}^{(t-1)}(X_k)) = B \cdot \prod_{q \in Q_i(X_k)} \exp \left( -L_{\text{VN-DM}}^{(t-1)}(x_{k,q}) \right),
\]

\[
= B \cdot \exp \left( - \sum_{q \in Q_i(X_k)} L_{\text{VN-DM}}^{(t-1)}(x_{k,q}) \right). \tag{5.17}
\]

Substitute (5.17), (5.5), and (5.8) in (5.13) (and note that \( x^y e^z = e^{y \log(x) + z} \)) to get
\[
p(x_{k,i} = 1|Y_k, L_{\text{VN-DM}}^{(t-1)}(X_k)) =
\]
\[
\frac{B \cdot \prod_{m=0}^{M-1} p_{Y|C}(y_{k,m}|c_{k,m} = 0)}{p(Y_k)} \sum_{\{j: c_k \in C_i, c_{k,j} = 1\}} \exp \left( y'_{k,j} - \sum_{q \in Q_i(X_k)} L_{\text{VN-DM}}^{(t-1)}(x_{k,q}) \right), \tag{5.18}
\]

Similarly, \( p(x_{k,i} = 0|Y_k, L_{\text{VN-DM}}^{(t-1)}(X_k)) \) is derived and given by
\[
p(x_{k,i} = 0|Y_k, L_{\text{VN-DM}}^{(t-1)}(X_k)) =
\]
\[
\frac{B \cdot \prod_{m=0}^{M-1} p_{Y|C}(y_{k,m}|c_{k,m} = 0)}{p(Y_k)} \sum_{\{l: c_k \in C_i^0, c_{k,l} = 1\}} \exp \left( y'_{k,l} - \sum_{r \in Q_i(X_k)} L_{\text{VN-DM}}^{(t-1)}(x_{k,r}) \right). \tag{5.19}
\]
Finally, from (5.18) and (5.19) one gets

\[
L_{DM \rightarrow VN}^{(t)}(x_{k,i}) = \max_{\{l:C_k \in C_1^l, c_k,i = l\} \atop \{i:C_k \in C_0^l, c_k,i = l\}} \left\{ y'_{k,l} - \sum_{r \in \mathcal{Q}_i(X_k)} L_{VN \rightarrow DM}^{(t-1)}(x_{k,r}) - \sum_{q \in \mathcal{Q}_i(X_k)} L_{VN \rightarrow DM}^{(t-1)}(x_{k,q}) \right\} - \max_{\{j:C_k \in C_1^j, c_k,j = j\} \atop \{j:C_k \in C_0^j, c_k,j = j\}} \left\{ y'_{k,j} - \sum_{q \in \mathcal{Q}_i(X_k)} L_{VN \rightarrow DM}^{(t-1)}(x_{k,q}) \right\},
\]

(5.20)

Recall, \( j \) describes a specific \( C_k \) which corresponds to a specific \( X_k \). Consequently, the index \( q \) in the equation above depends on \( j \). Similarly, \( r \) depends on \( l \).

After deriving the DM functionality in (5.20), the LDPC-PPM iterative decoder can be summarized in the following algorithm:

**Initialization**

- For all \( k \), for \( i = 0, 1, \ldots, \log(M) - 1 \), set \( L_{VN \rightarrow DM}^{(0)}(x_{k,i}) = 0 \).
- For all \( k \), for \( j = 0, 1, \ldots, M - 1 \), compute \( y'_{k,j} = y_{k,j} \cdot \log\left(\frac{1 + n_s}{n_b}\right) \).

**Start iterating**

- For all DMs, run the DM to compute \( L_{DM \rightarrow VN}^{(t)}(x_{k,i}) \). Send LLR to VNs.
- For all VNs, run the VN to compute LLR value for each edge connected to a CN. Send LLR to neighbor-CNs.
- For all CNs, run the CN to compute LLR. Send LLR to VNs.
- If maximum number of iteration reached (or, stopping rule hold), then exit iteration.
- For all VNs, run the VN to compute LLR value for each edge connected to a DM. Send LLR to neighbor-DMs.
- Repeat.

**After the last iteration**

- For all VNs, compute \( L(x) = \) sum of all messages from all incident edges. If \( L(x) > 0 \), then the VN value is 0, else VN value is 1.
5.3.3 LDPC-APPM iterative decoder

In this scheme, it is assumed that the output of the LDPC encoder is passed through an accumulator before entering the bit-to-PPM modulator. The APPM demodulator in this case is the same as described in [82]. However, as the outer code in this scheme is an LDPC code, the outer convolutional decoder in [82] is replaced by an LDPC decoder.

5.4 Density evolution for the LDPC-PPM iterative decoding

In this section, we devise density evolution technique for the design of LDPC codes for the LDPC-PPM coded modulation over the free space optical channel. In [83], the authors developed an EXIT analysis technique, where they assume that the messages propagating in the decoder have Gaussian distribution. However, the Gaussian assumption on these messages is not accurate. To show that, we gathered some message statistics. Fig. 5.3 plots the empirical pdfs of the PPM demodulator LLR output (decoder input), for both a transmitted 0 and a transmitted 1, given feedback from the LDPC decoder that is symmetrical-Gaussian with an SNR of -10 dB (SNR = $\mu^2/\sigma^2$). Fig. 5.4 presents such pdfs for a transmitted 0 for various SNR values. Fig. 5.5 presents such pdfs for a transmitted 0 for SNR = 5 dB and 10 dB. Note in the high-SNR situation, the pdfs correspond to discrete random variables. Lastly, Fig. 5.6 through Fig. 5.9 demonstrate the evolution of the pdfs of the messages send from the DM to the VN and vice versa as the iterations progressed. In these figures we consider a (6, 3)-regular LDPC code.

All the figures referenced in this section indicates that the Gaussian assumption is not an accurate assumption. Instead, we adopt the density evolution technique described below (Density evolution for discrete messages is described in [10] and for continuous messages it is described in [11]. See also [85]). This technique is used to find an accurate decoding threshold (defined later) for an irregular LDPC code ensemble, which is characterized by the VNs’s degree distribution $\lambda(z) = \sum_{i=1}^{L_{max}} \lambda_i z^{i-1}$, and the CNs’s degree distribution $\rho(z) = \sum_{i=1}^{R_{max}} \rho_i z^{i-1}$ s (both from
Figure 5.3: PPM demodulator output LLR pdfs for 0/1 transmissions at LDPC decoder feedback SNR= -10 dB. (Poisson channel parameters: $n_b=0.2$, $T_s=32$ns, $M=64$, $n_s/(MT_s)=-30$ dB photon/ns. Feedback input (from LDPC): Symmetric Gaussian, SNR=-10 dB.)

Figure 5.4: PPM demodulator output LLR pdfs for 0 transmission at LDPC decoder feedback SNR = -10, 0, and 5 dB. (Poisson channel parameters: $n_b = 0.2$, $T_s = 32$ns, $M=64$, $n_s/(MT_s) = -30$ dB photon/ns. Feedback input (from LDPC): Symmetric Gaussian, various SNRs. Txd symbol is zero)
Figure 5.5: PPM demodulator output LLR pdfs for 0 transmission at LDPC decoder feedback SNR = 5 and 10 dB. (Poisson channel parameters: $n_b=0.2$, $T_s=32$ns, $M=64$, $n_s/(MT_s)=-30$ dB photon/ns. Feedback input (from LDPC): Symmetric Gaussian. Txd symbol is zero. Note, as the feedback SNR from the LDPC increases, the PPM output tend to be independent from the feedback input.)

Figure 5.6: The pdf evolution of the messages send from the DM to the VN. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-29$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$. 
Figure 5.7: The pdf evolution of the messages send from the DM to the VN. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-30$ dB photon/ns, $n_b=0.2, T_s=32$ns, $M=64$.

edge perspective, that is, $\lambda_i$ is the fraction of edges in the code’s graph incident on degree-$i$ VNs, and $\rho_i$ is the fraction of edges incident on degree-$i$ CNs). In these distributions $L_{\text{max}}$ and $R_{\text{max}}$ are the maximal VN and CN degrees, respectively. Note, that in the VNs degree distribution only edges connecting them to the CNs are considered.

The evolution of the pdfs across the LDPC-PPM iterative decoder is depicted in Fig. 5.10. In this figure, each block computes an output pdf based on the input pdfs and the operation defined by the corresponding node functionality. Let us start by the DM block. This block takes as inputs the pdfs of the channel messages; call them $p_{\text{Ch}}(\xi)$, and the average pdf of the messages sent from the VN to the DM, $p_{4}^{(t-1)}(\xi)$. Note that the DM block has $M$ channel inputs corresponding to $c_{k,0}, c_{k,1}, \ldots, c_{k,M-1}$. Without loss of generality, assume that all of the VNs have value 0. Consequently, the distribution of $c_{k,0}$ is Poisson with mean $n_s+n_b$ and the distribution of $c_{k,i}, \ i \neq 0$ is Poisson with mean $n_b$. The pdf at the output of this
Figure 5.8: The pdf evolution of the messages send from the VN to the DM. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-29$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$.

Moving to the VN block, $p_1^{(t)}(\xi)$, is evaluated via computer simulation using the model in the block diagram in Fig. 5.11. Notationally, the distribution at the output of the DM is given by

$$p_1^{(t)}(\xi) = \mathcal{E}_{DM}(p_{Ch}(\xi), p_4^{(t-1)}(\xi)), \tag{5.21}$$

where $\mathcal{E}_{DM}$ refers to the DM evolution (average message density obtained by evolving the pdfs at the DM inputs).

Moving to the VN block, at the first half of an iteration, it takes the pdf $p_1^{(t)}(\xi)$, and the average pdf of the messages sent from the CN to the VN, $p_3^{(t-1)}(\xi)$, and computes the average pdf of the messages sent from the VN to the CN $p_2^{(t)}(\xi)$, as follows:

$$p_2^{(t)}(\xi) = p_1^{(t)}(\xi) \star \left[ \sum_{i=1}^{L_{\max}} \lambda_i \left( \star_{i-1} p_3^{(t-1)}(\xi) \right) \right]. \tag{5.22}$$

where the $\star$ refers to the convolution operation, and $\star_i$ stands for the convolution of $i$ pdfs, that is $\star_i p(\xi) = p(\xi) \star p(\xi) \star \ldots \star p(\xi)$ ($i$ factors). However, in the second
Figure 5.9: The pdf evolution of the messages send from the VN to the DM. The (6,3)-regular (4096, 2048) Lin LDPC code is used, where $n_s/(MT_s)=-30$ dB photon/ns, $n_b=0.2$, $T_s=32$ns, $M=64$.

half of an iteration, the VN block takes $p_3^{(t)}(\xi)$ and computes $p_4^{(t)}(\xi)$ as follows:

$$p_4^{(t)}(\xi) = \sum_{i=1}^{\lambda_{\max}} \bar{\lambda}_i \left( *_{i-1} p_3^{(t)}(\xi) \right),$$

(5.23)

where $\bar{\lambda}_i$ is the fraction of VNs that have degree $i$ ($\bar{\lambda}_i = \lambda_i/(i \int_0^1 \lambda(z)dz)$).

The last block is the CN block. It takes $p_2^{(t)}(\xi)$ and computes $p_3^{(t)}(\xi)$ as follows:

$$p_3^{(t)}(\xi) = \sum_{i=1}^{R_{\max}} \rho_i \mathcal{E}^{i-1}_{CN}(p_2^{(t)}(\xi)),$$

(5.24)

where $\mathcal{E}^{i-1}_{CN}(\cdot)$ refers to the evolution in a CN of degree $i$. This operation is evaluated as in [10].

Finally, we need to evaluate the average pdf of the VN LLR used in the decision, $p_d^{(t)}(\xi)$, from which one can calculate the probability of error-decisions, $P_e^{(t)}$. This pdf is given by

$$p_d^{(t)}(\xi) = p_1^{(t)}(\xi) \star \left[ \sum_{i=1}^{\lambda_{\max}} \lambda_i \left( *_i p_3^{(t)}(\xi) \right) \right].$$

(5.25)
Then, it is straightforward that

\[ P_e(t) = \int_{-\infty}^{0} p_d(t)(\xi) d\xi \quad (5.26) \]

Note, in the last equation it is assumed that the correct value for all VNs is 0.

Before describing how to find the threshold ratio \( n_s/n_b \), a summary of the density evolution procedure is presented below:

**Initialization**

- Set the pdfs \( p_{Ch}(\xi) \) (At the 0\(^{th}\) input given by (5.1), else given by (5.2)).
- Set \( p_4^{(0)}(\xi) = \delta(\xi) \) (where \( \delta(.) \) is the Dirac delta function.).
- Set \( p_3^{(0)}(\xi) = \delta(\xi) \).

**Start iterating**

- Find \( p_1^{(t)}(\xi) \) using (5.21).
- Find \( p_2^{(t)}(\xi) \) using (5.22).
• Find $p_3(t)(\xi)$ using (5.24).
• Find $p_4(t)(\xi)$ using (5.23).
• Find $p_d(t)(\xi)$ using (5.25).
• Compute $P_e(t)$ using (5.26).
• Repeat.

Now, with the algorithm described above, the error-free threshold ratio $n_s/n_b$ (called decoding threshold) is the minimal $n_s/n_b$ for which $P_e(t) \to 0$ after large number of iterations.

Note, to implement the above algorithm on a machine, it is required to quantize the messages [10], leading to discrete probability mass functions (pmf) which approximate the continues pdfs described above. As a result, the accuracy of this method depends on the number of quantization levels. In our implementation 64 quantization levels per unit are used, which showed a very good accuracy. Moreover, in the convolutions of the discrete pmf’s can be efficiently evaluated using the fast Fourier transform (FFT).

The threshold values for four LDPC code ensembles, as obtained using the above algorithm, are found. In all of them the threshold is given in the form $n_s/MT$ in dB, where the other Poisson channel parameters are $n_b = 0.2$, $M = 64$, and $T = 32ns$. The first example is the (6,3) regular LDPC code ensemble, for which the threshold $n_s/MT = -29.56$ dB photon/ns. In the second, we consider the rate-1/2 irregular LDPC code ensemble, which has the degree distributions $\lambda(z) = (42/264)z + (198/264)z^2 + (24/264)z^3$, and $\rho(z) = z^5$. The threshold of this ensemble is $n_s/MT = -29.78$ dB photon/ns. Note, the degree distributions in this example are the same as that for the JPL code in [83], where they got a threshold $n_s/MT = -29.83$ dB photon/ns assuming the Gaussian approximation. In the third example we evaluate the threshold for the ensemble, from which U of Arizona code (used later) is designed, it has $\lambda(z) = 0.285664z + 0.714336z^4$ and $\rho(z) = z^6$. Its threshold is $n_s/MT = -29.39$dB photon/ns. In the last example, the ensemble
with degree distributions $\lambda(z) = 0.33898z + 0.22373z^2 + 0.34124z^3 + 0.096045z^9$ and $\rho(z) = 0.59322z^4 + 0.40678z^7$ has threshold $n_s/MT = -29.84$ dB photon/ns.

5.5 Numerical results

In this section we present performance results for the different decoding schemes described in this chapter. We also compare performance of different LDPC codes, some were optimized for the AWGN channel, and others were optimized for the free-space optical channel.

In Fig. 5.12, we compare the performance of the iterative demodulator/decoder to that of the non-iterative one. The code used in this comparison is the (4096, 2048) LDPC code designed by Yifei and Ryan [86] to have an excellent performance on the AWGN channel (We call this code U of Arizona code). Note that the iterative demodulator/decoder achieves about 0.9 dB gain over the non-iterative one. Also in this figure, we compare the performance of the U of Arizona code to the performance of the (4096, 2048) serial turbo code (SCPPM) of JPL [82] (The code in [82] is a (15120, 7560) code, but here we simulated a (4096, 2048) code with the same structure and as in [82]), and to the performance of the (4085, 2047) Reed-Solomon PPM code (RSPPM). U of Arizona code have a coding gain of about 2 dB over the Reed-Solomon PPM code, but still about 0.4 dB from the serial turbo code.

In Fig. 5.13, we consider three (4096, 204) LDPC codes; the first is the U of Arizona code, which has only degree-2 and degree-5 VNs; the second has VNs with degrees 2, 3, 4, and 5; and the third has only degree-2 and degree-5 VNs. We compared the performance of the three codes when they decoded using both the LDPC-PPM iterative demodulator/decoder and the LDPC-APPM decoder. We see that the threshold of the code improves when a weak LDPC code is used. However, the code will suffer from a high error-floor. The use of an accumulator in the LDPC-APPM scheme, seems to improve the error-floor of the code, but it on the coast of the threshold.

In Fig. 5.14, we compare the performance of four (4096, 2048) LDPC codes; the
first is a (3, 6) regular LDPC code designed by Shu Lin et. al. [87]; the second is an irregular LDPC code also designed by Shu Lin et. al. [87]; the third is the U of Arizona code; and the fourth LDPC codes is a quasi-cyclic LDPC code (called DE LDPC) designed based on the degree distribution $\lambda(z) = 0.33898z + 0.22373z^2 + 0.34124z^3 + 0.096045z^9$ and $\rho(z) = 0.59322z^4 + 0.40678z^7$, which was obtained in the previous section using the density evolution tool. The first three codes were optimized for good performance over the AWGN channel, while the last one is optimized for a good performance over the free-space optical channel. Note that the DE LDPC code still behind the serial turbo code, but it closes the gap in performance from 0.4 dB to 0.3 dB.

Finally, in Fig. 5.15, we show the degradation in performance which can results from a mismatch between the real $n_s/n_b$ and the estimated one. We used the U of Arizona code and the iterative demodulator/decoder in our simulation. Note that an over-estimation in $n_s/n_b$ by 3 dB results in a degradation of 0.2 dB. On the other hand, an under-estimation in $n_s/n_b$ by 3 dB results in a degradation of 0.1 dB. Consequently, under-estimation in the value of $n_s/n_b$ is better that over-estimation.

5.6 Conclusion

In this chapter, we studied the performance of LDPC codes over the free-space optical channel. We compare the performance of three decoding schemes, and as expected the iterative demodulator/decoder outperform the non-iterative demodulator/decoder. The LDPC-APPM decoder increased the decoding threshold for all the studied codes, and so, it is not recommended. We proposed a density evolution tool and used it to design LDPC codes over the free-space optical channel. simulation showed that codes with better threshold (as predicted by the density evolution tool) have better performances in the waterfall region. Future work may include designing LDPC codes for the free-space optical channel with good threshold and good error-floor. Also, try to design LDPC code which performs better than the serial turbo code.
Figure 5.12: Bit error rate of the (4096, 2048) U of Arizona LDPC code with LDPC-PPM iterative and non-iterative demodulator/decoder (labeled LDPC↔PPM and LDPC→PPM, respectively). Performance also compared to the (4096, 2048) serial turbo code of JPL (labeled SCPPM), and to the (4085,2047) Reed-Solomon code (labeled RSPPM).
Figure 5.13: Performance comparison between the LDPC-PPM iterative demodulator/decoder and the LDPC-APPM decoder for several LDPC codes with different VNs degree distribution.
Figure 5.14: Performance comparison between two (4096, 2048) LDPC code designed by Shu Lin, the (4096, 2048) U of Arizona code, and a (4096, 2048) LDPC code with degree distributions $\lambda(z) = 0.33898z + 0.22373z^2 + 0.34124z^3 + 0.096045z^9$ and $\rho(z) = 0.59322z^4 + 0.40678z^7$ (labeled DE LDPC).
Figure 5.15: Performance of the (4096, 2048) U of Arizona code with LDPC-PPM iterative demodulator/decoder for different mismatch between the real $n_s/n_b$ and the estimated one.
CHAPTER 6

Conclusion

In this dissertation we have provided several analytical and numerical tools to help the designer recognize good ensembles of protograph-based G-LDPC codes. We have proposed a method for computing codeword-weight enumerators for finite-length and infinite-length protograph-based G-LDPC codes. The asymptotic results help identifying protograph-based G-LDPC code ensembles having minimum distance that grows linearly with code length. We have presented asymptotic codeword-weight enumerators results for several protograph-based G-LDPC code ensembles, which showed that these protographs have good minimum distance properties. The method is conceptually simple. However, the computational complexity becomes an impediment when some of the protograph’s constraint nodes have large dimensionality $k$. For this reason, we have proposed a conjecture which greatly reduces the computational complexity when analyzing such protographs.

We have also proposed methods for finding stopping set enumerators, and pseudocodeword enumerators for protograph-based G-LDPC code ensembles for both finite and infinite code lengths. The methods leverage the weight enumeration technique by transforming the stopping set, pseudocodeword, enumerator problem into a codeword weight enumerator problem. The asymptotic stopping set enumerator allows us to determine if a G-LDPC ensembles has a typical stopping set number, that is, has a minimum stopping set size that grows linearly with code length. Similarly, the asymptotic pseudocodeword-weight enumerator helps determine which G-LDPC code ensembles with minimum pseudocodeword-weight grow linearly with code length. We have provided example protographs for protograph-based codes with good stopping set and pseudocodeword properties.

Additionally, we have proposed a method for computing finite and asymptotic trapping set enumerators for protograph-based LDPC code ensembles. To compare
different ensembles, we classified the \((a, b)\)-trapping sets based on the ratio \(b/a = \Delta\), and defined the parameter \(\Delta\)-trapping set number as the size of the smallest, non-empty, trapping set with ratio \(b/a\) exactly \(\Delta\). The asymptotic results provide information about the existence of a typical \(\Delta\)-trapping set number. We believe the trapping set technique can be extended to protograph-based G-LDPC code ensembles, although this is an open problem. The difficulty is that trapping sets for G-LDPC codes never have been defined or studied in the literature.

Furthermore, we have developed an algorithm for estimating trapping set enumerators for a specific LDPC code given its parity-check matrix. We have used the algorithm to estimate the trapping set enumerators for the \((2640, 1320)\) Margulis code and several LDPC codes from the communications standards. The results indicated that the proposed algorithm is efficient in terms of its speed, computational complexity, and accuracy.

In addition, we have proposed a few G-LDPC protographs which include recursive systematic convolutional (RSC) constraint nodes, and designed G-LDPC codes based on these protographs. We have studied the performance of these codes via simulation. The results indicate that these codes have good performance in the waterfall region and have very low floors. Among these examples, the rate-1/6 and \(k = 100\) RSC-LDPC code, which showed no error-floor down to \(5 \times 10^{-9}\). This is very impressive considering its low-rate and short block-length. We have also studied reduced-complexity decoders for use in the decoding of the RSC constraint nodes.

Lastly, we have studied coded-modulation schemes with LDPC codes and pulse position modulation (LDPC-PPM) over the free-space optical channel. We have presented three different decoding schemes and compare their performances. In addition, we have developed a new density evolution tool for use in the design of LDPC codes with good performances over this channel.

Several interesting research problems still need to be studied. The following research problems are recommended for future consideration:

- Design of protograph-based codes with relatively high rates. Protograph-based
G-LDPC codes proposed in this dissertation, or in the literature, have low-to-moderate rates. Designing high-rate protograph-based G-LDPC codes is limited by the rate of the constituent constraint nodes, and these constraint nodes usually have somewhat low rates (e.g., rate-4/7 and rate-1/2). A possible solution may be found by trying to associate a protograph with a specific set of permutations so that the resultant parity-check matrix is not full rank.

- Derive approximations to the finite-length enumerators by finding tight bounds on these enumerators which depends on the block-length of the ensemble. In this dissertation we have presented methods for computing finite-length enumerators for protograph-based G-LDPC codes. In Appendix A, we have proposed bounds to reduce the computational complexity for computing these enumerators in the finite-length case. However, we still believe that it is possible to greatly reduce the amount of numerical computations by deriving approximations to these enumerators.

- Proof the conjecture which we proposed to reduces the computational complexity for computing the enumerators for protograph-based G-LDPC code ensembles.

- Define, study, and enumerate trapping sets for G-LDPC codes and protograph-based G-LDPC codes.

- Enhance the proposed algorithm for enumerating trapping sets for specific LDPC code given its parity-check matrix to distinguish between trapping sets based on their failure rate. In this work we have enumerated trapping sets as a graphical structure without any connection to their failure rate, but it will be nice to have an algorithm that enumerate trapping sets with failure rate above certain threshold. The problem is that the failure rate for trapping sets depends on the channel, the decoding algorithm, and the quantization of the iterative decoder messages.
• Extend the enumeration methods in this dissertation to include the multi-edge type G-LDPC codes [88].

• Study the error-floor for LDPC codes over the free-space optical channel.

• Design LDPC codes which can outperform the serial turbo codes on the free-space optical channel.
In this appendix, we describe a reliable method for solving the following Diophantine system which was shown to be crucial in Section 3.2. We are given a $K \times \mu$ binary matrix $M$ and a $1 \times \mu$ vector of nonnegative integers $w$. Then we are to find the solution to

$$M^T \cdot n^T = w^T,$$  \quad (A.1)

where $\sum_{k=1}^{K} n_k = N$ and $n_1, n_2, \ldots, n_K$ are positive integers. (Note, we use the transposed form of the system in Section 3.2 to simplify the linear algebra.)

Note that by adjoining an all-ones column to the matrix $M$, call this new matrix $\hat{M}$, the above system can be rewritten as

$$\hat{M}^T \cdot n^T = [N \mid w]^T. \quad (A.2)$$

As $n_k \leq N$, there are $N^K$ possibilities to search, which is generally impractical. However, the rank of $\hat{M}$ is $\mu + 1 = r$. So, find a transformation matrix $T$ such that $T \cdot \hat{M}^T = [I_r \mid B]$, where $I_r$ is the $r \times r$ identity matrix. Then, from the vector $T \cdot [N \mid w]^T = [p_1, p_2, \ldots, p_r]^T$ find the particular solution of the system, $n^{(p)} = [n_1^{(p)}, n_2^{(p)}, \ldots, n_K^{(p)}]$, as

$$n_k^{(p)} = \begin{cases} p_k, & k \leq r \\ 0, & \text{elsewhere} \end{cases}. \quad (A.3)$$

Thus, the general solution is

$$n^T = n^{(p)^T} + \begin{bmatrix} -B \\ I_{K-r} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{K-r} \end{bmatrix}. \quad (A.4)$$
The problem is now to find all vectors \([x_1, x_2, \ldots, x_{K-r}]\) which give a vector of positive integers \(n\). But from (A.4),

\[
[x_1, x_2, \ldots, x_{K-r}] = [n_{r+1}, n_{r+2}, \ldots, n_K].
\] (A.5)

Consequently, \(x_i\) is a positive integer, and so the search domain cardinality is reduced to \(N^{K-r}\).

The search domain can be further reduced by finding upper bounds on the \(x_i\)’s as follows. Any element in \(M\) has the value of either ‘0’ or ‘1’. Focusing on the \(j^{th}\) column of \(M\), from (A.1), the sum of the \(n_k\)’s which correspond to ‘1’ in this column is \(w_j\). This means that any of these \(n_k\)’s have to be less than or equal to \(w_j\). Now, as there are several columns in \(M\), more than one bound on each \(n_k\) may exists. From them, one selects the smallest. This is summarized as

\[
0 \leq n_k \leq \min_{\{j: m_{kj}=0\}} \{w_j\},
\] (A.6)

where \(m_{kj}\) is the element in the \(k^{th}\) row and the \(j^{th}\) column of \(M\). This bound greatly reduces the search domain, particularly when most of the components, \(w_j\)’s, satisfy \(w_j \leq N/2\). Consider when most of these components satisfy \(w_j \geq N/2\). Again, focusing on the \(j^{th}\) column of \(M\), from (A.1), the sum of the \(n_k\)’s which correspond to ‘1’ in this column is \(w_j\), and so the sum of the \(n_k\)’s which correspond to ‘0’ in this column is \(N - w_j\) This means that any of these \(n_k\)’s have to be less than or equal to \(N - w_j\). Consequently,

\[
0 \leq n_k \leq \min_{\{j: m_{kj}=0\}} \{N - w_j\}.
\] (A.7)

Combining the two bounds one gets

\[
0 \leq n_k \leq \min_{\{j: m_{kj}=1\}} \{w_j\}, \min_{\{j: m_{kj}=0\}} \{N - w_j\} \leq N.
\] (A.8)

From (A.5) the bounds on \(n_k\)’s above directly translate into bounds on \(x_i\)’s.
In this appendix we prove that if the protograph-based code ensemble has a typical minimum distance, then with probability close to one the minimum distance of a code in the ensemble increases linearly with the code length. This was first shown by Gallager [2] for regular LDPC code ensembles. The layout of the proof is as follows: First find a bound on the finite length weight enumerator, $A_d$. Then, use this bound to show that the $\sum_{d=1}^{\lfloor n\delta_{\min} \rfloor - 1} A_d \rightarrow 0$. Finally, show that this implies that $\Pr\{d_{\min} \leq \lfloor n\delta_{\min} \rfloor - 1\} \rightarrow 0$.

To obtain a bound on $A_d$ from (3.4), let us define the shorthand notation $B(c_j) = A^{c_j}(d_j)$ and $D = \prod_{i=1}^{n_v} \binom{N}{d_i}^{q_i-1}$. Also, the following inequalities from [2] will be used in this derivation:

$$\frac{1}{\sqrt{2\pi n \lambda(1-\lambda)}} \exp \left( nH(\lambda) - \frac{1}{12n \lambda(1-\lambda)} \right) < \left( \frac{n}{\lambda n} \right) < \frac{1}{\sqrt{2\pi n \lambda(1-\lambda)}} \exp \left( nH(\lambda) \right)$$

and the Stirling approximation

$$\sqrt{2\pi n} \cdot n^n \exp(-n) \leq n! \leq \sqrt{2\pi n} \cdot n^n \exp \left( -n + \frac{1}{12n} \right).$$

Now, from (B.1)

$$D > \prod_{i=1}^{n_v} \left[ \frac{1}{\sqrt{2\pi N \lambda_i(1-\lambda_i)}} \exp \left( N \cdot H(\lambda_i) - \frac{1}{12N \lambda_i(1-\lambda_i)} \right) \right]^{q_i-1},$$

where $\lambda_i = d_i/N$. Also,

$$B^{(c_j)} = \sum_{\binom{n}{n_j}} \frac{N!}{n_1! n_2! \ldots n_{K_j}!},$$

where $K_j$ is the number of codewords in the code corresponding to CN $c_j$. From (B.2),

$$\frac{N!}{n_1! n_2! \ldots n_{K_j}!} < \frac{\sqrt{2\pi N} \cdot N^n \exp \left( -N + \frac{1}{12N} \right)}{\prod_{i=1}^{K_j} \sqrt{2\pi n_i} \cdot n_i^{n_i} \exp(-n_i)}.$$
Moreover, because \(N^N = \prod_{i=1}^{K_j} n_i\), one can write \(N^N / \prod_{i=1}^{K_j} n_i\) as \(1/ \prod_{i=1}^{K_j} (n_i/N)^{n_i}\). But

\[
\prod_{i=1}^{K_j} (n_i/N)^{n_i} = \exp \left( \log \left( \prod_{i=1}^{K_j} \frac{n_i}{N} \right) \right),
\]

\[
= \exp \left( \sum_{i=1}^{K_j} n_i \log \left( \frac{n_i}{N} \right) \right),
\]

\[
= \exp \left( \sum_{i=1}^{K_j} \frac{n_i}{N} \log \left( \frac{n_i}{N} \right) \right),
\]

\[
= \exp (-N \cdot H(\rho)), \quad (B.6)
\]

where \(\rho = [\rho_1, \rho_2, \ldots, \rho_{K_j}], \rho_i = n_i/N\). Consequently, we may write (B.5) as

\[
\frac{N!}{n_1! n_2! \ldots n_{K_j}!} < \frac{1}{(2\pi N)^{\frac{K_j-1}{2}} \prod_{l=1}^{K_j} \sqrt{\rho_l}} \exp \left( N \cdot H(\rho) + \frac{1}{12N} \right). \quad (B.7)
\]

From (B.4) and (B.7) one obtain the following bound on \(B^{(c_j)}\):

\[
B^{(c_j)} < \frac{1}{(2\pi N)^{\frac{K_j-1}{2}}} \sum_{\{n\}} \exp \left( N \cdot H(\rho) + \frac{1}{12N} \right) \prod_{l=1}^{K_j} \frac{\sqrt{\rho_l}}{\sqrt{\rho_l}}, \quad (B.8)
\]

Furthermore, in (B.8)

\[
\sum_{\{n\}} \frac{\exp \left( N \cdot H(\rho) + \frac{1}{12N} \right)}{\prod_{l=1}^{K_j} \sqrt{\rho_l}} \leq |\{n\}| \cdot \max_{\{\rho\}} \left\{ \frac{\exp \left( N \cdot H(\rho) + \frac{1}{12N} \right)}{\prod_{l=1}^{K_j} \sqrt{\rho_l}} \right\},
\]

\[
= |\{n\}| \cdot \exp \left( N \cdot H(\rho^*) + \frac{1}{12N} \right),
\]

\[
\leq \left( \frac{N \sum_{i=1}^{q_{c_j}} \lambda_i^{(c_j)}}{a_{\min}^{(c_j)}} + 1 \right)^{K_j} \cdot \exp \left( N \cdot a^{(c_j)}(\lambda_j) + \frac{1}{12N} \right), \quad (B.9)
\]

where \(\rho^* = \arg \max_{\{\rho\}} \{H(\rho)\}\). The second equality holds for sufficiently large \(N\), where the exponential dominates the value of the function in the first inequality. To explain the second inequality, first note from (3.22) \(H(\rho^*) = a^{(c_j)}(\lambda_j)\). Second,
let \( \mathbf{1} \) be a vector of \( q_{c_j} \) ones, then 
\[ e_j = \mathbf{n} \cdot \mathbf{M}^{(c_j)} \cdot \mathbf{1} \geq \frac{d^{(c_j)}_{\min}}{d^{(c_j)}_{\min}} \sum_{i=2}^{K_j} n_i \geq d^{(c_j)}_{\min} \cdot n_i, \]
for any \( i = 2, 3, \ldots, K_j \). This means that any \( n_i \) can take at most \( e_j/d^{(c_j)}_{\min} + 1 \) values. Consequently, 
\[ |\{\mathbf{n}\}| \leq (e_j/d^{(c_j)}_{\min} + 1)^{K_j}. \]
Also as \( \mathbf{n} \cdot \mathbf{M}^{(c_j)} = \mathbf{d}_j \), then 
\[ e_j = \mathbf{d}_j \cdot \mathbf{1} = \sum_{i=1}^{q_{c_j}} d_i^{(c_j)} = N \sum_{i=1}^{q_{c_j}} \lambda_i^{(c_j)}. \]

Combining the bounds in (B.3), (B.8), and (B.9) one gets

\[
\prod_{j=1}^{n_c} B^{(c_j)} \leq \prod_{j=1}^{n_c} \left[ \frac{1}{2\pi N} \cdot \left( \frac{N \sum_{j=1}^{q_{c_j}} \lambda_i^{(c_j)}}{d^{(c_j)}_{\min}} + 1 \right) \right]^{K_j} \cdot \frac{\exp\left(\frac{N \cdot a^{(c_j)}(\mathbf{\lambda}_j) + \frac{1}{12N}}{\prod_{j=1}^{n_c} \sqrt{\rho_j}}\right)}{\prod_{j=1}^{n_c} \sqrt{2\pi N \lambda_i(1 - \lambda_i)}} \cdot \frac{\exp\left(\frac{N \cdot H(\mathbf{\lambda}_j) - \frac{1}{12N \lambda_i(1 - \lambda_i)}}{\sum_{j=1}^{n_c} q_{c_j}}\right)}{\prod_{j=1}^{n_c} \sqrt{\rho_j}} \cdot \exp\left\{ \frac{1}{12N} \left( n_c + \sum_{i=1}^{n_c} \frac{q_{v_i} - 1}{\lambda_i(1 - \lambda_i)} \right) \right\},
\]

where

\[
f(N, \mathbf{\lambda}) = \sqrt{\frac{2\pi}{\sum_{j=1}^{n_c} (q_{v_j} - 1) + \sum_{j=1}^{n_c} (K_j - 1)}} \cdot \prod_{j=1}^{n_c} \left( \frac{\sum_{i=1}^{q_{c_j}} \lambda_i^{(c_j)}}{d^{(c_j)}_{\min}} + 1 \right)^{K_j} \cdot \frac{\exp\left(\frac{N \cdot a^{(c_j)}(\mathbf{\lambda}_j) - \sum_{i=1}^{n_c} (q_{v_i} - 1) H(\lambda_i)}{\sum_{j=1}^{q_{c_j}} \lambda_i^{(c_j)}}\right)}{\prod_{j=1}^{n_c} \sqrt{\rho_j}} \cdot \exp\left\{ \frac{1}{12N} \left( n_c + \sum_{i=1}^{n_c} \frac{q_{v_i} - 1}{\lambda_i(1 - \lambda_i)} \right) \right\},
\]

is a polynomial in \( N \) for sufficiently large \( N \).

Next, using (B.10)

\[
A_d = \sum_{\{\mathbf{d}\}} \frac{\prod_{j=1}^{n_c} B^{(c_j)}}{D},
\]

\[
\leq |\{\mathbf{d}\}| \cdot \max_{\{\mathbf{d}\}} \left\{ \frac{\prod_{j=1}^{n_c} B^{(c_j)}}{D} \right\},
\]

\[
\leq |\{\mathbf{d}\}| \cdot f(N, \mathbf{\lambda}^*) \cdot \exp\left( N \cdot \left( \sum_{j=1}^{n_c} a^{(c_j)}(\mathbf{\lambda}_j^*) - \sum_{i=1}^{n_v} (q_{v_i} - 1) H(\lambda_i) \right) \right),
\]

\[
\leq \left( d + n_v - 1 \right) \cdot \left( n_v - 1 \right) \cdot f(N, \mathbf{\lambda}^*) \cdot \exp\left( N \cdot \tilde{r}(\tilde{\delta}) \right),
\]

where

\[
\mathbf{\lambda}^* = \arg \max_{\mathbf{\lambda}} \left\{ \sum_{j=1}^{n_c} a^{(c_j)}(\mathbf{\lambda}_j) - \sum_{i=1}^{n_v} (q_{v_i} - 1) H(\lambda_i) \right\},
\]

(B.13)
and from (3.19)\[
\tilde{r}(\tilde{\delta}) = \sum_{j=1}^{n_c} a^{(c_j)} (\lambda_j^*) - \sum_{i=1}^{n_v} (q_{c_i} - 1) H(\lambda_i^*). \tag{B.14}
\]
Also in (B.12), \(|\{d\}| = \binom{d+n_v-1}{n_v-1}\) because the right-hand side of the equation is the number of partitions of \(d\) into \(n_v\) values. From (B.12) note that \(A_d \to 0\) if \(\tilde{r}(\tilde{\delta}) < 0\). This implies that if \(\tilde{r}(\tilde{\delta})\) has a typical minimum distance \(\delta_{min}\), then \(\sum_{d=1}^{\lfloor n\delta_{min}\rfloor -1} A_d \to 0\).

Finally, recall Markov’s inequality, which states that if \(X\) is a random variable on nonnegative values, then
\[
Pr\{X \geq a\} \leq \frac{E[X]}{a}, \tag{B.15}
\]
for \(a \geq 0\). Now, define the random variable \(X_d\) as the number of codewords of weight \(d\) in a code drawn from the code ensemble. Note \(E[X_d] = A_d\) and denote the probability that a weight-\(d\) codeword exists by \(p(d) = Pr\{X_d \geq 1\}\). By Markov’s inequality, \(p(d) \leq A_d\). As a result,
\[
Pr\{d_{min} < \lfloor n\delta_{min}\rfloor\} \leq \sum_{d=1}^{\lfloor n\delta_{min}\rfloor -1} p(d) \leq \sum_{d=1}^{\lfloor n\delta_{min}\rfloor -1} A_d. \tag{B.16}
\]
As \(\sum_{d=1}^{\lfloor n\delta_{min}\rfloor -1} A_d \to 0\), then \(Pr\{d_{min} < \lfloor n\delta_{min}\rfloor\} \to 0\).
REFERENCES


