Asymptotic Weight Analysis of Low-Density Parity Check (LDPC) Code Ensembles

Thesis by

Sarah L. Sweatlock

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2008

(Defended April 29, 2008)

To Mr. Holland

Acknowledgements

This thesis never would have been possible without the support and guidance of my friends, family, coworkers and advisors. I have been incredibly lucky to have worked for and with some wonderfully friendly and brilliant people.

First and foremost, I would like to thank my thesis adviser, Robert J. McEliece, for taking me in and giving me an academic home. I am proud to have been one of his students, and very grateful that I made it to Caltech before his retirement. He is a wonderful adviser, offering encouragement when needed, and full of a contagious enthusiasm that made this whole process more fun than it should have been.

The staff members here at Caltech have been amazing to me. I am incredibly grateful to Shirley Slattery, Sydney Garstang, and Sheila Shull for taking care of me for the last few years. There was never a question they could not answer, never a problem that they could not solve. They quickly became some of my favorite people on campus, and will be sorely missed when I leave.

I was lucky enough to work with Kaiann Fu and Anna Bertiger on a few occasions. I would like to thank them both for numerous helpful and fun discussions. A hearty thank you also goes to all my current and former group members, Jeremy Thorpe, Mostafa El-Khamy, Edwin Soedarmadji, Farzad Parvaresh, Elette Boyle, Srinivas Aji, Weiyu Xu, and Bingwen Wang.

I would also like to thank the Information Processing Group at JPL. I spent many happy hours there in fruitful discussions with Ken Andrews, Sam Dolinar, Dariush Divsalar, Bruce Moision, Chris Jones, Mike Cheng, and Jon Hamkins.

In addition, I never would have been able to succeed at Caltech were it not for the people at Northwestern who taught me how to love research. I am full of gratitude

toward Profs. Mary Silber, William Kath, Gino Biondini, and Allen Taflove for helping me take that first step. Of course, I would not have made it to Northwestern were it not for the wonderful education I received at the Derryfield School. Thanks, especially, to Mr. Holland, who after all these years is still the best math teacher I have ever had.

Finally, I would like to thank my family and friends for their eternal support. A big thank you especially to my brother Tom Fogal, who always made time to solve my computer woes. Above all, thanks to my husband Luke for keeping me sane these last few years. Words cannot express how grateful I am to him for his unending confidence in me.

Abstract

With the invention of turbo codes in 1993 came increased interest in codes and iterative decoding schemes. Gallager's Regular codes were rediscovered, and irregular codes were introduced. Protograph codes were introduced and analyzed by NASA's Jet Propulsion Laboratory in the early years of this century. Part of this thesis continues that work, investigating the decoding of specific protograph codes and extending existing tools for analyzing codes to protograph codes.

The rest of this work focuses on a previously unknown relationship between the binary entropy function and the asymptotic ensemble average weight enumerator, which we call the *spectral shape* of the ensemble. This result can be seen as an extension of the Pless power-moment identities based on the discovery that the convex hull of the spectral shape is the Legendre transform of a function closely related to the moment-generating function of a codeword's weight.

In order to fully investigate this new relationship, tools needed to be designed to calculate the derivatives of the spectral shape as the equation describing an ensemble's spectral shape is rarely straightforward. For Gallager's regular ensembles, a formula for calculating derivatives of functions defined parametrically was required. For repeat-accumulate (RA) codes, a formula was needed for functions defined implicitly through a second function. Both formulas are similar to Faá di Bruno's formula for derivatives of compositions of functions.

Contents

List of Figures

List of Tables

Chapter 1

Background

1.1 Historical Framework[1]

The fundamental problem of communication is that of reproducing at one point a message selected at another point. — Claude E. Shannon

In his seminal 1948 paper [51], Claude Shannon derived the mathematical laws that govern how rapidly information can be reliably transmitted through a noisy channel. This mathematical framework became the basis for an entirely new field devoted to its study — *information theory*— and its sister discipline *error-correcting codes*. Shannon's noisy channel coding theorem asserts that for every channel there exists a maximum rate at which we can communicate with vanishing error probabilities. This maximum rate is known as the *capacity* of the channel. Shannon further proved that this capacity can be achieved by almost any extremely long code. This proof, however, was not constructive. An arbitrary long, random code may technically perform well, but the encoding and decoding times would be prohibitively large.

In the decades following Shannon's work, the ultimate goal of coding theory has been to construct capacity-achieving codes with manageable encoding and decoding times. One major success in this endeavor was the introduction of turbo codes in 1993

[1]Throughout this thesis, sections that contain significant contributions by the author will be denoted by a * in their section heading.

[10]. With turbo codes, came the introduction of *iterative decoding*, which bridged the gap between high performance and low complexity. Specifically, iterative decoding can achieve performance close to theoretical limits with a complexity that grows only linearly with the length of the code.

The discovery of turbo codes led to a flurry of research interest in the field, and, in particular, to the rediscovery of Gallager's 1963 work [26] on low-density parity check (LDPC) codes. Though Gallager's work had been largely forgotten due to the limited computational capabilities of his time, some interesting developments had been occurring. Most relevant to this thesis was the work of Tanner [56], which formally introduced the idea of using a bipartite graph to graphically represent a code.

The idea of irregular codes was introduced in 1998 by Luby et al. [34] as a way to improve upon Gallager's regular codes. Five years later, NASA's Jet Propulsion Lab (JPL) introduced the idea of a protograph code [58]. A protograph code is more structured than an irregular code, which allows for simpler code descriptions without sacrificing performance. Protograph codes are closely related to Tanner's codes created from seed graphs [57], and are an example of the multi-edge type construction introduced by Richardson [47].

With new codes came new theorems explaining their success. LDPC codes, with iterative decoding, have been shown to achieve excellent performance over many channels, nearly approaching capacity on the additive white Gaussian noise (AWGN) channel [48], and as code's length tends to infinity, achieving it on the binary erasure channel (BEC) [44].

1.2 Scope of This Thesis

In this chapter, we will provide the necessary background information that the rest of the thesis depends on. The spectral shape of an LDPC code ensemble will be introduced as an asymptotic version of the weight enumerator. While the weight enumerator can be used to accurately estimate the performance of a single finite length code, the spectral shape can be used to estimate the asymptotic performance of a ensemble of codes.

Various ensembles will be introduced, including Gallager's regular ensembles [26] and several generalizations of it by Litsyn and Shevelev [33]. We will also call the set of all codes the non-linear Shannon ensemble, and the set of all linear codes the linear Shannon ensemble. As Shannon [51] and Gallager [27] respectively proved, these ensembles are capacity-achieving. However, random codes are extremely difficult to decode.

Ideally, we would like to find ensembles that can be decoded using fast, iterative methods, but perform similarly to the Shannon ensembles. Since the spectral shape of an ensemble can be used to estimate its performance, we will look for LDPC ensembles whose spectral shapes are similar to that of the Shannon ensembles.

The first part of this thesis deals with how closely related the spectral shape of an ensemble is to that of the Shannon ensemble. The Pless power-moment identities [46] relate the moments of the distribution of the number of words of each weight to the weight enumerator of the dual code. In Chapter 2, we will review these identities and prove a similar result for the spectral shape. In Chapter 3, we will detail what we know about the spectral shapes of the ensembles introduced in our introductory material. Chapter 4 will detail the methods for taking derivatives of some relevant functions using principles from combinatorics and set theory.

The second part of this thesis focuses on more practical concerns. Decoding

algorithms and queuing problems will be discussed in Chapter 5. Finally, in Chapter 6, we will discuss methods of estimating the performance of a code without resorting to extensive simulation.

A review of relevant notation can be found in Table A-2 on page 113. Table A-3 on the following page details the special notation for the queuing and set theory problems.

1.3 The Basic Communications System

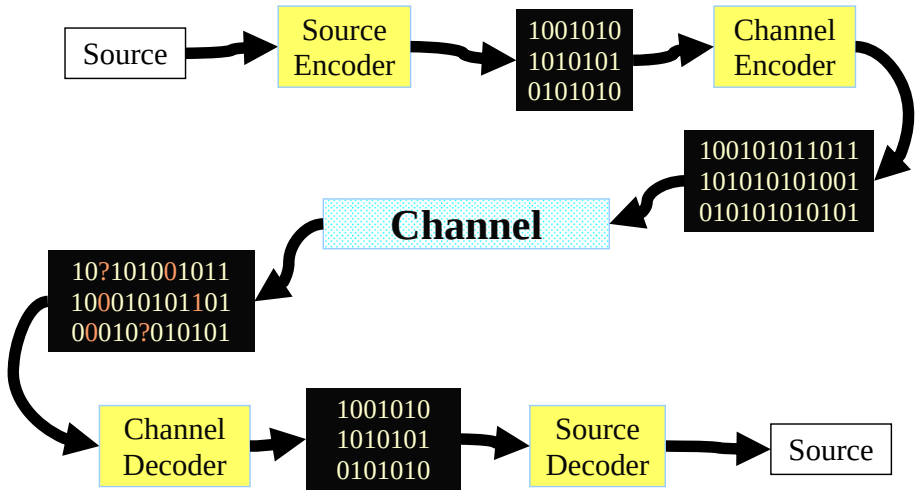


Figure 1.1: Basic Communications System

The basic set-up for a point-to-point communications system is shown in Figure 1.1. It all starts with a message or a source. This message can take many forms, but for our purposes we will assume that the source encoder turns it into an element of $GF(2)^k$, i.e., a binary vector of length k. The channel encoder then adds redundancy according to a specific *code*. This new vector is called a *codeword* and is an element of $GF(2)^n$. We call k the *dimension* of the code, and n its *length*.

The codeword is transmitted through either space or time using a *channel*. The channel usually has the property that it alters signals that pass through it, corrupting

the data they contain. At the end of the channel, a word is received. This received word may or may not be the codeword that was transmitted. The received word is sent to a channel decoder, which attempts to correct the errors made by the channel and returns the data to the source decoder, which identifies the original message.

As a simple example, consider burning a DVD: the raw footage from a camcorder is the source. A computer acts as the source and channel encoders, converting the footage to binary data and adding redundancy. The data is then burned to a DVD, which is the channel. The DVD may get scratched or dusty, altering some of the data, but a DVD player is able to read the disc, repair the altered data, and display the video.

1.4 Common Channel Models Used in This Thesis

There are myriad physical channels and channel models. This thesis deals exclusively with memoryless channels, which have the property that if and how a channel alters one bit is completely independent from if and how it alters any other. In mathematical terms, if the transmitted message is denoted $\mathbf{x}$ and the received message $\mathbf{y}$, $p(\mathbf{y}|\mathbf{x}) = \prod_i p(y_i|x_i)$. This is a large assumption, but it is often made as it greatly simplifies the mathematics without significantly distorting reality. This work will focus on the binary erasure channel (BEC) and the additive white Gaussian noise (AWGN) channel.

The binary erasure channel is defined by an erasure probability, p. Every bit that is transmitted has probability p of being *erased*. An erased bit arrives at the receiver as an unknown. The receiver knows that the bit was transmitted, but does not know its value. Any bit that is not erased arrives unscathed. The BEC makes no errors and alters no data other than erasing those bits. Interest in this channel has increased in the last decade as it can be seen as a simplified version of how data is corrupted in

transmission over the Internet; packets rarely arrive corrupted, but often are dropped, i.e., fail to arrive at all [35, 45].

The additive white Gaussian noise (AWGN) channel is defined by a signal-to-noise ratio E_b/N_0, the ratio of the energy in one bit of data to the energy of the surrounding noise. This value is often given in logarithmic scale in dBs. A bit received from the AWGN channel is the sum of the original bit value and an independent Gaussian random variable with mean zero and variance $N_0/2$. Gaussian noise is a reasonable approximation of the type of noise encountered in deep space communications [4].

1.5 Error-Correcting Codes

A widely circulated e-mail states:

> I cdnuolt blveiee taht I cluod aulaclty uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuan mnid! Aoccdrnig to rscheearch taem at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Scuh a cdonition is arppoiately cllaed Typoglycemia.
>
> Amzanig huh? Yaeh and yuo awlyas thought slpeling was ipmorantt.

While no such study was ever done at Cambridge University, the readability of the passage does suggest that spelling is not as important as one might have thought. The English language is redundant, and even when it is heavily garbled, it is still quite understandable. The goal of an error-correcting code is to duplicate that behavior for binary messages.

An (n, k) binary linear code $\mathcal{C}$ is a subspace of $\mathrm{GF}(2^n)$ with dimension k. The *rate* of a code $R = k/n$ is the ratio of the amount of information a word contains (k bits) to the length of the code (n bits). The focus here is on *linear* codes, which have the property that if $x \in \mathcal{C}$ and $y \in \mathcal{C}$ then $x + y \in \mathcal{C}$.

We start with the k-bit message. This message is multiplied by a $k \times n$ binary matrix, G, known as the *generator matrix*, creating a codeword of length n. The code is thus the rowspace of the generator matrix. A code can also be defined as the nullspace of a $(n-k) \times n$ binary matrix called the *parity check matrix*. The generator and parity check matrices are related. If the generator matrix, G is written in the form $G = [I_k A]$, where I_k is the identity matrix of size k, then a parity check matrix is given by $H = \left[A^T I_{n-k}\right]$.

A third description of a code is by a Tanner graph [56]. A Tanner graph is a bipartite graph consisting of n variable nodes, representing each of the n bits in the codeword, and $n - k$ check nodes, representing each constraint on the codeword. We will use circles for variable nodes and squares for check nodes. The Tanner graph is connected according to the parity check matrix, i.e., if $H_{ij} = 1$, there is an edge joining variable node i and check node j. To determine if a word is a codeword, the subset of variable connected to each check node is examined. If each such subset has even parity (i.e., sums to zero modulo 2), the word is a codeword.

Example 1. The Hamming Code

The (7,4) Hamming code can be defined by the 3×7 parity check matrix

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

or the 4×7 generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

or the Tanner graph shown in Figure 1.2. Note that, on a Tanner graph, variable nodes will be shown as numbered circles, while check nodes are shown as lettered squares. For larger Tanner graphs, the squares will simply contain plus signs.

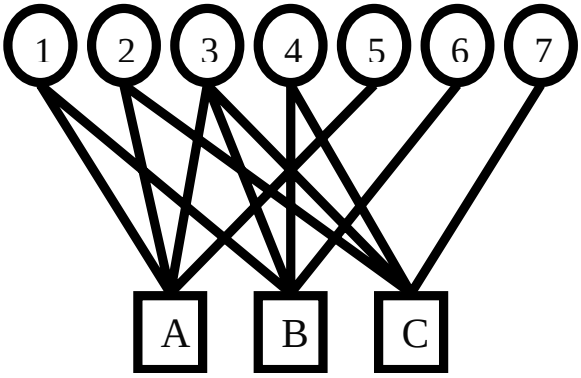


Figure 1.2: Hamming Code Tanner Graph

The Hamming code has dimension 4, and thus has $2^4 = 16$ codewords. In terms of the parity check matrix, x is a codeword if $Hx = 0$. In terms of the generator matrix, x is a codeword if $uG = x$ for some $u \in GF(2^k)$. In terms of the Tanner graph, a word is a codeword if the subsets connected to each check node have even parity. The 16 words are listed in Table 1.1

0000000	1111111	0100101	1011010
0001111	1110000	0101010	1010101
0010011	1101100	0110110	1001001
0011100	1100011	0111001	1000110

Table 1.1: Hamming Codewords

1.6 The Weight Enumerator

For any word, x, the Hamming *weight* of the word, $\mathrm{wt}\, x$, is defined as the number of non-zero components of x. For a binary word, this is simply the sum of the bits. The weight of a word can also be used to define a distance metric; the *distance* between any two words is the weight of their binary sum. For example,$x = 1101100$ has weight four, as does $y = 1101010$; the distance between x and y is $\mathrm{wt}\, x + y = \mathrm{wt}\, 0000110$, which is two. One of the most important descriptions of a linear code is its *weight enumerator*. The weight enumerator is a list of the number of codewords of each weight. In a linear code, the weight enumerator is of particular importance, as if there is a word, x, of weight j in the code, then there is also a word at distance j from every codeword y, i.e., $y + x$. Because of this, the weight enumerator is also the *distance enumerator*.

The distance enumerator is a powerful tool for estimating code performance with specific decoding schemes, which will be discussed further in Chapter 5. In general, however, knowing the distance between codewords allows us to calculate the probability that an error of a specific weight will cause the received word to be closer to the wrong codeword than it is to the transmitted codeword. For example, if the distance between any two codewords is always at least three, any error of weight one will be decodable, as the received word will remain closest to the transmitted word.

Example 2. The Hamming Code

The 16 Hamming codewords, and their weights, are given in Table 1.2. There is one word of weight zero, one word of weight seven, and seven words of weight three and four.

Word	Weight	Word	Weight
0000000	0	1111111	7
0001111	4	1110000	3
0010011	3	1101100	4
0011100	3	1100011	4
0100101	3	1011010	4
0101010	3	1010101	4
0110110	4	1001001	3
0111001	4	1000110	3

Table 1.2: Hamming Codewords and Their Weights

1.7 Ensembles

While the Hamming code is a useful tool for explanation, it is quite short. Longer codes, in general, perform better. Typically, the number of codewords increases exponentially in the dimension of the code. Further, as the code's length increases, it becomes harder to analyze specific codes, and we instead focus on groups, or ensembles, of codes that share certain properties. While A_j represents the number of words of weight j in a code, $\overline{A_j}$ represents the average of A_j over all the codes in the ensemble, equally weighted. Since this value is growing exponentially, we consider the limit of its logarithm, the *spectral shape* of the ensemble.

Mathematically, the spectral shape, $E(\theta)$, is given by Equation (1.1).

$$E(\theta) = \overline{\lim}_{n\to\infty} \frac{1}{n} \log \overline{A_{\theta n}} \tag{1.1}$$

The notation was chosen to be an E because this represents the first-order exponent of the asymptotic average weight enumerator, i.e.,

$$\overline{A_{\theta n}} = 2^{nE(\theta)+o(n)}.$$

Like the weight enumerator, the spectral shape is especially important for linear

codes, as it describes not just the codeword weights, but also the distance between the codewords. The spectral shape can thus be used to calculate bounds on asymptotic code performance. One simple example of such a bound, discussed in [16], will be addressed in chapter 6.

When we discuss the average number of words of a specific weight in an ensemble, we are assuming that an ensemble contains two pieces: a set of codes, and a probability distribution which describes how likely each code is to be chosen. In all cases, the probability distribution is assumed to be uniform; each code in the ensemble is equally likely. The set of codes varies, but there are a few standard cases.

Here, we will focus specifically on ensembles of low-density parity check (LDPC) codes. These codes are preferred over other types because their parity check matrices are sparse, which allows us to design efficient encoders and decoders. The following set of subsections briefly describes some ensembles and their spectral shapes. This is by no means an exhaustive list, but will serve to introduce the ensembles in this thesis.

1.7.1 The Shannon Ensemble

In his unprecedented work, Shannon [51] proved that almost any long code achieves capacity. Later, Gallager proved the same result for almost any asymptotically long linear code [27].

We will call the set of all linear codes of rate R the *Shannon ensemble*. This ensemble is defined by the set of all $(1-R)n \times n$ binary parity check matrices. Equivalently, a code in this ensemble is defined by a random parity check matrix in which every bit is independently chosen to be zero or one with equal probability. Note that, as the parity check matrix is dense, this is not an LDPC ensemble. It is, however, useful for comparisons.

Given this random parity check matrix, H, we need to calculate the expected

number of words of weight $j = \theta n$. A word, $\mathbf{x}$ will be a codeword if and only if $H\mathbf{x} = 0$. If $\mathbf{x}$ has weight h, then $H\mathbf{x}$ is just the binary sum of h of the columns in H. However, each of the columns is independent, and each of the bits within the columns is independent. Regardless of h, each bit in the resulting word is equally likely to be a zero or a one. The probability that $\mathbf{x}$ is a codeword is thus given by $2^{-(1-R)n}$.

There are, however, $\binom{n}{\theta n}$ words of weight θn. The expected number of codewords of weight θn is simply the product of the number of words and the probability of a word being a codeword,

$$A_{n\theta} = \binom{n}{\theta n} 2^{-(1-R)n}.$$

The spectral shape is the limit of the logarithm of this equation, and is given in the following equation,

$$\begin{aligned} E_0(\theta) &= \overline{\lim}_{n\to\infty} \frac{1}{n} \log \binom{n}{\theta n} 2^{-(1-R)n} \\ &= \mathcal{H}(\theta) - (1-R). \end{aligned} \tag{1.2}$$

Here, we have used the fact that, to first order in the exponent

$$\binom{n}{n\theta} = 2^{n\mathcal{H}(\theta)+o(n)}. \tag{1.3}$$

The binary entropy function, $\mathcal{H}(\theta)$, is defined as $\mathcal{H}(\theta) = -\theta \log \theta - (1-\theta)\log(1-\theta)$, and plays a crucial role in much of information theory. This particular relationship between the binomial coefficients and the entropy function can be verified using Stirling's formula.

We could also consider a non-linear Shannon ensemble, in which 2^{Rn} words were chosen at random from the 2^n possible words. This ensemble also has a spectral shape given by Equation (1.2), as the probability of any particular word being in the code

is still given by $2^{-(1-R)n}$.

1.7.2 Regular LDPC Codes

The regular codes exist in many forms, but were first conceived by Robert Gallager [26] in 1963 as the first kind of LDPC codes. Every code in a (j, k) regular ensemble has a parity check matrix with exactly j ones in each column and k ones in each row. The spectral shape is best described parametrically, as it was in Gallager's work, given here in Equation (1.4). Several spectral shapes are shown in Figure 3.1 on page 40.

$$\begin{aligned} Z(s) &= \frac{(1+e^s)^k + (1-e^s)^k}{2} \\ F(s) &= \log Z(s) \\ E(s) &= (sF'(s) - F(s)) + (j-1)\mathcal{H}(\boldsymbol{\Theta}) \\ \boldsymbol{\Theta}(s) &= \frac{1}{k}F'(s) \end{aligned} \tag{1.4}$$

1.7.3 Litsyn and Shevelev's LDPC Ensembles

Gallager's regular ensembles, though simply defined, have rather complicated spectral shapes. It is sometimes useful to consider generalizations of the regular ensembles that may help explain the various properties of the regular ensembles. The effect that such generalizations have on the spectral shape of the ensemble is of particular importance to this thesis.

In [33], Litsyn and Shevelev discuss the regular ensemble and numerous generalizations thereof, several of which share the same spectral shape. To summarize,

- In **Ensemble A**, the parity check matrix is chosen with uniform probability from the set of all $m \times n$ binary matrices having precisely j ones in each column and k ones in each row. The spectral shape is that of the regular ensembles,

given in Equation (1.4) [2].

- In **Ensemble B**, the parity check matrix is formed with j strips of size $\frac{m}{j} \times n$, where the first strip is the k-fold concatenation of the identity matrix of size k, and the rest of the strips are random permutations of the columns of the first strip. The spectral shape is that of the regular ensembles, given in Equation (1.4).

- In **Ensemble C**, the parity check matrix is chosen uniformly at random from the set of all $m \times n$ binary matrices with precisely j ones in every column. The spectral shape is given in Equation (1.5) and several examples are plotted in Figure 3.3 on page 44.

$$E_C(\theta) = \mathcal{H}(\theta) + (1-R)H(t) + j\theta \log(1-2t) \quad (1.5)$$

where t is the only root of

$$(1-2t)\log\left(\frac{1-t}{t}\right) = 2\frac{j\theta}{1-R} \quad (1.6)$$

- In **Ensemble D**, the parity check matrix is formed by starting with an all-zeros matrix, and flipping precisely j positions (not necessarily distinct) in every column. The spectral shape is the same as that for ensemble C, given in Equation (1.5).

- In **Ensemble E**, the parity check matrix is chosen uniformly at random from the set of all $m \times n$ binary matrices with precisely k ones in every row. The spectral shape is given in Equation (1.7) and several examples are plotted in Figure 3.4 on page 44.

[2] The form of the equation found in [33] can be found by replacing our e^s with their t.

$$E_E(\theta) = \mathcal{H}(\theta) + (1 - R) \log \frac{1 + (1 - 2\theta)^k}{2} \tag{1.7}$$

- In **Ensemble F**, the parity check matrix is formed by starting with an all zeros matrix, and flipping precisely k positions (not necessarily distinct) in every row. The spectral shape is the same as that of ensemble E, given in Equation (1.7).

- In **Ensemble G**, the parity check matrix is formed by starting with an all-zeros matrix of size $m \times n$ and flipping each bit with probability $k/n = j/m$. The spectral shape is given in Equation (1.8) and several examples are plotted in Figure 3.5 on page 46.

$$E_G(\theta) = \mathcal{H}(\theta) + (1 - R) \log \frac{1 + e^{-2k\theta}}{2} \tag{1.8}$$

- In **Ensemble H**, the parity check matrix is formed from a random regular bipartite $m \times n$ graph, with left degree k, and right degree j such that matrix entry $h_{il} = e$, where e is the number of edges between left node i and right node l. The spectral shape is that of the regular ensembles, given in Equation (1.4).

1.7.3.1 A New Litsyn-Shevelev Type LDPC Ensemble*

As in [33], let H be an $m \times n$ binary parity check matrix, and let k and l be constants independent of n. In ensemble E, H is chosen with uniform probability from the set of all $m \times n$ binary matrices with rows of weight k.

We define ensemble E* in which H is chosen with uniform probability from the set of all matrices such that, given a sequence $\{k_i\}_{i=1}^m$ of positive whole numbers, the ith row of the parity check matrix has weight k_i. We define the set of row sums as $\mathcal{K}$; $K \in \mathcal{K}$ if $k_i = K$ for some i. We let m_K be the number of rows of weight K.

Following the example set in [33], we seek the spectral shape of this ensemble. Let

$\mathcal{N}(C, n, \theta)$ be the number of codewords of weight $n\theta$ in the code defined by matrix H, and $\mathbb{E}(\mathcal{N}c, n, \theta))$ denote the expected value of this quantity over all the codes in ensemble E*. Let $b(n, \theta) = \text{Prob}(\text{word of weight } n\theta \text{ is a codeword})$.

$$\mathbb{E}(\mathcal{N}(C, n, \theta)) = \binom{n}{n\theta} b(n, \theta)$$

The spectral shape is defined as $E(\theta) = \overline{\lim}_{n\to\infty} \frac{1}{n} \log \mathbb{E}(\mathcal{N}(c, n, \theta))$.

Note that when we let n approach infinity, we hold the design rate of the code $R = 1 - \frac{m}{n}$ constant. Further, we also hold the ratios $\frac{m_K}{n}$ constant.

Using the identity regarding the binomial coefficients and the binary entropy function given in Equation (1.3), we can simplify part of the equation for the spectral shape.

$$\begin{aligned} E(\theta) &= \lim_{n\to\infty} \frac{1}{n} \log \binom{n}{n\theta} b(n, \theta) \\ &= \lim_{n\to\infty} \frac{1}{n} \log \binom{n}{n\theta} + \lim_{n\to\infty} \frac{1}{n} \log b(n, \theta) \\ &= \mathcal{H}(\theta) + \lim_{n\to\infty} \frac{1}{n} \log b(n, \theta) \end{aligned}$$

To analyze $b(n, \theta)$ we first note that the probability that a word of weight $n\theta$ is a codeword is independent of the location of the $n\theta$ ones in the word. The probability that any weight $n\theta$ word is a codeword is the same as the probability that the word with $n\theta$ ones followed by $n(1-\theta)$ zeros is a word. We will denote this word $1^{n\theta}0^{n-n\theta}$.

$1^{n\theta}0^{n-n\theta}$ will be a codeword if and only if the first $n\theta$ columns of the parity check matrix sum to zero modulo 2. For this to be true, the number of ones in the first $n\theta$ columns of each row must be even. Given that a row in a matrix has weight k_i, the probability that the first $n\theta$ columns contains an even numbers of ones is given by

$$P_i = \frac{1}{\binom{n}{k_i}} \sum_{j=0}^{\lfloor k/2 \rfloor} \binom{n\theta}{2j} \binom{n - n\theta}{k_i - 2j}.$$

This is Equation 155 of Litsyn and Shevelev's work [33], and, as they state, the limit of this as $n \to \infty$ is

$$P_i = \frac{1 + (1 - 2\theta)^{k_i}}{2} + o(1).$$

This represents one row, but the rows are independent. Thus, the probability that every row has an even number of ones in the first $n\theta$ positions is given by

$$\begin{aligned} \lim_{n\to\infty} b(n, \theta) &= \prod_{i=1}^{m} P_i \\ &= \prod_{i=1}^{m} \left(\frac{1 + (1 - 2\theta)^{k_i}}{2} \right) \\ &= \prod_{K \in \mathcal{K}} \left(\frac{1 + (1 - 2\theta)^{K}}{2} \right)^{m_K}. \end{aligned}$$

For large n, the logarithm of this probability is

$$\begin{aligned} \frac{1}{n} \log b(n, \theta) &= \frac{1}{n} \sum_{K \in \mathcal{K}} m_K \log \left(\frac{1 + (1 - 2\theta)^{K}}{2} \right) \\ &= \sum_{K \in \mathcal{K}} \frac{m_K}{n} \log \left(\frac{1 + (1 - 2\theta)^{K}}{2} \right). \end{aligned}$$

and therefore,

$$E_{E*}(\theta) = \mathcal{H}(\theta) + \sum_{K \in \mathcal{K}} \frac{m_K}{n} \log \left(\frac{1 + (1 - 2\theta)^{K}}{2} \right), \tag{1.9}$$

which clearly simplifies to Equation (1.7), the spectral shape of ensemble E, when all

rows have the same row weight.

1.7.4 RA Code Ensembles

RA or *repeat accumulate* codes were the first step toward designing codes composed of simpler pieces, which led to ARA codes [1] and the AR4A/AR4JA family of protograph codes now used as JPL standards, which will be discussed shortly. RA codes were first introduced in 1998 as a simple example of an LDPC code similar to a turbo code [20]. They are easiest to describe from the point of view of the encoder. A block of data of length k is repeated q times. This new block of data is fed to an *interleaver*, which applies a permutation to the order of the bits. Finally, the scrambled data is fed to an accumulator. An accumulator takes inputs $(u_1, u_2, \ldots u_{qk})$ and produces outputs $(x_1, x_2, \ldots x_{qk})$ according to the following progression:

$$\begin{aligned} x_1 &= u_1 \\ x_2 &= u_1 + u_2 \\ x_3 &= u_1 + u_2 + u_3 \\ &\vdots \\ x_{qk} &= u_1 + u_2 + \ldots + u_{qk}. \end{aligned}$$

The spectral shape of an RA code ensemble is given implicitly by Equation (1.10) [30]. Several RA spectral shapes are shown in Figure 3.6 on page 47.

$$\begin{aligned} E(\theta) &= \max_{0 \le u \le \min(2\theta, 2-2\theta)} \left\{ f(u,\theta) + \frac{1}{q}\mathcal{H}(\theta) \right\} \\ f(u,\theta) &\equiv -\mathcal{H}(u) + (1-\theta)\mathcal{H}\left(\frac{u}{2(1-\theta)}\right) + \theta\mathcal{H}\left(\frac{u}{2\theta}\right) \end{aligned} \tag{1.10}$$

1.7.5 Protograph Code LDPC Ensembles

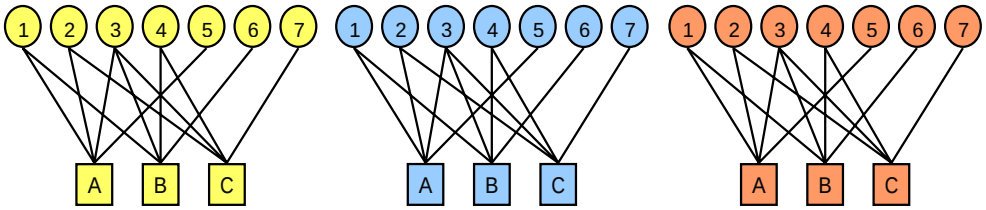
A *protograph* [59], [58] is a bipartite graph consisting of a set of variable nodes, a set of check nodes, a set of edges connecting one variable node to one check node. Variable nodes are usually denoted by circles, which are numbered, and check nodes are denoted by squares marked with letters. The protograph is equivalently described by a protomatrix M where $M_{c,v}$ is the number of edges connecting check node c to variable node v.

To create an ensemble of codes, the protograph is *lifted* by an integer L. First, L copies of the protograph are created. Now, for each $v \in V$, there are L variable nodes. We say that all of these nodes are in *class* v. Each check node or edge is also in a class defined in a similar fashion. The codes in the ensemble are formed by performing independent permutations within each class of edge. Each possible combination of permutations creates the Tanner graph for one code, and a uniform probability distribution on each of these graphs forms our protograph ensemble.

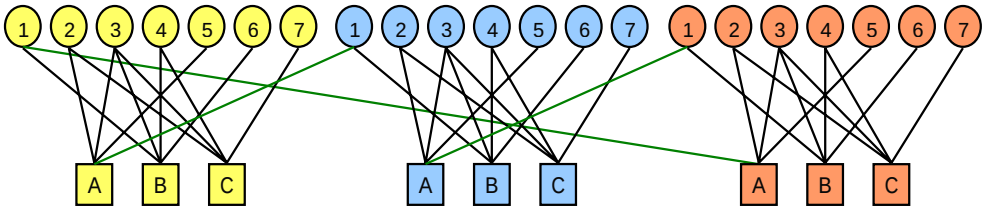
Each of these Tanner graphs has its own equivalent parity check matrix. In this matrix, the rows corresponding to check nodes of the same class have the same row sum, and the columns corresponding to variables of the same class have the same column sum. To create the parity check matrix from the protomatrix M, the element $M_{c,v}$ is replaced by an $L \times L$ binary matrix whose row and column sums are all $M_{c,v}$.

Example 3. The Hamming Code as a Protograph

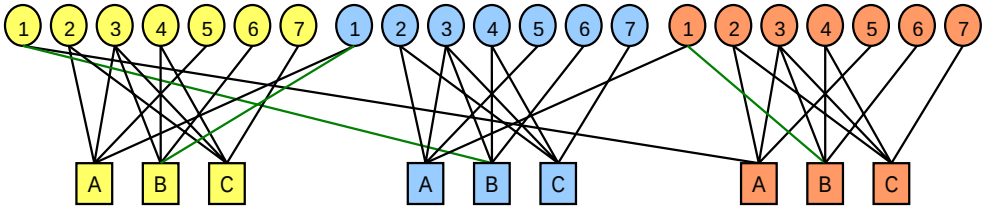
Normally, protographs are lifted such that the total length of the code is in the thousands. Here, the Hamming code will be used as a protograph, lifted 3 times, to create a code of length 21. The process is shown in Figure 1.3. First, the three copies are made. Then, a permutation is applied to each set of edges, starting with the edge that connects variable node 1 with check node A. This process is repeated for each edge.



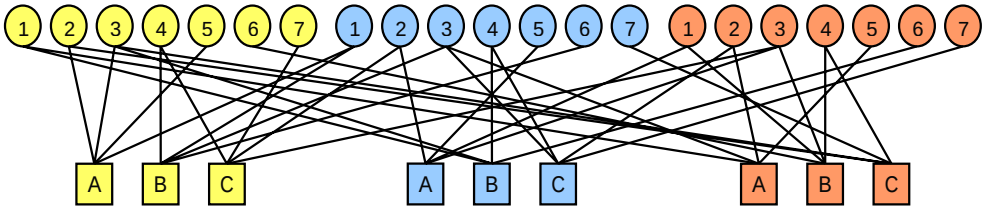
(a) First, the protograph is copied.



(b) Next, the edges between variable node 1 and check node A are permuted.



(c) Then, the edges between variable node 1 and check node B.



(d) Once all the edges have been permuted, the code is complete.

Figure 1.3: Lifting the Hamming Code

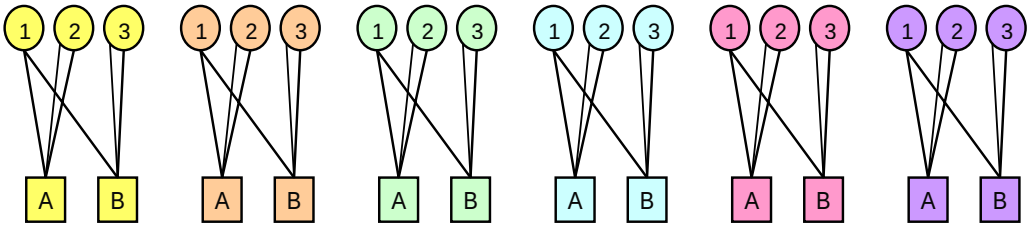
While any random permutation of the edges creates a code in the protograph ensemble, some codes are easier to describe than others. The use of a block circulant structure greatly reduces the amount of storage space required, and allows for simple encoding [5]. A circulant matrix is a square matrix formed by summing shifted versions of the identity matrix. The $n \times m$ protomatrix, M, is lifted to become a $nL \times mL$ parity check matrix, which is subdivided into nm $L \times L$ sub-matrices. The (i, j)th sub-matrix details the connections between variable nodes of class i and check nodes of class j as the sum of M_{ij} uniquely shifted versions of the identity matrix.

For example, the protomatrix $M = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix}$, is shown in Figure 1.4 being lifted by six using circulants. The six copies of the protograph are shown on top. The second row shows the permutations all applied, but the structure is not immediately obvious. The bottom row shows the same connections, but with the variable and check nodes regrouped according to class, which brings the circulant structure to light. Finally, the parity check matrix is given, which clearly shows the circulant structure.

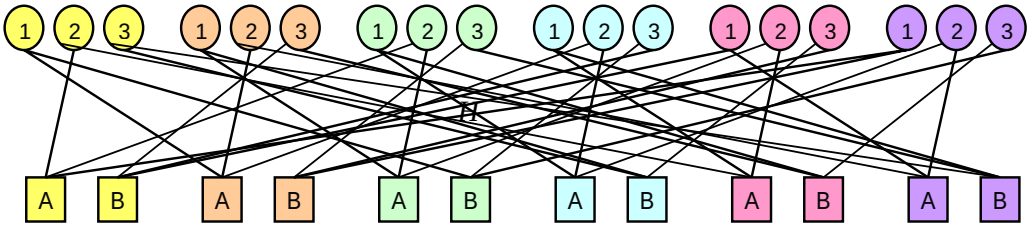
The spectral shape of a protograph code is found by the method of types and Sanov's theorem [23], [39] and is given by Equation (1.11). Figure 3.8 on page 57 shows several protograph spectral shapes.

$$\begin{aligned} E(\theta) &= \max_{\mathbf{\Theta}:\overline{\mathbf{\Theta}}=\theta} E(\mathbf{\Theta}) \\ &= \max_{\mathbf{\Theta}:\overline{\mathbf{\Theta}}=\theta} \sum_{c=1}^{m} \left(\Phi_c(\mathbf{\Theta}) - (\sum_{v=1}^{n} M_{cv} - 1)\mathcal{H}(\theta_v) \right) \end{aligned} \tag{1.11}$$

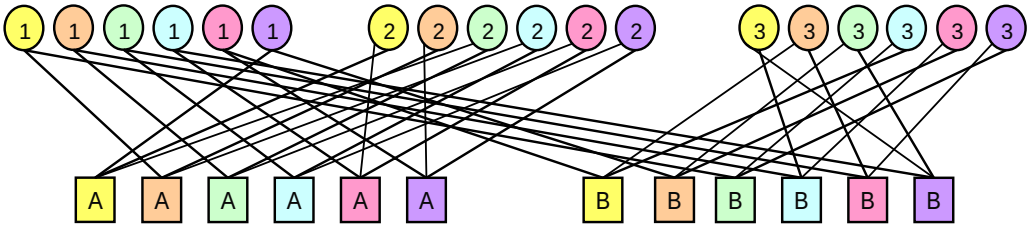
where M is the $n \times m$ protomatrix and $\Phi_c(\mathbf{\Theta})$ is the following constrained entropy



(a) Six copies of the protograph are created.



(b) The edges are permuted using circulants.



(c) The nodes are reordered by class, exposing the circulant structure.

$$H = \left(\begin{array}{cccccc}
0\,1\,0\,0\,0\,0 & 1\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0 \\
0\,0\,1\,0\,0\,0 & 0\,1\,0\,0\,0\,1 & 0\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,0 & 1\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,0 & 0\,1\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,1 & 0\,0\,1\,0\,1\,0 & 0\,0\,0\,0\,0\,0 \\
1\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,1 & 0\,0\,0\,0\,0\,0 \\
 & & \\
0\,0\,1\,0\,0\,0 & 0\,0\,0\,0\,0\,0 & 0\,0\,0\,1\,0\,1 \\
0\,0\,0\,1\,0\,0 & 0\,0\,0\,0\,0\,0 & 1\,0\,0\,0\,1\,0 \\
0\,0\,0\,0\,1\,0 & 0\,0\,0\,0\,0\,0 & 0\,1\,0\,0\,0\,1 \\
0\,0\,0\,0\,0\,1 & 0\,0\,0\,0\,0\,0 & 1\,0\,1\,0\,0\,0 \\
1\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0 & 0\,1\,0\,1\,0\,0 \\
0\,1\,0\,0\,0\,0 & 0\,0\,0\,0\,0\,0 & 0\,0\,1\,0\,1\,0
\end{array}\right)$$

(d) The parity check matrix has a block circulant structure.

Figure 1.4: Lifting with Circulants

maximization problem: Let c refer to a class of check nodes.

$$\begin{aligned} \Phi_c(\mathbf{\Theta}) = \max \mathcal{H}(p(\mathbf{x})) \qquad & \text{s.t.} \\ \sum p(\mathbf{x}) &= 1 \\ \sum \mathbf{W}(\mathbf{x}) p(\mathbf{x}) &= \alpha. \end{aligned} \tag{1.12}$$

In this equation, $\alpha_i = \theta_i M_{c,i}$ for the non-zero elements of row c in the protomatrix, and $\mathbf{x} \in \Omega_N$, the set of all binary vectors of length $N = \sum_i M_{c,i}$ which have even parity. The function $\mathbf{W}(\mathbf{x})$ is a vector of length n such that the first component is the weight of the first $M_{c,1}$ components of $\mathbf{x}$, the second component the weight of the next $M_{c,2}$ components of $\mathbf{x}$, etc.

The calculation of $\Phi_c(\mathbf{\Theta})$ is a convex problem, that will be addressed in Chapter 3. The maximization over all $\mathbf{\Theta}$ whose average is θ, however, is not convex. As the number of rows of the protomatrix (the number of classes of check nodes) increases, the landscape of $E(\mathbf{\Theta})$ develops more valleys and hills. Without calculating the value of the function at every point, one cannot be certain where the true maximum lies. For this reason, we are always calculating a lower bound on the spectral shape of a protograph code. In practice, however, using several starting points for the optimization, the spectral shape obtained in this fashion is indistinguishable from those found using the only other existing formula for protograph enumerators, given in [17]. The formula in [17] is quite similar to ours, and also involves a complicated optimization problem.

One benefit of calculating the spectral shape in this fashion is that it can calculate the spectral shape not of the codewords, but of the *stopping sets* of the code, simply by changing from $\mathbf{x} \in \Omega(N)$ to $\mathbf{x} \in \Omega^*(N)$, the set of all binary vectors of length N which have weight not equal to one. When decoding on the BEC, a stopping set is any set of variables that, if erased, cause the decoding to stop. As such, they determine

the performance. All codewords are stopping sets, but the converse is not true.

1.7.5.1 Equivalent Protographs

Many protographs which look structurally different have equivalent spectral shapes. Figure 1.5 shows three protographs representing ensembles which are contained within the regular (3,6) ensemble. The difference between the three protograph ensembles is that the ensemble featured on the left has no codes which contain double edges, while the center and right ensembles do contain double edges. However, all three of these code ensembles share their spectral shape with that of the regular (3,6) ensemble. While a regular ensemble always contains more codes than a protograph representation of the same ensemble, the difference is slight and cannot be distinguished in the spectral shape.

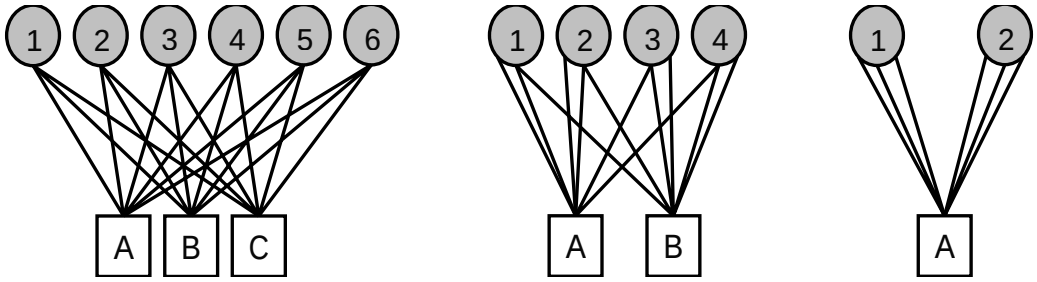


Figure 1.5: Equivalent Protographs

1.7.6 Codes Used in this Thesis

In later parts of this thesis, performance curves will be calculated for the AR4JA or AR4A groups of codes developed by the Information Processing Group at the Jet Propulsion Lab (JPL) specifically for deep-space applications. Here, the codes will be briefly described. Detailed descriptions of these codes can be found in [13] and [17].

The rate 1/2 AR4A and AR4JA protograph ensembles each contain three classes of check nodes, and five classes of variable nodes. One class of variable nodes is

punctured. A punctured variable is a variable that takes part in the encoding and decoding process, but is not transmitted. On a protograph, a punctured variable is denoted by an open, rather than shaded, circle. The protograph structure of each ensemble is shown in figures 1.6 and 1.7. The protomatrices are described in Equations (1.13) and (1.14). They are formed from the same basic building blocks: accumulators and repeaters. The final accumulator in the AR4JA code, however, is "jagged", hence the "J" in its name. Ensembles of different rates can be created by adding pairs of degree-four variables similar to variables 2 and 3. As an example of this, the rate 4/5 version of AR4A is shown in Figure 1.8.

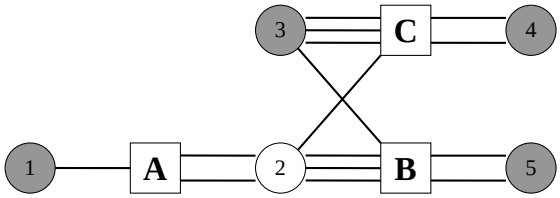


Figure 1.6: Rate 1/2 AR4A Protograph

$$H = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 & 2 \\ 0 & 1 & 3 & 2 & 0 \end{pmatrix} \tag{1.13}$$

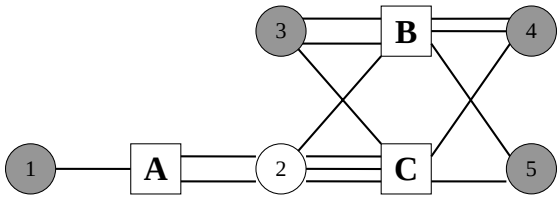


Figure 1.7: Rate 1/2 AR4JA Protograph

$$H = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 1 & 1 & 1 \\ 0 & 1 & 2 & 2 & 1 \end{pmatrix} \tag{1.14}$$

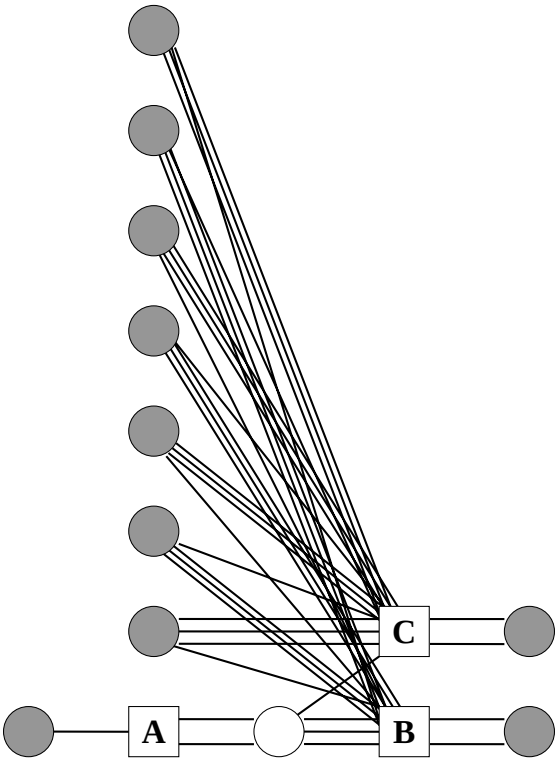


Figure 1.8: Rate 4/5 AR4A Protograph

Part I

Spectral Shapes

Chapter 2

Spectral Shapes and Cumulants

In this chapter, we will show that the spectral shape of an ensemble is closely related to that of the Shannon ensemble. In particular, we will show that if an ensemble has minimum dual distance $d^\perp$, then the first $d^\perp$ derivatives of the spectral shape at $\theta = 1/2$ will be precisely the first $d^\perp$ derivatives of the spectral shape of the Shannon ensemble, $\mathcal{H}(\theta) - (1 - R)$ at $1/2$.

2.1 Review of the Pless Identities and Dual Codes

Given a code $\mathcal{C}$ with parity check matrix H, the dual code, $\mathcal{C}^\perp$, is the code formed when H is used as a generator matrix. Every word in $\mathcal{C}^\perp$ is perpendicular to every word in $\mathcal{C}$. The weight enumerator of the dual code is denoted $A_j^\perp$, and its minimum distance is $d^\perp$. The enumerators for a code and its dual are related through the MacWilliams and Pless Identities.

We will follow the treatment of the Pless Identities given in Chapter sixteen of Berlekamp's text [8]. The number of codewords of weight j is denoted A_j. This also defines the probability that an arbitrarily chosen codeword will have weight j, i.e., $A_j/2^{Rn}$. This probability distribution allows us to calculate the average codeword weight, and higher-order moments as well. The Pless Identities relate these moments to the weight enumerator of the dual code, $A_j^\perp$. For binary codes, the rth moment

around $n/2$ is given in terms of the dual weight enumerator, according to Equation (2.1).

$$\sum_{j=0}^{n}(n/2-j)^r\frac{A_j}{2^{Rn}} = \frac{1}{2^r}\sum_{j=0}^{r}A_j^{\perp}F_r^{(j)}(n) \tag{2.1}$$

$$F_r^{(j)}(n) = \left(\frac{d^r}{dx^r}\cosh^{n-j}(x)\sinh^j(x)\right)_{x=0} \tag{2.2}$$

While Equation (2.2) is rather intimidating, for our purposes we will only need $F_r^{(0)}(n)$, as $A_j^{\perp}=0$ for $1\leq j\leq d^{\perp}$.

In the non-linear Shannon ensemble, 2^{Rn} words are chosen at random from the set of 2^n possible words. This is the same as letting each bit of each word be an independent random variable with probability one-half of being a one and probability one-half of being a zero. The moment-generating function for this Bernoulli variable is

$$\mathbb{E}\left\{e^{x(t-1/2)}\right\}=\frac{e^{-x/2}+e^{x/2}}{2}=\cosh(x/2)$$

and therefore the moment-generating function for the n independent bits that make up a codeword is $\cosh^n(x/2)$, which is very similar to $F_r^{(0)}(n)$. The right-hand side of Equation (2.1) contains 2^{-r}, which is precisely the ratio between the rth derivative of $\cosh^n(x/2)$ and $\cosh^n(x)$ at $x=0$. This leads to Theorem 2.1.

Theorem 2.1. *Consider a code, $\mathcal{C}$, whose dual code $\mathcal{C}^{\perp}$ has minimum distance $d^{\perp}$. The rth central moment of the weight of word in $\mathcal{C}$ is given by $\sum_{j=0}^{n}(n/2-j)^r\frac{A_j}{2^{Rn}}$. According to the Pless Identities, the first $d^{\perp}$ central moments are equivalent to the first $d^{\perp}$ central moments of the Shannon ensemble. In other words, the first $d^{\perp}$ terms (i.e., the constant term, $x, x^2, \ldots, x^{d^{\perp}-1}$) of the moment-generating function are the same.*

Since

$$\cosh^n(x) = 1 + \frac{nx^2}{2} + \left(\frac{n^2}{8} - \frac{n}{12}\right)x^4 + \ldots,$$

this gives us that the first few central moments are given by

$$\begin{aligned}
\sum_{j=0}^{n}(n/2-j)^0\frac{A_j}{2^{Rn}} &= 1 \\
\sum_{j=0}^{n}(n/2-j)^1\frac{A_j}{2^{Rn}} &= 0 \\
\sum_{j=0}^{n}(n/2-j)^2\frac{A_j}{2^{Rn}} &= n \\
\sum_{j=0}^{n}(n/2-j)^3\frac{A_j}{2^{Rn}} &= 0 \\
\sum_{j=0}^{n}(n/2-j)^4\frac{A_j}{2^{Rn}} &= 3n^2 - 2n \\
&\vdots
\end{aligned}$$

or,

$$\mathbb{E}\left\{e^{(j-n/2)x}\right\} = \cosh^n(x) \text{ up to degree } d^{\perp}. \tag{2.3}$$

For simplicity, we will use $\Psi(s)$ to denote the moment-generating function of the weight of a codeword in a specific code or ensemble, and $\Psi_0(s)$ to denote the moment-generating function of the weight of a codeword in the Shannon ensemble. We will also use $\psi(s) = \log \Psi(s)$ and $\psi_0(s) = \log \Psi_0(s)$ to denote their respective cumulant-generating functions.

Notation	Definition
V	a finite set
J	a finite set of n integers
$w(v)$	a function which takes $v \in V$ to $j \in J$
V_j	$\{v \in V : w(v) = j\}$
m_j	$\lvert V_j \rvert$
$Z(s)$	$\sum_{v \in V} e^{-sw(v)} = \sum_{j \in J} m_j e^{-sj}$
$F(s)$	$-\log Z(s)$
$\widetilde{F}(x)$	$\mathcal{L}\{F(s)\}$, the Legendre transform [28, 7, 2]
$G(x)$	$\max\{\sum_{j \in J} p_j \log m_j : \sum_{j \in J} p_j j = x\}$ for $x \in [0, n]$

Table 2.1: Notation for Theorem 2.2

2.2 A Legendre Transform Theorem for Spectral Shapes*

In this section, we will introduce and prove Theorem 2.2[3], which is needed to connect the cumulant generating function for codeword weight to the convex hull of the ensemble's spectral shape. The following section will deal with its application.

We will first need the notation described in Table 2.1. The function $Z(s)$ is similar to the moment-generating function. Using $\Psi(s)$ to denote the moment-generating function, $Z(s) = 2^{Rn}\Psi(-s)$. The cumulant-generating function, $\psi(s) = \log \Psi(s)$, is related to $F(s)$ similarly, i.e., $F(s) = -Rn - \psi(-s)$. The function $G(x)$ is the convex hull of the logarithm of the weight enumerator.

If we consider the Taylor series representation of $\psi(s)$ and $F(s)$ around $s = 0$, and use ψ_k and F_k to denote the coefficient of s^k in $\phi(s)$ and $F(s)$, respectively, then

$$F_k = \begin{cases} \psi_k & \text{if } k \text{ is odd} \\ -\psi_k & \text{if } k \text{ is even.} \end{cases}$$

We will need to consider $F(s)$ and not $\psi(s)$ because $F''(s)$ is easily shown to be

positive, and the Legendre transform is only defined for convex functions.

Theorem 2.2. *Aji, McEliece, Sweatlock [3]*

$$\widetilde{F}(x) - \log n \leq G(x) \leq \widetilde{F}(x)$$

In particular,

$$\frac{1}{n}\widetilde{F}(x) - \frac{1}{n}\log n \leq \frac{1}{n}G(x) \leq \frac{1}{n}\widetilde{F}(x).$$

Thus, in the limit as $n \to \infty$, $G(x) = \widetilde{F}(x)$. In words, this means that a function related to the asymptotic cumulant-generating function of a codeword's weight and the convex hull of the spectral shape are Legendre transforms of each other.

Proof. This proof is easily accomplished by looking at each inequality separately.

Proof that $\widetilde{F}(x) - \log n \leq G(x)$

$$\widetilde{F}(x) := \max_s \{xs - F(s)\}$$

To find the maximum, we differentiate with respect to s, yielding the expression $x = F'(s)$, or

$$x = \frac{\sum_{v \in V} e^{-sw(v)} w(v)}{\sum_{v \in V} e^{-sw(v)}} = \sum_{v \in V} q(v) w(v)$$

for some probability distribution q over V.

Using this value for x, we can show that $sx - F(s) = H(q)$.

$$
\begin{aligned}
sx - F(s) &= s\sum_{v\in V} q(v)w(v) + \log\sum_{\nu\in V} e^{-sw(\nu)} \\
&= \sum_{v\in V} -q(v)\log e^{-sw(v)} + \sum_{v\in V} q(v)\log\sum_{\nu\in V} e^{-sw(\nu)} \\
&= \sum_{v\in V} -q(v)(\log e^{-sw(v)} - \log\sum_{\nu\in V} e^{-sw(\nu)}) \\
&= \sum_{v\in V} -q(v)\left(\log\frac{e^{-sw(v)}}{\sum_{\nu\in V} e^{-sw(\nu)}}\right) \\
&= \sum_{v\in V} -q(v)\log q(v) \\
&= H(q)
\end{aligned}
$$

From the distribution q over V, we can convert to a distribution p over J, where for $j = w(v)$, $p_j = m_j q(v)$, and $\sum p_j j = x$. So,

$$
\begin{aligned}
H(q) &= \sum_{v\in V} -q(v)\log q(v) \\
&= \sum_{v\in V} -\frac{p_{w(v)}}{m_{w(v)}}\log\frac{p_{w(v)}}{m_{w(v)}} \\
&= \sum_{j\in J} -p_j\log\frac{p_j}{m_j} \\
&= \sum_{j\in J} -p_j\log p_j + p_j\log m_j \\
&= H(p) + \sum_{j\in J} p_j\log m_j.
\end{aligned}
$$

To summarize,

$$
\begin{aligned}
\widetilde{F}(x) &:= \max_s\{xs - F(s)\} \\
&= \max_q\{H(q) : \sum_{v\in V} q(v)w(v) = x\} \\
&= \max_p\{H(p) + \sum_{j\in J} p_j\log m_j : \sum p_j j = x\} \\
&\leq \log n + G(x)
\end{aligned}
$$

where the last line is obtained by bounded $H(p)$ by $\log n$, and noting that by definition $G(x) \geq \sum_{j \in J} p_j \log m_j$ when we assume that $\sum p_j j = x$.

Proof that $G(x) \leq \widetilde{F}(x)$

For any i and real s,

$$\begin{aligned} m_i &\leq \sum_j m_j e^{-s(j-i)} \\ &= e^{si} \sum_j m_j e^{-sj}, \end{aligned}$$

as the summation contains only positive values, one of which is m_i. Taking the logarithm on both sides yields

$$\begin{aligned} \log m_i &\leq si + \log \sum_j m_j e^{-sj} \\ \log m_i &\leq si - F(s) = \widetilde{F}(i). \end{aligned}$$

Then, we weight by p_j, sum, and let $\sum p_j j = x$ to find the desired result,

$$\begin{aligned} \sum_{j \in J} p_j \log m_j &\leq \sum_{j \in J} p_j (sj - F(s)) \\ \sum_{j \in J} p_j \log m_j &\leq s \sum_{j \in J} p_j j - F(s) \\ \{\sum_{j \in J} p_j \log m_j : \sum_{j \in J} p_j j = x\} &\leq sx - F(s) \\ G(x) &\leq \widetilde{F}(x). \end{aligned}$$

□

2.3 Connecting the Pless Identities to the Spectral Shape*

The Pless Identities show that the first $d^\perp$ terms in the moment-generating function for a codewords weight are equivalent to those of the Shannon ensemble. In this section, we'll detail the steps necessary to prove that when applying Theorem 2.2 to transform from the cumulant-generating function to the spectral shape, the first $d^\perp$ terms in the resulting series still match.

2.3.1 The Cumulant-Generating Function

We start with the Pless Identities, which give us that $\Psi(x) = \Psi_0(x)$ up to degree $d^\perp$. We would prefer to deal with the cumulant-generating functions, however, so we need to first take the logarithm on both sides of the equation. We need to prove that doing so does not alter the property that the first $d^\perp$ derivatives match.

Consider a function $M(x) = \sum_{i=0}^{\infty} a_i x^i$. We would like to find the power series for $C(x) = \log M(x)$ around $x = 0$. We will denote this power series $\sum_{i=0}^{\infty} b_i x^i$. Equating the derivatives of the two series shows that

$$\begin{aligned} b_0 &= \log a_0 \\ b_1 &= \frac{a_1}{a_0} \\ b_2 &= \frac{-a_1^2}{2a_0^2} + \frac{a_2}{a_0} \\ b_3 &= \frac{a_1^3}{3a_0^3} - \frac{a_1 c_2}{a_0^2} + \frac{a_3}{a_0}. \end{aligned}$$

Note that d_n depends only on those c_j such that $j \leq n$. This will be true for all n, as $C(x)$ is a composition of two functions ($\log x$ and $M(x)$) and the nth derivative of a composition contains only the first n derivatives of the composing functions. This

is formalized in Faá Di Bruno's formula, which is discussed more formally in Chapter 4.

Thus, given that the moment-generating functions $\Psi(s)$ and $\Psi_0(s)$ match up to degree $d^\perp$, so will the cumulant-generating functions $\psi(s)$ and $\psi_0(s)$. From the series representation of the cumulant-generating functions, we can create the series for the functions $F(s)$ and $F_0(s)$ where $F(s) = -Rn - \psi(-s)$. These two convex functions will also clearly match up to degree $d^\perp$. We are then prepared to take the Legendre transform.

2.3.2 The Legendre Transform

We are now prepared to use Theorem 2.2 and take the Legendre transform of $F(x)$ and $F_0(x)$, yielding the convex hulls of the logarithms of the weight enumerators $G(s)$ and $G_0(s)$. However, we need to show that taking the Legendre transform preserves our matching property.

Consider the Legendre transform of a convex power series $f(x) = \sum a_i x^i$

$$\begin{aligned} \widetilde{f}(s) &= \max_x \{sx - f(x)\} \\ s &= f'(x) = \sum i a_i x^{i-1} \\ sx - f(x) &= \sum i a_i x^i - \sum a_i x^i \\ &= \sum (i-1) a_i x^i \end{aligned}$$

The Legendre transform is thus given parametrically, with $s(x) = \sum i a_i x^{i-1}$ and $\widetilde{f}(s(x)) = \sum (i-1) a_i x^i$. Derivatives of functions defined parametrically are also discussed in chapter 4. Much like in Faá di Bruno's formula for function composition, the first n derivatives of a parametrically defined function depend only on $a_1, a_2, \ldots a_n$.

We know that the first $d^\perp$ central moments (i.e., the zeroth through $d^\perp$-1 moments)

of our code and the non-linear Shannon ensemble code match, which implies that $F(s)$ is tangent to $F_0(s)$ at $s = 0$ with a degree of tangency equal to $d^{\perp}$. The Legendre transform sets up a bijection between s in moment space and x in spectral shape space as $s = G'(x)$ and $x = F'(s)$ [2]. So, the point $s = 0$ in moment-space corresponds to the point $x = F'(0)$ in spectral shape space.

$$\begin{aligned} F'(s) &= \frac{-Z'(s)}{Z(s)} \\ &= \frac{-\sum A_j(-j)e^{-sj}}{2^{Rn}\Psi(-s)} \\ F'(0) &= \frac{\sum jA_j}{2^{Rn}} \\ &= \sum j\frac{A_j}{2^{Rn}}, \text{ the first moment} \end{aligned}$$

If $d^{\perp}$ is at least two, $x = F'(0) = F_0'(0) = n/2$ (we usually look at the spectral shape in terms of fractional weight θ, and this is the point $\theta = 1/2$.) If, however, $d^{\perp} < 2$, $F'(0) \neq F_0'(0) = n/2$ and the spectral shapes will not be tangent to each other as the point $s = 0$ corresponds to different values of $x = \theta n$. This is the case for Litsyn and Shevelev's [33] ensembles C, D, and G, which were introduced in Chapter one and will be further studied in the following chapter.

2.4 Conclusions*

We have glossed over one small detail — the relation between the convex hull of the spectral shape and the spectral shape itself. If a function is convex at its global maximum, then for some neighborhood around that maximum point, the convex hull and the function are equivalent. It is not trivial to show, however, that the spectral shape will be convex at its maximum point for a generic ensemble. Spectral shapes of protograph ensembles, for example, are not guaranteed to have continuous first derivatives if they contain more than one check node. However, for a wide class of

useful ensembles, there is no difference between the spectral shape and its convex hull in some neighborhood around the maximum point. When this is the case, Theorem 2.3 applies.

Theorem 2.3. *Given an ensemble whose spectral shape is convex at its maximum, and whose dual minimum weight, $d^{\perp}$, is at least two, the degree of tangency between the ensemble's spectral shape, $E(\theta)$ and that of the Shannon ensemble, $E_0(\theta) = \mathcal{H}(\theta) - (1 - R)$, is at least $d^{\perp}$ at $\theta = 1/2$.*

Theorem 2.3 suggests that if we are looking for an ensemble whose spectral shape is similar to that of Shannon ensemble, we should look for codes whose minimum dual distance is large. Since the minimum distance of the dual code is no larger than the minimum row weight of the parity check matrix, this suggests that ensembles whose parity check matrices have high row weight have spectral shapes more similar to the that of the Shannon ensemble.

Chapter 3

Spectral Shapes of Specific Ensembles

In this chapter, we will detail what is known about the spectral shapes of certain ensembles. The minimum weight of the dual code is not always known. The dual of LDPC codes are called low-density generator matrix (LDGM) codes, as the LDPC parity check matrix is the generator matrix for the dual code. The minimum distance of the LDGM code is clearly no bigger than the minimum row weight in its generator matrix. Little work has been done on showing that it is no smaller. An input/output weight enumerator for finite-length regular LDGM codes (the dual to Gallager's regular LDPC codes) has been derived [29], but not studied asymptotically for small weights. The results in this section assume the following conjecture:

Conjecture 1. *In the limit as the code's length approaches infinity, the minimum distance is precisely the minimum row weight of the generator matrix. Thus, the minimum dual weight is the minimum row weight of the parity check matrix.*

We will return to this conjecture in the concluding remarks for this chapter.

3.1 Regular Ensemble*

The spectral shape of a regular ensemble is given in Equation (1.4), but repeated here for convenience. Spectral shapes for several rate one-half ensembles are shown

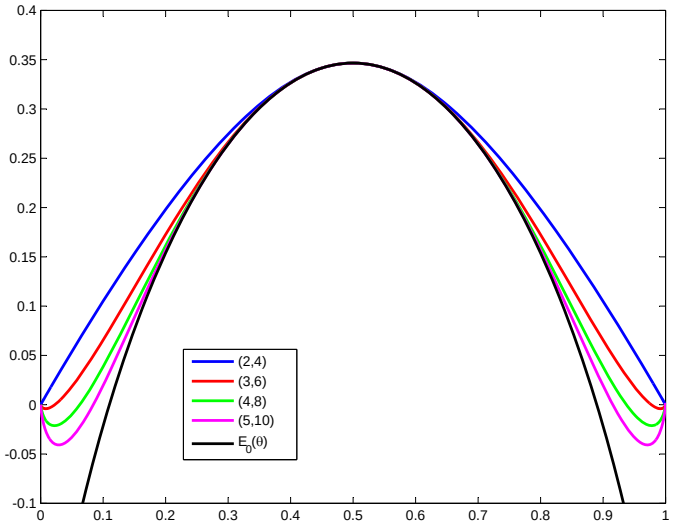


Figure 3.1: Spectral Shapes of Regular Ensembles

in Figure 3.1, along with the spectral shape of the rate one-half Shannon ensemble.

$$\begin{aligned} Z(s) &= \frac{(1+e^s)^k + (1-e^s)^k}{2} \\ F(s) &= \log Z(s) \\ E(s) &= (sF'(s) - F(s)) + (j-1)\mathcal{H}(\mathbf{\Theta}) \\ \mathbf{\Theta}(s) &= \frac{1}{k} F'(s) \end{aligned}$$

We will define the (j,k) regular ensemble as the set of all binary parity check matrices with j ones in each column and k ones in each row. The dual to the (j,k) regular ensemble LDPC is a (j,k) regular low-density generator matrix (LDGM) code. The minimum distance of the LDGM code is clearly no more than k, as each of the rows is a codeword and has weight k.

Using our formula for parametric derivatives, discussed in Chapter 4, we can calculate the derivatives of the spectral shape $E(\theta)$ with respect to θ. We can then

compare those to the derivatives of the shifted entropy function, $E_0(\theta) = \mathcal{H}(\theta) - (1 - R)$, which is the spectral shape of the Shannon ensemble of rate R. Since the ith derivative is exponentially related to i, we will list the ratio between them. At $\theta = \frac{1}{2}$, the derivatives of the entropy function are given by Equation (3.1).

$$\mathcal{H}^{(i)}\left(\frac{1}{2}\right) = \begin{cases} 0 & i \text{ odd} \\ -2^i(i-2)! & i \text{ even} \end{cases} \tag{3.1}$$

Table 3.1 lists the ratios between the even derivatives of the spectral shape of rate 1/2 regular ensembles and the entropy function. Odd derivatives, which are all zero, are not listed[1]. While the analysis in Chapter 2 explains the leading ones in each column, it does not explain why every single one of these values is an integer. Further, each element, decreased by one, is a multiple of $j(k-1)$.

Derivative	Ensemble						
i	(2,4)	(3,6)	(4,8)	(5,10)	(6,12)	(7,14)	(8,16)
2	1	1	1	1	1	1	1
4	-5	1	1	1	1	1	1
6	31	-14	1	1	1	1	1
8	-209	1	-27	1	1	1	1
10	1471	136	1	-44	1	1	1
12	-10625	-164	1	1	-65	1	1
14	78079	-1364	365	1	1	-90	1
16	-580865	3361	-419	1	1	1	-119
18	4361215	12496	1	766	1	1	1
20	-32978945	-53864	-5319	-854	1	1	1

Table 3.1: Ratios of the Derivatives of the Spectral Shape of (j, k) Regular Ensembles to Those of the Entropy Function

We would like to find the radius of convergence the Taylor series expansion of $E(\theta)$ around $\theta = 1/2$, the coefficient of $(\theta - 1/2)^i$. Given a series, $f(x) = \sum a_i(x-c)^i$, the

[1]Rate 1/2 regular ensembles have even row sums, which implies that the all ones word, 1^n is always a valid codeword. When this is the case, for every codeword (x) of weight θn, there is a word $(x + 1^n)$ with weight $(1 - \theta)n$. Thus, the spectral shape is an even function and all odd derivatives are zero.

radius of convergence is given by $\liminf_{i\to\infty} |(a_i)|^{-1/i}$. In this notation, $a_i = \frac{1}{i!}\frac{d^i E}{d\theta^i}$. Finding this value for $i > 30$ is quite time consuming, and unfortunately the limit is not reached quickly. However, if we factor out the derivatives of the entropy function and the factorial, we can say something about the radius. The radius of convergence of the entropy function is 1/2. If we let b_i be the ratio between the ith derivative of the spectral shape and the entropy function, then the radius of convergence of the series is $\frac{1}{2}\lim(b_i)^{-1/i}$. Since the spectral shape is only defined for $0 < \theta < 1$, we expect the radius of convergence at $\theta = 1/2$ to be less than 1/2. In the ensembles we have tested, $(b_i)^{-1/i} < 1$ for all $i > k + 2$, which is consistent with our expectations.

For example, for the $(3, 6)$ ensemble, for even i, the first 30 $(b_i)^{-1/i}$ are (1., 1., 0.644, 1., 0.612, 0.654, 0.597, 0.602, 0.592, 0.580, 0.594, 0.569, 0.614, 0.563, 0.592). With only fifteen values, we cannot precisely define the limit, but the sequence appears to be approaching something around 0.6, suggesting that the radius of convergence of the Taylor Series is approximately 0.3. This value is supported by Figure 3.2.

3.2 Litsyn and Shevelev's Ensembles*

Ensembles A, B, and H have spectral shapes like the regular ensembles, which were discussed in the previous section. Here, we will discuss the spectral shapes of ensembles C, E*, and G. While Litsyn and Shevelev's work [33] does show plots of the spectral shapes of their ensembles, they do not include comparisons to that of the Shannon Ensemble.

3.2.1 Ensemble C

Recall that in ensemble C, the parity check matrix is chosen with uniform probability from the set of all $m \times n$ binary matrices with exactly j ones in each column. The spectral shapes for various rate 1/2 codes are shown in Figure 3.3, along with the

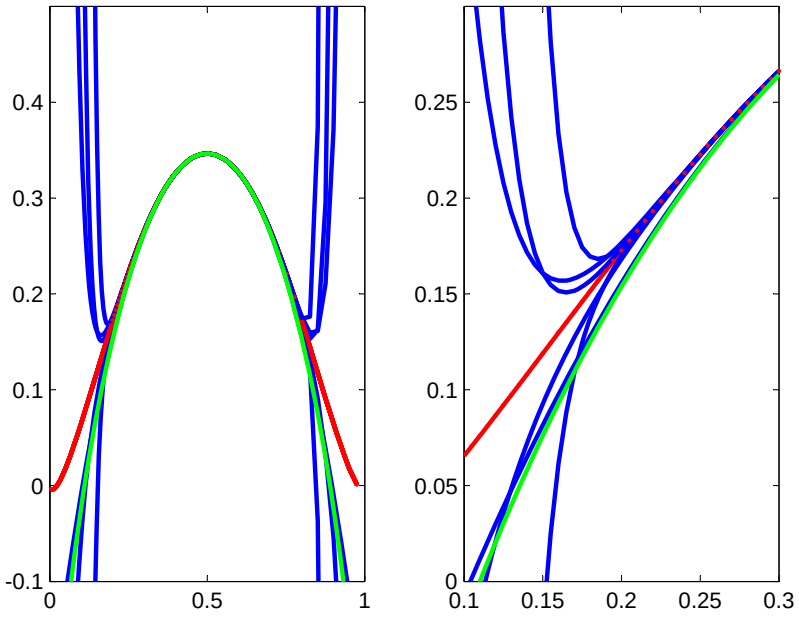


Figure 3.2: The spectral shape of the (3,6) regular ensemble is reconstructed from its Taylor series. The red curve shows the spectral shape, the blue shows the reconstruction with between 20 and 30 terms in the series. The green line is the binary entropy function. The full curve is shown on the left, and a smaller region is shown on the right. As expected, the radius of convergence is approximately 0.3.

appropriately shifted entropy function. Note that the spectral shapes do not match the shifted entropy function, even at the maximum. We expected this is because there is a possibility that rows in the parity check matrix will have weight less than two. Empty rows in the parity check matrix mean that the code is not actually the rate it was designed to be. Rows of weight one force their attached variable nodes to always be zero. When that happens, the first moment, the average codeword weight, is not at $\theta = 1/2$. In other words, the minimum dual distance is less than two, and so Theorem 2.3 does not apply.

3.2.2 Ensemble E*

In ensemble E, the parity check matrix is constrained to have precisely k ones in each row. The spectral shapes for various rate 1/2 ensembles of this type are shown in Figure 3.4.

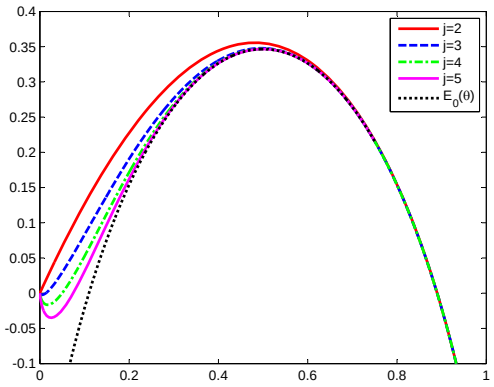


Figure 3.3: Spectral Shapes for Ensemble C

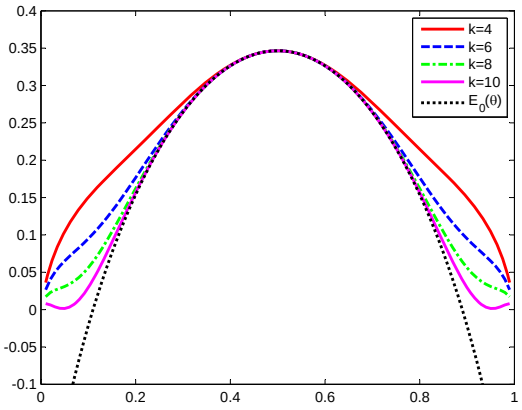


Figure 3.4: Spectral Shapes for Ensemble E

Since ensemble E can be seen as a special case of ensemble E*, we will discuss the derivative matching properties of ensemble E^* in which row i has sum k_i for a prescribed list of k_is. The minimum distance of the dual code is clearly no more than $\min k_i$. We expect the derivatives of the spectral shape at $\theta = 1/2$ to match those of the Shannon ensemble up to this degree.

We start from Equation (1.9), which gave the spectral shape of the E^* ensemble, and is repeated here for convenience:

$$E_{E*}(\theta) = \mathcal{H}(\theta) + \sum_{K \in \mathcal{K}} \frac{m_K}{n} \log\left(\frac{1 + (1-2\theta)^K}{2}\right).$$

Let us focus near the point $\theta = \frac{1}{2}$. In this neighborhood, $(1 - 2\theta)$ is small, and so

$$\log\left(\frac{1 + (1-2\theta)^K}{2}\right) = (1-2\theta)^K - \log 2 + \mathcal{O}((1-2\theta)^{2K})$$

$$E(\theta) = \mathcal{H}(\theta) + \sum_{K \in \mathcal{K}} \frac{m_K}{n} \left((1-2\theta)^K - \log 2 + \mathcal{O}((1-2\theta)^{2K})\right).$$

Note that $\sum \frac{m_K}{n} = \frac{m}{n} = 1 - R$ and so

$$E(\theta) = \mathcal{H}(\theta) - (1-R)\log 2 + \sum_{K \in \mathcal{K}} \frac{m_K}{n} \left((1-2\theta)^K + \mathcal{O}((1-2\theta)^{2K})\right).$$

In the neighborhood around $\theta = \frac{1}{2}$, the difference between spectral shape and the shifted entropy function $h(x) - (1 - R)$ is of order $(1 - 2\theta)^{k^*}$ where k^* is the smallest element in $\mathcal{K}$, and the smallest row-weight of the parity check matrix. Thus, as expected, the derivatives of the spectral shape will be identical to those of the Shannon ensemble up to the k^*th derivative.

3.2.3 Ensemble G

Parity check matrices in this ensemble begin as the all-zeros matrix, and each entry is flipped with probability $k/n = j/m$. Spectral shapes for various rate $1/2$ codes are shown in Figure 3.5. As with ensemble C, there is a possibility that a row in this parity check matrix has weight less than two, which means that the average word weight is not $n/2$, so we do not expect these spectral shapes to match the shifted entropy function.

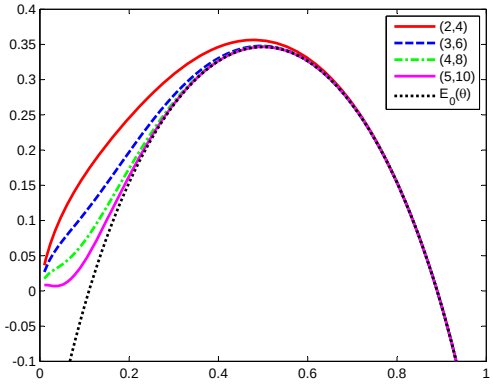
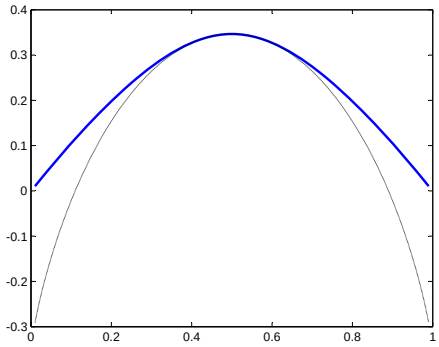


Figure 3.5: Spectral Shapes for Ensemble G

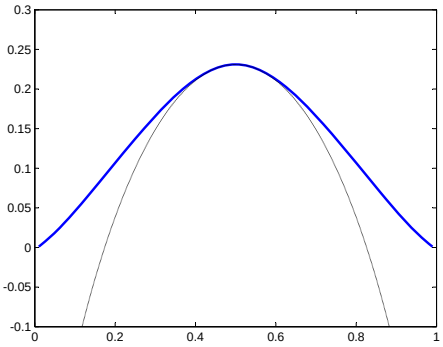
3.3 RA Codes*

An RA(q) code has rate $R = \frac{1}{q}$, so we cannot create pictures of codes with the same rate as we have for previous ensembles. Figure 3.6 shows the spectral shapes for various values of q.

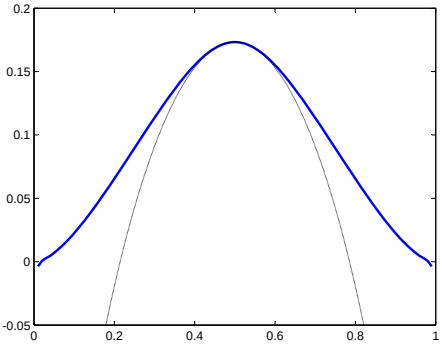
The spectral shape for an RA code is defined implicitly. Unfortunately, the formula for calculating derivatives of functions defined implicitly (given in Equation (4.4)) is much more complicated than that of the functions defined parametrically. Once again, we display not the actual derivatives, but the ratio of the derivatives to the derivatives of the entropy function. Table 3.2 lists the ratio for the first few even derivatives; the



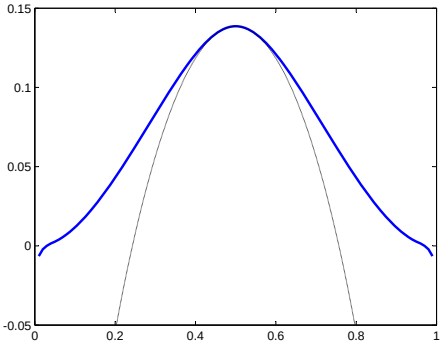
(a) RA(2)



(b) RA(3)



(c) RA(4)



(d) RA(5)

Figure 3.6: Enumerators for RA Codes of Various Rates

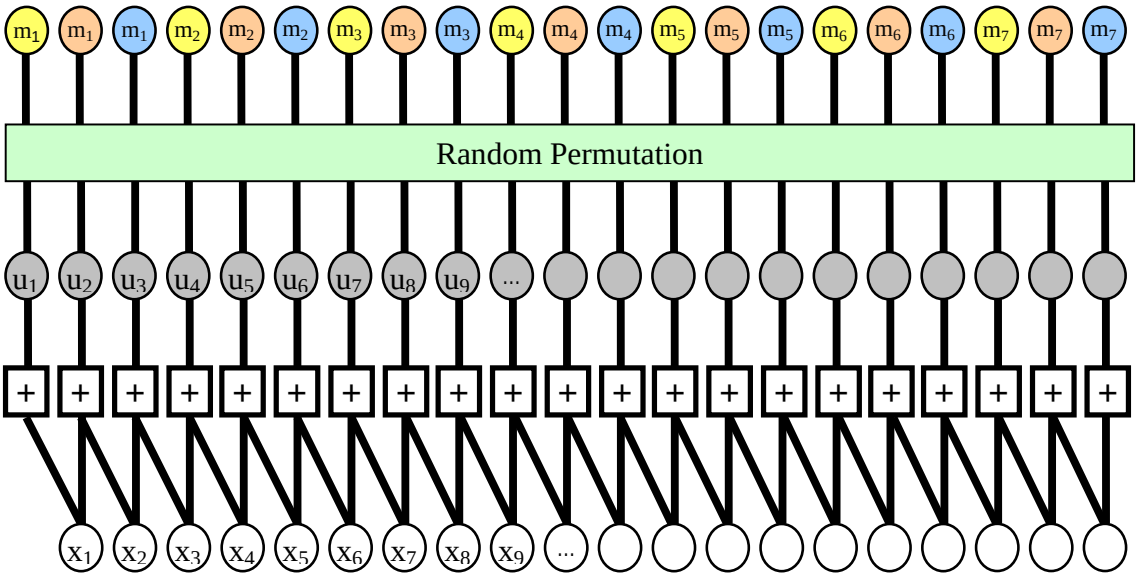


Figure 3.7: The Tanner graph for the RA(3) code is shown. The first row contains seven message bits, each repeated three times. These bits are all punctured (i.e., not transmitted). The message bits are randomly connected to another set of punctured variable nodes, denoted by u_i. These variable nodes are then accumulated, using a row of check nodes, denoted by the plus sign. The final result is the set of transmitted variables x_i.

first few odd derivatives are all zero. Once again, we notice that these ratios are all integers. We know that the degree of tangency, which in this case is four, should be at least the minimum weight of the dual code. However, this is one case where the degree of tangency actually exceeds the minimum distance of the dual code.

Derivative	RA(2)	RA(3)	RA(4)	RA(5)
2	1	1	1	1
4	-5	-11	-17	-23
6	31	121	271	481

Table 3.2: Ratios of the Derivatives of the Spectral Shapes of RA Codes to the Derivatives of the Entropy Function

To calculate the actual minimum weight of the dual to the RA codes, we need to examine the Tanner graph representation of the codes, shown in Figure 3.7. As this Figure suggests, for every i, $x_i + x_{i-1} = u_i$. If we rearrange these equations, we find

that

$$\begin{aligned} x_1 &= u_1 \\ x_2 &= u_1 + u_2 \\ x_3 &= u_1 + u_2 + u_3 \\ &\vdots \\ x_{qk} &= u_1 + u_2 + \ldots + u_{qk} \end{aligned}$$

where we have assumed that it is an RA(q) code with k message bits, and thus the code's length is $n = qk$.

Words in the dual code are found from writing parity check equations, i.e., if $x_i + x_j + x_k = 0$, then the word of weight three, with ones in positions i, j, and k, is a word in the dual code. If, for instance, $u_1 = u_2 = m_j$ for some message bit j, then $x_2 = m_j + m_j = 0$ and thus there is a word of weight one in the dual code. This, however, is unlikely to occur. Whatever u_1 happens to be, the probability that $u_2 = u_1$ is $\frac{q-1}{qk-1}$, which tends to zero as k approaches infinity.

Words of weight two are found when $x_i + x_j = u_{i+1} + u_{i+2} + \ldots + u_j = 0$. This is most likely to happen when $j - i$ is as small as possible, i.e., when identical message bits are connected to consecutive us. In this case, $x_{i-2} + x_i = u_{i-1} + u_i$. Whatever message u_{i-1} is connected to, the probability that u_i is connected to the same message is $(q-1)/qk$. Since there are qk possible choices for i, the expected number of words of weight two is $q - 1$. This is the most likely way to get words of weight two, as it only involves restrictions on two different us. If $j - i = s$, the probability drops to $\mathcal{O}((qk)^{1-s/2})$.

It is simple to show that there are at least $q - 1$ words of weight three. For any j, $x_1 + x_j + x_{j+1} = u_1 + u_{j+1}$. Regardless of what u_1 is, there are precisely $q - 1$ values of j such that $u_1 = u_{j+1}$. Other methods of obtaining words of weight three have

expected values that decrease with k.

The number of words of weight four, however, is $\mathcal{O}(k)$. This is easy to show because $x_i + x_{i+1} + x_j + x_{j+1} = u_{i+1} + u_{j+1}$. Clearly, for every value of k, there are $\binom{q}{2}$ choices for i and j.

The minimum weight of the dual to an RA(q) code is thus independent of q. Often, the minimum weight is two, as the expected number of words of weight q is $q-1$. The minimum weight is never more than three, as there are always at least $q-1$ words of weight three. The spectral shape of the RA code matched that of the Shannon ensemble up to degree four, which suggests that perhaps it is not the minimum dual weight that matters, but the minimum dual weight for which the number of words is significant with respect to n.

3.4 Protograph Codes*

Equation (1.11) was introduced as the spectral shape of a protograph code. It involved an entropy maximization problem, $\Phi_c(\boldsymbol{\Theta})$, which we will investigate here.

Let c refer to a class of check nodes in an $m \times n$ protomatrix M.

$$\begin{aligned} \Phi_c(\boldsymbol{\Theta}) = \max \mathcal{H}(p(\mathbf{x})) &= -\sum_{\mathbf{x}\in\Omega_N} p(\mathbf{x}) \log p(\mathbf{x}) (s.t.) \\ \sum p(\mathbf{x}) &= 1 \\ \sum \mathbf{W}(\mathbf{x}) p(\mathbf{x}) &= \alpha \end{aligned}$$

where $\alpha_i = \theta_i M_{c,i}$ for the non-zero elements of row c in the protomatrix, and $\mathbf{x} \in \Omega_N$, the set of all binary vectors of length $N = \sum_i M_{c,i}$ which have even parity. The function $\mathbf{W}(\mathbf{x})$ is a vector of length n such that the first component is the weight of the first $M_{c,1}$ components of $\mathbf{x}$, the second component the weight of the next $M_{c,2}$ components of $\mathbf{x}$, etc. The set of valid values for α is a closed set that will be denoted

$\mathcal{K}$. Since α is directly related to $\mathbf{\Theta}$, we will refer to this problem as $\Phi(\alpha)$ for simplicity.

Example 4. If we are dealing with the protograph $M = [3, 4]$, and $\mathbf{\Theta} = [0.5, 0.5]$, then $\alpha = [1.5, 2]$. Ω is the set of binary vectors of length 7 that have even parity. For a vector $\mathbf{x} \in \Omega$, $W(\mathbf{x})_1$ is the weight of the first 3 components of $\mathbf{x}$ and $\mathbf{W}(\mathbf{x})_2$ is the weight of the last 4 components of $\mathbf{x}$. This is detailed in Table 3.3. Note that while there are 64 possible choices for $\mathbf{x}$, $\mathbf{W}(\mathbf{x})$ takes one of only 10 values.

$\mathbf{x}$	$\mathbf{W}(\mathbf{x})$	$\mathbf{x}$	$\mathbf{W}(\mathbf{x})$	$\mathbf{x}$	$\mathbf{W}(\mathbf{x})$	$\mathbf{x}$	$\mathbf{W}(\mathbf{x})$
0000000	0 0	0100001	1 1	1000001	1 1	1100000	2 0
0000011	0 2	0100010	1 1	1000010	1 1	1100011	2 2
0000101	0 2	0100100	1 1	1000100	1 1	1100101	2 2
0000110	0 2	0100111	1 3	1000111	1 3	1100110	2 2
0001001	0 2	0101000	1 1	1001000	1 1	1101001	2 2
0001010	0 2	0101011	1 3	1001011	1 3	1101010	2 2
0001100	0 2	0101101	1 3	1001101	1 3	1101100	2 2
0001111	0 4	0101110	1 3	1001110	1 3	1101111	2 4
0010001	1 1	0110000	2 0	1010000	2 0	1110001	3 1
0010010	1 1	0110011	2 2	1010011	2 2	1110010	3 1
0010100	1 1	0110101	2 2	1010101	2 2	1110100	3 1
0010111	1 3	0110110	2 2	1010110	2 2	1110111	3 3
0011000	1 1	0111001	2 2	1011001	2 2	1111000	3 1
0011011	1 3	0111010	2 2	1011010	2 2	1111011	3 3
0011101	1 3	0111100	2 2	1011100	2 2	1111101	3 3
0011110	1 3	0111111	2 4	1011111	2 4	1111110	3 3

Table 3.3: Example of $\mathbf{x}$ and $W(\mathbf{x})$ for $[3, 4]$ Protograph

We solve the constrained maximization problem (1.11) for the spectral shape of a protograph ensemble by taking the derivative of the Lagrangian. To simplify the notation, let $\mathbf{x}_i$ be the ith vector in Ω. Let $p_i = p(\mathbf{x}_i)$ and $\mathbf{w}_i = \mathbf{W}(\mathbf{x}_i)$. We will use λ as the Lagrange multiplier for the constraint that $\sum p_i = 1$ and $\mathbf{s}$ for the constraints that $\sum \mathbf{w}_i p_i = \alpha$.

$$\begin{aligned}
\mathcal{L}(p) &= \mathcal{H}(P) - \mathbf{s} \cdot \sum \mathbf{w}_i p_i - \lambda \sum p_i \\
\mathcal{L}(p) &= -\sum_i p_i \log p_i - \mathbf{s} \cdot \sum \mathbf{w}_i p_i - \lambda \sum p_i
\end{aligned}$$

$$\frac{d\mathcal{L}(p)}{dp_i} = -(1 + \log p_i) - \mathbf{s} \cdot \mathbf{w}_i - \lambda$$

In order for the derivative of the Lagrangian to be zero,

$$p_i = \frac{e^{-s \cdot \mathbf{w}_i}}{e^{\lambda+1}},$$

but since the denominator is just a constant, we remove λ from the equation and recognize that the probability distribution p must be the Boltzmann probability density

$$p(\mathbf{x}) = \frac{e^{-\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}}}{\sum_{\mathbf{x} \in \Omega} e^{-\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}}}.$$

Let $Z(\mathbf{s}) = \sum_i e^{-\mathbf{s} \cdot \mathbf{w}_i}$ and $F(\mathbf{s}) = -\log Z(s)$.

Then,

$$\begin{aligned}
\Phi(\alpha) &= -\sum p(\mathbf{x}) \log p(\mathbf{x}) \\
&= -\sum \frac{e^{-\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}}}{Z(\mathbf{s})} \log \frac{e^{-\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}}}{Z(\mathbf{s})} \\
&= -\sum \frac{e^{-\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}}}{Z(\mathbf{s})} \left((-\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}) + F(\mathbf{s}) \right) \\
&= \sum (\mathbf{W}(\mathbf{x}) \cdot \mathbf{s}) p(\mathbf{x}) - F(\mathbf{s}) \\
&= \sum (\mathbf{W}(\mathbf{x}) p(\mathbf{x}) \cdot \mathbf{s}) - F(\mathbf{s}) \\
&= \alpha \cdot \mathbf{s} - F(\mathbf{s}).
\end{aligned}$$

This final equation may look familiar. It defines $\Phi(\alpha)$ as the Legendre transform

[28, 7, 2] of $F(\mathbf{s})$, which gives us that $\nabla F(\mathbf{s}) = \alpha$ and $\nabla\Phi(\alpha) = \mathbf{s}$. This sets up our bijection: for every $\alpha \in \mathcal{K}$, there exists one and only one $\mathbf{s} \in \mathcal{R}^n$ such that $\Phi(\alpha) = \mathbf{s} \cdot \alpha - F(\mathbf{s})$ [2]. Solving this problem then simplifies to finding the $\mathbf{s}$ such that $\nabla F(\mathbf{s}) = \sum_i \frac{\mathbf{w}_i e^{-\mathbf{s}\cdot\mathbf{w}_i}}{\sum_j e^{-\mathbf{s}\cdot w_j}} = \alpha$.

Our code first tries Broyden's method to solve this problem. If Broyden's method cannot find a solution (which typically happens along the border of $\alpha \in \mathcal{K}$), Newton's method is used. These methods are described in the following subsection. A reader already familiar with the methods will find little new information there.

3.4.1 Newton's and Broyden's Methods

Chapter 11 of Nocedal and Wright's text [42] deals with the problem of solving nonlinear equations. In our case, we are trying to find a specific value of $\mathbf{s}$ such that $\nabla F(\mathbf{s}) = \alpha$. In their notation, $r(\mathbf{s}) = \nabla F(\mathbf{s}) - \alpha$, where $r(\mathbf{s})$ is a vector of length n. In general, such a system can have no solutions, a single solution, or many solutions. In our case, we know the solution is unique because of the Legendre transform relationship.

Assuming that we have a suitable starting position, $\mathbf{s}_0$, Newton's method begins by calculating $J(\mathbf{x}_0)$, the Jacobian of $r(\mathbf{s}_0)$, which in our case is the Hessian of $F(\mathbf{s}_0)$. The algorithm is iterative, and subscripts denote the iteration, i.e., $\mathbf{x}_5$ is the vector after 5 iterations, not the fifth component of $\mathbf{x}$. The algorithm in its purist form is given in Algorithm 1. In practice, however, the addition of a backtracking line search can improve convergence times, and a stopping condition should be added. This algorithm is given in Algorithm 2.

Algorithm 1. Newton's Algorithm

- Choose $\mathbf{s}_0$;
- Calculate the Jacobian, $J(\mathbf{s}_0)$
- for k=0,1,2,...

- Solve the Newton equation for a new direction p_k

$$J(\mathbf{s}_k)p_k = -r(\mathbf{s}_k)$$

- Set $\mathbf{s}_{k+1} = \mathbf{s}_k + p_k$
- Calculate $J(\mathbf{s}_{k+1})$

Algorithm 2. Newton's Algorithm with Backtracking Linesearch

- Choose $\mathbf{s}_0$ and set tolerance ϵ
- Set $\beta_0 = 1$, $k = 0$
- Choose $\rho \in (0, 1)$
- Calculate the Jacobian, $J(\mathbf{s}_0)$
- while $r(\mathbf{s}_k) > \epsilon$
  - Solve the Newton equation for a new direction p_k

$$J(\mathbf{s}_k)p_k = -r(\mathbf{s}_k)$$

  - Choose β_k by line search
    - Set $\beta_k = 1$
    - while $(r(\mathbf{s}_k) + \beta_k p_k) > r(\mathbf{s}_k)$
      $\beta_k = \rho\beta_k$
  - Set $\mathbf{s}_{k+1} = \mathbf{s}_k + \beta_k p_k$
  - Calculate $J(\mathbf{s}_{k+1})$
  - Set $k = k + 1$

Broyden's method is an approximation to Newton's method that approximates the Jacobian at each step, which is often much faster than calculating it. This approximation is carefully constructed to mimic the behavior of the true Jacobian as each step is taken. To calculate this update, the difference between successive values of $\mathbf{s}$ and $r(\mathbf{s})$ must be known. The difference between $\mathbf{s}_{k+1}$ and $\mathbf{s}_k$ is already known; it is $x_k = \beta_k p_k$. Let $y_k = r(\mathbf{s}_{k+1}) - r(\mathbf{s}_k)$. With this new notation, Broyden's method is given in Algorithm 3.

Algorithm 3. Broyden's Algorithm

- Choose $\mathbf{s}_0$ and set tolerance ϵ
- Set $\beta_0 = 1$, $k = 0$
- Choose $\rho \in (0, 1)$
- Set B_0 equal to the Jacobian $\mathbf{s}_0$
- while $r(\mathbf{s}_k) > \epsilon$
  - Solve the Newton equation for a new direction p_k

$$B_k p_k = -r(\mathbf{s}_k)$$

  - Choose β_k by line search
    - Set $\beta_k = 1$
    - while $(r(\mathbf{s}_k) + \beta_k p_k) > r(\mathbf{s}_k)$ $\beta_k = \rho\beta_k$
  - Set $\mathbf{s}_{k+1} = \mathbf{s}_k + \beta_k p_k$
  - Set $x_k = \beta_k p_k$ and $y_k = r(\mathbf{s}_{k+1}) - r(\mathbf{s}_k)$
  - Update B_{k+1}

$$B_{k+1} = B_k + \frac{(y_k - B_k x_k)x_k'}{x_k' x_k}$$

  - Set $k = k + 1$

3.4.2 Specific Protograph Enumerators*

Since the spectral shape of a protograph is found numerically, and only approximately, calculating the derivatives is not feasible. However, we can still compare the enumerators to the shifted entropy function visually, and see what we notice. We expect the minimum weight of the dual code to be equal to the minimum row weight of the protomatrix. If this were true, protomatrices with higher row sums would have spectral shapes more like the shifted entropy function. Here we will focus on rate 1/2 protomatrices with a single row, corresponding to a single class of check node. In this case, the outer non-convex maximization problem does not exist, and thus the spectral shape is easier to calculate and far more accurate.

Figure 3.8 shows the spectral shapes for various protograph ensembles. There are three sets of pictures corresponding to three different check node weights. In all cases,

the full enumerator is shown on the left and a close-up of the behavior near zero is shown on the right. This allows us to quickly identify which ensembles have positive zero crossings, which indicates better code performance. This will be discussed further in chapter 6. Note that the ensembles with higher check node degree are more likely to have this property, and they also remain close to the shifted entropy function $E_0(\theta)$ longer, suggesting that their degree of tangency is higher.

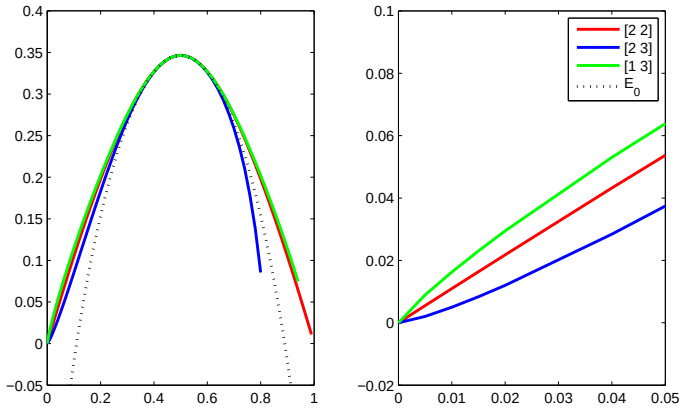
As mentioned in Chapter 1, one of the benefits of our method for calculating protograph spectral shapes is that it can be easily altered to calculate the growth rate of the stopping set enumerators. These growth rates are shown in Figure 3.9. Note that protograph ensembles with larger check degree are more likely to have a growth rate which is negative up to some value of θ, denoted θ_s. In theory, this means that asymptotically long codes will have no stopping sets smaller than $\theta_s n$. In practice, however, any finite length code will have smaller stopping sets [25].

3.5 Conclusions

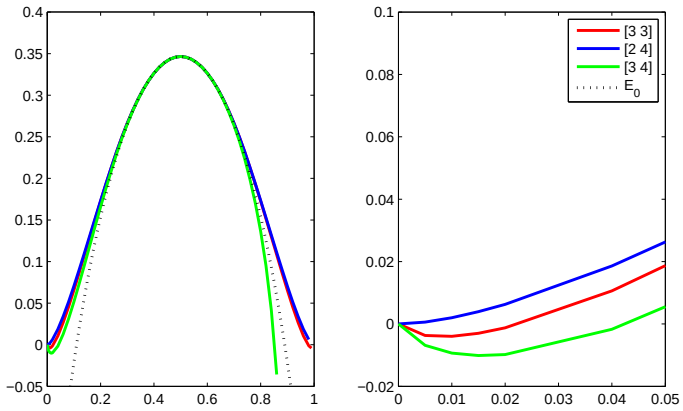
In this chapter, we reviewed the spectral shapes of several ensembles, and examined the degree of tangency between their spectral shapes and the shifted entropy function, which is the spectral shape of the Shannon ensemble. The hope is that ensembles whose spectral shapes closely mimic that of the Shanon ensemble will contain codes whose performance mimics that of the Shannon ensemble. In this chapter, we saw that ensembles whose parity check matrices have high row weight have spectral shapes with higher degrees of tangency to the spectral shape of the Shannon ensemble.

We also saw that ensembles like Litsyn and Shevelev's ensembles C and G [33], that place no restrictions on the row weight of the parity check matrix, have spectral shapes that differ greatly from the shifted entropy function.

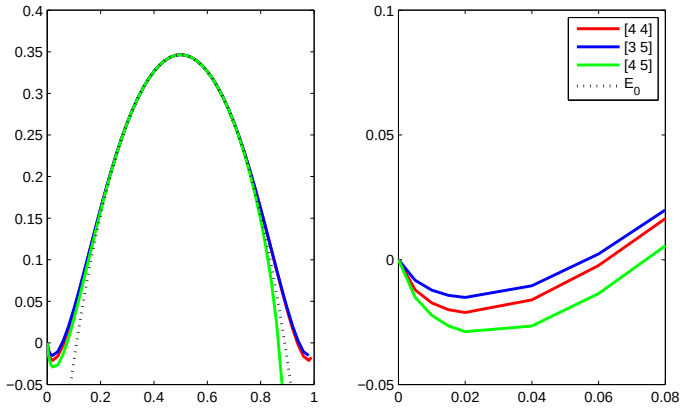
Further study of the dual codes to these ensembles may be needed to prove a



(a) Protographs with Small Check Node Degrees

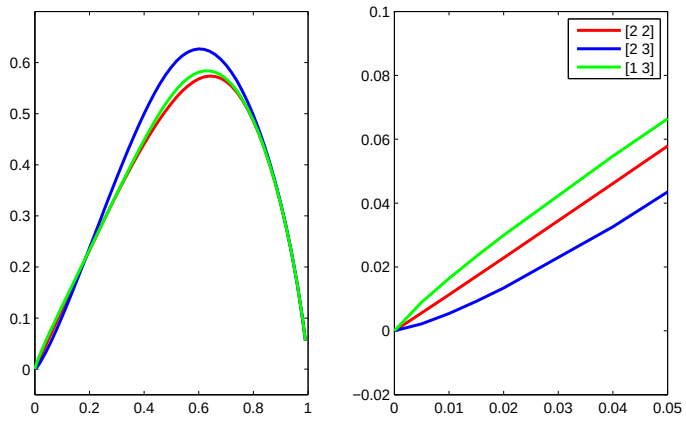


(b) Protographs with Mid-Size Check Node Degrees

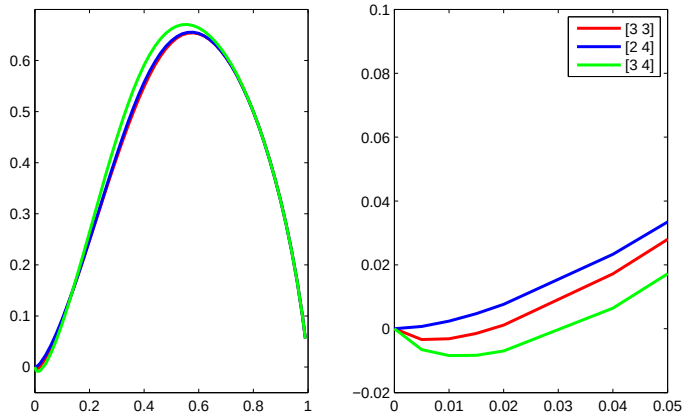


(c) Protographs with Large Check Node Degrees

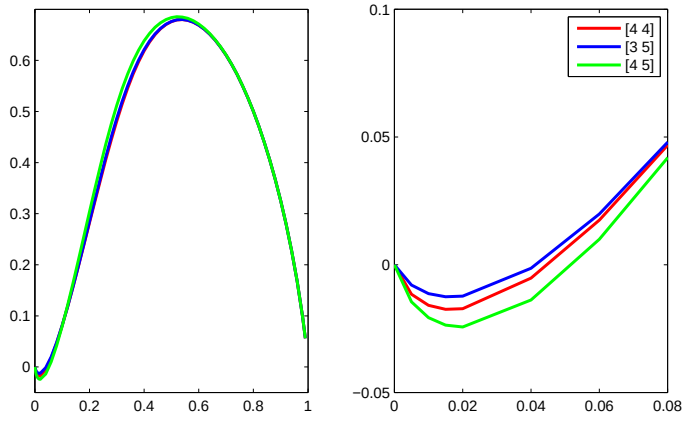
Figure 3.8: Spectral Shapes of Protograph Ensembles



(a) Protographs with Small Check Node Degrees



(b) Protographs with Mid-Size Check Node Degrees



(c) Protographs with Large Check Node Degrees

Figure 3.9: Stopping Set Enumerators for Protograph Ensembles

stronger result than Theorem 2.3, which stated that the degree of tangency was at least $d^{\perp}$, the minimum dual distance. If the ensemble's parity check matrices are constrained to have row weights at least k, it is certainly still possible to have words in the dual code of smaller weight. As the code's length increases, however, this becomes less likely.

If every row in a parity check matrix has weight k, then there are at least $\mathcal{O}(n)$ words of weight k in the dual code. The sum of any two random rows is likely to have weight $2k$, and thus there are at least $\mathcal{O}(n^2)$ words of that weight. For regular codes, the number of ones in each column is fixed, thus it will always be possible to select two rows such that their sum creates a word of weight $2k-2$. There are, then, at least $\mathcal{O}(n)$ words of weight $2k-2$.

There are far fewer words of weight less than k, and of weight $k+1$ through $2k-3$. For a regular ensemble, we suspect that the expected number of words of these weights is $o(n)$. If this is the case, then the results in this chapter support a stronger hypothesis than Theorem 2.3, i.e., that the ith derivative of the spectral shape of an ensemble at $\theta = 1/2$ is precisely equal to that of the spectral shape of the Shannon ensemble if the number of words in the dual ensemble of weight i is $o(n)$. This would suggest that, while the minimum dual weight is useful for predicting the spectral shape of the primal ensemble, knowing even a few more points in the dual enumerator would allow for a much more accurate description of the primal code's spectral shape.

Chapter 4

Combinatoric Derivatives of Relevant Functions

4.1 Introduction and Notation

To apply Theorem 2.2 in Chapter 2 we needed a few results regarding the derivatives of functions constructed from other functions. We needed Faá di Bruno's famous formula for calculating the derivatives of function compositions, and a new formula to differentiate functions defined parametrically. The parametric derivatives formula is always needed to calculate the derivatives of the spectral shape of Gallager's regular LDPC ensembles. For RA code ensembles, a formula is needed for differentiating functions defined implicitly. In this chapter, we will give combinatoric formulas for each of these three derivatives. Much of this chapter is based on the first few pages of Chapter 5 of Stanley and Fomin's combinatorics text [53], though we will alter their notation slightly.

A *sequence* is a mapping from the nonnegative integers $\mathbb{N}$, or the positive integers $\mathbb{P}$, to a fixed field K of characteristic zero. Typically, such a sequence will be denoted by $f[0], f[1], f[2], \ldots$. The *exponential generating function* for a sequence $f[n]$ is the formal power series

$$E_f(x) = \sum_{n \geq 0} f[n] \frac{x^n}{n!}.$$

The original sequence $f[n]$ can be recovered from the series $E_f(x)$ by formal differentiation:

$$f[n] = \frac{d^n}{dx^n} E_f(x)|_{x=0}.$$

This important pairing is summarized with the symbol

$$f[n] \leftrightarrow E_f(x).$$

The problems to be addressed in this section are as follows: Given sequences $f : \mathbb{P} \to K$, with $f[1] = 1$ and $g : \mathbb{N} \to K$, with $g[0] = 1$, construct sequences $k[n]$, and $h[n]$ such that

$$\begin{aligned} E_k(x) &= E_f(E_g(x)) \\ E_h(x) &= E_g(E_f^{(-1)}(x)). \end{aligned}$$

To construct these sequences, we will need the notion of set theory and number theory partitions. We will use the capital letter Λ to refer to the former, and lowercase λ to refer to the latter. A *number theory* partition n is a multiset of positive integers that sum to n. For instance, $\lambda = \{1, 1, 2, 2, 2\}$ is a partition of 8. Functions of λ are defined to be the product of the functions of each λ_i, i.e., if $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$, then for a generic function f, $f[\lambda] = f[\lambda_1] f[\lambda_2] \ldots f[\lambda_k]$. A partition can also be characterized by its multiplicities $m_j = \#\{i : \lambda_i = j\}$. For example, the partition $\lambda = \{1, 1, 2, 2, 2\}$ is denoted by its multiplicities as $1^2 2^3$ and $f[\lambda] = f[1]^2 f[2]^3$.

A set theory partition of n is a sectioning of the integers from 1 to n. Each set theory partition corresponds to a number theory partition. This number theory partition is a list of the size of each subset. The set theory partition $\{(1)(2)(34)(56)(78)\}$ has corresponding number theory partition $\lambda = \{1, 1, 2, 2, 2\}$.

Let $\#\lambda$ be the number of parts in a given partition λ and r denote the number of

singletons in the set theory partition (the number of subsets with only one number). Finally, let $\mathcal{N}(\lambda)$ be the number of set theory partitions that collapse into number theory partition λ.

$$\mathcal{N}(\lambda) = \frac{n!}{\prod_{i=1}^{n}(i!)^{m_i} m_i!}$$

Table 4.1 lists a few examples of each type of partition for $n = 5$.

Num. Theory Partition	$\#\lambda$	r	$\mathcal{N}(\lambda)$	(Some) Set Theory Partitions
$\{1,2,2\}$	3	1	15	$(1)(23)(45), (1)(24)(35), (1)(25)(34)$ $(2)(13)(45), (2)(14)(35), (2)(15)(34), \ldots$
$\{5\}$	1	0	1	(12345)
$\{2,3\}$	2	0	10	$(12)(345), (14)(234), (124)(35), \ldots$
$\{1,4\}$	2	1	5	$(1)(2345), (1345)(2), (1245)(3),$ $(1235)(4), (1234)(5)$

Table 4.1: Examples of Set and Number Theory Partitions

4.2 Faá di Bruno and Function Composition

Faá di Bruno's formula [21] calculates the derivatives of a function defined as the composition of two functions. While Faá di Bruno's result is the most well known, he was not the first or the last to study this particular problem. It is possible that the first calculation is due to Arbogast [6] in 1800. More historical framework is found in Johnson's work [31]. More recently, research has focused on extending these results to the multivariate case [12]. In the next section, we will extend it to a different kind of function, one that is defined parametrically.

Theorem 4.1. *Faá di Bruno's Formula Given two functions and their corresponding sequences*

$$\begin{aligned} E_f(x) &\leftrightarrow f[n] \\ E_g(x) &\leftrightarrow g[n], \end{aligned}$$

The exponential generating function for their composition,

$$E_k(x) = E_f(E_g(x)),$$

can be described by the corresponding sequence $k[n]$ where

$$\begin{aligned} k[n] &= \sum_{\lambda \in P(n)} \mathcal{N}(\lambda) f[\#\lambda] g[\lambda] \\ P(n) &= \left\{ \lambda : \sum_j \lambda_j = n \right\}. \end{aligned}$$

Example 5. Some Examples of $P(n)$ and $k[n]$

$$\begin{aligned} P(1) &= \{1\} \\ P(2) &= \{\{2\}, \{1,1\}\} \\ P(3) &= \{\{3\}, \{2,1\}, \{1,1,1\}\} \\ P(4) &= \{\{4\}, \{3,1\}, \{2,2\}, \{2,1,1\}, \{1,1,1,1\}\} \\ k[0] &= 1 \\ k[1] &= f[1] \\ k[2] &= f[1]g[2] + f[2] \\ k[3] &= f[1]g[3] + 3f[2]g[2] + f[3] \\ k[4] &= f[1]g[4] + 4f[2]g[3] + 3f[2]g[2]^2 + 6f[3]g[2] + f[4] \end{aligned}$$

4.3 Parametric Functions*

This section deals with the problem of how to take the derivatives of functions defined parametrically. We assume that we have two generating functions $E_g(t) \leftrightarrow g[n]$ and $E_f(t) \leftrightarrow f[n]$. We assume that while $E_f(t)$ is a bijection, inverting the function analytically is intractable. Further, we assume that the problem has been scaled such that $E'_f(0) = f[1] = 1$. We have defined the new generating function

$$E_h(x) = E_g(E_f^{(-1)}(x)) \leftrightarrow h[n]$$

and we are looking for the sequence $h[n]$. In simpler language, $h[n]$ is the nth derivative of a function defined parametrically as $(E_g(t), E_f(t))$ evaluated at the point $t = 0$,

$$h[n] = \frac{d^n E_g}{dE_f^n}|_{t=0}.$$

We need to introduce new notation for the multiplicity of one in the partition, $r = m_1$, as this plays a crucial role in the theorem. When necessary, we will use subscripts, i.e., r_λ to distinguish one partition's r value from that of another partition.

Theorem 4.2. *Parametric Derivatives Theorem*

$$\begin{aligned} E_h(x) &= E_g(E_f^{(-1)}(x)) \leftrightarrow h[n] && (4.1)\\ h[n] &= \sum_{\lambda \in V(n)} (-1)^{\#\lambda - r} \mathcal{N}(\lambda) f[\lambda] g[r+1] && (4.2)\\ V(n) &= \left\{ \lambda : \sum_{j=1}^{\#\lambda} \max(\lambda_j - 1, 1) = n - 1 \right\} && (4.3) \end{aligned}$$

The proof of this theorem is given in the appendix.

Example 6. Some Examples of $V(n)$ and $h[n]$

$$
\begin{aligned}
V(1) &= \emptyset \\
V(2) &= \{\{2\},\{1\}\} \\
V(3) &= \{\{3\},\{2,2\},\{2,1\},\{1,1\}\} \\
V(4) &= \{\{4\},\{3,2\},\{3,1\},\{2,2,2\},\{2,2,1\},\{2,1,1\},\{1,1,1\}\}
\end{aligned}
$$

$$
\begin{aligned}
h[1] &= g[1] \\
h[2] &= g[2] - g[1]f[2] \\
h[3] &= g[3] - 3g[2]f[2] - g[1]f[3] + 3g[1]f[2]^2 \\
h[4] &= g[4] - 6g[3]f[2] - 4g[2]f[3] + 15g[2]f[2]^2 - g[1]f[4] + \\
&\quad +10g[1]f[2]f[3] - 15g[1]f[2]^3
\end{aligned}
$$

4.4 Implicit Functions*

Assume that we are given $g(x, y) = 0$, and that this uniquely defines y in terms of x, though in a manner that is too complicated to be practical. Let $F(x) = f(x, y)$. We seek the total derivatives of F with respect to x, i.e., $\frac{dF}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx}$.

If we take the derivative of $g(x, y(x)) = 0$ with respect to x, we find that $\frac{\partial g}{\partial x} + \frac{\partial g}{\partial y}\frac{dy}{dx} = 0$, which simplifies to

$$\frac{dy}{dx} = \frac{\frac{\partial g}{\partial x}}{\frac{\partial g}{\partial x}}.$$

This gives us our first derivative

$$\frac{dF}{dx} = \frac{\partial f}{\partial x} - \frac{\frac{\partial f}{\partial y}\frac{\partial g}{\partial x}}{\frac{\partial g}{\partial y}}.$$

This also sets up our recursion

$$\frac{d^n F}{dx^n} = \frac{\partial}{\partial x}\frac{d^{n-1}F}{dx^{n-1}} - \frac{\frac{\partial g}{\partial x}}{\frac{\partial g}{\partial y}}\frac{\partial}{\partial y}\frac{d^{n-1}F}{dx^{n-1}}.$$

If we use subscripts to denote partial derivatives, i.e.,

$$f_x = \frac{\partial f}{\partial x}$$

and

$$f_{xx} = \frac{\partial^2 f}{\partial x^2},$$

the first few derivatives are given by

$$\begin{aligned}
\frac{dF}{dx} &= f_x - \frac{f_y g_x}{g_y} \\
\frac{d^2F}{dx^2} &= f_{xx} - \frac{f_y g_{yy} g_x^2}{g_y^3} + \frac{f_{yy} g_x^2}{g_y^2} + \frac{2 f_y g_{xy} g_x}{g_y^2} - \frac{2 f_{xy} g_x}{g_y} - \frac{f_y g_{xx}}{g_y}
\end{aligned}$$

$$\begin{aligned}
\frac{d^3F}{dx^3} &= f_{xxx} - \frac{3 f_y g_{yy}^2 g_x^3}{g_y^5} + \frac{3 f_{yy} g_{yy} g_x^3}{g_y^4} + \frac{9 f_y g_{yy} g_{xy} g_x^2}{g_y^4} + \frac{f_y g_{yyy} g_x^3}{g_y^4} - \frac{3 f_{xy} g_{yy} g_x^2}{g_y^3} \\
&\quad - \frac{6 f_{yy} g_{xy} g_x^2}{g_y^3} - \frac{f_{yyy} g_x^3}{g_y^3} - \frac{3 f_y g_{yy} g_{xx} g_x}{g_y^3} - \frac{3 f_y g_{xyy} g_x^2}{g_y^3} - \frac{6 f_y g_{xy}^2 g_x}{g_y^3} + \frac{6 f_{xy} g_{xy} g_x}{g_y^2} \\
&\quad + \frac{3 f_{yy} g_{xx} g_x}{g_y^2} + \frac{3 f_{xyy} g_x^2}{g_y^2} + \frac{3 f_y g_{xxy} g_x}{g_y^2} + \frac{3 f_y g_{xy} g_{xx}}{g_y^2} - \frac{3 f_{xxy} g_x}{g_y} - \frac{f_y g_{xxx}}{g_y} - \frac{3 f_{xy} g_{xx}}{g_y}.
\end{aligned}$$

With each subsequent differentiation, the number of x derivatives increases by one, as we might expect. The number of derivatives in y varies, but it is always

balanced; the denominator has exactly as many derivatives in y as the numerator. Every time we take a derivative in y, we divide by g_y. The nth derivative always contains the term f_{x^n}, but all the other terms involve g in some way.

Unlike with function composition and parametric functions, these derivatives cannot be described with simple partitions. We instead need partition pairs. Each element of the partition will be an ordered pair. The sum of these ordered pairs will be the pair (n, k) We will assume the convention that in each pair, the first element corresponds to x derivatives and the second to y. For example, with $(n, k) = (2, 3)$, one possible partition is $\lambda = (2, 1)(0, 1)(0, 1)$. However, some of these derivatives are of f and others are of g. We will assume that the first pair corresponds to the derivatives of f, and we will call this first pair μ. Thus, the partition λ above corresponds to the term $f_{xxy}g_x^2$. The multiplicities of elements of the partition are defined pairwise, i.e., $m_{i,j}$ is the multiplicity of pair (i, j) in the partition.

Conjecture 2. *Equation* (4.4) *gives a formula for the nth derivative of* $F(x) = f(x, y)$ *when* y *is defined by* $g(x, y) = 0$.

$$\frac{d^n F}{dx^n} = \sum_{k=0}^{2n-1} \frac{(-1)^k}{g_y^k} \sum_{\lambda \in K(n,k)} C(\lambda) f_\mu g_\lambda, \tag{4.4}$$

where

$$C(\lambda) = \frac{1}{\prod_{ij} m_{i,j}!} \frac{n!}{\prod_i \lambda_{i_x}!} \frac{(k-1)!\lambda_{1_y}}{\prod_i \lambda_{i_y}!}.$$

The set of partitions $K(n, k)$ is a subset of the set of all partitions of x degree n and y degree k. The restrictions on $K(n, k)$ come directly from the recursive formula for calculating the derivatives. First, even though k is often bigger than n, neither a specific element nor the sum of any pair can be larger than n. This corresponds to the idea that we are taking only n derivatives, so no single f or g term can have more than n derivatives. The pair $(0, 1)$ can only refer to f (i.e., it can only be the first

pair), as the term g_y cannot appear in the numerator. Finally, every time we take a derivative w.r.t y, we then multiply by a negative sign and g_x/g_y. Because of this, the number of y derivatives (k) must also be equal to the number of terms involving g. Since there is always one and only one term involving only f, there are a total of $k+1$ pairs in the partition.

The coefficient $C(\lambda)$ is the product of the multinomial coefficients that describe how many ways we can partition the n derivatives in x and $k-1$ derivatives in y. This number is divided by the multiplicities because the derivatives are indistinguishable. Only $k-1$ derivatives in y are considered because, except for the term f_{x^n} (which contains no y derivatives) every term in the summation contains at least one y derivative of f. Put in another way, $C(\lambda)$ is precisely the number of set theory partitions that collapse into number theory partition λ if we assume that if μ_y is not empty, it contains the number one.

Conjecture 2 is known to be accurate up to at least the eighth derivative. To our knowledge, standardized, consistent notation does not yet exist for partitions of pairs. Ongoing work on this and other derivatives will hopefully lead to such notation and a conclusive proof of this conjecture.

Part II

Practical Considerations for Protograph Codes

Chapter 5

Iterative Decoding of LDPC Codes

In this chapter, we will discuss the iterative decoding of LDPC codes. In the first section, we will introduce the idea of maximum likelihood decoding. In the second section, we will discuss the iterative decoding framework. Finally, the last section discusses the effects of adding an input buffer to the decoding system.

5.1 Bounded Distance and Maximum Likelihood Decoding

In the first chapter, we defined the weight enumerator A_j as the number of words of weight j in a given code. For any linear code, $A_0 = 1$, meaning that there is precisely one codeword of weight zero. For good codes, $A_j = 0$ for all j less than some value d, called the *minimum distance*. A code with minimum distance d can always correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors using a *bounded distance decoder* (BDD). Imagine the codeword vectors as points in space. No two words are closer together than the minimum distance, d. If we draw spheres around each codeword of radius $\left\lfloor \frac{d-1}{2} \right\rfloor$, then, no two spheres will overlap. If no more than $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors are made by the channel, the received word will lie within the sphere of the transmitted word, and thus be correctly decoded.

Figure 5.1 illustrates this decoding. The smaller circles represent codewords, and the large circles a radius of $\left\lfloor \frac{d-1}{2} \right\rfloor$. The red codeword was transmitted. If less than

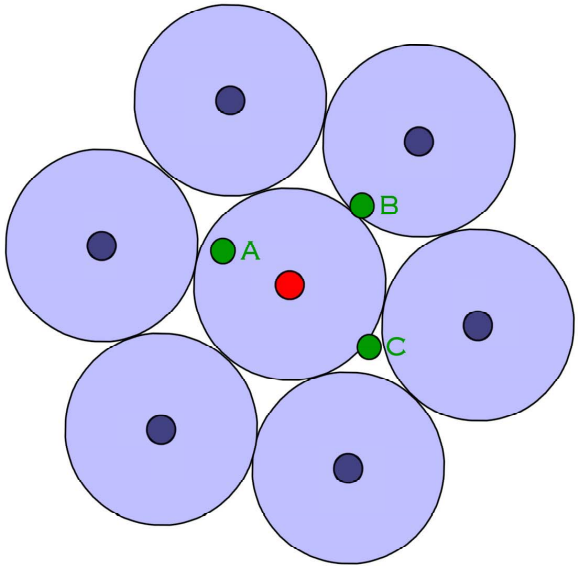


Figure 5.1: Decoding Spheres

$\lfloor \frac{d-1}{2} \rfloor$ errors are made, the received word resembles the green circle labeled A and is clearly within the sphere for the desired codeword. If slightly more errors are made, the result could be something like green circles B or C. A bounded distance decoder would make an error in both of these cases.

However, the circle labeled C, though outside the sphere of radius $\lfloor \frac{d-1}{2} \rfloor$ is still closer to the transmitted codeword than any other codeword. A maximum likelihood decoder always finds the closest codeword to the received word. Because of this, it can decode more than $\lfloor \frac{d-1}{2} \rfloor$ errors some of the time. Using the weight enumerator, or spectral shape, bounds on the maximum likelihood decoding error probability can be found. One simple example of such a bound, discussed in [16], will be addressed in chapter 6.

For a code of rate R and length n, there are 2^{Rn} codewords. A maximum likelihood decoder has to find the distance between the received word and each of the codewords in order to choose the smallest one. So, while the maximum likelihood decoder can correct the most errors, its complexity grows exponentially with the length of the codewords. For this reason, iterative decoding methods, which have complexity that

grows linearly with the length of the codewords, are much preferred.

5.2 Introduction to Message Passing Decoding

When discussing iterative decoding, it is helpful to think of the code in its Tanner graph form. Each bit of the received message is represented by a variable node. Each variable node is connected to a set of check nodes. In a valid codeword, the subset of variable nodes that each check node is connected to has even parity.

Messages are passed from variable nodes to check nodes and back again along the edges that connect them. These messages consist, in general, of information about the value of each variable node. Message passing algorithms like this are a specific case of the general class of belief propagation algorithms.

5.2.1 Message Passing on the BEC

Message passing is easiest to understand on the binary erasure channel (BEC). As described in chapter one, this channel introduces no errors, but erases some message bits. In the Tanner graph representation, then, each variable node either knows with certainty what its value is, or it does not. Decoding starts when the variable nodes send messages to their adjacent check nodes that indicate whether or not the variable node knows its value. The check nodes examine the messages they received from their adjacent variable nodes. If all the adjacent variables but one knew their value, the check node can determine the value of the remaining node because even parity is required. A round is completed when all the check nodes that can make this calculation send a message to the last variable node, letting it know its value. Check nodes connected to variables that all know their value can be removed from the decoding process. The cycle then repeats. With every round, more variable nodes learn their true values, until all is known or no more progress can be made. When no

more progress can be made, the set of erasures remaining is known as a *stopping set*.

Example 7. Decoding the Hamming Code on the BEC

Figure 5.2 illustrates the decoding of the Hamming code on the BEC.

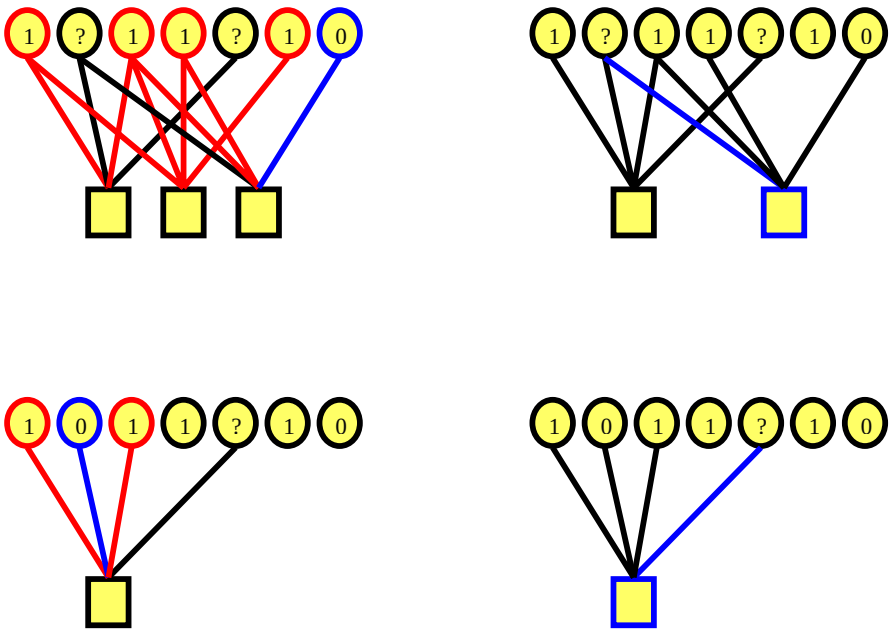


Figure 5.2: Decoding the Hamming code on the BEC starts on the top-left, with each variable node sending its value to the check nodes (red for "one" and blue for "zero"). On the top-right, the third check node sends a message to the second variable node, letting it know that it is a zero. In the bottom row, the process repeats, and every variable learns its identity.

5.2.2 Message Passing on Other Channels

Message passing on more complicated channels is more mathematically challenging, but conceptually the same. Each variable node has some idea of what its value is, those values are sent to the check nodes, and the check nodes send information back to the variable nodes.

Several possibilities exist for what the actual messages should be, and with each set of messages there are different update rules at every variable and check node. The

outgoing message on any edge, e, is a function of the incoming message along all the *other* edges connected to the node, and *not* of the incoming message on e. This is a crucial element of the proofs for the success of message passing decoding and will be discussed further in section 6.3.

Discussing all the possible messages is beyond the scope of this thesis. One idea is to use the likelihood ratio $\lambda = \frac{P(x=0)}{P(x=1)}$. Here, we will use the log-likelihood ratio $\Lambda = \log \lambda$.

In the log-likelihood domain, the update rules are quite simple. The outgoing message on any edge leaving a variable node is the sum of all the incoming messages on the other edges connected to that variable node. At a check node, the outgoing message u is given in terms of the incoming messages on other edges, x and y,by Equation (5.1).

$$u = \text{sign}(xy)[\min(|x|, |y|) - \log(1 + e^{(-||x|-|y||)}) + \log(1 + e^{(-||x|+|y||)})] \tag{5.1}$$

5.3 Iterative Decoders with Input Buffers*

Iterative LDPC decoders can either be designed to perform a fixed number of iterations on each noisy received word, or to stop iterating when a preset stopping condition has been met. In the former case, a fixed amount of time is required to process each word; in the latter, most words are decoded more rapidly and a few take longer.

Noisy words from the channel arrive at the decoder bit by bit. Since the decoder must operate on an entire word, it cannot begin decoding until the entire word has arrived. While one word is being decoded, the bits that compose the next are arriving. These bits must be stored in a buffer large enough to hold at least one noisy codeword.

If the buffer can only hold one such word, the decoder must be able to finish decoding each word before the last bit of the next arrives; the decoder cannot spend more time decoding any one word than the time between word arrivals. If an additional buffer is included, the decoder can spend more time decoding a particularly difficult word, as long as the subsequent words can be decoded fast enough to prevent buffer overflow. This system is shown in Figure 5.3.

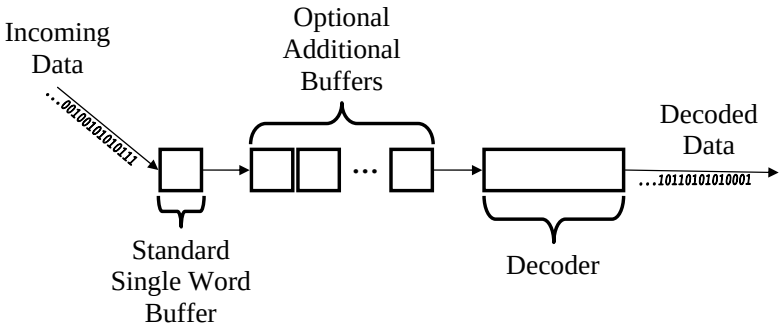


Figure 5.3: An LDPC Decoder with an Input Buffer That Holds One or More Words

When considering the effect of buffering on the decoding system, there are three important parameters: the number of received noisy words that fit in the additional buffer, the amount of time between word arrivals, and the average amount of time needed to decode a word. This average depends not only on the code's parameters, but also on the amount of noise present. Since each iteration of the decoder takes a constant amount of time, an iteration can be seen as the unit of time; we consider the average number of iterations needed to decode a word and the number of iterations between word arrivals.

This idea has been studied extensively in many applications [50, 38, 11, 60] and in many ways our analysis parallels theirs. In [50, 38], Rovini and Martinez propose a buffering system that is slightly different from ours. The difference occurs when a word arrives to find the buffer is full. In their system, the arriving word is discarded or *blocked*. In our system, the word currently being decoded is thrown out. If the additional buffer can hold β words, their system is a D/G/1/β + 1 system with

blocking, where the D stands for deterministic arrival times, the G for a general service distribution, a single decoder, and a system that can hold at most $\beta + 1$ words. Our system is a variant of a *pre-emptive* system. In a pre-emptive system the server (or decoder) knows how much time each customer (or word) will take to complete service and picks customers from the queue based on that information. Our system does not presume to know that much. Rather, it assumes that if the buffer is full, then the word currently being worked on has likely had a fair amount of time in the decoder, and since it has not yet decoded, its likely to never decode. Rather than continue to work on this word, it discards it and begins to work on a fresh word. Though not technically correct, we will call our system *pre-emptive buffer control*.

With a blocking system, one has to be careful to impose some maximum on the total number of iterations allowed (the iteration cap), to avoid becoming stuck on one undecodable word. This is not necessary in pre-emptive buffer control, though it can improve performance slightly. Blocking tends to yield better performance when the incoming data rate is almost the same as the average decoding speed, but this is an unfavorable situation in which we would almost never practically operate.

A mathematical description for the blocking system is given in [50], based on the study of a random walk in Feller's popular probability textbook [22]. A similar description for pre-emptive buffer control based on a random walk is described in [55]. Here, we follow a different mathematical approach in an attempt to gain more insight into the problem. Our approach is based on that of Berlekamp and McEliece [9], with ample help from Kleinrock's text [32].

5.3.1 The Model

We assume that the standard buffer of size one is used to collect incoming bits until an entire word has arrived, and that the entire word then travels instantaneously from the standard buffer to the additional buffer. Additionally, we assume words are

transfered instantaneously from the additional buffer to the decoder when needed. This rapid transfer may not be entirely accurate, but the delay could be included into the model by adding a constant amount to the number of iterations needed to decode [41]. We further assume that the number of iterations needed to decode a word is an independent and identically distributed (i.i.d.) random variable with a known distribution. In practice, consecutive words may have correlated noise, violating this assumption. As noted in [9], this could be avoided by including an interleaver in the decoding system.

Finally, we assume that we are using *pre-emptive buffer control*; when the buffer is full and a new word arrives, the decoder releases the word it was working on and starts decoding a new word. With this method, there are never buffer overflows, errors occur when a word is ejected from the decoder before it has finished decoding.

Our results will again focus on a group of codes from the AR4JA family of protograph codes described in [18], [19], and [13]. This family contains codes of several rates (R) and information block lengths (k). For this analysis, we focus on the $k = 1024$ codes of rate $\frac{1}{2}, \frac{2}{3}$, and $\frac{4}{5}$, and the $R = \frac{1}{2}$, $k = 4096$ code. We also consider the $R = 7/8$, $k = 7154$ block-circulant code C2, which is also described in [13]. Our distributions for the decoding times were obtained from simulations of decoding noisy words obtained from a simulated additive white Gaussian noise (AWGN) channel at various noise levels. These noise levels are characterized by E_b/N_0, the ratio of the bit energy to the noise power spectral density.

5.3.2 Mathematical Analysis

Using the notation in [9], let W_n denote the number of iterations that the n-th word must wait in the additional buffer before decoding of that word begins, and let D_n denote the number of iterations needed to decode it. Assume that the additional buffer holds β words and that a word arrives every a iterations. The decoder and the

additional buffer hold a total of $\beta+1$ words. If it has not already decoded, word n is ejected from the system when word $n+\beta+1$ arrives. Thus, we have an error whenever the combination of the waiting and decoding time for a word exceeds $a(\beta+1)$.

Let $w_n(k) = \text{Prob}(W_n = k)$ and $d(k) = \text{Prob}(D_n = k)$. In most cases, we will only have data for a fixed number of decoder iterations; let M denote that maximum. We are looking for the steady-state behavior of the system, denoted by

$$w(k) = \lim_{n\to\infty} w_n(k).$$

To find this, we note that the waiting times obey a recursion. Given W_n, the next word arrives a iterations later, but has to wait an additional D_n iterations for the nth word to decode before it can begin decoding, unless the n-th word decodes before the $n+1$st word arrives, in which case $W_{n+1} = 0$. Further, after $a\beta$ iterations, if a word has not already entered the decoder, it will be forced in. Thus,

- $W_1 = 0$
- $W_{n+1} = (W_n - a + D_n)_+^{a\beta}$

where $(x)_+^y = \min(\max(x,0), y)$. This leads to a recursion on the $w_n(k)$.

$$w_{n+1}(k) = \begin{cases} \sum_{j=0}^{M} d(j) \sum_{i=0}^{a-j} w_n(i) & \text{if} \quad k = 0 \\ \sum_{j=0}^{M} d(j) w_n(k+a-j) & \text{if} \quad 1 \le k < a\beta \\ \sum_{i=0}^{a\beta} w_n(i) \sum_{j=a\beta-i+a}^{M} d(j) & \text{if} \quad k = a\beta \\ 0 & \text{if} \quad k > a\beta \end{cases}$$

In practice less than a hundred iterations are required to find the limiting distri-

bution $w(k)$. As stated above,

$$\text{Prob(overflow error)} = \text{Prob}(W_n + D_n > a(\beta + 1)).$$

Thus, convolving $w(k)$ with $d(j)$ and summing the probability in the tail yields the probability of errors due to overflow. This does not include the probability that the decoder finished decoding but chose the wrong codeword. Since histograms of the decoding time are usually obtained through simulation of a decoder, and that decoder usually has a maximum number of iterations, the probabilities of error given by this method will only be valid when $a(\beta + 1)$ is strictly less than M, the number of iterations the decoder allowed. In our case, $M = 200$. Further, in our simulation of standard decoding, a word *never* decoded to the wrong word; all errors were due to the decoder reaching the cap of 200 iterations without meeting the stopping rule.

Figures 5.4, 5.5, and 5.6 show an example of the probability distribution functions of the decoding time, waiting time, and total time spent in the system (the system time), respectively, on linear and log scales, obtained from the histogram for the decoding times for the $R = 1/2$, $k = 4096$ AR4JA code at $E_b/N_0 = 1.35$ dB (where the average number of iterations to decode, I_{avg}, is 22.49) with $a = 24$ and an additional buffer of size 2. Notice that, while the distribution of the decoding times in Figure 5.4 has a heavy tail that extends to 200 iterations (and probably beyond), most of the probability is concentrated between 20 and 30 iterations. The distribution of the waiting times in Figure 5.5 shows a large spike at zero, implying that most words are decoded immediately upon their arrival. On the logarithmic scale, a spike at $a\beta$ is visible. This spike occurs because if one word has to wait the maximum amount of time before it can enter the decoder, there is a reasonably large probability that the next word will wait the maximum as well, and so on, until the decoder has caught up with the queue. The total system time distribution shown in Figure 5.6 is the

convolution of the decoding and waiting times distributions. Since the waiting time distribution is dominated by the spike at zero, the system time distribution resembles the decoding time distribution, and it reflects the fluctuations inherent in obtaining the decoding time distributions from simulations.

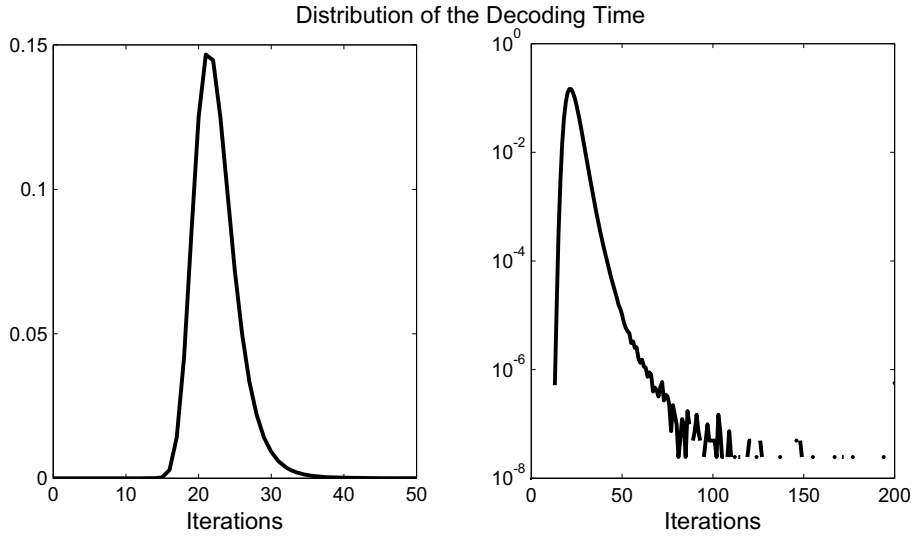


Figure 5.4: Distribution of Decoding Times, AR4JA, $k = 4096$, $R = 1/2$

5.3.3 Latency

In practice, we would have to include a buffer of size β at the output of our decoder to allow for data to move through the decoder at a constant rate. The average latency is thus not of much practical concern. However, the normalized average weight time does give us a way to compare many different situations and draw some conclusions about when the system is likely to perform well.

With any block decoder, there is a certain amount of latency introduced because the decoder cannot start working until the entire block has arrived. Now that we have introduced a buffer, we must not only wait for the entire word to arrive, but that word might also have to wait in the buffer for the decoder to be available. Since any word spends at most $a(\beta + 1)$ iterations in the buffer and decoder, the maximum

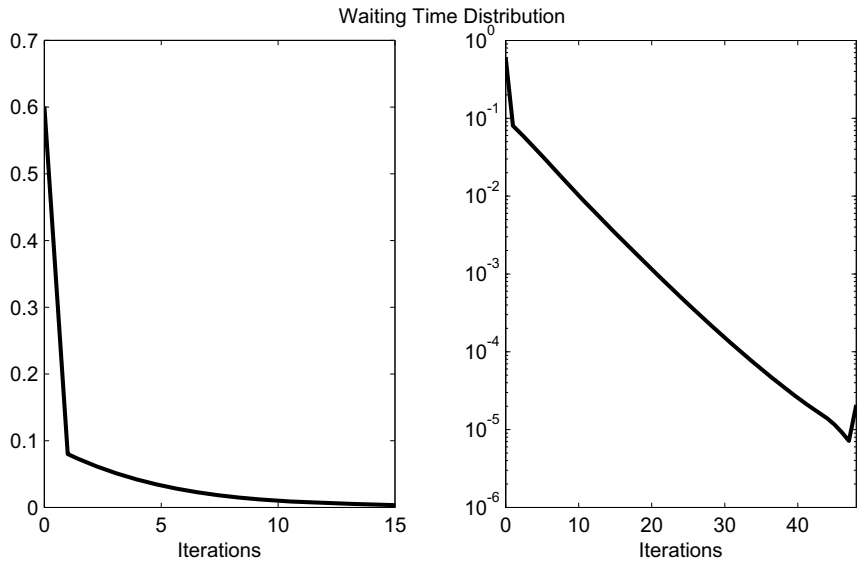


Figure 5.5: Distribution of Waiting Times, AR4JA, $k = 4096$, $R = 1/2$, $a = 24$, $\beta = 2$

possible time from the arrival of the first bit until the word leaves the decoder is $a(\beta + 2)$.

Let $\overline{W}_a(\beta)$ be the average number of iterations a word waits in the additional buffer of size β with incoming words every a iterations. Table 5.1 gives $\overline{W}_a(\beta)$ for several combinations of a and β for the rate 1/2, $k = 4096$ AR4JA code at $E_b/N_0 =$ 1.35 dB ($I_{avg} = 22.49$). The average waiting time is nearly $a\beta$, the maximum, when $a < I_{avg}$, and is very small when $a >> I_{avg}$. The averages for various buffer sizes obey $\overline{W}_a(\beta) < \overline{W}_a(\beta + 1)$, and, as β increases, for all $a > I_{avg}$ the average waiting time approaches a limit, the average wait time in an infinite queue.

Figure 5.7 illustrates these trends. For each code, six representative values of a were chosen. These values of a are the same values that are used in the the series of performance curves in figures 5.9, 5.10, 5.11, 5.12, and 5.13. At every E_b/N_0 value such that the word error rate (WER) with a fixed 200 iterations on each word, which we call WER(200), is less than 0.01, the average waiting time was calculated for additional buffers of size one and five. Decreasing the threshold value further

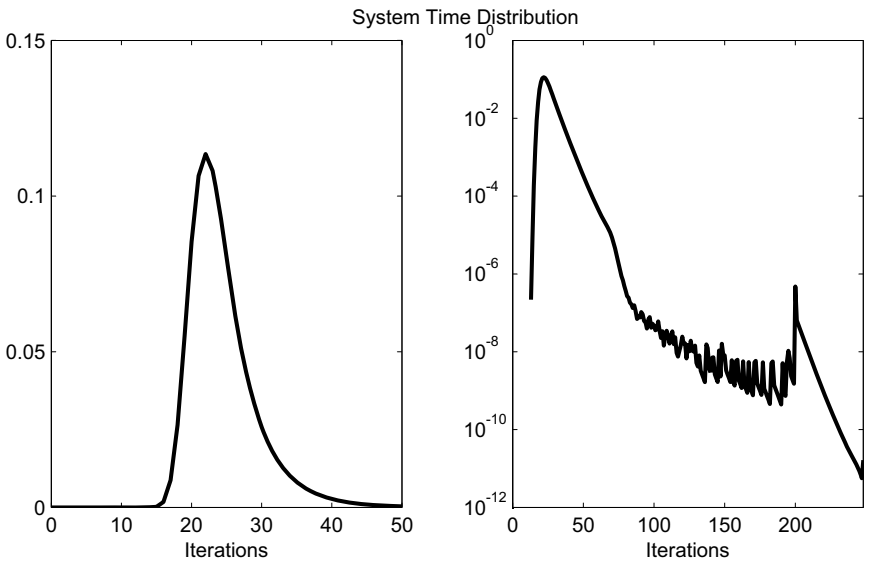


Figure 5.6: Distribution of Service Times, AR4JA, $k = 4096$, $R = 1/2$, $a = 24$, $\beta = 2$

from 0.01 to 0.001 would remove the three outliers seen on right side of the plot. Requiring that WER(200) be below a threshold essentially requires that our decoding distribution, obtained from the histogram, is a reasonable approximation of the true distribution. When WER(200) is high, too much of the tail has been collapsed into a single point at 200 iterations.

The average waiting times were normalized by their maximum value, $a\beta$, and plotted against $\rho = \frac{I_{avg}}{a}$. This value of ρ, commonly called the utilization factor in queuing theory, represents the load on the system. When ρ is small, the decoder runs, on average, much faster than the incoming data rate. When $\rho = 1$, the incoming data rate and the average decoder speed are identical. When $\rho > 1$, the incoming words are arriving faster than they can be decoded, and many are ejected before they can finish decoding. For a buffer of infinite length, we would expect to see a discrete step in the normalized average waiting time from zero to one at $\rho = 1$. This is supported by the increase in the sharpness of the transition between the trends for a buffer of size one and a buffer of size five. It is remakable that across the wide variation in codes and decoding time distributions, the normalized waiting times as a function

of ρ and the size of the additional buffer seem to lie along the same curve. Given only the ratio between the average decoding speed and the incoming data rate, we can determine the normalized average waiting time. The plot shows that if $\rho < 0.5$, i.e., that the incoming data rate is half the average decoding speed, the normalized average waiting time is nearly zero. This implies that most words do not have to wait in the additional buffer, which suggests that most words finish decoding before the next arrives. Based on this fact, we can expect near optimal performance even with an additional buffer that holds only one word.

a	$\overline{W}_a(1)$	$\overline{W}_a(2)$	$\overline{W}_a(3)$	$\overline{W}_a(4)$	$\overline{W}_a(5)$
22	16.1	36.9	58.7	80.7	103
23	5.82	7.57	7.94	8	8.02
24	1.93	1.98	1.98	1.99	1.99
30	0.0638	0.0647	0.0651	0.0656	0.0661
40	0.00264	0.0029	0.00314	0.00343	0.00343
50	0.000499	0.000679	0.000893	0.000893	0.000893

Table 5.1: Average Waiting Times For AR4JA, $R = 1/2$, $k = 4096$, at $E_b/N_0 = 1.35$dB

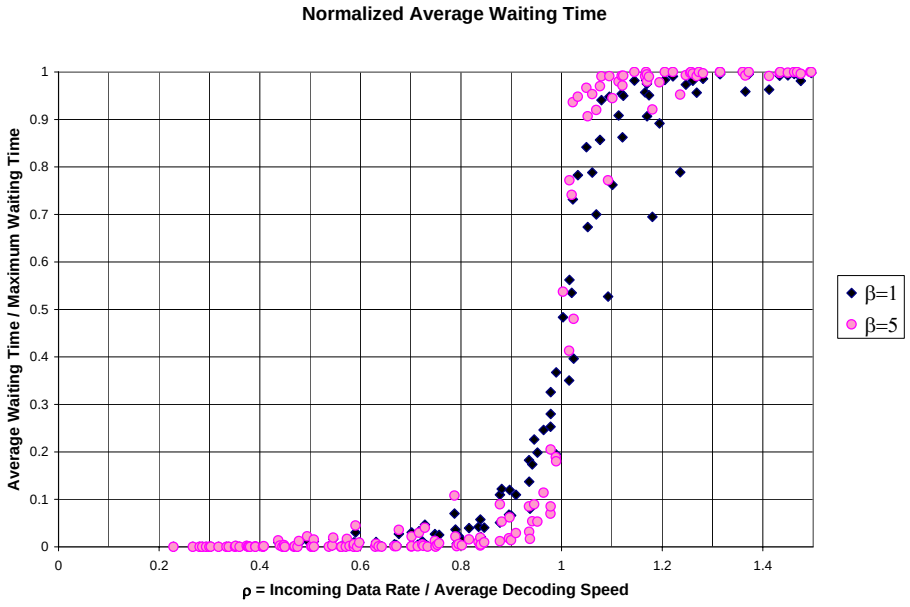


Figure 5.7: Normalized Average Waiting Times as a Function of ρ

5.3.4 Sensitivity to the Quality of the Distribution

Since the distribution for the decoding times is based on simulation data which is difficult to determine accurately at low WER values, it is important to consider what happens when the distribution is altered. Starting with the original distribution from the rate 1/2, $k = 4096$ AR4JA code at $E_b/N_0 = 1.35$ dB, which is denoted D, we consider four other distributions derived from it. The distribution D contains 201 values. The first 200 contain the probability that a word decoded correctly in 0–199 iterations. The final bin contains the probability of word errors; those that did not decode correctly in 199 iterations.

- D is our original distribution.
- D_2 is equivalent to D except for the last item, which was set to zero. The values were then rescaled to ensure that the probability distribution was still valid. This effectively sets the WER to zero, a common situation when the decoder simulations have not been running long enough to have found a word error.
- To create D_3, the probability of a word decoding in 99–199 iterations correctly was set to zero, but the WER was left intact. The distribution was then rescaled.
- To create D_4, the entire tail was removed; the probability of word error and correct decoding in 99–199 iterations was set to zero, and then the distribution was rescaled.
- In D_5, the value representing the WER has been halved, and the distribution rescaled.

Table 5.2 shows the WER obtained for these distributions when the interarrival time was set at 30 iterations. Though the distributions were altered, the average

number of iterations in each case was still 22.49 iterations. Note that the results for D_4 are overly optimistic; some tail is needed for the results to be accurate. Between D_2 and D_3, D_3 is closer to the actual performance, thus, having the correct WER is more important than having the correct tail. Further, if the simulation has not been running long enough to have any word errors, as in D_2, including any WER (even if it is off by a factor of 2, as in D_5) is more accurate than leaving the WER at zero.

Distribution	$\beta = 1$	$\beta = 2$	$\beta = 3$	$\beta = 4$	$\beta = 5$
D	1.32e-005	2.17e-006	1.16e-006	9.23e-007	8.58e-007
D_2	1.24e-005	1.37e-006	3.55e-007	1.21e-007	5.63e-008
D_3	1.23e-005	1.29e-006	8.02e-007	8.02e-007	8.02e-007
D_4	1.15e-005	4.90e-007	5.02e-011	6.58e-013	1.61e-016
D_5	1.28e-005	1.77e-006	7.56e-007	5.22e-007	4.57e-007

Table 5.2: WER for Distorted Distributions

5.3.5 Optimization of the Iteration Cap

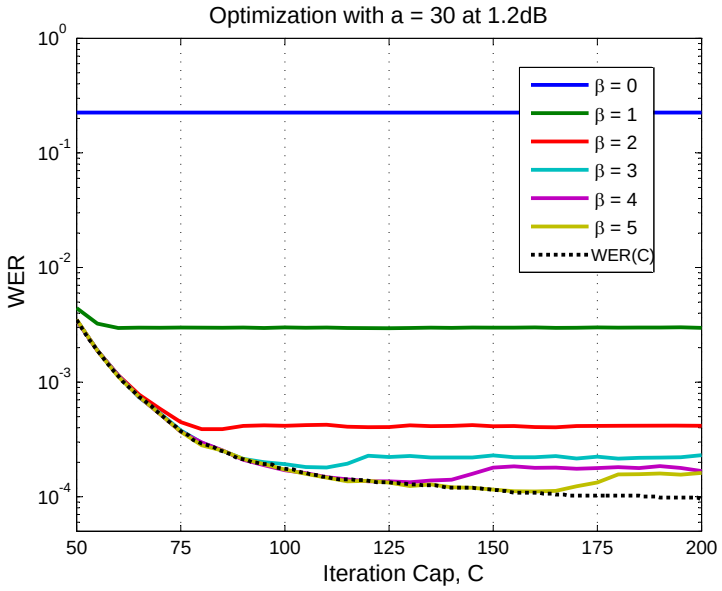


Figure 5.8: Optimization of WER with Respect to the Iteration Cap

If a word arrives while the buffer and the decoder are empty, it can begin decoding immediately. If it has not finished decoding, after β words have arrived, the buffer

will be full. When the $\beta+1$st word arrives, if the original word is still in the decoder, it will be ejected to make room for the new word. Thus, the maximum number of iterations that can be performed on any codeword is $a(\beta+1)$. We can, however, cap the maximum number of iterations lower than that to improve performance.

Figure 5.8 shows the WER for the rate 1/2, $k = 4096$ AR4JA code at $E_b/N_0 =$ 1.2 dB with $a = 30$. The average number of iterations needed to decode at this E_b/N_0 value is 26.8664. For buffers of size zero through 5, the buffer and decoder were simulated for values of the iteration cap between 50 and 200, and 20 million word arrivals were simulated at each point. The dotted line, denoted "WER(C)", is independent of a. It shows the performance of a fixed iteration decoder performing C iterations on each word. Notice that when the iteration cap is small, the WER of the buffered decoder is limited by the performance of the fixed iteration decoder. The constant WER for high values of the iteration cap is a result of the methods iteration cap; whether the maximum is set at 180 or 200 does not matter if the method only allows for 150 iterations.

The minimum WER occurs around halfway between $a\beta$, where the curve begins to deviate from the fixed iteration curve, and $a(\beta+1)$ where it reaches a constant value. Thus, a reasonable approximation for the ideal iteration cap is $a(\beta+\frac{1}{2})$. The intuition for this minimum is as follows; suppose that a difficult word is in the decoder, and words arrive until the buffer fills up and the decoder ejects its current word. The next word will only have a iterations at its disposal. However, there is a greater probability that this word would have decoded between iterations a and $a+\frac{a}{2}$ than that the previous word was going to decode in the last $\frac{a}{2}$ iterations before it was ejected. So, if difficult words are ejected a few iterations sooner, what we lose in WER for the difficult words is far less than what we gain by letting the next word (which is likely to be less difficult) have a few more iterations available.

In practice, however, setting the iteration cap perfectly only improves the WER

by a small amount. For the sake of simplicity, all our results presented here were produced with an iteration cap equal to the minimum of 200 and the iteration cap imposed by the method.

5.3.6 Performance Curves

Figures 5.9, 5.10, 5.11, 5.12, and 5.13 are performance curves for the $R = \frac{1}{2}$, $k = 4096$ AR4JA code, whose average decoding times are given in Table 5.3, the $k = 1024$ family of AR4JA codes, whose average decoding times are given in Table 5.4, and C2, whose average decoding times are given in Table 5.5. Each Figure contains several plots. Each plot is for a constant interarrival time and shows the WER curve found with buffers of size zero through five. The WER obtained from allowing 200 iterations on every word, which is equivalent to setting $a = 200$ and $\beta = 0$, is also shown and again labeled WER(200). For convenience, a graph of of the average number of iterations to decode at each E_b/N_0 value accompanies each sequence of plots.

For the rate 1/2 codes, the curves are obtained through simulation. At each point, the number of words simulated varies between 10 million and 200 million. The simulations were stopped between these values when at least 100 errors had been found. Note that these 100 errors are not independent events, as one difficult word can reduce the number of iterations available for several words following it. The rest of the curves were obtained from the mathematical analysis. The mathematical analysis could not be used for the rate 1/2 codes because with relevant interarrival times and buffers of size four or five, the iteration cap imposed by the method is greater than 200, and the distribution for the decoding times does not include the possibility of more than 200 iterations.

Of particular interest are the sets of images that show what happens when the interarrival time is increased from below the average number of iterations needed to above it. Even if the additional buffer only holds two or three words, the transition

is sharp; when the interarrival time is less than the average number of iterations to decode, the WER is unacceptably high, but when the interarrival time is increased to just a few iterations above the average decoding time, the WER quickly drops to near optimal levels. Further, note that as predicted by the analysis of the waiting times, the word error rate of the buffered decoder, even with an additional buffer that holds only one word, is nearly optimal when the number of iterations between word arrivals is greater than twice the average number of iterations to decode.

Also, note that the distance between the curves for buffers of size zero and one is much larger than the distance between the 200-iteration WER and the buffer of size one curve. Including a single buffer decreases the word error rate by far more than increasing the size of the buffer from one to two.

AR4JA, $k = 4096$ Rate 1/2	
E_b/N_0 (dB)	I_{avg}
1.05	35.0939
1.1	31.5612
1.15	28.9237
1.2	26.8664
1.25	25.1865
1.3	23.7456
1.35	22.4932

Table 5.3: Average Number of Iterations to Decode $k = 4096$ AR4JA

5.3.7 Conclusion

Using the performance of an iterative LDPC decoder performing 200 iterations on each word as a benchmark, we have shown that adding a buffer large enough to hold only a few noisy received words allows the system to operate much faster while achieving the same word error rate. Over a wide range of codes and noise levels, the system can handle incoming data rates almost as high as the average decoding speed.

AR4JA, $k = 1024$					
Rate 1/2		Rate 2/3		Rate 4/5	
E_b/N_0 (dB)	I_{avg}	E_b/N_0 (dB)	I_{avg}	E_b/N_0 (dB)	I_{avg}
1.4	23.6116	2	34.9007	3	27.7324
1.5	20.2983	2.2	20.0406	3.2	16.1086
1.6	17.9357	2.4	14.3357	3.4	10.9234
1.7	16.3232	2.6	11.7384	3.6	8.8097
1.8	15.0695	2.8	10.1553	3.8	7.5657
1.9	14.0370	3	8.9971	4	6.6795
2.2	11.7419			4.3	5.7128

Table 5.4: Average Number of Iterations to Decode $k = 1024$ AR4JA

C2, $k = 7154$ Rate 7/8	
E_b/N_0 (dB)	I_{avg}
3.6	17.3389
3.7	9.8870
3.8	7.4820
3.9	6.3648
4.0	5.6138

Table 5.5: Average Number of Iterations to Decode C2

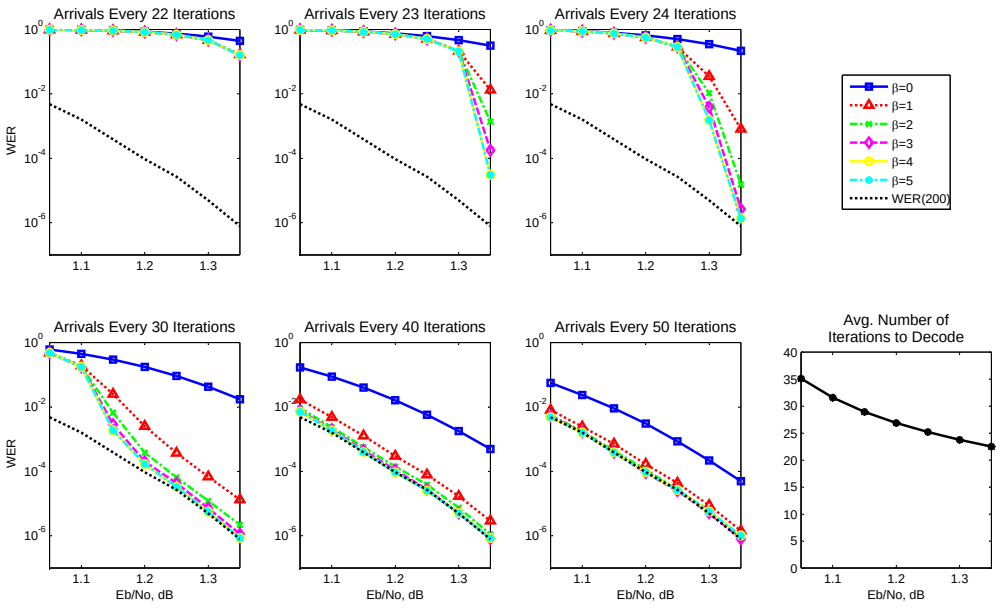


Figure 5.9: Performance for a Variety of Interarrival Times: AR4JA, rate 1/2, $k = 4096$

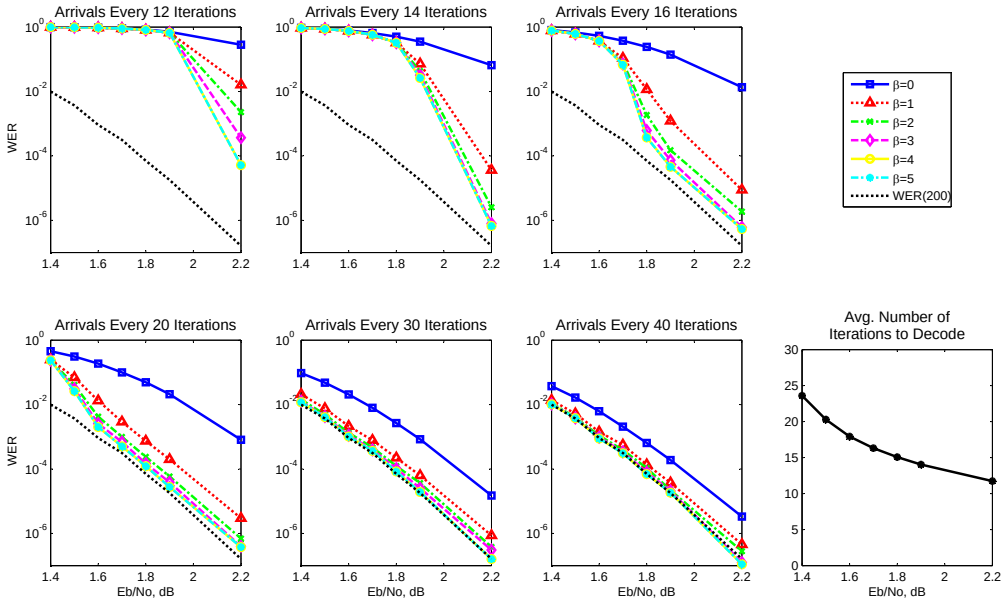


Figure 5.10: Performance for a Variety of Interarrival Times: AR4JA, rate 1/2, $k = 1024$

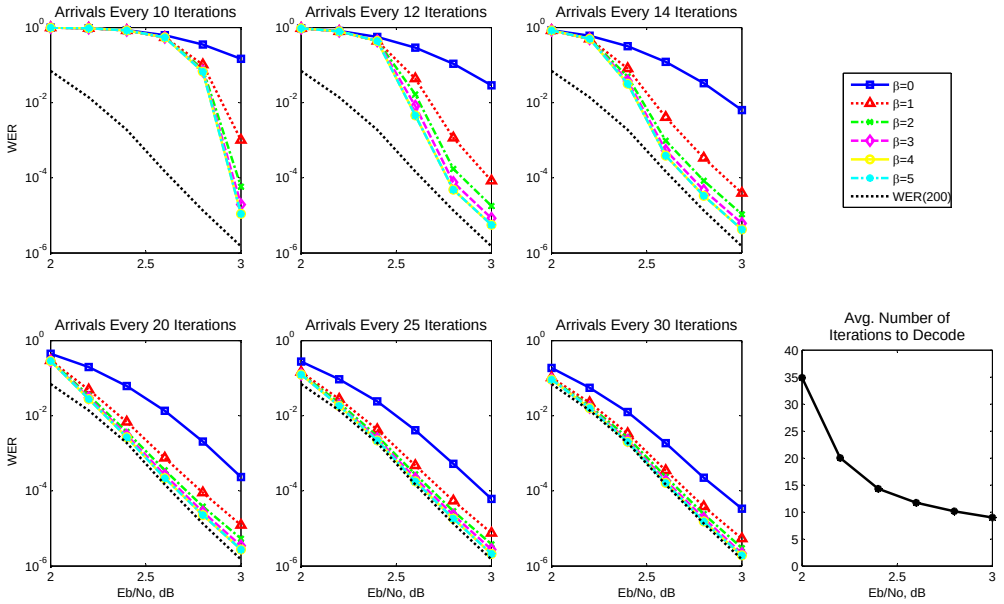


Figure 5.11: Performance for a Variety of Interarrival Times: AR4JA, rate 2/3, $k = 1024$

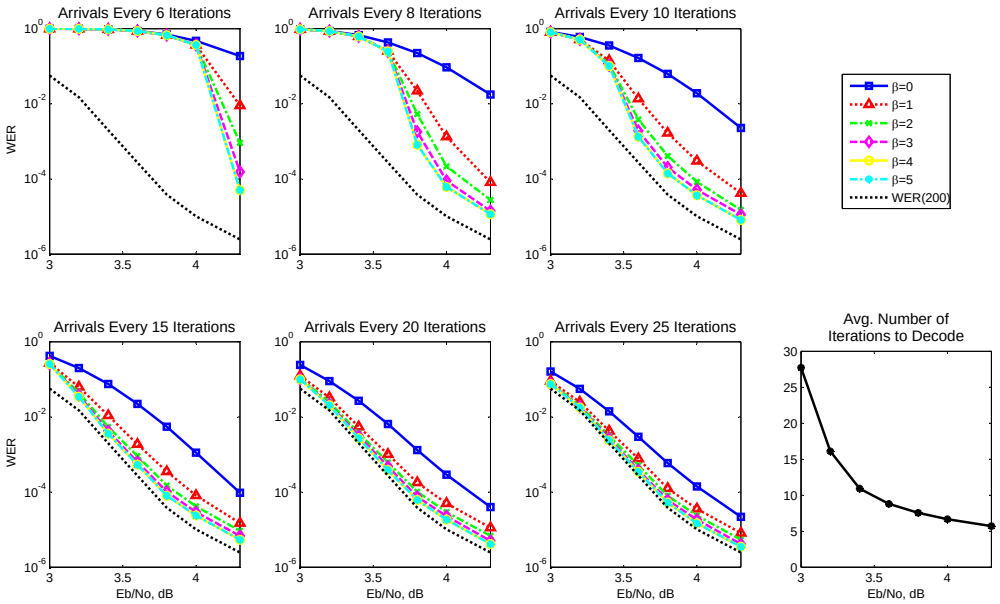


Figure 5.12: Performance for a Variety of Interarrival Times: AR4JA, rate 4/5, $k = 1024$

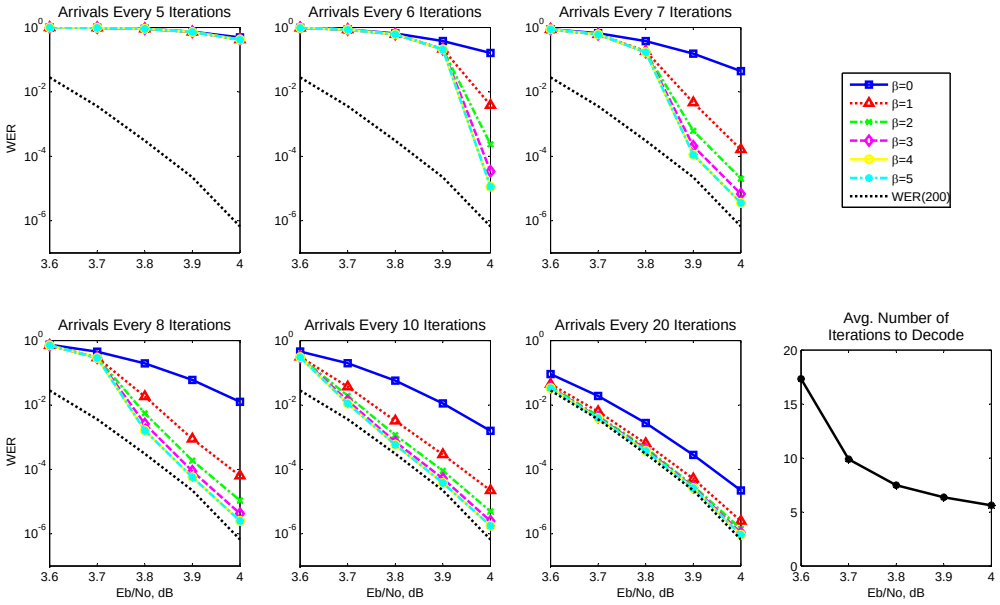


Figure 5.13: Performance for a Variety of Interarrival Times: C2, rate 7/8, $k = 7154$

The ratio between them, ρ, is a crucial parameter. The word error rates are high when $\rho \geq 1$, but as ρ is decreased from 1, they rapidly approach the benchmark of WER(200). When $\rho = 2/3$, the E_b/N_0 loss with respect to the benchmark is on the order of tenths of a dB. For $\rho < 1/2$, there is virtually no loss.

Chapter 6

How to Find a Good Protograph

In this chapter, we will discuss ways to determine which protograph ensembles are likely to contain codes that will be successful. In the first section, the definition of a "good" code will be introduced and we will discuss how we can use the spectral shape to determine which ensembles contain good codes. We will also use the spectral shape to calculate some performance bounds. The second section deals with density evolution, and the use of simulated annealing to find good codes with high density evolution thresholds on the binary erasure channel (BEC). In the final section, we will discuss minimum cycle size.

6.1 Applications of the Spectral Shape*

An ensemble of codes is said to be "good" if the probability of error can be made arbitrarily small for some nonzero rates [36]. All code ensembles that have a minimum distance that grows linearly with the code's length are "good" [37]. One way to determine this is to find the zero crossing of the spectral shape. If $E(\theta)$ represents the spectral shape, the number of words of weight θn is $e^{nE(\theta)+o(n)}$. If the spectral shape of a code is negative for small values of θ, then crosses zero at θ_c and becomes positive for larger values of θ, then as n grows, the number of words with weight less than $n\theta_c$ will decrease exponentially. Thus, there will be codes in the ensemble for any

given value of n such that the minimum weight is approximately equal to $\theta_c n$ [15]. For instance, the $(3, 6)$ regular ensemble contains good codes because the spectral shape crosses the axis at $\theta_c = 0.0227$. For very large n, we expect the distance of the code to be roughly $0.0227n$.

Section 3.4.2 contains plots of several rate 1/2 protograph ensembles. In Table 6.1, approximate values of θ_c are tabulated for those ensembles. Also tabulated are the approximate values of θ_s, the value of θ at which the growth rate of the stopping set enumerator crosses zero. The tabulated values are found by a bisection method that stops when the objective function has an absolute value that is less that 10^{-5}, so they are not the exact zero crossings, but a reasonable approximation of them. An entry of "-" in the Table is used for ensemble whose spectral shapes are never negative.

Protomatrix	θ_c	θ_s
$\begin{pmatrix} 2 & 2 \end{pmatrix}$	-	-
$\begin{pmatrix} 2 & 3 \end{pmatrix}$	-	-
$\begin{pmatrix} 1 & 3 \end{pmatrix}$	-	-
$\begin{pmatrix} 2 & 4 \end{pmatrix}$	-	-
$\begin{pmatrix} 3 & 3 \end{pmatrix}$	0.0227	0.0180
$\begin{pmatrix} 3 & 4 \end{pmatrix}$	0.0428	0.0322
$\begin{pmatrix} 4 & 4 \end{pmatrix}$	0.0627	0.0453
$\begin{pmatrix} 3 & 5 \end{pmatrix}$	0.0569	0.0416
$\begin{pmatrix} 4 & 5 \end{pmatrix}$	0.0746	0.0526

Table 6.1: Zero Crossings of the Spectral Shape and the Growth Rate of the Stopping Set Enumerator for Several Protograph Ensembles

In [16], Divsalar derives a simple, tight bound on the minimum signal-to-noise ratio at which a code can reliably transmit information on the Gaussian channel. This bound is given in Equation (6.1).

$$\left(\frac{E_b}{N_0}\right)_{\min} = \frac{1}{R} \max_{0 \le \theta \le (1-R)} \left\{ \left(1 - e^{-2E(\theta)}\right) \frac{1-\theta}{2\theta} \right\} \tag{6.1}$$

Using the enumerators shown in section 3.4.2, we can calculate the value of this bound for each protograph ensemble. The minimum possible signal-to-noise ratio is given by the rate of the code, so it is fair to compare codes of the same rate. Table 6.2 contains a list of several bounds on the threshold values, given in decibels [1]. The AR4JA code has the best (i.e., lowest) threshold of the rate 1/2 codes. In general, the Table shows that for simple codes the threshold decreases as the maximum check node degree increases.

Protomatrix	Bound on Threshold (dB)
$\begin{pmatrix} 2 & 2 \end{pmatrix}$	3.3644
$\begin{pmatrix} 2 & 3 \end{pmatrix}$	1.5031
$\begin{pmatrix} 1 & 3 \end{pmatrix}$	5.4647
$\begin{pmatrix} 2 & 4 \end{pmatrix}$	0.8727
$\begin{pmatrix} 3 & 3 \end{pmatrix}$	0.7938
$\begin{pmatrix} 3 & 4 \end{pmatrix}$	0.5417
$\begin{pmatrix} 4 & 4 \end{pmatrix}$	0.4262
$\begin{pmatrix} 3 & 5 \end{pmatrix}$	0.4299
$\begin{pmatrix} 4 & 5 \end{pmatrix}$	0.3665
AR4JA	0.3598
$R = 1/2$ Shannon Ensemble	0.3081
$\begin{pmatrix} 3 & 0 & 3 \\ 0 & 3 & 3 \end{pmatrix}$	-0.3885
$\begin{pmatrix} 3 & 0 & 3 \\ 0 & 3 & 4 \end{pmatrix}$	-0.4101
$R = 1/3$ Shannon Ensemble	-0.4533

Table 6.2: Bounds on the Minimum E_b/N_0 Value at which decoding will succeed based on the spectral shape

In this section, we compared the zero crossings of the spectral shape, the growth rate of the stopping set enumerator, and a bound on AWGN performance for several simple protograph codes. In each case, better codes were found by increasing the number of edges in the protograph.

[1] A number in decibels can be converted to the actual number according to the formula x dB $= 10\log_{10}(x)$.

6.2 Density Evolution on the BEC

On the binary erasure channel (BEC) each symbol is erased with probability p. An (n, k) linear code with rate $R = k/n$ will never be able to decode if $p > 1 - R$, as $1 - R$ is the fraction of the data that is redundant. Accepting that p must be less than $1 - R$, we use a tool called *density evolution* to calculate the maximum value of p such that decoding is likely to succeed. This allows us to compare codes of the same rate, as if one code can decode at 90% of $1 - R$ and the other cannot, it would be fair to say that the former code is better.

Density evolution looks at the evolution of the probability densities as the decoder performs successive iterations. Let us assume that we are dealing with a large code in a regular (j, k) ensemble. The assumption that the code is large is necessary because this analysis assumes that the local structure of the code is tree-like. Initially, each variable has probability p of being erased. Each variable sends this message to all its adjoining check nodes. The check node can determine the value of an adjoining node if all the other nodes are not erased, which occurs with probability $(1 - p)^{k-1}$. Thus, the probability that a check node of degree k tells a variable node that its value is unknown is $1 - (1 - p)^{k-1}$. The outgoing message from a variable node of degree j will be an erasure if all of its adjoining check nodes and the channel do not know its value. This will happen with probability $p_1 = p(1 - (1 - p)^{k-1})^{j-1}$. Thus, after one iteration, the probability that a variable node is still erased is p_1. After two iterations, the probability will be $p_2 = p(1 - (1 - p_1)^{k-1})^{j-1}$. After three iterations, the probability will be $p_3 = p(1 - (1 - p_2)^{k-1})^{j-1}$ and so on. If this probability tends to zero, then decoding will succeed. The largest value of p such that $p_n \to 0$ is the code's threshold on the BEC. Table 6.3 lists the threshold values for several rate 1/2 regular codes.

The same reasoning holds for protographs, although the execution becomes more

Ensemble	Threshold p
(2,4)	0.3336
(3,6)	0.4294
(4,8)	0.3834
(5,10)	0.3415

Table 6.3: Density Evolution Thresholds for Rate 1/2 Regular Codes

complicated as the degrees of each node are no longer constant. Using the same process, however, density evolution thresholds for given protographs can be calculated quite simply. For example, the AR4A protograph has a density evolution threshold on the BEC of 0.4418; the AR4JA protograph's threshold is 0.4387. These and other protograph thresholds are given in Table 6.4

Protomatrix	Threshold
Rate 1/2 Code	0.5
$\begin{pmatrix} 2 & 4 \end{pmatrix}$	0.4450
$\begin{pmatrix} 3 & 3 \end{pmatrix}$	0.4294
$\begin{pmatrix} 3 & 4 \end{pmatrix}$	0.4094
AR4JA	0.4387
AR4A	0.4418
Rate 1/3 Code	0.667
$\begin{pmatrix} 3 & 0 & 3 \\ 0 & 3 & 3 \end{pmatrix}$	0.5206
$\begin{pmatrix} 3 & 0 & 3 \\ 0 & 3 & 4 \end{pmatrix}$	0.5070

Table 6.4: Maximum Erasure Probability p at which Decoding Can Succeed

6.2.1 Simulated Annealing*

Once calculation of the threshold becomes possible, it is natural to ask whether we can optimize protographs using this information. One way of accomplishing this goal is through simulated annealing, as discussed in [59] and [37]. A starting protograph with the desired number of check and variable nodes is selected. Edges in this protograph are added, removed, or moved according to some probability distribution. These

changes are accepted if they increase the threshold. If they decrease the threshold, they are accepted with some small probability that decreases with how long the algorithm has been running.

This is not the only optimization method known to find codes with high thresholds on the BEC. Other possibilities include linear programming [35] and differential evolution [54, 52].

Other constraints are imposed on the system. The size of the initial protograph is the size of the final protograph, and smaller protographs have fewer degrees of freedom. We expect to get different results for a 4x8 protograph than a 2x4, even though they have the same rate. We also restrict the maximum check node degree. High check node degrees, while potentially beneficial to theoretical decoding performance, increase the decoding time dramatically. Every iteration, the decoder updates first the variable nodes, and then the check nodes. The update rules are simple at the variable nodes, and complicated at the check nodes, as described in chapter 5. The maximum degree of the check node, then, primarily determines how much time an iteration takes. Limiting the check node degree thus limits decoding times.

Starting with a protograph with four transmitted variables, one punctured variable, and three check nodes (like the AR4A and AR4JA codes), simulated annealing found a protograph with a density evolution threshold of 0.4834, slightly higher than the AR4A and AR4JA codes.

$$M = \begin{bmatrix} 2 & 1 & 1 & 2 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 1 \end{bmatrix}$$

The second variable node is the punctured node. Unfortunately, the spectral shape for this protograph is positive at 0.0025, making it unlikely that this is a "good" code. Repeating the annealing process several times yields other codes with high thresholds,

but none which are "good" codes.

6.2.2 Simulated Annealing with Constraints*

To find good codes that also have high density evolution thresholds, we must adjust the simulated annealing algorithm by choosing a small value, like $\theta = 0.005$. With every change in the annealing process, the value of the spectral shape at θ is calculated. If this value is negative, the change is accepted or rejected in the normal fashion. If, however, the value of the spectral shape is positive at that point, the change is always rejected.

Using the same parameters as the AR4A and AR4JA codes, i.e., a 3x5 protomatrix with the second variable punctured, simulated annealing found an ensemble defined by the protomatrix

$$H = \begin{pmatrix} 1 & 1 & 2 & 0 & 1 \\ 0 & 2 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \end{pmatrix},$$

which has density evolution threshold on the BEC of 0.4443. The spectral shape for this ensemble appears to remain negative for some time, crossing the x-axis somewhere between $\theta = 0.008$ and $\theta = 0.009$.

Calculating the zero crossing of the spectral shape is difficult, and could not be done on every simulated annealing run. As an example, however, it was attempted for the annealing of an unconstrained 2x3 protomatrix. Figure 6.1 is a scatter plot of the results for eighty-two 2x3 protomatrices. With such a small protograph, the calculations are quite simple, allowing us to explore a large region of the space.

While this is only a small collection of the possible protomatrices, a trend is noticeable. The density evolution threshold on the BEC seems to be negatively correlated with the zero crossing of the spectral shape. This data suggests that one

cannot have high threshold and a large minimum distance. This same relationship was observed for the irregular ensembles in [14]. Table 6.5 lists a few such values for the 2x3 protomatrices. Protomatrices with many edges are more likely to have high zero crossings and low thresholds. Protomatrices with few edges have high thresholds but low zero crossings.

Protomatrix	Density Evolution Threshold	θ_c
$\begin{pmatrix} 6 & 3 & 1 \\ 1 & 5 & 3 \end{pmatrix}$	0.3968	0.1009
$\begin{pmatrix} 6 & 3 & 1 \\ 2 & 1 & 4 \end{pmatrix}$	0.3932	0.1010
$\begin{pmatrix} 4 & 4 & 1 \\ 1 & 3 & 5 \end{pmatrix}$	0.4052	0.1013
$\begin{pmatrix} 4 & 4 & 1 \\ 2 & 2 & 5 \end{pmatrix}$	0.4034	0.1014
$\begin{pmatrix} 2 & 2 & 1 \\ 2 & 0 & 2 \end{pmatrix}$	0.5962	0.005
$\begin{pmatrix} 1 & 2 & 1 \\ 3 & 0 & 2 \end{pmatrix}$	0.5855	0.005
$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 0 & 3 \end{pmatrix}$	0.5851	0.0051
$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \end{pmatrix}$	0.6063	0.008

Table 6.5: Protomatrices, Density Evolution Thresholds, and Zero Crossings of the Spectral Shape

For the specific goal of improving performance on the BEC, one might argue that it would be better to plot not the zero crossing of the spectral shape (θ_c), but the zero crossing of the growth rate of the stopping set enumerator (θ_s). Recall that a stopping set is any set of variable nodes that, if erased, cause an erasure decoder to stop. The growth rate of the stopping set enumerator is defined analogously to the spectral shape, i.e., if S_j represents the number of stopping sets of size j, the growth rate of the stopping set enumerator is given by $\overline{\lim}_{n\to\infty} \frac{1}{n} \log S_j$. Figure 6.2 is a scatter plot of the results for a 4x8 protomatrix. Unfortunately, for a protomatrix this large

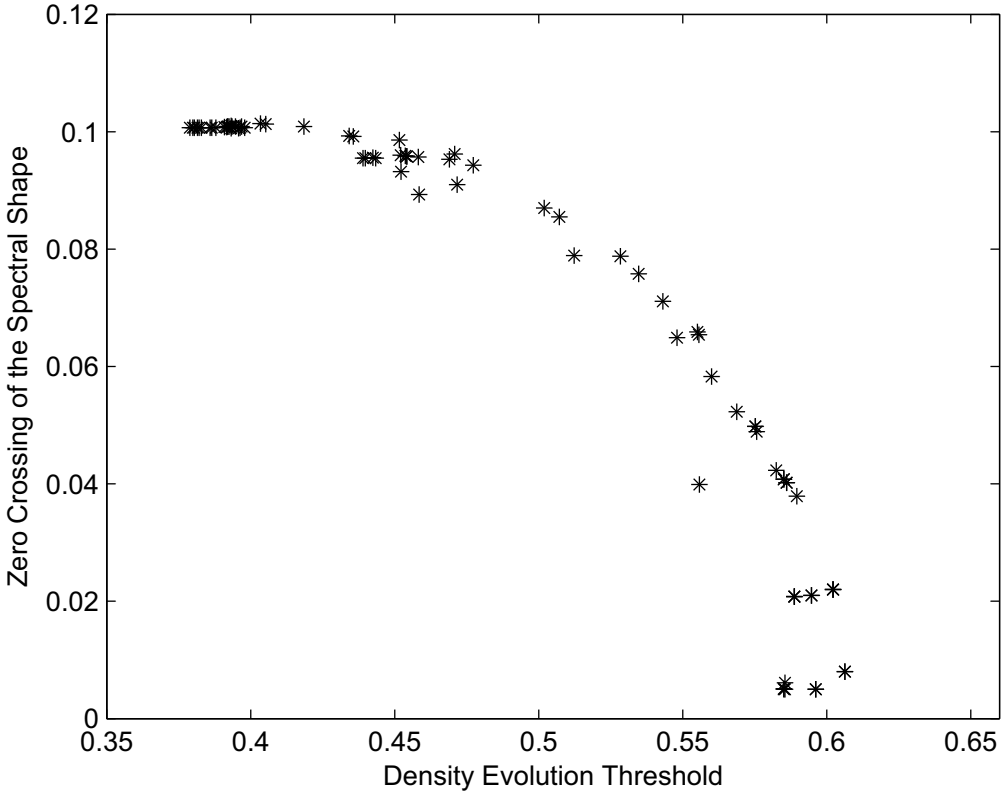


Figure 6.1: Simulated Annealing on a 2x3 Protograph

calculation times were long and only a small region of the space could be explored. The correlation between θ_s and the threshold does not seem as striking here only because the axes are so much smaller. Since all codewords are stopping sets, the zero crossings of the spectral shape and the growth rate of the stopping set enumerator are strongly correlated with each other, and so we see similar results when comparing them to density evolution thresholds.

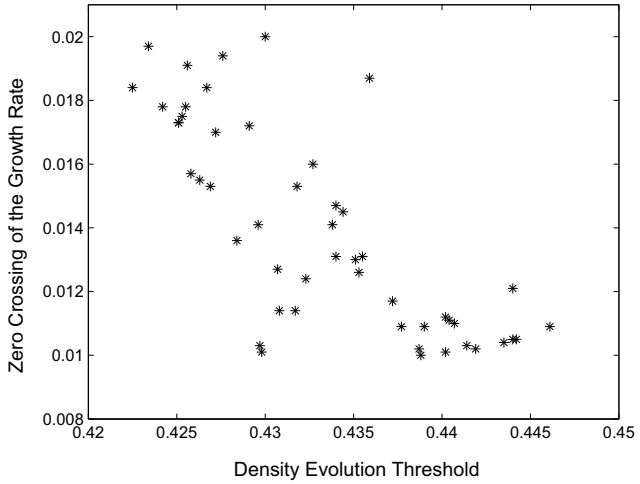


Figure 6.2: Simulated Annealing on a 4x8 Protomatrix

6.3 Cycles and Girth

Much effort has gone put into finding the minimum cycle size, or *girth* of codes, and to design codes with high girth [61],[40],[43],[24]. Existing proofs of the success of message passage decoding rely on the messages remaining independent, which requires a tree-like structure (i.e., no cycles) [49]. Since hundreds of iterations may be performed, we are interested in determining when this assumption is no longer valid.

In [26], Gallager calculates an upper bound on the girth of codes in a regular ensemble. Starting with any node, he considers the tree created by following its

edges to its neighbors, and their neighbors, and so on. The number of nodes in this tree grows exponentially. For any code's length n, there exists some point where the number of nodes in the tree exceeds n. At this point, some node is in the tree twice, and thus a cycle has been formed. In this section, we will review that technique and extend it to protograph codes.

6.3.1 Gallager's Method for Regular Codes

Assume that we have a regular (j, k) ensemble. Then if we start with any variable node, there are $j(k-1)$ variables nodes in the next layer, i.e., it is connected to j check nodes, each of which is connected to $k-1$ other variable nodes. So, counting our first node, we have touched $1+j(k-1)$ nodes. Each of the new nodes is connected to $(j-1)$ new check nodes, connected to $(k-1)$ additional variable nodes, etc. If we use L to denote how many layers there are in the tree, the last layer contains $j(k-1)^{L-1}(j-1)^{L-2}$ variable nodes. So, in a tree with L layers, we will touch

$$T_v(L) = 1 + \sum_{i=2}^{L-1} j(k-1)^{i-1}(j-1)^{i-2} \text{ nodes.}$$

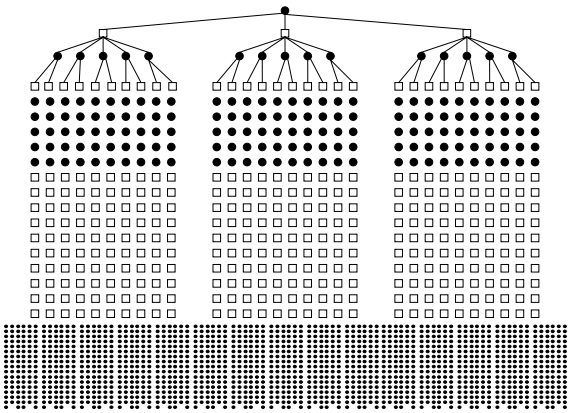


Figure 6.3: Tree Structure

For some value of L denoted L^*, $T_v(L^*) > n$. At his point, we will have touched

some node more than once, creating a cycle. Figure 6.3 shows this tree for the (3,6) ensemble. The first variable node is connected to 3 check nodes, each of which is connected to 5 other variable nodes, for a total of 15 variable nodes at this level. Each of these variable nodes is connected to two other check nodes, which are connected to five more check nodes, for a total of 150 variable nodes in the third level. At this point, it becomes difficult to actually draw the tree structure. but this illustrates the main point: the tree gets very large very fast.

Calculating $T_v(L)$ gives us the number of variable nodes in the tree. By also considering the check nodes, we can further clarify a bound on the cycle size. We denote by $T_c(L)$ the number of check nodes connected to the first L rows of variable nodes. Each variable is connected to $j-1$ new check nodes. Table 6.6 shows the number of variables in each level, and the number of checks they are connected to. Their cumulative sums yield the values of $T_v(L)$ and $T_c(L)$, which are also shown. Using this Table, we can calculate an upper bound on the girth of codes in the $(3, 6)$ ensemble, shown in Table 6.7.

L	Number of Variable Nodes	Number of Check Nodes	$T_v(L)$	$T_c(L)$
1	1	3	1	3
2	15	30	16	33
3	150	300	166	333
4	1,500	3,000	1,666	3,333
5	15,000	30,000	16,666	33,333

Table 6.6: Number of Nodes in Each Level for the (3,6) Ensemble

6.3.2 Protographs Codes — An Example*

With regular codes, we were only able to consider the total number of variable and check nodes. With protographs, we can consider each class of variable and check node seperately. For instance, let us consider the protograph codes from the ensemble defined by the protomatrix $H = [3, 4]$. In this ensemble, there are two types of

n	Upper Bound on Girth
$n < 16$	4
$16 \leq n < 66$	6
$66 \leq n < 166$	8
$166 \leq n < 666$	10
$666 \leq n < 1,666$	12
$1,666 \leq n < 6,666$	14
$6,666 \leq n < 16,666$	16

Table 6.7: Bounds on the Minimum Cycle Size in the (3,6) Ensemble

variable nodes. The first type has degree three, the second degree four. Every check node is connected to three of the first type and four of the second.

Let us call the variables with degree 3 "type one" and the variables with degree 4 "type two". Let us start with a type one node. This node is connected to 3 checks, which are each connected to 2 other type ones and 4 type twos. So, the original type one node is connected to 6 type ones and 12 type twos. That is the first iterations. Each of these type 1s is connected to 2 other checks, each of these checks is connected to 2 other type ones and 4 type twos. So in subsequent iterations, each type one is connected to 4 type ones and 8 type twos.

If we start with a type two node, in the first iteration it is connected to 12 type ones and 12 type twos. In subsequent iterations, a type two node is connected to 9 type ones and 9 type twos.

For $i > 2$, if x_i represents the number of type ones at step i, and y_i the number of type twos, then $x_{i+1} = 4x_i + 9y_i$ and $y_{i+1} = 8x_i + 9y_i$. The values of $T(L)$ must now be separated not only between variables and check nodes, but also between types of variable nodes. We differentiate this using extra subscripts. The first is either a c or a v, depending on whether we are counting check nodes or variable nodes, as before. The second subscript is the type of node. Using this notation, $T_{v_1}(L) = \sum_{i=1}^{L} x_i$ and $T_{v_2}(L) = \sum_{i=1}^{L} y_i$.

Next, the check nodes must be considered. If we start with a type 1 variable node,

it is connected to 3 check nodes. A type 2 variable node is connected to 4. Assuming $i > 1$, if there are x_i type ones and y_i type twos in level i of the variable nodes, then the number of check nodes connected to them is $2x_i + 3y_i$, as each variable is already connected to one check node in the previous level.

Table 6.8 details T_{v_1}, T_{v_2}, and T_c at various levels, starting at both a type one and a type two variable node. These values are used to generate Table 6.9, which gives the bounds on the minimum cycle size for this protograph ensemble based on the number of copies of the protograph $C = n/2$. For example, a length 1600 code has $C = 800$. According to Table 6.8, there are plenty of variables in level three, but not enough check nodes. Two of the check nodes in the third level must be the same, creating a cycle of size ten.

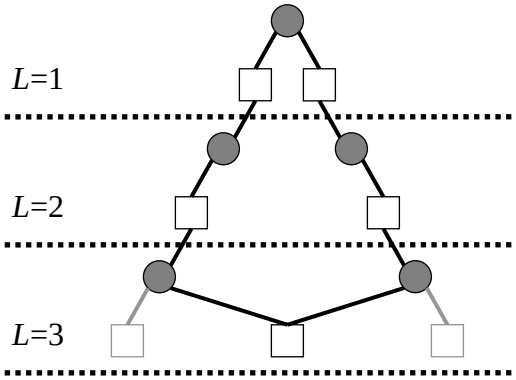


Figure 6.4: A cycle of size ten is formed when there are not enough check nodes in the third level.

	Starting From Type One			Starting From Type Two		
L	$T_{v_1}(L)$	$T_{v_2}(L)$	$T_c(L)$	$T_{v_1}(L)$	$T_{v_2}(L)$	$T_c(L)$
1	1	0	3	0	1	4
2	7	12	51	12	13	64
3	139	168	783	168	217	988
4	2,071	2,628	12,027	2,628	3,301	15,160
5	31,939	40,224	184,554	40,224	50,737	233,660

Table 6.8: $T_v(L)$ and $T_c(L)$ for $[3, 4]$ Protograph

C	Maximum Girth
$4 \leq C < 14$	4
$14 \leq C < 64$	6
$64 \leq C < 218$	8
$218 \leq C < 988$	10
$988 \leq C < 3302$	12
$3302 \leq C < 15,160$	14
$15,160 \leq C < 50,738$	16

Table 6.9: Bounds on Minimum Cycle

6.3.3 Protograph Codes in General*

The example in the previous section was simple because there was only one kind of check node. When calculating the number of variable nodes or check nodes in each layer of the tree, one must keep track not only of the number of nodes in the previous layer, but where they came from. At every layer, we calculate the number of variables of type j that came from check nodes of type i. We then calculate the number of check nodes of type i that come from these variables of type j. We must do this starting from each type of variable node to get the best results.

We have used Matlab to calculate the number of nodes of each type at each level, and thus $T_{v_j}(L)$ and $T_{c_i}(L)$ for all variables of type j and check nodes of type i for any protograph ensemble. In order for there to be no cycles created, the lifting of the protograph has to be larger than the maximum of all the $T_{v_j}(L)$ and $T_{c_j}(L)$.

For the AR4A and AR4JA ensemble, considering only the maximum T_v and the maximum T_c, we arrive at the results summarized in Table 6.10. For instance, for either code, if we only lift by 20, for a total code's length of 80 transmitted variables, then in the second level, two check nodes must be the same, creating a cycle of size six. Similarly, a code of size 16,000 (i.e., lifted by 4000) will have a cycle of size fourteen.

	AR4A		AR4JA	
Level	$\max T_v$	$\max T_c$	$\max T_v$	$\max T_c$
1	1	3	1	3
2	9	30	9	28
3	103	357	93	343
4	1279	4488	1195	4016
5	15989	55851	13961	47383

Table 6.10: Maximum Nodes Touched by AR4A and AR4JA

6.3.4 Conclusions

The results of this section suggest that cycles are more prevalent than one might have thought. The assumption that messages passed in iterative decoders are independent clearly breaks down after only a handful of iterations. However, since message passing works, it must be that this assumption is important only to the proofs, and not actually to the success of decoding. As Gallager wrote in Chapter 4 of his seminal work [26],

> This lack of independence can be ignored, however, on the reasonable assumption that the dependencies have a relatively minor effect and tend to cancel each other out somewhat.

This idea is echoed by Richardson and Urbanke in [49], where, after proving that for large enough code lengths the decoding performance converges to the cycle-free performance, they remark that

> The required size of n [codeword length], according to the proofs, could be absurdly large. The proofs, however, are very pessimistic. We assume, for example, that any loop effectively introduces an error into every message it affects. ... In practice, however, large loops may have only a small effect ....

Appendix

A-1 Proof of the Parametric Derivatives Formula, Theorem 4.2*

While Equation (4.3) is much clearer if $f[1] = 1$, the proof is less obvious so we will need to prove a related, but more complicated, result given in Theorem A-1.

Theorem A-1. *Let* $f_\lambda = \frac{f^{(\lambda_1)}(t)}{f'(t)^{\lambda_1}} \frac{f^{(\lambda_2)}(t)}{f'(t)^{\lambda_2}} \ldots$[1] *and* $g_{r+1} = \frac{g^{(r+1)}(t)}{f'(t)^{r+1}}$. *Then, the* n*th derivative of* g *with respect to* f *is given by*

$$h_n(t) = \frac{d^n g}{df^n}(t) = \sum_{\Lambda \in V(n)} (-1)^{\#\lambda - r} f_\lambda g_{r+1},$$

where

$$V(n) = \left\{ \lambda : \sum_{j=1}^{\#\lambda} \max(\lambda_j - 1, 1) = n - 1 \right\}.$$

We have changed the summation over number theory partitions λ to one over set theory partitions Λ, with the understanding that λ now refers to the corresponding number theory partition to Λ. When $f'(0) = 1$ and we evaluate the equation at $t = 0$, this simplifies directly to Theorem 4.2 on page 64.

This result can be proved by induction on n, noting that the statement is true for $n = 1$, as

$$h_1(t) = \frac{dg}{df} = g_1$$

[1] With this definition, $f_1 = 1$.

If we then assume that the formula is true for $h_n(t)$, we will show that it is true for $h_{n+1}(t)$.

To get $h_{n+1}(t)$, we need to differentiate $h_n(t)$ by f. To accomplish this, we first differentiate with respect to t and then divide by $f'(t)$. Each term corresponding to a specific partition will be differentiated by the product rule, yielding several new terms in h_{n+1}, each corresponding to taking the derivative w.r.t. t of either g_{r+1}, a specific f_j, or the denominator.

We start with a specific Λ, which contains the numbers 1 through ξ partitioned into $\#\Lambda$ sets. The term corresponding to this Λ is

$$(-1)^{\#\lambda-r} f_\lambda g_{r+1} = (-1)^{\#\lambda-r} \prod_i \frac{f^{\lambda_i}(t)}{f'(t)^{\lambda_i}} \frac{g^{(r+1)}(t)}{f'(t)^{r+1}}.$$

Taking the derivative will yield terms of three types:

- If we take the derivative w.r.t. t through g, then divide by $f'(t)$, the result will be

  $$(-1)^{\#\Lambda-r} N(\lambda) \prod_i \frac{f^{\lambda_i}(t)}{f'(t)^{\lambda_i}} \frac{g^{(r+2)}(t)}{f'(t)^{r+2}},$$

  which corresponds to a new partition found by including a new set of size 1 in λ. Equivalently, this derivative creates a new partition $\eta = \{\Lambda, (\xi+1)\} \in V(n+1)$. This new partition has $\#\eta = \#\Lambda + 1$ and ${}_\eta = r_\Lambda + 1$, so $(-1)^{\#\Lambda - r_\Lambda} = (-1)^{\#\eta - r_\eta}$. Thus, this new term is precisely equal to

  $$(-1)^{\#\eta-r} N(\eta) \prod_i \frac{f^{\eta_i}(t)}{f'(t)^{\eta_i}} \frac{g^{(r+1)}(t)}{f'(t)^{r+1}} = (-1)^{\#\eta-r} f(\eta) g(r+1).$$

- If we take the derivative w.r.t. t through $f^{\lambda_i}(t)$ then divide by $f'(t)$, the f^{λ_i} term will be replaced by an f^{λ_i+1} term. There will also be an additional power of $f'(t)$ in the denominator. This corresponds to a new partition that is formed by taking $\lambda_i > 1$ and increasing it by one. In terms of Λ, this new partition

is created by including $\xi + 1$ in the ith set of Λ. This changes neither the number of singletons, nor the number of total sets, leaving the coefficient of -1 unchanged. This new partition is clearly a member of $V(n+1)$.

- If the total denominator is called $f'(t)^L$, the derivative w.r.t t through $f'(t)$ is $\frac{-Lf''(t)}{f'(t)^{L+1}}$. When we divide by $f'(t)$, we will arrive at $\frac{-Lf''(t)}{f'(t)^{L+2}}$, essentially creating a new partition found by including a new set of size two in λ. To accomplish this, both $\xi + 1$ and $\xi + 2$ will be incorporated into Λ to create an additional partition of size two. This new partition will also be a member of $V(n+1)$. Since the new partition has an additional pair in it, its size has increased by one, which accounts for the negative sign.

While we have proved that each term in $h_n(t)$, when differentiated, yields only terms that should be in $h_{n+1}(t)$, we must now show that every element of $V(n+1)$ can be found in this way.

Many elements of $V(n+1)$ are simple to identify. For instance, if the partition Λ is such that the largest number is contained in a set of size one, or is contained in a set of size three or larger, then removing it reveals the element of $V(n)$ that, when differentiated, yields a term corresponding to Λ. When, however, the largest element of Λ is contained in a pair, this is less trivial. Consider the partition $\Lambda = \{(15)(3)(24)\} \in V(4)$. If the largest term, 5, is removed, the partition is still in $V(4)$. Even removing 5 and 4 results in a partition still in $V(4)$. If the largest element of Λ is contained in a pair, then it must have resulted from taking the derivative of something through the denominator. In the denominator, f_1 is raised to a power $L = 1 + m_1 + 2m_2 + 3m_3 + \ldots$. When this is differentiated, the result is $-L$ times the appropriate functions. Thus, the derivative of a term corresponding to Λ must create L new partitions.

The L partitions created are described as follows:

- One partition is formed by adding a new pair $(\xi + 1, \xi + 2)$ to the partition.
- m_1 partitions are formed by adding $(\xi + 1)$ as a singleton, and including $\xi + 2$ in each existing singleton, creating a pair.
- $2m_2 + 3m_3 + 4m_4 + \ldots$ partitions are formed by replacing any element not in a singleton with $\xi + 1$, and including the displaced element as a new pair with $\xi + 2$.

Thus, the anti-derivative of a term corresponding to Λ when the largest number in Λ is located inside a pair is the partition formed when the largest number is removed, and its partner used to replace the second-largest number. A few examples of partitions and the partitions created by their differentiation are shown in Table A-1.

Partition	Derivative Partitions		
	through f	through g	through the denominator
(1)		(1)(2)	(1)(23), (13)(2)
(1)(23)	(1)(234)	(1)(23)(4)	(1)(23)(45), (15)(23)(4) (1)(25)(43), (1)(24)(35)
(1)(23)(456)	(1)(237)(456) (1)(23)(4567)	(1)(23)(456)(7)	(1)(23)(456)(78), (18)(23)(456)(7) (1)(28)(456)(73), (1)(38)(456)(72) (1)(23)(457)(68), (1)(23)(476)(58) (1)(23)(756)(48)

Table A-1: Examples of Partitions After Differentiation

We have now completed our induction. Our base case, $h_1(t)$ is true. Assuming that the theorem then holds for $h_n(t)$, then every partition in $V(n)$ is present as a term in $h_n(t)$. If we differentiate $h_n(t)$ with respect to t and divide by $f'(t)$, each term is uniquely related to a partition in $V(n + 1)$ with the sign as predicted by theorem A-1.

A-2 Notation and Definitions

Notation	Description
$\mathcal{C}$	a code
$\mathcal{C}^{\perp}$	the dual code
k	code dimension
	OR number of ones in each row of a LDPC parity check matrix
n	length of the code
GF(2)	Galois Field of two elements
BEC	binary erasure channel
AWGN	additive white Gaussian noise
R	code rate
G	generator matrix
H	parity check matrix
A_j	weight enumerator
$E(\theta)$	spectral shape
	$\overline{\lim}\frac{1}{n}\log A_j$
w	input weight
h	output weight
$E_0(\theta)$	spectral shape of Shannon ensemble
	$= \mathcal{H}(\theta) - (1-R)$
$\mathcal{H}(\theta)$	binary entropy function
	$= -\theta\log\theta - (1-\theta)\log(1-\theta)$
$\mathcal{H}(\mathbf{\Theta})$	entropy function
	$-sum_i\theta_i\log\theta_i$
q	number of times data is repeated in an RA code
M	protomatrix
$\Omega(n)$	set of all even vectors of length n
$\Omega^*(n)$	set of all vectors of length n with weight ≥ 2
d	minimum distance of a code
	minimum codeword weight in a linear code
$d^{\perp}$	minimum distance of the dual code
$A_j^{\perp}$	dual code weight enumerator
θ_c	value at which $E(\theta)$ crosses x-axis
θ_s	value at which the stopping set growth rate crosses x-axis
$T_v(L)$	number of variable nodes touched in a tree with L layers
$T_c(L)$	number of check nodes touched in a tree with L layers
$\Psi(x)$	moment-generating function for codeword weight
$\Psi_0(x)$	moment-generating function for Shannon ensemble codeword weight
$\psi(x)$	cumulant-generating function for codeword weight
$\psi_0(x)$	cumulant-generating function for Shannon ensemble codeword weight

Table A-2: General Notation

Queuing Notation	
W_n	waiting time of nth word
D_n	decoding time of nth word
M	maximum number of iterations allowed
β	size of the additional buffer
a	number of decoder iterations between word arrivals
$w_n(k)$	probability that $W_n = k$
$d(k)$	probability that $D_n = k$
$w(k)$	$\lim_{n\to\infty} w_n(k)$
WER(200)	decoder performance with a fixed 200 iterations for each word
ρ	utilization factor, I_{avg}/a
I_{avg}	average number of iterations to decode a word
$\overline{W}_a(\beta)$	average waiting time with arrivals every a iterations and an additional buffer of size β

Partition Notation	
λ	number theory partition
Λ	set theory partition
$\#\lambda$	number of sets in a set theory partition number of elements in a number theory partition
$\mathcal{N}(\lambda)$	number of set theory partitions that collapse into λ
r	number of sets of size one
$\mathcal{O}(n^i)$	less than or equal to a constant times n^i

Table A-3: Special Notation

Bibliography

[1] A. Abbasfar, D. Divsalar, and K. Yao. Accumulate Repeat Accumulate Codes. In *Proc. IEEE International Symposium on Information Theory*, 2004.

[2] S. Aji, S. Fogal, R. McEliece, and B. Wang. Constrained Entropy, Free Energy and the Legendre Transform. In *International Symposium on Communication Theory and Applications*, pages 110–115, Ambleside, 2005.

[3] S. Aji, R. McEliece, and S. Sweatlock. On the Taylor Series of Asymptotic Ensemble Weight Enumerators. In *International Symposium on Communication Theory and Applications*, Ambleside, 2007.

[4] K. Andrews, S. Dolinar, D. Divsalar, and J. Thorpe. Design of Low-Density Parity-Check (LDPC) Codes for Deep Space Applications. *JPL InterPlanetary Network (IPN) Progress Report 42-159*, Nov. 2004.

[5] K. Andrews, S. Dolinar, and J. Thorpe. Encoders for Block-Circulant LDPC Codes. In *Proc. IEEE International Symposium on Information Theory*, 2005.

[6] L. F. A. Arbogast. *Du Calcul des Derivations*. Levrault, Strasbourg, 1800.

[7] V. Arnold. *Mathematical Methods of Classical Mechanics*. Springer-Verlag, 1979.

[8] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[9] E. R. Berlekamp and R. J. McEliece. Average-Case Optimized Buffered Decoders. In J. K. Skwirzynski, editor, *The Impact of Processing Techniques on*

Communications. Dordrecht:Martinus Nijhoff, 1985. (Volume 91E in NATO ISI Series).

[10] C. Berrou, A. Glavieux, and P. Thitimajshimia. Near Shannon limit error-correcting coding and decoding. In *Proc. IEEE International Conference on Communcations*, 1993.

[11] G. Bosco, G. Montorsi, and S. Benedetto. Decreasing the Complexity of LDPC Iterative Decoders. *IEEE Communications Letters*, 9, July 2005.

[12] G. Constantine and T. Savits. A Multivariate Faà Di Bruno with Applications. *Transactions of the American Mathematical Society*, 348, 1996.

[13] Consultative Committee for Space Data Systems (CCSDS). Low Density Parity Check Codes for use in Near-Earth and Deep Space Applications, Sept. 2007. Orange book, available at http://public.ccsds.org/publications/archive/131x1o2e1.pdf.

[14] C. Di. *Asymptotic and Finite-Length Analysis of Low-Density Parity-Check Codes*. PhD thesis, Ecole Polytechnique Fdrale de Lausanne (EPFL), 2004.

[15] C. Di, R. Urbanke, and T. Richardson. Weight distributions: How deviant can you be? In *Proc. International Symposium on Information Theory*, 2001.

[16] D. Divsalar. A Simple Tight Bound on Error Probability of Block Codes with Application to Turbo Codes. *Telecommunications and Mission Operations (TMO) Progress Report 42-139*, November 1999.

[17] D. Divsalar. Ensemble weight enumerators for Protograph LDPC codes. In *Proc. IEEE International Symposium on Information Theory*, 2006.

[18] D. Divsalar, S. Dolinar, and C. Jones. Low-Rate LDPC Codes with Simple Protograph Structure. In *Proc. IEEE International Symposium on Information Theory*, 2005.

[19] D. Divsalar, S. Dolinar, and C. Jones. Protograph Based LDPC Codes with Minimum Distance Linearly Growing with Block Size. In *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2005.

[20] D. Divsalar, H. Jin, and R. J. McEliece. Coding theorems for turbo-like codes. In *Proc. Allerton Conference on Communication, Control, and Computing*, 1998.

[21] C. F. Faà di Bruno. Note sur une novelle formule du calcul differentiel. *Quart. J. Mathematics*, 1, 1855.

[22] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. John Wiley & Sons, 2nd edition, 1971.

[23] S. L. Fogal, R. J. McEliece, and J. Thorpe. Enumerators for Protograph Ensembles of LDPC Codes. In *Proc. IEEE International Symposium on Information Theory*, 2005.

[24] M. Fossorier. Quasi-Cycle Low-Density Parity-Check Codes from Circulant Permutation Matrices. *IEEE Transactions on Information Theory*, 50, Aug. 2004.

[25] K. Fu. *Finite-Length and Asymptotic Analysis and Design of LDPC Codes for Binary Erasure and Fading Channels*. PhD thesis, University of Michigan, 2007.

[26] R. Gallager. *Low Density Parity Check Codes*. M.I.T. Press, 1963.

[27] R. Gallager. *Information Theory and Reliable Communication*. Wiley, 1968.

[28] D. Glickenstein. The Legendre Transform, January 2000. http://math.arizona.edu/∼glickenstein/tex/legendre.pdf.

[29] C.-H. Hsu and A. Anastasopoulos. Asymptotic Weight Distribution of Irregular Repeat-Accumulate Codes. In *Proc. IEEE Global Communications Conference (GLOBECOM)*, 2005.

[30] H. Jin and R. J. McEliece. RA Codes Achieve AWGN Channel Capacity. In *Applied Algebra, Algebraic Algebra and Error-Correcting Codes*. Springer, 1999.

[31] W. P. Johnson. The Curious History of Faà di Bruno's Formula. *The American Mathematical Monthly*, 109, March 2002.

[32] L. Kleinrock. *Queuing Systems*, volume 1 and 2. Wiley, 1975.

[33] S. Litsyn and V. Shevelev. On Ensembles of Low-Density Parity-Check Codes: Asymptotic Distance Distributions. *IEEE Transactions on Information Theory*, April 2002.

[34] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman. Analysis of low-density codes and improved designs using irregular graphs. In *Proc. 30th Association for Computing Machinery (ACM) Symp. on the Theory of Computing*, 1998.

[35] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. Spielman, and V. Stemann. Pratical loss-resilient codes. In *Proc. 29th Association for Computing Machinery (ACM) Symp. on the Theory of Computing*, 1997.

[36] D. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45:399–431, March 1999.

[37] D. C. J. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[38] A. Martinez and M. Rovini. Iterative Decoders Based on Statistical Multiplexing. In *Proc. 3rd International Symposium on Turbo Codes and Related Topics*, 2003.

[39] R. J. McEliece. *The Theory of Information and Coding*. Addison-Wesley, Reading, MA, 1977.

[40] O. Milenkovic, D. Leyba, and N. Kashyap. Shortened Array Codes of Large Girth. *IEEE Transactions on Communications*, 52, Aug. 2006.

[41] B. Moision. Buffer sizes and losses for SCPPM. (inter-office memorandum 331.2005.2.25).

[42] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[43] M. O'Sullivan. Algebraic Construction of Sparse Matrices with Large Girth. *IEEE Transactions on Information Theory*, 52, Feb. 2006.

[44] P. Oswald and A. Shokrollahi. Capacity-achieving sequences for the erasure channel. *IEEE Transactions on Information Theory*, 48, Dec. 2002.

[45] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, UC Berkeley, 1997.

[46] V. Pless. Power Moment Identities on Weight Distributions in Error Correcting Codes. *Information and Control*, 6:147–152, June 1963.

[47] T. Richardson. Multi-Edge Type LDPC Codes, May 2002. Presented at the Workshop Honoring Prof. McEliece on his 60th Birthday.

[48] T. Richardson, M. A. Shokrollahi, and R. Ubranke. Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes. *IEEE Transactions on Information Theory*, 47:619–637, Feb. 2001.

[49] T. Richardson and R. Urbanke. The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding. *IEEE Transactions on Information Theory*, 47, Feb. 2001.

[50] M. Rovini and A. Martinez. On the Addition of an Input Buffer to an Iterative Decoder for LDPC Codes. In *Proc. IEEE Vehicular Technology Conference*, 2007.

[51] C. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, Oct. and July 1948.

[52] A. Shokrollahi and R. Storn. Design of efficient erasure codes with differential evolution. In *Proc. IEEE International Symposium on Information Theory*, 2000.

[53] R. P. Stanley and S. Fomin. *Enumerative Combinatorics, Volume 2.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 1999.

[54] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11, 1997.

[55] S. L. Sweatlock, S. Dolinar, and K. Andrews. Buffering Requirements for Variable Iterations LDPC Decoders. In *Proc. Information Theory and Applications (ITA) Workshop*, 2008.

[56] R. M. Tanner. A Recursive Approach to Low Complexity Codes. *IEEE Transactions on Information Theory*, 27, Sept. 1981.

[57] R. M. Tanner. On graph constructions for LDPC codes by quasicyclic extensions. In M. Blaum, P. Farrell and H. van Tillborg, editor, *Information, Coding and Mathematics*, pages 209–219. Kluwer Academic Publishers, 2002.

[58] J. Thorpe. Low-Density Parity-Check (LDPC) Codes Constructed from Protographs. *JPL InterPlanetary Network (IPN) Progress Report 42-154*, Aug. 2003.

[59] J. Thorpe. *Analysis and Design of Protograph Based LDPC Codes and Ensembles*. PhD thesis, California Institute of Technology, 2005.

[60] J. Vogt and A. Finger. Increasing the Throughput of Iterative Decoders. *Electronics Letters*, 37, June 2001.

[61] H. Zhang and J. Moura. Large-Girth LDPC Codes Based on Graphical Models. In *Proc. IEEE Workshop on Signal Processing Advances in Wireless Communications*, 2003.