2020-07-29

Applications of Mathematical Optimization Methods to Digital Communications and Signal Processing

Spencer Giddens
Brigham Young University

Applications of Mathematical Optimization Methods to Digital Communications and Signal Processing

Spencer Giddens

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mathematics

Brigham Young University

ABSTRACT

Applications of Mathematical Optimization Methods to Digital Communications and Signal Processing

Spencer Giddens
Department of Mathematics, BYU
Master of Science

Mathematical optimization is applicable to nearly every scientific discipline. This thesis specifically focuses on optimization applications to digital communications and signal processing. Within the digital communications framework, the channel encoder attempts to encode a message from a source (the sender) in such a way that the channel decoder can utilize the encoding to correct errors in the message caused by the transmission over the channel. Low-density parity-check (LDPC) codes are an especially popular code for this purpose. Following the channel encoder in the digital communications framework, the modulator converts the encoded message bits to a physical waveform, which is sent over the channel and converted back to bits at the demodulator. The modulator and demodulator present special challenges for what is known as the two-antenna problem. The main results of this work are two algorithms related to the development of optimization methods for LDPC codes and the two-antenna problem.

Current methods for optimization of LDPC codes analyze the degree distribution pair asymptotically as block length approaches infinity. This effectively ignores the discrete nature of the space of valid degree distribution pairs for LDPC codes of finite block length. While large codes are likely to conform reasonably well to the infinite block length analysis, shorter codes have no such guarantee. Chapter 2 more thoroughly introduces LDPC codes, and Chapter 3 presents and analyzes an algorithm for completely enumerating the space of all valid degree distribution pairs for a given block length, code rate, maximum variable node degree, and maximum check node degree. This algorithm is then demonstrated on an example LDPC code of finite block length. Finally, we discuss how the result of this algorithm can be utilized by discrete optimization routines to form novel methods for the optimization of small block length LDPC codes.

In order to solve the two-antenna problem, which is introduced in greater detail in Chapter 2, it is necessary to obtain reliable estimates of the timing offset and channel gains caused by the transmission of the signal through the channel. The timing offset estimator can be formulated as an optimization problem, and an optimization method used to solve it was previously developed. However, this optimization method does not utilize gradient information, and as a result is inefficient. Chapter 4 presents and analyzes an improved gradient-based optimization method that solves the two-antenna problem much more efficiently.

Keywords: optimization, signal processing, digital communications, low-density parity-check (LDPC) codes, two-antenna problem, aeronautical telemetry, parameter estimation

Acknowledgements

First off, I would like to thank my advisor Dr. Jared Whitehead for his excellent support and mentorship from the very beginning of my research career. Despite the fact that I did not have any previous research experience, he had enough confidence in me to give me a major role in my first research project. Had he not provided me with that experience, I would not be where I am today.

Second, I want to thank my pseudo-advisor Dr. Willie Harrison. I appreciate the opportunity he gave me to begin research in a different field via a study abroad experience, and the opportunity to continue research in that field afterwards. I greatly admire his work ethic, integrity, and positive outlook on life, and I am grateful for the many hours he spent helping me both with my research and my career decisions.

I also want to thank the final member of my committee, Dr. Michael Rice, for inviting me to work on a project that allowed me to continue working in the field I began studying with Dr. Harrison. He consistently managed to balance high expectations for my work with a high level of confidence in my abilities, encouraging me to grow as a researcher.

I am grateful to the many other faculty members (both inside and outside BYU), TAs, and students I have had the opportunity to learn from during my time at BYU, especially Elliot Brown, Brooke Mosby, Michael Hansen, and Gabriella Smith, who made the ACME program as enjoyable as it was difficult.

Outside of school, I want to thank my family, especially my parents, for their unwavering emotional and financial support as I have worked to pursue my goals. I also want to thank the countless friends that have supported me. I wish I could list all of you by name, but I need to save some room for my thesis.

Most importantly, I want to thank my wife, Caitlin. I am so grateful for the many sacrifices she makes on my behalf. Her long hours at work to support us financially and her sacrifices of time to allow me to work on research do not go unnoticed. She is truly my best friend and my number 1 fan.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Optimization is of interest to virtually every scientific discipline, though the exact way in which it is used varies drastically. Mathematics is interested in optimization from a purely theoretical standpoint. By considering optimization methods generally, rather than only in the context of a specific problem, a unifying theory can be developed that allows optimization techniques to be successfully applied across multiple disciplines.

Other disciplines are also interested in optimization, but their interest is typically to solve a practical problem. In business, an organization may seek to maximize their profits. In computer science, a common goal is to develop algorithms that minimize computational cost while achieving the desired results. A civil engineer, meanwhile, might be interested in maximizing the efficiency of a new public roadway. Despite these differences, the task at hand is fundamentally the same: to determine, out of a set of possible choices, the choice which leads to the "best" outcome. It should come as no surprise, given all of this, that optimization also has applications in communication theory and signal processing.

This chapter first introduces the formal mathematical definitions related to optimization as they will be used throughout this work. Next, it provides introductory material necessary to understand the fields of communication theory and signal processing, to which the optimization methods developed later on will be applied. After providing this material as context, the chapter concludes by listing the contributions made by this thesis to the fields of communication theory and signal processing.

1.1 Overview of Optimization Methods

Depending on the context and application, optimization could either refer to minimization or maximization. As is common in optimization literature, the following definitions will assume that minimization is the goal. We note that it is usually straightforward to convert a minimization method to a maximization method if desired, and that in some circumstances maximization is equivalent to the minimization of a slightly modified problem [2].

Definition 1.1. An optimization problem in *standard form* is written

$$\begin{aligned} \underset{\mathbf{x}\in X}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0 \\ & h_j(\mathbf{x}) = 0 \end{aligned}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, p$. In standard form, $X \subset \mathbb{R}^n$ is called the *search space*, $f : X \to \mathbb{R}$ is called the *objective function*, the $g_i : X \to \mathbb{R}$ are called *inequality constraints*, and the $h_i : X \to \mathbb{R}$ are called *equality constraints*. If $m = 0$ and $p = 0$, the optimization problem is *unconstrained*.

Optimizing an objective function f could refer to either finding the minimum value of f, denoted f^*, or to finding the value(s) of $\mathbf{x} \in X$, denoted $\hat{\mathbf{x}}$, such that $f(\hat{\mathbf{x}}) = f^*$. For this work, we use the latter definition, and we denote the set of such values by $\underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$.

Definition 1.2. An *optimization method* is defined to be any algorithm whose goal is to find

$$\underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$$

where f is the objective function of an optimization problem in standard form.

This definition encompasses a large range of algorithms. Which algorithm to use and how effectively the algorithm performs the optimization are dependent on the information known about the optimization problem. Almost certainly, the easiest optimization problems to solve are those where the objective function and the search space are continuous, convex, and first and second order partial derivative information for the objective function is known and easily computed. The first order partial derivatives can be organized into a vector known as the gradient, and second order partial derivatives can be organized into a matrix called the Hessian. Newton's method performs famously well as an optimization method under these circumstances, but that is not the only method available. Entire books have been written containing optimization methods useful for when the objective function and search space are convex [3, 4]. Non-convex optimization has also been of interest lately, especially in light of the current popularity of deep neural networks [5]. When the Hessian is unknown or costly to obtain (or approximate), first order methods like gradient descent are also typically effective [6]. In the case where neither first order nor second order information is available, a myriad of derivative-free and blackbox optimization methods may be viable [2]. Finally, it should be mentioned that an optimization problem with a discrete search space is usually far more difficult to solve than a problem with a continuous search space [7].

When choosing an optimization method for a particular problem, the general rule of thumb is to choose a method that utilizes as much information about the problem as is available. As explained in [2], if the gradient of the objective function is known and easy to compute, a gradient-based method will almost always outperform a method that does not use the gradient. It is also worth mentioning here that the majority of optimization methods of interest are iterative. While there do exist some methods whereby the optimization problem may be solved by direct computation, these are only applicable under special circumstances, and in the case where they are applicable, the solution tends to be easy enough that further study of the optimization method itself is unnecessary.

A natural question to ask is how can optimization methods be compared to determine which is best? There are a number of metrics that can be considered when comparing algorithms. First and probably most obvious is the accuracy of the algorithm. The closer an optimization method gets to the true minimizer, the more accurate the method. Related to the accuracy is the variance of an algorithm. In other words, if an optimization method is used repeatedly on the same problem, how much variance is there in the results? One may also consider the number of iterations required by the optimization method to produce its results. Similarly, the number of objective function evaluations required by the algorithm may be of interest. Note these last two metrics are usually related, but not always the same thing. Usually, there is a trade-off between the accuracy/variance of an optimization method,

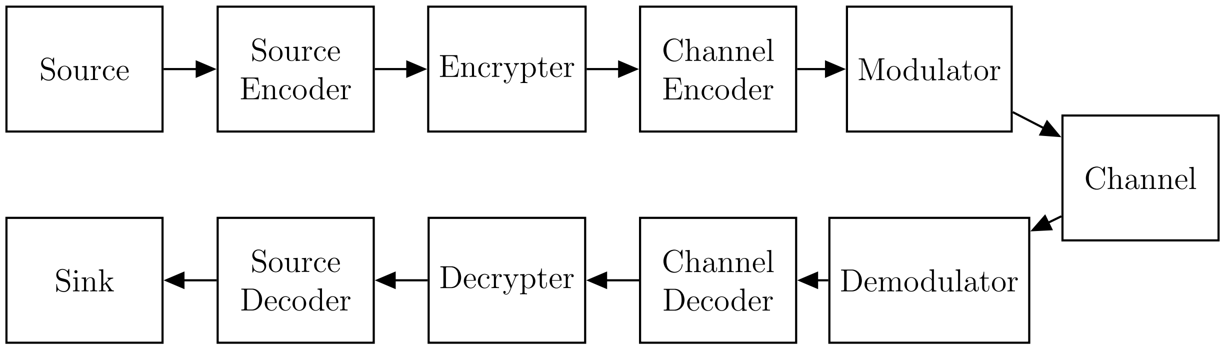

Figure 1.1: The general framework for digital communications.

and the number of iterations or objective function evaluations required, and therefore the best optimization method may differ for different problems.

To conclude this section, it is worth discussing one (though not the only) limitation of optimization algorithms. Even the best optimization methods are not guaranteed to successfully solve every problem, and some problems may be impossible to solve entirely without a bit of luck. A classic example of this type of problem is when the objective function is highly non-convex with a large number of local minima. Iterative optimization methods under these circumstances are unlikely to converge to the true minimizer, and in some cases may never do so unless initialized in very specific regions of the search space. Thus, it is crucial to have a firm understanding of the search space and the structure of the objective function when developing an optimization method for a specific problem.

1.2 Digital Communications and Signal Processing

In this section, we briefly introduce the fields of digital communication and signal processing. Digital communication refers to the transmission of a message from a sender, also called the source, to a receiver, also called the sink. A diagram depicting digital communication is shown in Figure 1.1. This diagram depicts the path that a message travels to get from the source to the sink. First, the message is passed through the source encoder. The source encoder encodes the message into a sequence of bits in a manner that attempts to use the fewest number of bits possible to completely represent the information in the message.

Next, the encoded message is encrypted using techniques from cryptography. After that, the encrypted message is passed through the channel encoder. The channel encoder encodes the encrypted sequence of bits into blocks of bits called *codewords*. This is done in a way that allows transmission errors to be corrected later on. The modulator then converts this final bit sequence to a waveform which is transmitted over a physical channel, such as a wire or the air for wireless channels. At the receiver, the demodulator's job is to detect and interpret the waveform that was sent over the channel. It converts the waveform back into bits, which are then decoded and decrypted by inverting the processes of the encoders and the encrypter, before finally making their way to the sink.

It is likely that an optimization problem can be formulated that relates to any of the steps in the digital communication framework, but this work focuses on developing optimization methods for problems in two of these steps in particular. Chapter 3 discusses an optimization method that could lead to an improved algorithm sometimes used for the channel decoder, while Chapter 4 develops and analyzes an optimization method used in the demodulator of a specific communication scheme. The unifying theme between the optimization methods of both chapters is that transmission of the message over the channel leads to a variety of errors in the message, and the improvement of the systems used to correct these errors can be formulated as an optimization problem.

1.2.1 Channel Encoder Errors. The first type of error in the transmission that typically needs to be corrected is called a bit error.

Definition 1.3. A *bit error* occurs when the received sequence incorrectly has a 0 where a 1 was located in the transmitted sequence (or vice-versa) usually due to noise in the channel. This can be measured by either comparing the bit sequence before the channel encoder with the bit sequence after the channel decoder, or by comparing the bit sequence after the channel encoder with the bit sequence before the channel decoder. The goal of the channel encoder is to encode the message in such a way that the decoder is able to correct bit errors caused by the channel. An encoder/decoder pair that does this is called an *error-correcting* *code.*

The performance of error-correcting codes can be measured by considering the BER, or the number of bit errors divided by the number of bits, after the channel decoder. Some codes perform better than others, but there is a limit to how well these codes can perform. More specifically, Shannon's noisy-channel coding theorem states that for a given amount of noise in a channel, there exists a code which can communicate messages virtually error-free up to a maximum communication rate (i.e. bits per unit time), called the Shannon capacity [8]. To date, a few error-correcting codes have been shown to achieve Shannon capacity, including turbo codes [9], polar codes [10], and low-density parity-check (LDPC) codes [11]. This work focuses on the optimization of LDPC codes, which will be introduced in more detail in Chapter 2.

1.2.2 Modulation Errors. We now shift to introduce a different type of transmission error that occurs in the demodulator for a specific problem in aeronautical telemetry called the *two-antenna problem*. Note that the two-antenna problem is discussed in greater detail in [12] and [1], but is summarized here for completeness. Aircraft, especially military aircraft, collect data while in flight that need to be transmitted to the ground in real time. To accomplish this, a transmitter antenna is placed on the bottom of the aircraft that sends information to a receiver on the ground. However, under some circumstances, the body of the aircraft may prevent the transmitted signal from reaching the ground antenna. To counter this, a second antenna is placed on the top of the aircraft that transmits the same message as the first antenna. When the signal from one of the two antennas is detectable by the ground receiver, the message can be transmitted successfully. However, this second antenna complicates matters when the signals from both antennas are simultaneously available. In this case, the two signals self-interfere, making it difficult for the demodulator to correctly determine the intended message. This self-interference is known as the two-antenna problem, and is depicted in Figure 1.2.

To solve the two-antenna problem, a number of variables must be estimated at the

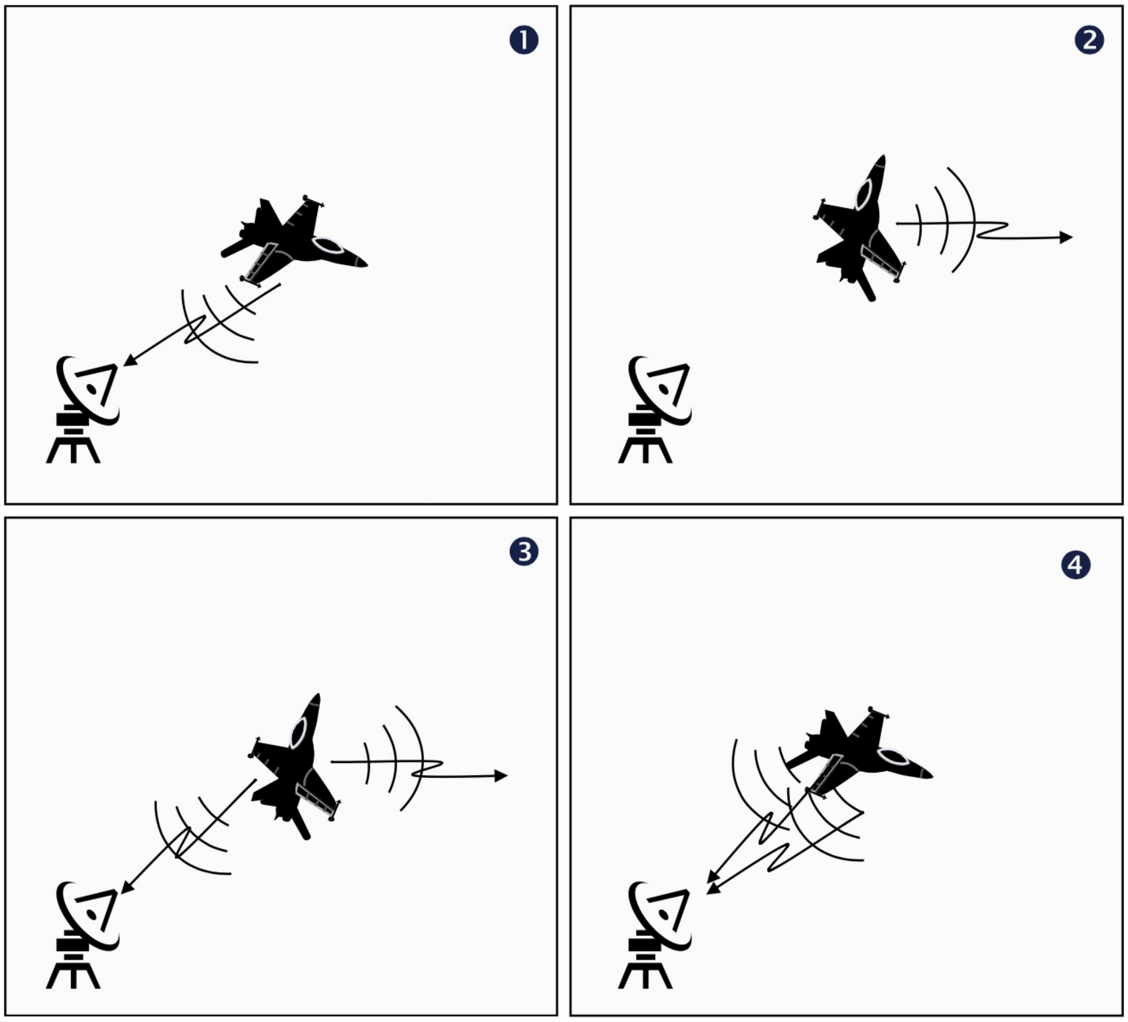

Figure 1.2: Depiction of the two-antenna problem (from [1]).

demodulator. The one that will be primarily discussed in this work is the timing offset.

Definition 1.4. The *timing offset* is the difference between the expected time of the received signal and the time it was actually received. In a communication system, the timing offset must be estimated and corrected in order to accurately demodulate the signal. Since there are two signals in the two-antenna problem, there are two timing offsets to estimate at the demodulator. The joint estimator of these offsets can be formulated as an optimization problem, which will be further explained in Chapter 2.

1.3 Contributions

This work makes a number of important contributions towards optimization methods used in digital communication and signal processing. First, a unique objective function and search space are developed for the improvement of LDPC codes, leading to a novel algorithm for the enumeration of the search space. This algorithm is presented and analyzed in the context of the objective function and its possibility for improving upon current state-of-the-art optimization methods for LDPC codes. Next, an improved optimization method is developed to solve an optimization problem necessary to estimate the timing offset. The previously used algorithm, developed in [1], was a modification on the popular derivative-free simplex algorithm developed by Nelder and Mead [13], and will be explained in greater detail in Chapter 2. The improved algorithm, explained in Chapter 4, uses first order gradient information to decrease the temporal complexity of the estimation. The algorithm chosen is compared to other potential algorithms, and is shown to outperform the previous one.

The outline of this thesis is as follows. Chapter 2 provides the background necessary to understand the algorithms developed in Chapter 3 and Chapter 4, including a thorough explanation of LDPC codes, the derivation of the estimators for the two-antenna problem, and the presentation of the previously used timing offset estimator. Next, Chapter 3 develops and analyzes the algorithm for enumerating the search space of the LDPC code optimization problem and puts it into the context of current LDPC optimization methods. Then, Chapter

4 develops and analyzes the improved timing offset estimator, before providing detailed comparisons with the previous algorithm. Chapter 5 then summarizes and concludes the work.

One of the fascinating things about optimization methods is that they are applicable to a wide range of fields. As a result, a full understanding of the details and impacts of an optimization algorithm cannot be obtained without first acquiring the necessary domain knowledge. This chapter provides the digital communication and signal processing background required to fully appreciate the algorithms presented in Chapter 3 and Chapter 4. For LDPC codes, this chapter provides a more detailed explanation of the codes, their properties, and the current state of their optimization in the literature. For the two-antenna problem, the chapter derives the needed estimators and explains the method previously used to obtain these estimates.

Before proceeding, we provide a brief description of notation used throughout the thesis. If x is a vector, we denote the ith element of x by x_i (zero-based indexing is assumed). To denote the ith through jth elements of x, we write $x_{i:j}$. If j is not written explicitly in this notation (i.e. $x_{i:}$), we mean all elements of x from the ith one to the end of the vector.

2.1 LDPC codes

This section provides all of the background necessary to understand the contributions made in Chapter 3 towards the optimization of LDPC codes. Note that the descriptions of LDPC codes and background provided are based on the presentation of the same content in [14]. First, some basic definitions related to LDPC codes are given and illustrated with an example. Then, the constraints on the search space are presented. Finally, the section concludes with a discussion of the current state of LDPC code optimization, and where this work fits into the literature.

2.1.1 Basic Definitions. The goal of the channel encoder is to encode its input, a series of message bits, into a series of coded bits (i.e., a codeword) such that the channel decoder is able to correct errors that occur in the series of coded bits as they are transmitted through the channel. There are many ways to design effective channel encoders and decoders, with varying degrees of success, and LDPC codes represent a particularly effective design. These codes were invented by Robert Gallager in the 1960s [15], though not popularized until rediscovered by MacKay in the 1990s [16, 11, 17].

We provide a high-level understanding of LDPC codes to provide context for their optimization. First, the series of incoming message bits are broken into blocks of k bits, usually denoted by row vectors. Then, an additional $m = n - k$ bits, called parity-check bits, are added to the end of each block to make codewords of n total bits. The specific code used, which is fixed in advance, selects subsets of bits within the codeword and requires that the mod-2 sum of these bits is 0 (i.e. has even parity). The values of the parity-check bits are selected such that all of these mod-2 sums are satisfied. Furthermore, the mod-2 sums defined by the code uniquely define a matrix, called the parity-check matrix and denoted by H, which when multiplied by a block of n bits produces the all-zero vector if and only if the block represents a valid codeword. This provides the decoder with a way to detect if an error has occurred during the transmission. Finally, we define the rate of the code to be k/n, since every n coded bits correspond precisely to k message bits.

Example 2.1. Let $n = 6$, $k = 3$, and assume that a particular block of message bits is $\mathbf{m} = [1, 1, 1]$. The codeword associated with these message bits is $\mathbf{c} = [1, 1, 1, c_3, c_4, c_5]$, where c_3, c_4, and c_5 are the parity check bits, which are yet to be determined. An individual LDPC code is defined by its list of parity-check constraints. If we assume that the specific code we are using requires

$$\begin{aligned} c_0 + c_1 + c_2 + c_4 + c_5 &= 0 \mod 2 \\ c_0 + c_2 + c_3 + c_4 + c_5 &= 0 \mod 2 \\ c_0 + c_1 + c_2 + c_3 + c_5 &= 0 \mod 2, \end{aligned} \tag{2.1}$$

then it can be determined by solving a system of linear equations that $c_3 = 1$, $c_4 = 1$, and

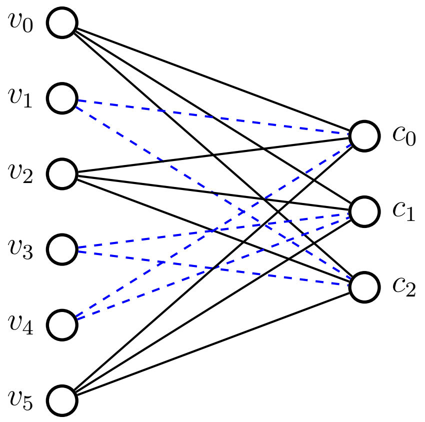

Figure 2.1: The Tanner graph for (2.2). The variable nodes are on the left and the check nodes are on the right. The variable nodes correspond to bits in the received codeword, and have maximum degree 3. Notice that 6 of the 15 total edges (blue, dashed) are connected to variable nodes of degree 2, while the other 9 of the 15 (black, solid) are connected to variable nodes of degree 3. So, $\lambda = (0, \frac{6}{15}, \frac{9}{15})^T$ for this code. All 15 edges are connected to check nodes of degree 5, so $\rho = (0, 0, 0, 0, \frac{15}{15})^T$.

$c_5 = 0$. The parity check matrix for (2.1) is

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

Note it is easily verified that $\mathbf{c}H = \mathbf{0}$.

It is worth mentioning here that LDPC codes used in practice utilize much longer block lengths than the code in the example. In order for the decoder to be computationally feasible, the parity check matrix needs to be sparse, or in other words have low-density (hence the name low-density parity-check codes). This is accomplished by increasing the codeword block length n and using small subsets of the coded bits for our parity checks. So, strictly speaking, the example shown is not an LDPC code, but it is a helpful illustration.

Definition 2.2. LDPC codes are commonly described using bipartite Tanner graphs [18], a graphical representation of the parity check matrix. In these graphs, one set of nodes, called the *variable nodes*, represents the codeword, with each node corresponding to one coded bit. The other set of nodes, called the *check nodes*, represents the parity-check constraints. An undirected edge is drawn from variable node i to check node j if there is a 1 in the ith row and jth column of the parity check matrix. This representation is used mainly due to its convenience for visualizing the algorithm used by the decoder.

Definition 2.3. A *cycle* in a graph is defined to be a path between nodes along edges such that the first node is the same as the last node, and no other node in the path repeats.

Definition 2.4. The *degree* of a node is defined to be the number of edges connected to that node.

Definition 2.5. A distribution on the variable nodes, denoted by λ, is defined to be a vector of length equal to the maximum variable node degree, where the ith component of λ is the proportion of edges connected to variable nodes of degree $i + 1$. A distribution on the check nodes is defined analogously and is denoted by ρ. A tuple containing λ and ρ is commonly referred to as a *degree distribution pair*. Finally, an LDPC code *ensemble*, or *family*, is the collection of codes with the same degree distribution pair. See Figure 2.1 for an example of these definitions.

If the received bits are not a codeword, the decoder uses the parity-check bits to correct the errors via belief propagation [19, 20], a message-passing decoding algorithm that has been shown to perform very close to the theoretical Shannon capacity [8, 21], making LDPC codes very powerful in many applications [22]. Details on the implementation of the belief propagation algorithm can be found in [20], but they are not important to this work, so they are not summarized here.

2.1.2 Search Space Constraints. The error-correcting capabilities of LDPC codes are dependent on the configuration of the parity-check constraints that define individual codes.

In other words, different parity-check matrices, or equivalently, different arrangements of the edges in the Tanner graph, result in different post-decoder BERs (recall the bit error rate was defined in Chapter 1 to be the number of bit errors divided by the number of bits). An optimization problem interested in the best error-correction capabilities might attempt to find the specific LDPC code which minimizes the BER. However, the search space for such an optimization problem would be prohibitively complex given typical codeword block lengths and the resulting size of the parity check matrices. Fortunately, LDPC codes within the same code family tend to have very similar error-correction capabilities, so optimization problems tend to use the space of valid degree distribution pairs as the search space.

Let d_v be the maximum variable node degree, d_c be the maximum check node degree, n be the block length of the code (the length of the codewords), n_e be the number of edges present in the bipartite graph, and $m = n(1-r)$, where r is the rate of the code. For a degree distribution pair to be valid, it must adhere to the following:

$$\sum_{i=0}^{d_v-1} \lambda_i = 1, \tag{2.3}$$

$$\sum_{i=0}^{d_c-1} \rho_i = 1, \tag{2.4}$$

$$\sum_{i=0}^{d_v-1} \frac{\lambda_i}{i+1} = \frac{n}{n_e}, \tag{2.5}$$

$$\sum_{i=0}^{d_c-1} \frac{\rho_i}{i+1} = \frac{m}{n_e}, \tag{2.6}$$

$$\lambda_i \geq 0, \tag{2.7}$$

$$\rho_i \geq 0. \tag{2.8}$$

In addition, we require $\lambda_0 = \rho_0 = 0$ in order to ensure there are no nodes of degree one, which are useless for decoding. The constraints in (2.3), (2.4), (2.7), and (2.8) are due to the fact that λ and ρ define distributions on the proportion of edges connected to variable nodes and check nodes, respectively, of certain degrees. Meanwhile, constraints (2.5) and (2.6) prevent logical fallacies such as, for example, $\lambda = (0, \frac{7}{15}, \frac{8}{15})^\intercal$, meaning 7 edges are connected to variable nodes of degree 2, an obvious impossibility.

2.1.3 Optimization Problem. Today, the use of LDPC codes is widespread and well-documented. As a result, there have been a number of methods developed to try to optimize the code construction process (e.g., [22, 23, 24]). Let X be the set of all valid degree distribution pairs and let f be the function mapping elements of this set to the average BER for the code family described by the element of X. The optimization problem can be written in standard form as

$$\begin{aligned} \underset{(\lambda,\rho)\in X}{\text{minimize}} \quad & f(\lambda,\rho) \\ \text{subject to} \quad & -\lambda_i \leq 0 \\ & -\rho_i \leq 0 \\ & \sum_{i=0}^{d_v-1} \lambda_i - 1 = 0 \\ & \sum_{i=0}^{d_c-1} \rho_i - 1 = 0 \\ & \sum_{i=0}^{d_v-1} \frac{\lambda_i}{i+1} - \frac{n}{n_e} = 0 \\ & \sum_{i=0}^{d_c-1} \frac{\rho_i}{i+1} - \frac{m}{n_e} = 0. \end{aligned} \tag{2.9}$$

Most of the methods developed to date for the optimization of LDPC codes, consider code design asymptotically as the block length approaches infinity [22, 25]. While there are some methods out there which seem to avoid directly assuming infinite block length in their optimization [23, 24, 26], these methods still rely on exceptionally large block lengths for performance estimations or otherwise utilize infinite block length analysis to some degree. Although these optimization methods have yielded positive results for large codes, they ignore the underlying structure of LDPC code degree distributions, and carry no guarantees for code design with shorter block lengths. More specifically, the infinite block length assumption ignores the right-hand side of the constraints in (2.5) and (2.6). At finite block lengths, those constraints have the effect of discretizing the space of valid LDPC code degree distributions, and the discretized space is progressively more sparse as the block length decreases. Thus, the degree distribution pairs produced by common optimization methods are not, strictly speaking, likely to be in the space of valid degree distribution pairs for finite block lengths, so the results of such optimization can only be utilized approximately in any practical code. For codes of smaller block length, many of the assumptions made in infinite block length LDPC code optimization are much farther from the truth than for large codes. Furthermore, smaller codes require more "rounding" with the degree distributions to find a valid code than do larger codes.

For the various reasons mentioned so far, it should be apparent that removing the infinite block length assumption and optimizing directly over the discrete valid degree distribution pair search space is of interest. In order to accomplish this, there must first be a method developed to enumerate the discrete search space in a way that allows an optimization method to "walk" through it. Chapter 3 presents and analyzes a novel algorithm that does exactly that. It is also worth mentioning here that asymptotic optimization assumes there are no cycles in the graph, which are well known to hinder performance [24], and are obviously unavoidable in a finite block length code. Though there are methods to reduce the number of small cycles in finite length LDPC codes [24, 26], even these techniques require *good* degree distributions as a starting point, which can be better accomplished for small codes through discrete code enumeration.

2.2 Two-Antenna Problem

In [12], the solution to the two-antenna problem was shown theoretically. Rice *et al.* [1, 27] then implemented these ideas on a practical aeronautical telemetry system, including developing an optimization method to estimate the timing offset. Chapter 4 of this work extends [1] by improving upon this optimization method. This section summarizes the background information from [1] necessary to understand the contributions made in Chapter

4.

First, the two-antenna problem is formulated mathematically. Next, the estimators for the variables required to solve the two-antenna problem are derived, and the estimator for the timing offset is shown to require an optimization method. This section concludes with the presentation of the optimization problem in standard form and a description and analysis of the optimization method previously used to solve the problem.

2.2.1 Problem Formulation.

At the modulator step of Figure 1.1, the coded bits that come out of the channel encoder are transformed into a modulated signal or waveform that can be sent over the channel. In this formulation, the complex-valued low-pass equivalent waveforms [28] are used to represent the real-valued bandpass signals. The continuous time complex-valued low-pass equivalent waveform is denoted $s(t)$. When represented in discrete time, we write the signal as $s(nT)$, where T is the spacing (in units of time) between samples, and n is an integer. In this case, $s(nT)$ represents the nth sampled value of the continuous waveform $s(t)$. Note that for the purposes of this work, the signal is complex-valued.

Here are some important definitions.

Definition 2.6. The *timing offset* is the delay (in units of time) of the signal that was actually received relative to the expected signal.

Definition 2.7. The complex-valued *attenuation* refers to the reduction of signal strength due to the channel.

Definition 2.8. The *power spectral density* of a signal is the power in the signal as a function of the frequency.

Definition 2.9. The *energy per bit*, usually denoted E_b, is the ratio of the amount of energy in the signal to the number of bits transmitted.

Definition 2.10. A *white complex-valued wide-sense stationary random process* $z(t)$ is a complex-valued random process with

$$\mathrm{E}\{z(t)\} = 0, \quad \text{all } t \tag{2.10}$$

$$\mathrm{E}\{z(t+\tau)z^*(t)\} = 2N_0\delta(\tau), \tag{2.11}$$

where $\mathrm{E}\{\cdot\}$ is the expectation operator, $z^*(t)$ is the complex conjugate of $z(t)$, and $\delta(\tau)$ is the Dirac delta function. The power spectral density of $z(t)$ is the Fourier transform of (2.11)

$$S_z(f) = 2N_0. \tag{2.12}$$

Definition 2.11. Finally, the *signal-to-noise ratio* is the ratio of the power in the signal to the power in the noise. Normalizing by the number of bits gives a measurement for this quantity, commonly expressed as E_b/N_0 in units of dB. Higher values of E_b/N_0 correspond to a less corrupted signal at the demodulator.

For the two-antenna problem, two signals carrying the same information, denoted $s_0(t)$ and $s_1(t)$, are transmitted simultaneously from the upper and lower antennas. Both of these signals experience timing offsets, denoted respectively as ϵ_0 and ϵ_1, as well as a complex-valued attenuation, respectively h_0 and h_1. In addition, assume the received signal experiences thermal noise modeled by a complex-valued zero-mean white Gaussian random process, denoted by $z(t)$. Assume $z(t)$ has power spectral density $2N_0$. The received signal can be written in continuous time as

$$r(t) = h_0 s_0(t-\epsilon_0) + h_1 s_1(t-\epsilon_1) + z(t). \tag{2.13}$$

Now assume the signal is sampled at T-spaced intervals. In the discrete-time domain, define $\tau_0 = \epsilon_0/T$, $\tau_1 = \epsilon_1/T$, and $w(nT)$ to be the sampled contribution of $z(t)$. Assume that for each n, $w(nT)$ is a sample from a complex-valued Gaussian random variable with zero mean and variance $2\sigma^2 = 2N_0/T$, and that the samples are independent and identically distributed (iid). Then, the received signal can be written in discrete time as

$$r(nT) = h_0 s_0((n - \tau_0)T) + h_1 s_1((t - \tau_1)T) + w(nT). \tag{2.14}$$

Note that (2.13) and (2.14) are identical to (1) and (2) from [1].

The goal is to obtain estimates of h_0, h_1, τ_0, and τ_1 from $r(nT)$ in order to accurately reconstruct the intended signal. To accomplish this, a sequence of L_p *pilot bits* is periodically inserted after every L_d data bits into the signal sent by each of the two antennas. At the receiver, these pilot bits can be used to construct effective estimators for the desired values.

2.2.2 Estimators. For the ARTM CPM waveform utilized in this work, bits in the codeword are mapped to symbols at a rate of 2 bits/symbol. Then these symbols are modulated to form the waveform that is sent over the channel, which is then sampled at a rate of N samples/symbol at the demodulator. The details of the ARTM CPM waveform and corresponding modulation are beyond the scope of this work, but can be found in [29]. Assume $N = 4$ samples/symbol, so there are $N_p = N \times L_p/2$ samples corresponding to the pilot bits. Also assume without loss of generality that the samples corresponding to the modulated pilot bits begin at index $n = 0$ and denote these samples by $p_0(nT)$ and $p_1(nT)$ for $n = 0, 1, \ldots, N_p - 1$. We can now write (2.14) for the pilot sequence as

$$r(nT) = h_0 p_0((n - \tau_0)T) + h_1 p_1((t - \tau_1)T) + w(nT). \tag{2.15}$$

Stacking the received samples into an $N_p \times 1$ vector yields the equivalent matrix-vector equation

$$\mathbf{r} = \mathbf{Ph} + \mathbf{w} \tag{2.16}$$

where $\mathbf{P}$ is the $N_p \times 2$ matrix

$$\mathbf{P} = \begin{bmatrix} p_0(-\tau_0 T) & p_1(-\tau_1 T) \\ p_0((1-\tau_0)T) & p_1((1-\tau_1)T) \\ \vdots & \vdots \\ p_0((N_p-1-\tau_0)T) & p_1((N_p-1-\tau_1)T) \end{bmatrix}, \tag{2.17}$$

$\mathbf{h}$ is the 2×1 vector

$$\mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \end{bmatrix}, \tag{2.18}$$

and $\mathbf{w}$ is the $N_p \times 1$ vector

$$\mathbf{w} = \begin{bmatrix} w(0) \\ w(T) \\ \vdots \\ w((N_p-1)T) \end{bmatrix}. \tag{2.19}$$

Note that $\mathbf{P}$ is a function of the parameter vector

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_0 \\ \tau_1 \end{bmatrix}. \tag{2.20}$$

Since the elements of $\mathbf{w}$ are iid complex-valued zero-mean circularly symmetric Gaussian random variables with variance $2\sigma^2$, the conditional probability density function of $\mathbf{r}$ can be written as

$$f(\mathbf{r}|h_0, h_1, \tau_0, \tau_1) = \frac{1}{(2\pi\sigma^2)^{N_p}} \exp\left\{ -\frac{1}{2\sigma^2} |\mathbf{r} - \mathbf{Ph}|^2 \right\}. \tag{2.21}$$

We seek to maximize (2.21), which is equivalent to minimizing

$$|\mathbf{r} - \mathbf{Ph}|^2. \tag{2.22}$$

Combining (2.22) with the maximum likelihood (ML) estimate of $\mathbf{h}$ via least squares produces the estimators for $\boldsymbol{\tau}$ and $\mathbf{h}$:

$$\hat{\boldsymbol{\tau}} = \underset{\boldsymbol{\tau}\in\mathcal{T}}{\operatorname{argmin}} \left\{ \left| \left[\mathbf{I}_{N_p} - \mathbf{P}\left(\mathbf{P}^\dagger\mathbf{P}\right)^{-1}\mathbf{P}^\dagger \right] \mathbf{r} \right|^2 \right\}, \tag{2.23}$$

$$\hat{\mathbf{h}} = \left(\hat{\mathbf{P}}^\dagger\hat{\mathbf{P}}\right)^{-1}\hat{\mathbf{P}}^\dagger\mathbf{r}, \tag{2.24}$$

where $\mathbf{P}$ is assumed to be a function of $\boldsymbol{\tau}$, $\mathbf{P}^\dagger$ is the Hermitian (conjugate transpose) of $\mathbf{P}$, $\hat{\mathbf{P}} = \mathbf{P}(\hat{\boldsymbol{\tau}})$, and $\mathcal{T}$ is the search space for $\boldsymbol{\tau}$. Note that (2.23), (2.24), and their respective derivations are equivalent to (13a), (13b), and their respective derivations from [1] when the frequency offset is assumed to be 0.

2.2.3 Optimization Problem. A thorough understanding of the search space $\mathcal{T}$ is necessary in order to perform the optimization required for (2.23). First, the pilot sequence is detected within the waveform by correlating the sampled waveform with the known pilot samples. Thus, at least one of τ_0 and τ_1 must be within one sample time of the true value (i.e. between -0.125 and 0.125). In addition, due to the physical dimensions of the aircraft and commonly used symbol rates, the differential delay, or the difference between τ_0 and τ_1, is restricted to be less than one symbol time (less than or equal to three sample times) in absolute value[1]. This means that if, without loss of generality, $-0.125 \leq \tau_0 \leq 0.125$, then $\tau_0 - 0.875 \leq \tau_1 \leq \tau_0 + 0.875$, so overall both τ_0 and τ_1 are bounded between -1 and 1.

Next, the nature of the relationship between $\boldsymbol{\tau}$ and $\mathbf{P}$ is too complicated to permit the real time computation of $\mathbf{P}$ on the required hardware for arbitrary values of $\boldsymbol{\tau}$ in the assumed interval. In [1], this problem was solved by pre-computing and saving $\mathbf{P}$ for quantized values of $\boldsymbol{\tau}$. A quantization parameter, Q, was fixed, and $\mathbf{P}$ was pre-computed and stored for all values of $\boldsymbol{\tau}$ such that each τ_0 and τ_1 were integer multiples of $1/Q$ between -1 and 1. For $Q = 32$, this results in $\mathcal{T}$ consisting of a 65×65 grid of possible values, and the optimizer seeks to find the value of $\boldsymbol{\tau} \in \mathcal{T}$ that minimizes the objective function.

[1]If the dimensions of the airborne test article and the symbol rate define a differential delay in excess of one symbol time, common practice is to insert a delay at the transmitter to force the differential delay to be less than one symbol time at the receiver.

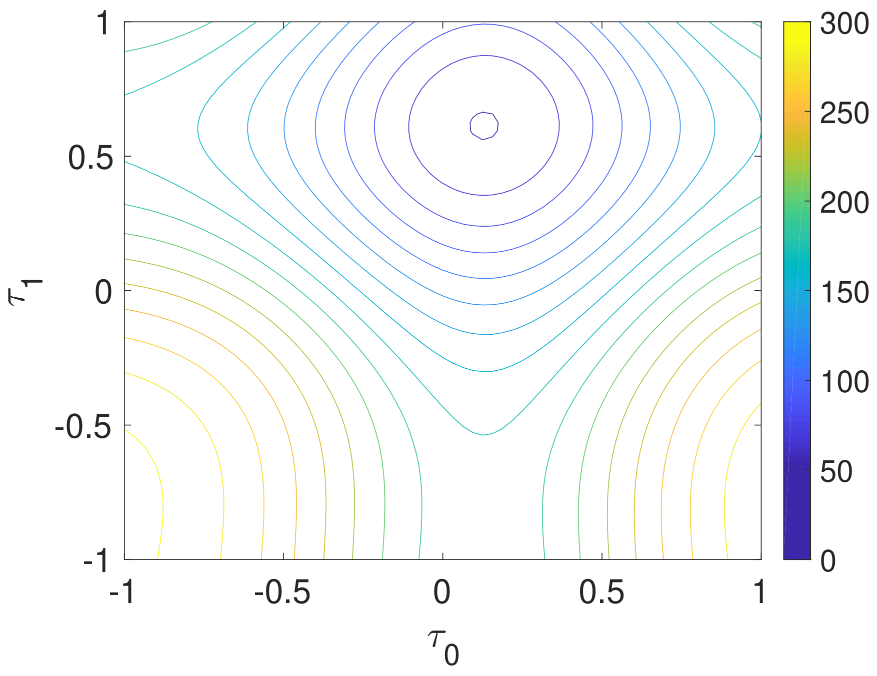

Figure 2.2: A contour plot of the objective function $F(\boldsymbol{\tau}) = \left|\left[\mathbf{I}_{N_p} - \mathbf{P}\left(\mathbf{P}^\dagger\mathbf{P}\right)^{-1}\mathbf{P}^\dagger\right]\mathbf{r}\right|^2$ for a specifically chosen $\mathbf{r}$.

This optimization problem can be written in standard form as

$$\underset{\boldsymbol{\tau}\in\mathcal{T}}{\text{minimize}} \quad f(\boldsymbol{\tau}), \tag{2.25}$$

where the objective function is

$$f(\boldsymbol{\tau}) = \left|\left[\mathbf{I}_{N_p} - \mathbf{P}\left(\mathbf{P}^\dagger\mathbf{P}\right)^{-1}\mathbf{P}^\dagger\right]\mathbf{r}\right|^2. \tag{2.26}$$

Note this optimization problem does not have any listed constraints because in this case it was more convenient to integrate the constraints into the creation of the search space.

Finally, the memory storage analysis performed in [1] can be applied directly to the situation in this work, so in addition to pre-computing values for $\mathbf{P}$, values of $(\mathbf{P}^\dagger\mathbf{P})^{-1}$ are also pre-computed and stored. The interested reader is referred to [1] for a more detailed explanation.

2.2.4 Previous Method. The objective function (2.26) is shown in Figure 2.2. For this figure, $\mathbf{r}$ was constructed using $\boldsymbol{\tau} = (0.123, 0.6)^{\intercal}$, $\mathbf{h} = (1/\sqrt{2}, -1/\sqrt{2})^{\intercal}$, and noise samples in $\mathbf{w}$ were generated assuming a signal-to-noise ratio E_b/N_0 of 10 dB. Note the obvious minimizer at approximately the value of $\boldsymbol{\tau}$ used to generate $\mathbf{r}$, showing that minimizing this objective function is equivalent to estimating $\boldsymbol{\tau}$. An optimization algorithm over the search space $\mathcal{T}$ would attempt to find the closest $\boldsymbol{\tau}$ on the grid to the true minimizer.

Finally, [1] developed an optimization algorithm used to estimate (2.23) by searching over $\mathcal{T}$. This algorithm modifies the famous simplex method created by Nelder and Mead [13] so that the simplices always operate on the grid points contained in $\mathcal{T}$. Though the algorithm does accurately estimate the timing offset, it is costlier to run than is necessary. It is clear from Figure 2.2 that the objective function demonstrates a level of convexity over the search space $\mathcal{T}$. For convex optimization problems, it is well known that gradient-based methods are optimal. This fact is noted in [1], but gradient-based methods are not explored in detail because it was hypothesized that either the direct computation or approximation of the gradient would be prohibitively expensive, or that it would be impossible to guarantee that gradient-based iterates would remain in $\mathcal{T}$.

Chapter 4 demonstrates that gradient-based optimization methods are in fact feasible for this problem. Furthermore, a gradient-based algorithm is developed that outperforms the modified Nelder-Mead approach from [1] in terms of computational complexity, while maintaining the same level of accuracy in the timing offset estimation.

Chapter 3. Search Space Enumeration for LDPC Code Optimization

As mentioned in Chapter 2, this chapter presents a novel algorithm for enumerating the discrete search space of valid degree distribution pairs for the optimization of LDPC codes. Note that this chapter is a modified version of [14], adapted to flow well with the rest of the thesis. First, the algorithm is derived and explained, and a temporal complexity analysis is performed. Next, the objective function is visualized and is shown to demonstrate a pattern over the search space that can be exploited by iterative optimization methods to solve the LDPC code optimization problem in (2.9). The chapter concludes with a discussion of how the algorithm could be utilized in future works to optimize LDPC codes differently from current methods.

3.1 Algorithm Derivation and Analysis

Using constraints (2.3) through (2.8), this section derives an algorithm to completely enumerate every valid degree distribution pair for a given block length, code rate and maximum variable and check node degrees. For convenience, the analysis is broken into steps.

(i) We show that, given a fixed number of edges in the bipartite graph and one particular valid degree distribution pair, we can find all other valid degree distribution pairs with the same number of edges.

(ii) We demonstrate that there are a minimum and maximum number of edges possible for a fixed block length, fixed code rate, and fixed maximum variable and check node degrees.

(iii) We explain a way to find one particular valid degree distribution pair given any number of edges between the minimum and maximum number possible.

The combination of these three steps yields an algorithm that can enumerate every valid degree distribution pair for a finite block length LDPC code. The section concludes with a complexity analysis for the algorithm.

3.1.1 Step 1: Finding Remaining Valid Pairs. For the first step, assume a fixed number of edges, n_e, and a given valid degree distribution pair, (λ, ρ). All other valid degree distribution pairs can be found from these assumptions. If $\hat{\lambda}$ is any other valid variable node degree distribution, then $\hat{\lambda} = \lambda + h$ for some step h.

Since equations (2.3) and (2.5) must hold for λ and $\hat{\lambda}$ (note $\lambda_0 = \hat{\lambda}_0 = 0$, so $h_0 = 0$), then

$$\sum_{i=1}^{d_v-1} h_i = 0, \tag{3.1}$$

$$\sum_{i=1}^{d_v-1} \frac{h_i}{i+1} = 0. \tag{3.2}$$

This set of constraints can be represented in matrix form as

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{d_v} \\ 1 & 1 & \cdots & 1 \end{bmatrix} h = 0, \tag{3.3}$$

$$\implies Ch = \begin{bmatrix} 1 & 0 & -\frac{2}{4} & -\frac{4}{5} & \cdots & \frac{2(3-d_v)}{d_v} \\ 0 & 1 & \frac{6}{4} & \frac{9}{5} & \cdots & \frac{3d_v-6}{d_v} \end{bmatrix} h = 0. \tag{3.4}$$

where C is the constraint matrix obtained by Gaussian elimination.

From (3.4), it is clear that for h to be in the null space of C as required, the degrees of freedom can be decreased by two, so

$$h_1 = \sum_{i=3}^{d_v-1} \frac{2(i-2)}{i+1} h_i, \tag{3.5}$$

$$h_2 = \sum_{i=3}^{d_v-1} \frac{3-3i}{i+1} h_i. \tag{3.6}$$

Next, constraint (2.7) needs to be met for both λ and $\hat{\lambda}$, so

$$0 \le \hat{\lambda}_i, \tag{3.7}$$

$$\implies 0 \le \lambda_i + h_i, \tag{3.8}$$

$$\implies -h_i \le \lambda_i. \tag{3.9}$$

These constraints can be combined with (3.5) and (3.6) to produce the matrix inequality

$$\begin{bmatrix} -\frac{2}{4} & -\frac{4}{5} & \cdots & \frac{2(3-d_v)}{d_v} \\ \frac{6}{4} & \frac{9}{5} & \cdots & \frac{3d_v-6}{d_v} \\ -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ & & \cdots & \\ 0 & 0 & \cdots & -1 \end{bmatrix} h_{3:} = Ah_{3:} \le \lambda_{1:}. \tag{3.10}$$

where the $\le$ operates element-wise. Note A is dependent on d_v, but not on λ or n_e.

Applying (3.9) and (2.7) to the given valid λ shows that $-1 \le h_i \le 1$ for all i. In addition, if λ is changed by some amount h, each h_i must be an integer multiple of $\frac{i+1}{n_e}$. This is because the number of edges connected to variable nodes of degree $i+1$ must always be an integer multiple of $i+1$, and n_e is assumed fixed.

Finally, we summarize how all other valid degree distribution pairs can be found given a fixed number of edges, n_e, and one valid pair in particular, (λ, ρ). First, for all $i \ge 3$, create a set of values, denoted S_i for h_i by taking all integer multiples of $\frac{i+1}{n_e}$ in the interval $[-1, 1]$. Use these lists to construct a set of vectors

$$H = \left\{ h \middle| h_i \in S_i, i \ge 3; h_0 = 0; h_1 = \sum_{i=3}^{d_v-1} \frac{2(i-2)}{i+1} h_i; h_2 = \sum_{i=3}^{d_v-1} \frac{3-3i}{i+1} h_i \right\}. \tag{3.11}$$

Now, remove all vectors from H that fail to satisfy (3.10) to form a new set,

$$\hat{H} = \left\{ h \middle| h \in H, Ah_{3:} \leq \lambda_{1:} \right\}. \tag{3.12}$$

The result of adding λ to all elements in $\hat{H}$ is the set of all valid variable node degree distributions for a fixed number of edges. Notice all previously done analysis applies for ρ as well as λ, so to find all valid degree distribution pairs, simply repeat the described steps with ρ in place of λ.

3.1.2 Step 2: Bounds on Number of Edges.

For the second step, assume without loss of generality a fixed block length, n, fixed code rate, r, and fixed maximum variable and check node degrees, d_v and d_c, respectively. There must be a minimum number of edges possible for a valid LDPC code. To find this, first note that there are n variable nodes and $m = n(1-r)$ check nodes. Each variable and check node must have at least two edges, as having less than this would make them a useless node for the belief propagation algorithm. So, the minimum number of edges possible based on the variable nodes is $2n$, and the minimum number possible based on the check nodes is $2m$. Since $0 < r < 1$, $m < n$ by construction, so $2n$ forms a tighter bound on the minimum number of edges. Thus, $n_e^{\min} = 2n$. In other words, the minimum possible number of edges in the Tanner graph for an LDPC code is twice the block length of the code.

The maximum number of edges possible for a valid LDPC code can be found in the following way. For the belief propagation algorithm, there cannot be more than one edge connecting any two nodes. Thus, each variable node can have at most the minimum of d_v and m edges. Similarly, each check node can have at most the minimum of d_c and n edges. So, since there are n variable nodes, the maximum number of edges possible based on the variable nodes is $n \times \min\{d_v, m\}$. Likewise, the maximum number of edges possible based on the check nodes is $m \times \min\{d_c, n\}$. The actual maximum number of edges possible, $n_e^{\max}$, is thus given by the formula $n_e^{\max} = \min\{n \times \min\{d_v, m\}, m \times \min\{d_c, n\}\}$.

```
Algorithm 1: Minimum edges pair generation
Result: Generate valid pair for min number of edges
Fix block length $n$;
Fix code rate $r$;
Fix max variable node degree $d_v$;
Fix max check node degree $d_c$;
$m := n(1-r)$;
$n_e^{\min} := 2n$;
$n_e^{\max} := \min\{n \times \min\{d_v, m\}, m \times \min\{d_c, n\}\}$;
$\lambda := (0, 2n, 0, \ldots, 0)^T$;
$\rho := (0, 2m, 0, \ldots, 0)^T$;
$k := 2m$;
while $k < 2n$ do
    $i :=$ idx of first non-zero elt. of $\rho$;
    Decrement $\rho_i$ by $i+1$;
    Increment $\rho_{i+1}$ by $i+2$;
    $k = k+1$;
end
$\lambda = \lambda / n_e^{\min}$;
$\rho = \rho / n_e^{\min}$;
return $(\lambda, \rho)$
```

3.1.3 Step 3: Obtaining a Valid Pair for Each Number of Edges.

For the third step, we explain an algorithm to find a valid degree distribution pair for every number of edges between the minimum and maximum number possible, beginning with the minimum. First, fix a block length, a code rate, and maximum variable and check node degrees. Second, use step 2 to determine the minimum and maximum number of edges for these fixed values. Then, set $\lambda = (0, 2n, 0, \ldots, 0)^T$ and $\rho = (0, 2m, 0, \ldots, 0)^T$. Set $k = 2m$ and while $k < 2n$, do the following. First, let i be the index of the first non-zero element of ρ. Second, decrement ρ_i by $i+1$. Third, increment ρ_{i+1} by $i+2$. Finally, increment k by 1. When $k = 2n$, divide both λ and ρ by $n_e^{\min}$ to obtain a valid degree distribution pair for the minimum possible number of edges. This algorithm is detailed in Algorithm 1.

To generate valid degree distribution pairs for the remaining possible number of edges, set $k = n_e^{\min}$ and while $k \leq n_e^{\max}$, follow the same steps described in the previous paragraph, but do them for both λ and ρ simultaneously. The degree distribution pair produced at each step k forms a valid degree distribution pair for k edges.

```
Algorithm 2: Valid pair generation
  Result: Generate all valid degree distribution pairs
  Run Algorithm 1;
  Initialize $(\Lambda_j)$ to store valid variable node distributions for $j$ edges;
  Initialize $(R_j)$ to store valid check node distributions for $j$ edges;
  $k := n_e^{\min}$;
  while $k \leq n_e^{max}$ do
      Construct $\hat{H}_\lambda$ as in (3.12) for $\lambda$;
      Construct $\hat{H}_\rho$ as in (3.12) for $\rho$;
      Set $\Lambda_k = \lambda + \hat{H}_\lambda$;
      Set $R_k = \rho + \hat{H}_\rho$;
      $\lambda = k\lambda$;
      $\rho = k\rho$;
      $i :=$ idx of first non-zero elt. of $\lambda$;
      Decrement $\lambda_i$ by $i + 1$;
      Increment $\lambda_{i+1}$ by $i + 2$;
      $i :=$ idx of first non-zero elt. of $\rho$;
      Decrement $\rho_i$ by $i + 1$;
      Increment $\rho_{i+1}$ by $i + 2$;
      $k = k + 1$;
      $\lambda = \lambda/k$;
      $\rho = \rho/k$;
  end
  return $((\Lambda_j), (R_j))$;
```

In summary, this section showed it is possible to completely enumerate the entire space of valid degree distribution pairs and provided analysis leading to the following algorithm for doing so. First, fix all necessary variables and run Algorithm 1. Then, use step 3 to generate one particular valid degree distribution pair for each number of edges between the minimum and maximum. Finally, use step 1 and the generated pairs to generate all valid degree distribution pairs for all possible number of edges. This algorithm is detailed in pseudocode in Algorithm 2. For a given number of edges, all valid variable node degree distributions for that number of edges and all valid check node degree distributions for that same number of edges can together form a valid pair. So, the algorithm returns two sequences of sets of valid variable and check node distributions, indexed by the number of edges.

3.1.4 Complexity Analysis. This section concludes with a complexity analysis for the number of floating-point operations (FLOPs) required for Algorithm 1 and Algorithm 2. For Algorithm 1, 8 FLOPs are required prior to the **while** loop. The **while** loop runs $2n - 2m = 2n - 2(n(1-r)) = 2nr$ times and requires 6 FLOPs each time, for a total of $12nr$ FLOPs. Finally, the divisions at the end of the algorithm require d_v and d_c FLOPs, respectively. All together, Algorithm 1 requires $12nr + d_v + d_c + 8 = \mathcal{O}(nr)$ FLOPs.

A similar, though far more complicated, analysis can be performed for Algorithm 2. Due to space considerations, we will only provide a summary of the analysis and present upper and lower bounds on the resulting complexity. The dominating contribution to the number of FLOPs required for the algorithm comes from the construction of $\hat{H}_\lambda$ and $\hat{H}_\rho$. It can be shown that for each k, the construction of these two sets requires

$$\begin{aligned} 2k \log\left(\frac{d_v d_c}{9}\right) &+ (2d_v^2 - 5d_v - 7)\frac{(2k)^{d_v-3}(3!)}{d_v!} \\ &+ (2d_c^2 - 5d_c - 7)\frac{(2k)^{d_c-3}(3!)}{d_c!} \end{aligned} \tag{3.13}$$

FLOPs. Within the **while** loop, k ranges from $n_e^{\min}$ to $n_e^{\max}$, so a lower bound on the number of FLOPs in (3.13) can be found by replacing k with $n_e^{\min} = 2n$, and an upper bound can be found by replacing k with $n_e^{\max}$, which for simplicity we will assume is nd_v. As a lower bound, constructing $\hat{H}_\lambda$ and $\hat{H}_\rho$ requires

$$\mathcal{O}\left(d_v^2\frac{(4n)^{d_v-3}}{d_v!} + d_c^2\frac{(4n)^{d_c-3}}{d_c!}\right) \tag{3.14}$$

FLOPs, while as an upper bound the construction requires

$$\mathcal{O}\left(d_v^2\frac{(2nd_v)^{d_v-3}}{d_v!} + d_c^2\frac{(2nd_v)^{d_c-3}}{d_c!}\right) \tag{3.15}$$

FLOPs.

The **while** loop runs $n_e^{\max} - n_e^{\min} = nd_v - 2n = \mathcal{O}(nd_v)$ times, so the total number of

FLOPs required for Algorithm 2 is bounded below by

$$\mathcal{O}\left(nd_v^3\frac{(4n)^{d_v-3}}{d_v!}+nd_vd_c^2\frac{(4n)^{d_c-3}}{d_c!}\right) \tag{3.16}$$

and bounded above by

$$\mathcal{O}\left(nd_v^3\frac{(2nd_v)^{d_v-3}}{d_v!}+nd_vd_c^2\frac{(2nd_v)^{d_c-3}}{d_c!}\right). \tag{3.17}$$

Note that these bounds are polynomial in the block length n, but exponential in the maximum variable and check node degrees, d_v, and d_c. As a result, the algorithm becomes computationally intractable if large maximum degrees are allowed. Fortunately, the enumeration algorithms are useful primarily for small block lengths and the low density requirement for LDPC codes ensures reasonably small d_v and d_c parameters.

3.2 Objective Function Visualization

This section provides an example for Algorithm 2. The example uses exceptionally small values for block length and maximum variable and check node degrees to facilitate visualization of the discrete nature of the space. Fix the block length $n = 100$, the code rate $r = 1/2$, the maximum variable node degree $d_v = 4$, and the maximum check node degree $d_c = 5$. Algorithm 2 is run on this example and the results are visualized for all of the corresponding pairs for $n_e = 202$ edges.

The three dimensional plot in Figure 3.1 represents the set of all valid variable node distributions for 202 edges. Since $\lambda_0 = 0$ and $d_v = 4$, there are only three possible degrees of freedom, plotted as the three axes in the plot. Algorithm 2 produces only two valid degree distributions for $n_e = 202$.

The plot shown in Figure 3.2 represents the corresponding set of all valid check node distributions. Since $d_c = 5$ and $\rho_0 = 0$, there are four degrees of freedom, but only ρ_2, ρ_3, and ρ_4 are plotted. The construction of $\hat{H}$ in (3.12) reduces the degrees of freedom to only

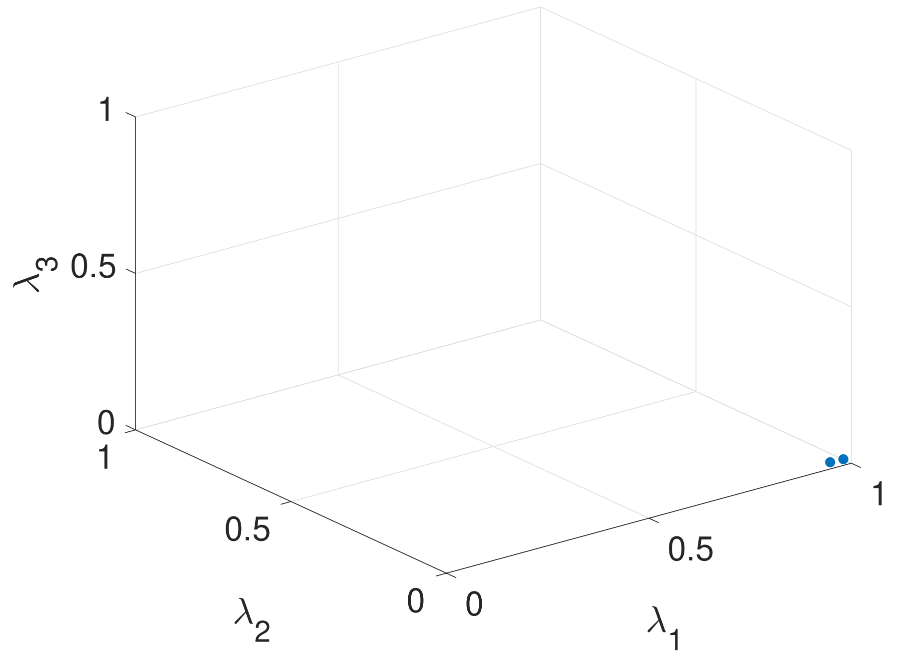

Figure 3.1: A visualization of the valid space of variable node degree distributions for $n_e = 202$. Note there are only two valid distributions, located at the bottom right of the figure.

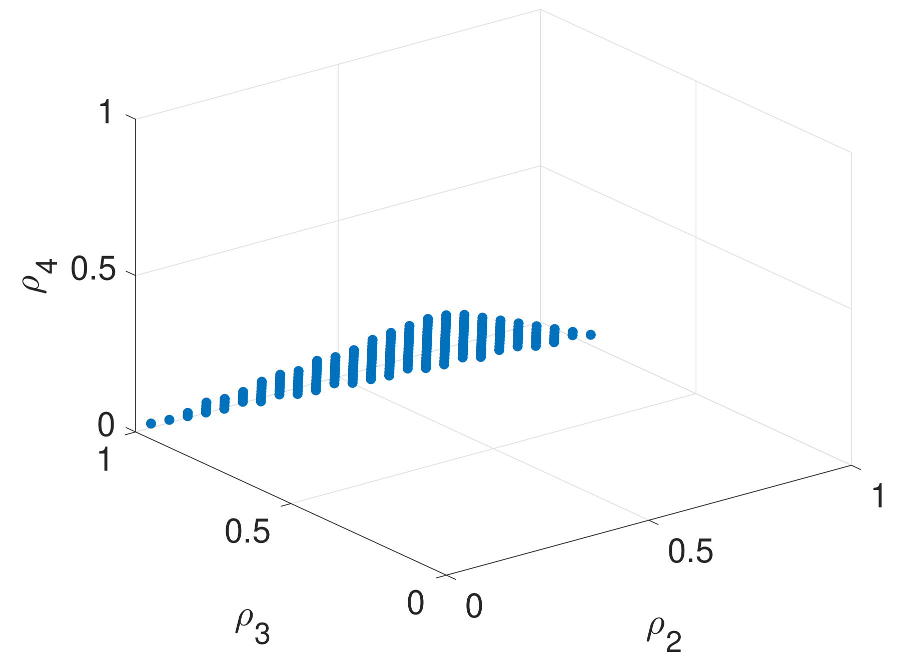

Figure 3.2: A visualization of the valid space of check node degree distributions for $n_e = 202$. Note the set forms a subset of a plane in $\mathbb{R}^4$.

two, and therefore the valid space of check node distributions is a discrete subset of a plane in $\mathbb{R}^4$. Note any combination of a variable node distribution from Figure 3.1 and a check node distribution from Figure 3.2 forms a valid degree distribution pair for $n_e = 202$.

3.3 Towards Optimization

The complete enumeration of degree distribution pairs presented in this work allows for LDPC code optimization that is both flexible and accurate at small block lengths. Asymptotic optimization methods by nature are only capable of optimizing objective functions that can be evaluated theoretically for infinite block length codes. In contrast, the enumeration of all valid pairs combined with standard discrete optimization methods, such as those found in [2], allows for the optimization of any objective function that takes as an input a valid degree distribution pair. This opens the door for the optimization of LDPC codes with respect to objective functions that utilize simulation of real codes, among other things. In addition, since the search space for these methods will consist entirely of valid degree distribution pairs of finite block length, an optimal pair obtained from such methods is guaranteed to be optimal for the finite case, rather than only for the asymptotic case.

This chapter concludes with an example of an objective function that can be optimized using our enumeration method. Again, fix the block length $n = 100$, rate $r = 1/2$, maximum variable node degree $d_v = 4$, and maximum check node degree $d_c = 5$. The goal is to find the degree distribution pair with these constraints that produces the LDPC code family that minimizes the average BER through a simulated additive white Gaussian noise (AWGN) channel at a fixed $E_b/N_0 = 0.1$ dB. Define the objective function to accept a degree distribution pair, run the simulations, and output the average BER for that pair.

Figure 3.3 and Figure 3.4 visualize the value of this objective function for every possible degree distribution pair shown in Figure 3.1 and Figure 3.2. The plot in Figure 3.3 is constructed by plotting the two degrees of freedom (h_3 and h_4) for the check node distributions against the objective function value for the left-most valid variable node distribution, while

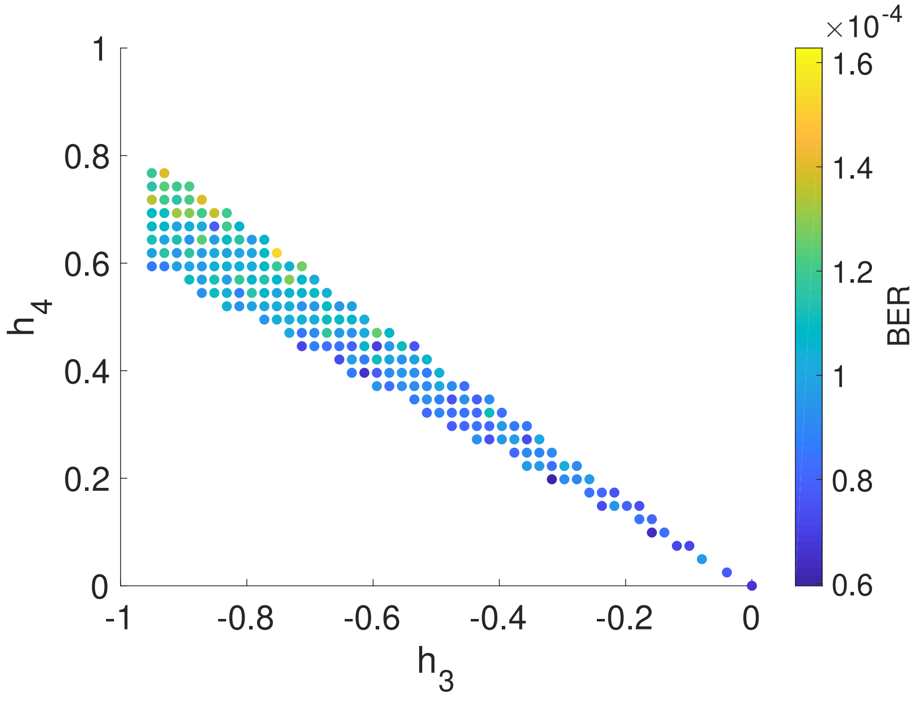

Figure 3.3: A visualization of the continuous dependence of BER on location in the space of valid check node degree distributions for the left-most variable node degree distribution.

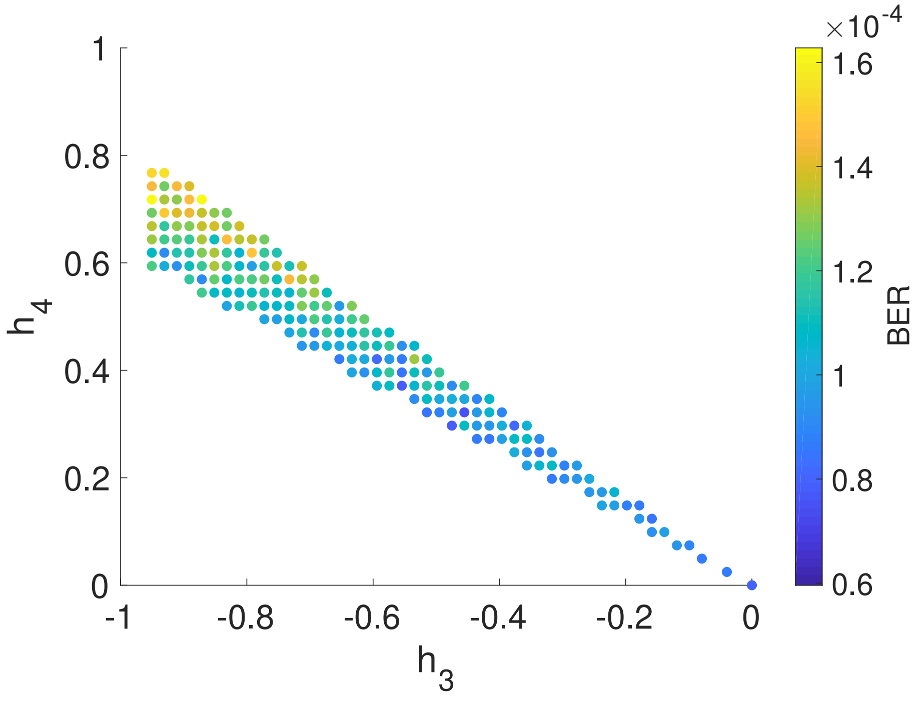

Figure 3.4: A visualization of the continuous dependence of BER on location in the space of valid check node degree distributions for the right-most variable node degree distribution.

Figure 3.4 is constructed similarly, but for the right-most valid variable node distribution. There is a clearly discernible pattern to the BER as a function of the degree distribution pair that can be exploited by discrete optimization methods to produce the optimal pair for this objective function. Note that for $n_e = 202$ like in the visualizations, the optimal pair is $\big(\lambda, \rho\big) = \big((0, \frac{196}{202}, \frac{6}{202}, 0)^T, (0, 0, \frac{24}{202}, \frac{128}{202}, \frac{50}{202})^T\big)$, which corresponds to $(h_3, h_4) = (-\frac{64}{202}, \frac{40}{202})$ for the left-most variable node degree distribution. However, a true optimization would also need to account for degree distribution pairs with $n_e \neq 202$ when optimizing the objective function.

Chapter 4. Timing Offset Estimator for the Two-Antenna Problem

As mentioned in Chapter 2, this chapter presents an improved algorithm to estimate the timing offset in order to solve the two-antenna problem. First, the algorithm is presented and analyzed, and various modifications and hyperparameters specific to this context are discussed. A complexity analysis for both the improved algorithm and the previously used modified Nelder-Mead algorithm is then performed. The chapter concludes by comparing the two algorithms and conclusively determining that the one presented in this work outperforms the previous method.

4.1 Algorithm Derivation and Analysis

This section first presents the gradient-based algorithm used to estimate (2.23). Next, the hyperparameters chosen for the algorithm are provided, as well as justification for these values. This section concludes with an analysis of the temporal complexity and memory storage requirements for both the new gradient-based algorithm and the modified Nelder-Mead algorithm from [1].

```
Algorithm 3: Generic Gradient Descent
  Result: Estimate minimizer of objective function
  Define objective function $f$;
  Define gradient $\nabla f$;
  Set initial guess $\boldsymbol{\tau}^{(0)}$;
  Fix step size $\gamma$;
  $i = 0$;
  while termination condition not met do
      $\boldsymbol{\tau}^{(i+1)} = \boldsymbol{\tau}^{(i)} - \gamma \nabla f(\boldsymbol{\tau}^{(i)})$;
      $i = i + 1$;
  end
  return $\boldsymbol{\tau}^{(i)}$
```

4.1.1 Gradient Approximation. Gradient descent is easily the most popular gradient-based optimization algorithm due to its simplicity and effectiveness. This algorithm first assumes access to the objective function, f, as well as the gradient of the objective function, ∇f. A step size, γ, and an initial guess, $\boldsymbol{\tau}^{(0)}$, are fixed in advance. The algorithm then iterates, seeking to improve its guess for the minimizer $\boldsymbol{\tau}$ with each iteration, until some predetermined termination condition is met. At each iteration, the current guess for $\boldsymbol{\tau}$ is updated according to the rule $\boldsymbol{\tau}^{(i+1)} = \boldsymbol{\tau}^{(i)} - \gamma \nabla f(\boldsymbol{\tau}^{(i)})$. The gradient of a function gives the direction of greatest increase, so its negative is the direction of greatest decrease. Thus, the gradient descent algorithm attempts to improve its guess for the function minimizer at each iteration by moving the guess some step size in the direction of the negative gradient, with the ultimate goal of converging to the true minimizer. The algorithm is shown in pseudocode in Algorithm 3.

Utilizing the gradient descent algorithm for the timing offset estimator requires access to gradient information. The appendix of [1] derives the partial derivatives of an equivalent version of (2.22) with respect to both τ_0 and τ_1, and these partial derivatives can be combined into a vector to form a gradient that could be used for the gradient descent algorithm. However, computing this gradient requires knowledge of the channel gains $\mathbf{h}$. As a result, the true gradient cannot be used in an optimization algorithm for (2.23) because the $\mathbf{h}$ is not estimated until after the optimization algorithm is run in practical systems.

Rather than define ∇f to be the true gradient, it is defined to be the function that approximates the gradient using linear interpolation as described in [2]. Let $T = \{\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \boldsymbol{\tau}_2\}$ be a set of 3 vector values for $\boldsymbol{\tau}$. Also define $\mathbf{Y} = \left[f(\boldsymbol{\tau}_1) - f(\boldsymbol{\tau}_0),\ f(\boldsymbol{\tau}_2) - f(\boldsymbol{\tau}_0)\right]^{\mathsf{T}}$. From Proposition 9.1 in [2], the set T is poised for linear interpolation if and only if the matrix $\mathbf{L} = \left[\boldsymbol{\tau}_1 - \boldsymbol{\tau}_0,\ \boldsymbol{\tau}_2 - \boldsymbol{\tau}_0\right]$ is invertible. If a given set T is poised for linear interpolation, then the gradient, denoted by $\boldsymbol{\alpha}$, can be approximated near the set T by solving the linear system $\mathbf{L}^{\mathsf{T}}\boldsymbol{\alpha} = \mathbf{Y}$.

The approximate gradient function is defined to be

$$\nabla f(\boldsymbol{\tau}) = \boldsymbol{\alpha} = \mathbf{L}^{-\intercal}\mathbf{Y}, \tag{4.1}$$

where $\mathbf{Y}$ and $\mathbf{L}$ are constructed as described using

$$T = \{\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \boldsymbol{\tau}_2\} = \{\boldsymbol{\tau}, \boldsymbol{\tau} + [0,\ 1/Q]^\intercal, \boldsymbol{\tau} + [1/Q,\ 0]^\intercal\}. \tag{4.2}$$

Note multiples of $1/Q$ are added in order to guarantee that $\boldsymbol{\tau}_i \in \mathcal{T}$ for all i.

4.1.2 Modification and Hyperparameters. In addition to the gradient approximation, a few additional adjustments are made to the standard gradient descent algorithm for it to function appropriately as an estimator under the conditions of the two-antenna problem. First, since values for $\mathbf{P}$ and $(\mathbf{P}^\dagger\mathbf{P})^{-1}$ are only pre-computed and stored for $\boldsymbol{\tau} \in \mathcal{T}$, it is necessary to ensure that all of the iterates $\boldsymbol{\tau}^{(i)}$ remain in the search space $\mathcal{T}$. This is accomplished primarily by rounding both components of $\boldsymbol{\tau}^{(i)}$ to the nearest multiple of $1/Q$ at each iteration. For the edge case where a component of one of the iterates is less than -1, that component is shifted to -1, and similarly if the component is greater than 1. Finally, in the case where one of the elements used to construct T in the evaluation of ∇f has a component less than -1 or greater than 1, the same component of all elements of T is shifted to -1 or 1, respectively, before constructing $\mathbf{Y}$.

Next, a reasonable termination condition must be established for the **while** loop. There are a few termination conditions often used in practice for gradient descent algorithms. For example, it is common to terminate the loop when the evaluated gradient drops below a fixed, small, positive threshold, implying close proximity to a critical point of the objective function. However, since the algorithm can only approximate the gradient on the discrete search space $\mathcal{T}$, continuous values of the gradient cannot be obtained, making it unlikely to find a threshold value that works universally for objective functions constructed with any arbitrary received vector $\mathbf{r}$. It is also common to terminate the loop when the absolute value of the difference between the objective function evaluated at consecutive iterates drops below a fixed, small, positive threshold. Unfortunately, this condition has the same pitfall as the previous one, as continuous values of the objective function are unattainable over a discrete search space.

The algorithm presented in this work uses a termination condition different from, but inspired by the examples in the previous paragraph. Rather than terminate when consecutive iterates are close enough together to yield similar objective function evaluations, it terminates the loop when an iterate is identical to a previous iterate. If it were not terminated at that point, the discreteness of the space would force the iterates to cycle. This termination condition would almost certainly never be met for a continuous search space, but the condition works exceptionally well for the discrete search space $\mathcal{T}$. It also means that the approximated gradient is close enough to 0 to prevent the next iterate from stepping too far in the search space, likely implying proximity to a critical point. Finally, an additional termination condition is added that is typical of gradient descent algorithms. To avoid extremely rare cases where the iterates may wander for too long without repeating, the algorithm terminates after a previously fixed maximum number of iterations are reached.

The first hyperparameter that must be fixed for the gradient descent algorithm is the initial guess $\boldsymbol{\tau}^{(0)}$. It seems natural to set the initial guess to be the value in the search space that is most likely *a priori* to represent the true timing offset. Since the true value of the timing offset for both τ_0 and τ_1 is symmetrically distributed with mean 0, set $\boldsymbol{\tau}^{(0)} = [0\ 0]^\intercal$.

The second, and far more complicated, hyperparameter to tune is the step size γ. For a discrete search space, using too small of a value for γ would result in very slow convergence, or at worst would prevent the algorithm from ever leaving the initial guess. On the other hand, too large of a value for γ would result in far more error in the timing offset estimation. The goal is to find the value of γ that results in the convergence of the gradient descent algorithm in the fewest number of iterations. A typical grid search is used to accomplish this.

For the grid search algorithm, first fix a few values that will be held constant for the estimators regardless of the true values of $\boldsymbol{\tau}$ and $\mathbf{h}$. Set $N = 4$ samples/symbol, $Q = 32$ parts/sample, and $L_p = 124$ pilot bits, meaning $N_p = 248$ samples. The approximate gradient descent algorithm is then simulated for each γ out of a carefully selected set of possible values under various values of E_b/N_0 (dB) and true $\mathbf{h}$.

Each simulation first randomly selects 1000 true $\boldsymbol{\tau}$ values such that $-0.125 \leq \tau_0 \leq 0.125$ and $-1 \leq \tau_1 \leq 1$. Note it could have equivalently selected $\boldsymbol{\tau}$ such that $-1 \leq \tau_0 \leq 1$ and $-0.125 \leq \tau_1 \leq -0.125$. Next, simulated values for the received vector of samples $\mathbf{r}$ are obtained using (2.16) with the given true value of $\mathbf{h}$ and values in $\mathbf{w}$ sampled from a complex-valued Gaussian distribution with mean zero and variance corresponding to the fixed value of E_b/N_0 (dB). For each of these simulations, the average number of objective function evaluations required for the approximate gradient descent algorithm to converge is recorded, as well as if the algorithm is "successful."

Definition 4.1. The timing estimator is *successful* if either the Euclidean distance between the estimated $\hat{\boldsymbol{\tau}}$ and the true $\boldsymbol{\tau}$ is at most $\frac{\sqrt{2}}{Q}$, or $f(\hat{\boldsymbol{\tau}}) \leq f(\boldsymbol{\tau})$. The first success condition guarantees that the estimated $\hat{\boldsymbol{\tau}}$ is one of the four nearest grid points to the true $\boldsymbol{\tau}$, while the second condition takes into account the possibility that, due to noise, the minimizer of the objective function may not be exactly the true value of $\boldsymbol{\tau}$, in which case the best the estimator can do is find the minimizer over the search space.

Figure 4.1 plots the average number, over the 1000 simulations, of objective function evaluations required for convergence against various possible step sizes for different values of E_b/N_0 (dB). Similarly, Figure 4.2 plots the number of successful estimations out of the 1000 simulations for these same step sizes and values of E_b/N_0. These simulations were run for various true values of $\mathbf{h}$, but to avoid redundancy only plots for $\mathbf{h} = [1/\sqrt{2} \;\; -1/\sqrt{2}]^\intercal$ are shown.

Note that different values of $\mathbf{h}$ produce similarly shaped plots, but over different values

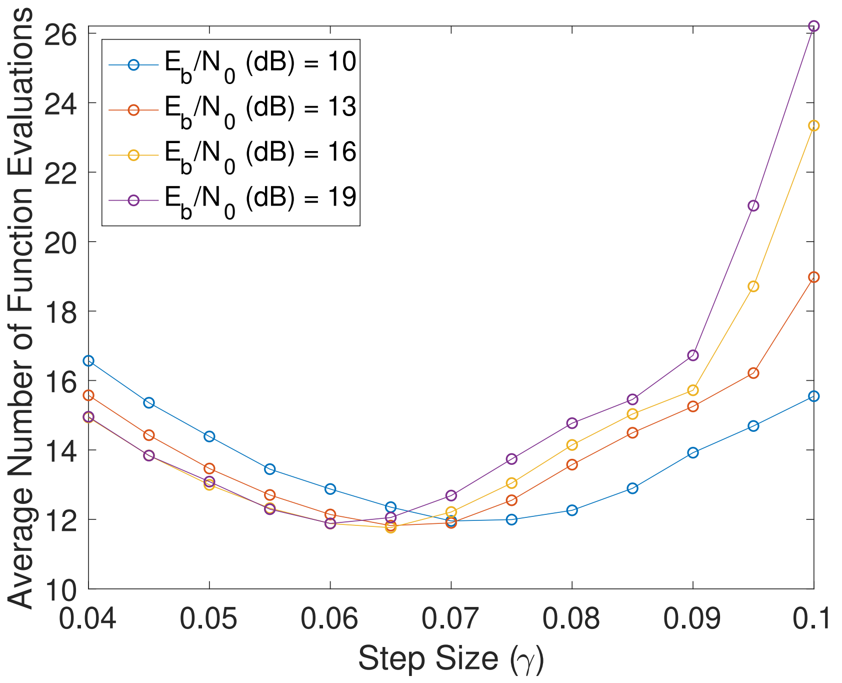

Figure 4.1: Plot of the average number of objective function evaluations necessary for convergence of the approximate gradient descent algorithm for various step sizes assuming $\mathbf{h} = [1/\sqrt{2} \;\; -1/\sqrt{2}]^\intercal$.

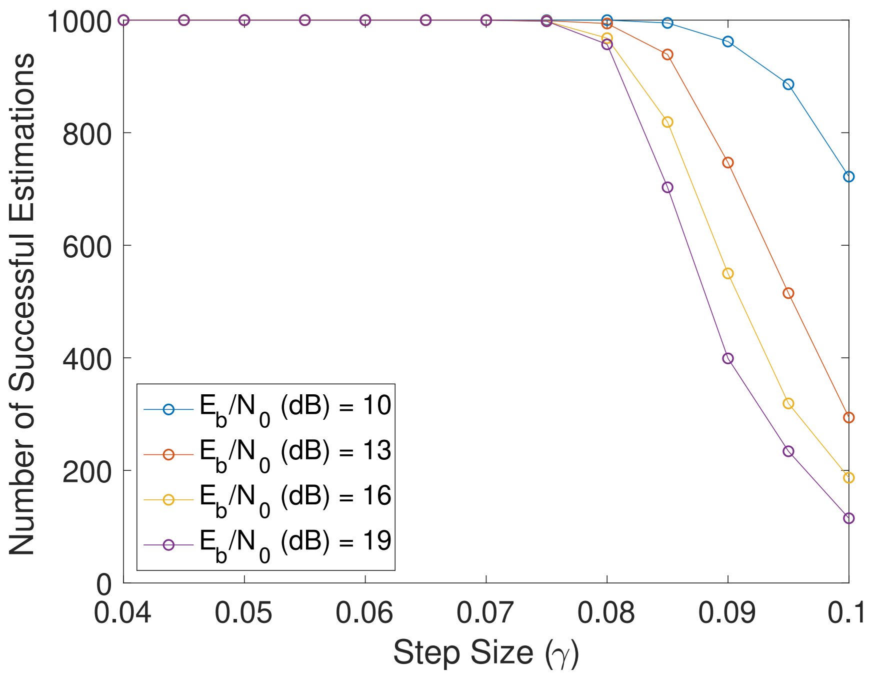

Figure 4.2: Plot of the number of successful timing offset estimations out of the 1000 simulations performed for various step sizes assuming $\mathbf{h} = [1/\sqrt{2} \;\; -1/\sqrt{2}]^\intercal$.

of γ. These differences are insignificant for values of $\mathbf{h}$ of the same magnitude (as measured by the norm), but have a scaling effect on the values of γ for $\mathbf{h}$ of different magnitudes. For example, the plot shown in Figure 4.1 assumes $||\mathbf{h}|| = 1$. A value of $\mathbf{h}$ such that $||\mathbf{h}|| = 2$ results in an equivalent plot, but with all step size values multiplied by a factor of $||\mathbf{h}||^2 = 4$.

We seek a value of γ such that successful convergence is obtained with the fewest number of function evaluations over arbitrary values of $\mathbf{h}$ and E_b/N_0. As the range of values for γ varies depending on $||\mathbf{h}||^2$, some form of normalization is required in order to compare different potential values. It is impossible to know *a priori* the magnitude of $\mathbf{h}$, but this quantity can be estimated from the known quantity $\mathbf{r}$ since

$$||\mathbf{r}|| = ||\mathbf{Ph} + \mathbf{w}|| \leq ||\mathbf{Ph}|| + ||\mathbf{w}|| \approx ||\mathbf{Ph}|| \tag{4.3}$$

for values of E_b/N_0 in the considered range and

$$||\mathbf{Ph}|| \approx ||\mathbf{P}||||\mathbf{h}|| \tag{4.4}$$

for typical values of $\mathbf{P}$ and $\mathbf{h}$. A reasonably accurate approximation for $||\mathbf{h}||$ is obtained by simulating values of $\mathbf{w}$ for various E_b/N_0, producing $\mathbf{r} = \mathbf{Ph} + \mathbf{w}$, then using a least squares approximation to solve $||\mathbf{r}|| = m||\mathbf{h}|| + b$ for m and b. These values are pre-computed and stored in order to appropriately normalize the step size.

The simulations shown in Figure 4.1 and Figure 4.2, as well as simulations using additional true values of $\mathbf{h}$, indicate virtually 100% accuracy using an unnormalized $\gamma \leq 0.06$. They also indicate that on average the fewest number of function evaluations are required when $0.06 \leq \gamma \leq 0.08$, making $\gamma = 0.06$ a natural choice for the unnormalized step size. The step size is then normalized based on the received $\mathbf{r}$ so that

$$\gamma = \frac{0.06}{||\mathbf{h}||^2} = \frac{0.06}{((||\mathbf{r}|| - b)/m)^2}, \tag{4.5}$$

where the unnormalized step size 0.06, b, and m are all pre-computed and stored. To conclude the step size discussion, it is worth mentioning that for different values of N, Q, and L_p, the best choice for the unnormalized γ is unlikely to remain 0.06, so the step size analysis would have to be redone.

Algorithm 4: Modified Gradient Descent

```
Result: Estimate minimizer of objective function
Define objective function $f$ using (2.23);
Define approximate gradient $\nabla f$ using (4.1);
Set initial guess $\boldsymbol{\tau}^{(0)} = [0\ 0]^\intercal$;
Fix step size $\gamma = \frac{0.06}{((||\mathbf{r}||-b)/m)^2}$;
Initialize an empty set $S$;
Fix maximum number of iterations $max\_iters$;
$i = 0$;
while $i < max\_iters$ & $\boldsymbol{\tau}^{(i)} \notin S$ do
    Add $\boldsymbol{\tau}^{(i)}$ to $S$;
    $\boldsymbol{\tau}^{(i+1)} = \text{round}\big(\boldsymbol{\tau}^{(i)} - \gamma \nabla f(\boldsymbol{\tau}^{(i)})\big)$;
    $i = i + 1$;
end
$V = \{\boldsymbol{\tau}^{(i)}, \boldsymbol{\tau}^{(i)} + [0,\ 1/Q]^\intercal, \boldsymbol{\tau}^{(i)} + [1/Q,\ 0]^\intercal, \boldsymbol{\tau}^{(i)} + [1/Q,\ 1/Q]^\intercal\}$;
return $\text{argmin}_{v \in V} f(v)$
```

Algorithm 4 summarizes the gradient-based algorithm used to estimate the timing offset, including the various modifications to standard gradient descent and the hyperparameter tuning. First, the objective function f is defined using (2.23) and the approximate gradient is defined using (4.1). Next, fix the initial guess $\boldsymbol{\tau}^{(0)}$ and step size γ as described, before initializing a set to hold the iterates $\boldsymbol{\tau}^{(i)}$, and fixing a maximum number of iterations. Then, iterate by adding $\boldsymbol{\tau}^{(i)}$ to the set and setting the next iterate based on the previous iterate, the step size, and the approximate gradient evaluation.

To conclude the algorithm, define

$$V = \{\boldsymbol{\tau}^{(i)}, \boldsymbol{\tau}^{(i)} + [0,\ 1/Q]^\intercal, \boldsymbol{\tau}^{(i)} + [1/Q,\ 0]^\intercal, \boldsymbol{\tau}^{(i)} + [1/Q,\ 1/Q]^\intercal\}, \tag{4.6}$$

where i corresponds to the iteration count at the end of the loop. Since the gradient at step i is approximated using the first three elements of V, the approximation is likely to be most accurate for some value of $\boldsymbol{\tau}$ roughly equal to the average of these three values. So, V consists of the four elements of the search space closest to the most accurate approximation of the critical point, meaning any $\boldsymbol{\tau} \in V$ could represent the search space value closest to the critical point of the objective function. Thus, compute the objective function value for all V and return the $\boldsymbol{\tau}$ with the smallest objective function value.

4.1.3 Complexity Analysis.

Section 4.1 concludes with an analysis of the temporal and spatial complexity of the approximate gradient descent algorithm, as well as the modified Nelder-Mead algorithm from [1] for comparison. These complexities are expressed as functions of the number of samples/symbol N, the number of pilot symbols n_s (recall from Chapter 2 that n_s is equal to half of the number of pilot bits L_p) , and the quantization parameter Q. Note that the practical implementation of the timing estimator occurs directly in hardware, where there is a large difference between the complexity of a real addition/subtraction and a real multiplication/division. For this reason, rather than lump them together by comparing the number of FLOPs between the algorithms, the real additions/subtractions and real multiplications/divisions are compared separately.

Objective Function. As both algorithms require objective function evaluations, we first count the number of real multiplications/divisions, additions/subtractions, and the amount of memory storage required for an individual objective function evaluation. Note that the memory storage analysis performed in [1] still applies to the situation in this work, so assume that values for $\mathbf{P}$ and $(\mathbf{P}^{\dagger}\mathbf{P})^{-1}$ are pre-computed and stored for all $\boldsymbol{\tau} \in \mathcal{T}$.

A simple count shows that each function evaluation requires

$$12N^2n_s^2 + 20Nn_s \tag{4.7}$$

real multiplications/divisions, and

$$10N^2n_s^2 + 14Nn_s - 1 \tag{4.8}$$

real additions/subtractions. Rather than store the entire matrix $\mathbf{P}$ for all $\boldsymbol{\tau} \in \mathcal{T}$, a column of $\mathbf{P}$ is stored for each possible τ_i, and these columns are combined into $\mathbf{P}$ for a given $\boldsymbol{\tau}$. So, to store values of P for all quantized τ requires storing

$$(2Q + 1)(4Nn_s) = 8QNn_s + 4Nn_s \tag{4.9}$$

real values. In addition, the memory storage required for $(P^{\dagger}P)^{-1}$ is

$$8(2Q + 1)^2 = 32Q^2 + 32Q + 8 \tag{4.10}$$

real values. Combining (4.9) and (4.10), this amounts to a total memory storage requirement of

$$32Q^2 + 32Q + 8QNn_s + 4Nn_s + 8 \tag{4.11}$$

real values.

Approximate Gradient Descent. The approximate gradient descent algorithm requires real multiplications/divisions, real additions/subtractions, and memory storage above and beyond what is required for every objective function evaluation. Prior to the **while** loop, this algorithm requires the computation of the initial step size, three objective function evaluations, and one gradient approximation. The initial step size computation requires

$$Nn_s + 3 \tag{4.12}$$

multiplications/divisions, and

$$Nn_s \tag{4.13}$$

additions/subtractions. The requirements for the three objective function evaluations were already discussed. Finally, the gradient approximation involves solving a small system of linear equations and requires approximately 18 real multiplications/divisions and 5 real additions/subtractions. All together, the portion of the algorithm before the **while** loop requires

$$(Nn_s + 3) + 3(12N^2n_s^2 + 20Nn_s) + 18 = 36N^2n_s^2 + 61Nn_s + 21 \tag{4.14}$$

real multiplications/divisions, and

$$(Nn_s) + 3(10N^2n_s^2 + 14Nn_s - 1) + 5 = 30N^2n_s^2 + 43Nn_s + 2 \tag{4.15}$$

real additions/subtractions.

Next, each iteration of the **while** loop requires 3 objective function computations, one gradient approximation, and some overhead, like computing the next step size/iterate. Together, these combine for

$$3(12N^2n_s^2 + 20Nn_s) + (18) + (5) = 36N^2n_s^2 + 60Nn_s + 23 \tag{4.16}$$

real multiplications/divisions, and

$$3(10N^2n_s^2 + 14Nn_s - 1) + (5) + (5) = 30N^2n_s^2 + 42Nn_s + 9 \tag{4.17}$$

real additions/subtractions per-iteration.

After the **while** loop portion, this algorithm evaluates one more objective function value, and has a little bit of additional overhead. Combining these, we have

$$12N^2n_s^2 + 20Nn_s + 2 \tag{4.18}$$

real multiplications/divisions, and

$$10N^2n_s^2 + 14Nn_s + 1 \tag{4.19}$$

real additions/subtractions.

Finally, in addition to the values stored for the objective function computations, the computation of the step size requires the storage of 3 more real values, bringing the total amount of memory storage for this algorithm to

$$32Q^2 + 32Q + 8QNn_s + 4Nn_s + 11 \tag{4.20}$$

real values.

Modified Nelder-Mead. Similar to the approximate gradient descent algorithm, the modified Nelder-Mead algorithm from [1] requires some additional real multiplications/divisions, real additions/subtractions, and memory storage on top of what is required for the objective function computation. The analysis of that algorithm is provided here as well for the sake of comparison. Prior to any iterations, the modified Nelder-Mead algorithm builds the initial simplex, requiring 3 objective function evaluations and 3 additions. In total, the pre-iteration requirement for this algorithm is

$$3(12N^2n_s^2 + 20Nn_s) = 36N^2n_s^2 + 30Nn_s \tag{4.21}$$

real multiplications/divisions, and

$$3(10N^2n_s^2 + 14Nn_s - 1) + (3) = 30N^2n_s^2 + 42Nn_s + 2 \tag{4.22}$$

real additions/subtractions.

Each iteration of the modified Nelder-Mead algorithm requires one objective function evaluation and one simplex manipulation. There are a variety of types of simplex manipulations that could occur, and each has a slightly different (though admittedly insignificant) complexity. Using the requirements for the worst case yields

$$12N^2n_s^2 + 20Nn_s + 7 \tag{4.23}$$

real multiplications/divisions, and

$$10N^2n_s^2 + 14Nn_s + 5 \tag{4.24}$$

real additions/subtractions for each iteration.

Strictly speaking, the post-iteration portion of the algorithm only contributes 2 multiplications. However, we choose to include the switch from the standard to the unit simplex here as well, since this is only performed once during the algorithm. So, this step requires 4 multiplications/divisions, and 8 additions/subtractions.

The modified Nelder-Mead algorithm requires no memory storage other than what is required for the objective function computation. So the requirement is

$$32Q^2 + 32Q + 8QNn_s + 4Nn_s + 8 \tag{4.25}$$

real values.

4.2 Comparison to Previous Method

This section compares the approximate gradient descent optimization method proposed in this work with the modified Nelder-Mead method used previously. The accuracy and variance of the estimators are compared first. The section concludes with a comparison of the computational complexity required for the convergence of each of the two methods.

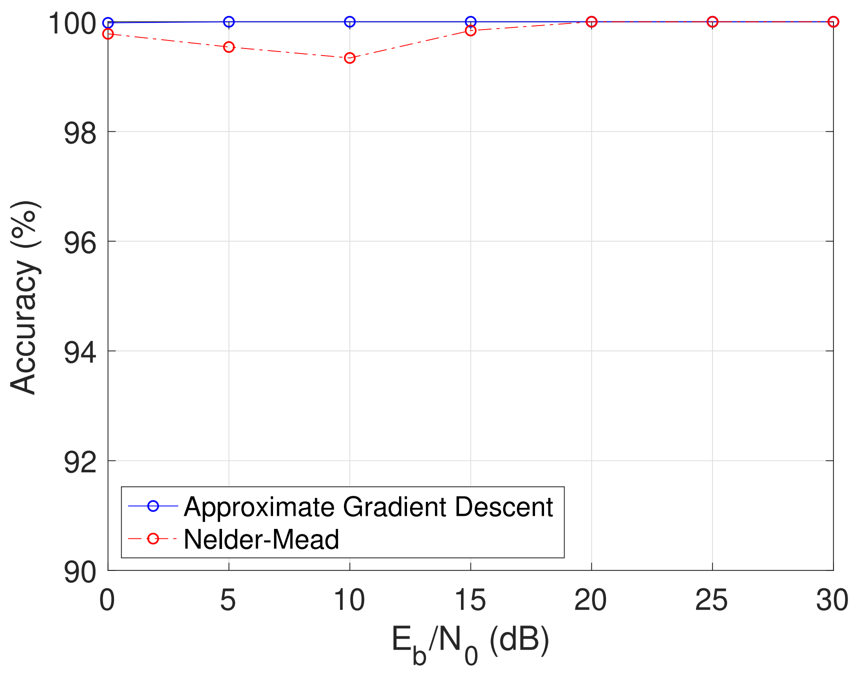

Figure 4.3: Accuracy (as measured by the percentage of successful estimates) of both the approximate gradient descent and modified Nelder-Mead algorithms as functions of E_b/N_0 (dB).

4.2.1 Accuracy and Variance. The accuracy of the two algorithms is first compared, measured as the percentage of successful timing estimations, where a successful estimation is defined as in Definition 4.1. As it is obviously impossible to simulate the accuracy of the estimator for all possible true values of $\boldsymbol{\tau}$, a test case is simulated instead. This allows for the variance of the two estimators to be compared with the Cramér-Rao bound (CRB), a lower bound on the variance dependent on the true $\boldsymbol{\tau}$. The true $\boldsymbol{\tau}$ is set to $[0.123,\ 0.6]^\intercal$, a value chosen to not be on the grid represented by the search space in order to avoid artificially good results. Then 5000 noise vectors $\mathbf{w}$ are randomly generated for various values of E_b/N_0, the corresponding values of $\mathbf{r}$ are constructed, and the timing offset is estimated with both algorithms for each $\mathbf{r}$. At each simulated value of E_b/N_0, the percentage of successful estimates are recorded. The results are plotted in Figure 4.3. Note that since the channel estimator is simply a least-squares approximation based on the result of the timing estimator, its accuracy is primarily a function of the timing estimator accuracy.

Figure 4.3 shows that the approximate gradient descent algorithm achieves virtually 100 percent accuracy for all simulated values of E_b/N_0. On the other hand, while the modified Nelder-Mead algorithm still obtains above 99 percent accuracy for all simulated values, the accuracy is noticeably lower than the approximate gradient descent algorithm for smaller values of E_b/N_0.

Another quantity of interest is the variance of the estimators. To compare the variance between the two algorithms, the same simulations are run as those run for the accuracy. In addition to recording the number of successes at each E_b/N_0, the simulated estimator variance is recorded for each algorithm for six real-valued quantities: τ_0, τ_1, and both the real and imaginary parts of h_0 and h_1. The computed variances of τ_0 and τ_1 are summed to estimate the error variance for the timing estimator, while the computed variance of the real and imaginary parts of h_0 and h_1 are summed to estimate the error variance for the channel estimator. Finally, the CRB on the variance of each estimator is computed for reference. The Fisher information matrix necessary for the computation of the CRB for each estimator is derived in the Appendix of [1], except that it adds an additional row and column to account for the frequency estimator. Since this work assumes the frequency offset is zero, only the upper-left 6×6 submatrix of the derived Fisher information matrix is used for the computation of the CRB.

The results of these simulations are seen in Figure 4.4 and Figure 4.5. The timing estimator variance in Figure 4.4 shows that both algorithms achieve roughly the same variance, and that this variance is close to the CRB for $E_b/N_0 \leq 20$ dB. At higher values of E_b/N_0, the variance reaches a lower bound due to the quantization of the search space. The channel estimator variance in Figure 4.5 has similar characteristics to the timing estimator variance, except that it does not face the same quantization issue, so the simulated variance continues to decrease for $E_b/N_0 > 20$ dB.

4.2.2 Convergence Complexity.

This subsection compares the computational complexity required for convergence for each of the two algorithms. Assume, as has been done

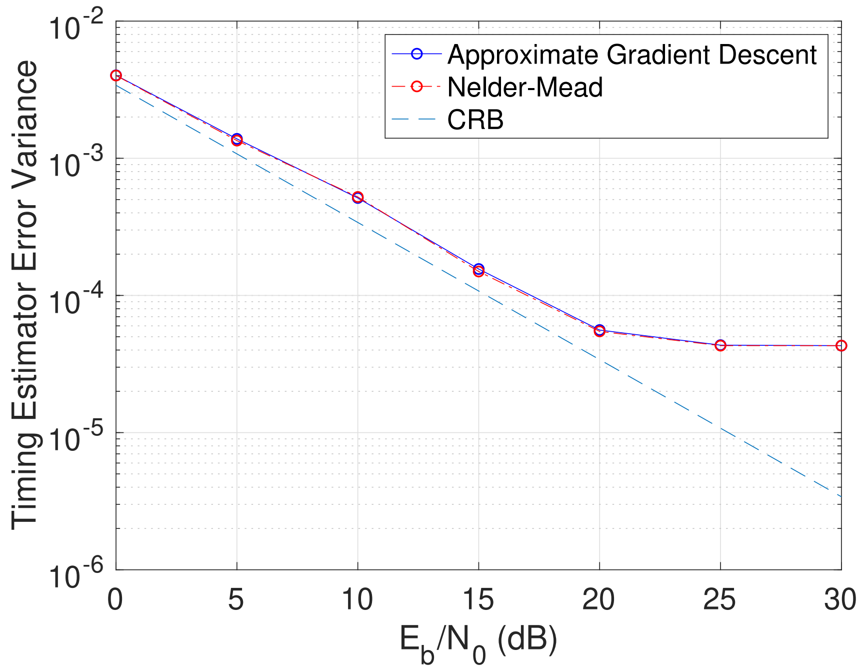

Figure 4.4: Error variance for the timing estimator for both algorithms, as well as the CRB for this variance, as a function of E_b/N_0 (dB).

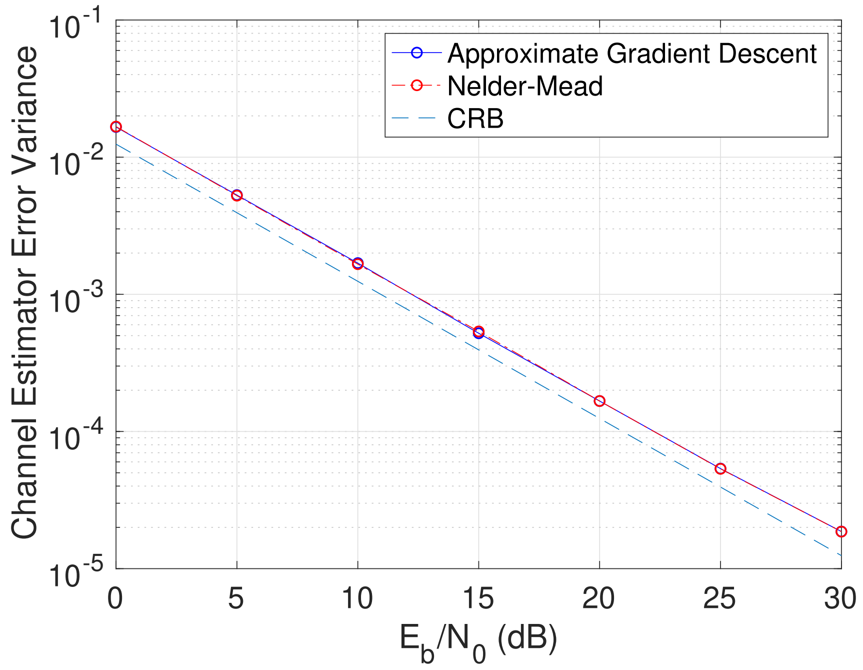

Figure 4.5: Error variance for the channel estimator for both algorithms, as well as the CRB for this variance, as a function of E_b/N_0 (dB).

up to this point, that $N = 4$ samples/symbol, $n_s = 62$ pilot symbols, and $Q = 32$ quantized parts per symbol. First, using the complexity equations for each algorithm from Section 4.1.3, we build a table of the number of real multiplications/divisions, real additions/subtractions, and real values of memory storage necessary for each algorithm under these assumptions.

Algorithm Complexity Comparison		
	Gradient Descent	Nelder-Mead
Average Number of Iterations		
Iterations	5.229	35.117
Multiplications/Divisions		
Pre-Iteration	2,229,293	2,221,584
Per-Iteration	2,229,047	743,015
Post-Iteration	743,010	4
Avg Total Cost	14,628,088	28,314,133
Additions/Subtractions		
Pre-Iteration	1,855,786	1,855,538
Per-Iteration	1,855,545	618,517
Post-Iteration	618,513	8
Avg Total Cost	12,177,025	23,576,079
Memory Storage		
Memory Storage	98,283	98,280

Table 4.1: Comparison of the number of iterations, real multiplications/divisions, additions/subtractions, and memory storage required for various portions of the two algorithms. The units for numerical values in the table are "number of iterations" for the first section, "real operations" for the next two sections, and "real values to store" for the memory section.

Equations (4.14), (4.16), and (4.18) give the number of real multiplications/divisions performed by Algorithm 4 prior to the **while** loop, at each iteration of the **while** loop, and after the **while** loop, respectively. Equations (4.15), (4.17), and (4.19) provide the same information for real additions/subtractions. For the modified Nelder-Mead algorithm, (4.21), (4.23), and the additional 4 mentioned in the text give the number of real multiplications/divisions, while (4.22), (4.24), and the additional 8 mentioned in the text give the number of real additions/subtractions. Finally, (4.20) and (4.25) provide the memory storage requirements for approximate gradient descent and modified Nelder-Mead, respectively. We plug in our values for N, n_s, and Q, into each of these equations, and summarize the results in Table 4.1.

It is immediately apparent that both the per-iteration and post-iteration complexity is significantly higher for approximate gradient descent than for modified Nelder-Mead. The per-iteration difference can be easily explained by the fact that one iteration of gradient descent requires three objective function evaluations, while one iteration of modified Nelder-Mead only requires one. The difference in post-iteration complexity is due to the additional function evaluation necessary for approximate gradient descent discussed at the end of Section 4.1.2.

Given the computational complexity comparison between the two algorithms, it is clear that for approximate gradient descent to outperform modified Nelder-Mead, it must be able to converge to the timing estimate in roughly three times fewer iterations. To test this, we fix $E_b/N_0 = 10$ (dB) and we fix a set

$$H = \{[1,\ -1]^\intercal/\sqrt{2},\ [2,\ -2]^\intercal/\sqrt{2}, [0.5,\ -0.5]^\intercal/\sqrt{2},\ [1,\ -2]^\intercal/\sqrt{2},\ [2,\ -1]^\intercal/\sqrt{2}\} \tag{4.26}$$

of values for $\mathbf{h}$. Then for each true $\mathbf{h} \in H$, 2500 true values of $\boldsymbol{\tau}$ are drawn uniformly such that $-0.125 \leq \tau_0 \leq 0.125$ and $-1 \leq \tau_1 \leq 1$, and values of $\mathbf{r}$ are constructed using (2.16) with the noise $\mathbf{w}$ drawn according to the assumption for E_b/N_0. On each constructed $\mathbf{r}$, $\boldsymbol{\tau}$ is estimated using both the approximate gradient descent algorithm and the modified Nelder-Mead algorithm, and the number of real additions/subtractions and real multiplications/divisions required for convergence are counted for both algorithms.

A histogram comparing the results for real additions/subtractions is shown in Figure 4.6, and a histogram comparing the results for real multiplications/divisions is shown in Figure 4.7. Despite the fact that the approximate gradient descent algorithm requires more operations per-iteration and post-iteration than does modified Nelder-Mead, the total number of iterations required is reduced significantly enough that the total number of operations necessary for convergence is actually much lower.

It is commonly known that gradient-based optimizers work best for convex optimization problems, and this case is no exception. The approximate gradient descent algorithm is far

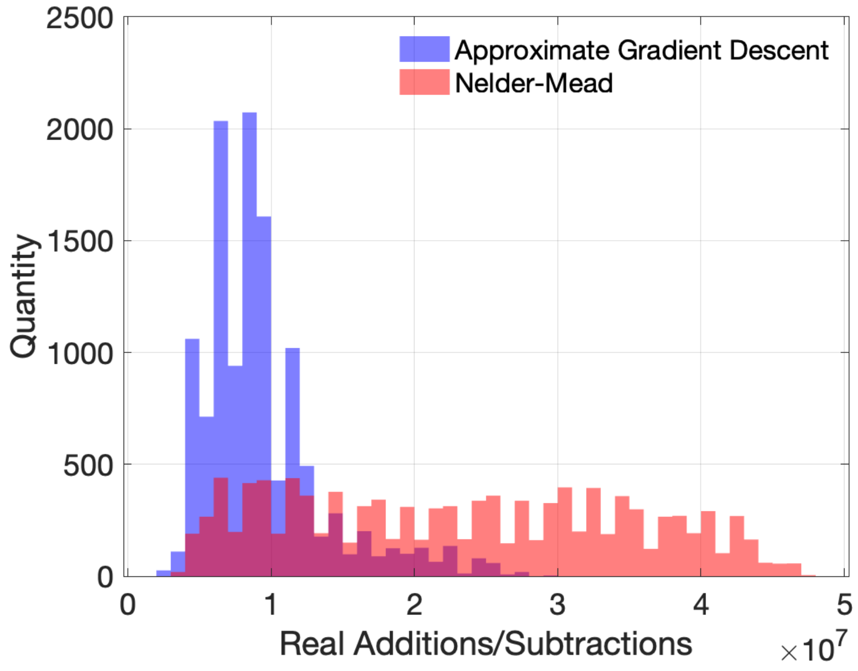

Figure 4.6: Histogram of the number of real additions/subtractions required for convergence for both algorithms.

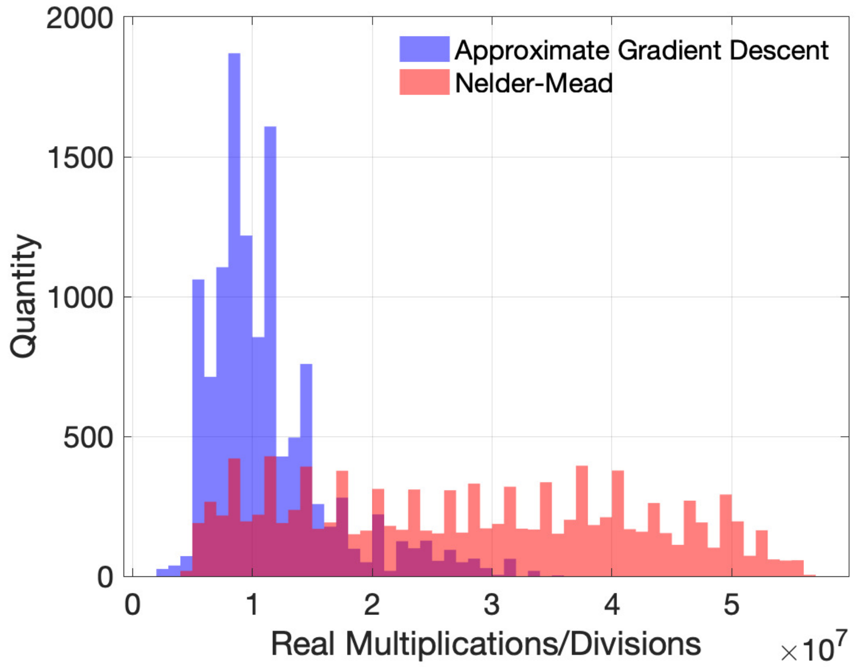

Figure 4.7: Histogram of the number of real multiplications/divisions required for convergence for both algorithms.

less computationally expensive than the modified Nelder-Mead algorithm used previously, and these improvements can be made without sacrificing the accuracy or variance of the estimator. Thus, it is clear that the algorithm set forth in this chapter is preferred over the previous algorithm.

Chapter 5. Conclusion

This chapter concludes the thesis. The contributions of this work are summarized, and the major results are highlighted. Finally, a discussion of potential future work is provided.

5.1 Summary of Major Results

This work focused on applications of mathematical optimization to digital communications and signal processing. Algorithms were developed for the purpose of solving specific optimization problems related to the channel encoder/decoder and the modulator/demodulator within a communications system.

For the channel encoder/decoder, an algorithm was developed and analyzed for completely enumerating the valid space of degree distribution pairs for LDPC codes of finite block length. The necessary mathematical analysis to demonstrate the validity and complexity of the method was performed, and the algorithm was run on an example to visualize the results.

For the modulator/demodulator, an improved optimization method for the timing estimator in the two-antenna problem was developed and analyzed. This gradient-based method refuted the claim that gradient information could not be used in this application. The design choices and hyperparameters for the algorithm were explored in detail, and a complexity analysis of the algorithm was performed. Simulations comparing the new algorithm with the previous algorithm showed a significant improvement in the speed of the algorithm without sacrificing the accuracy.

5.2 Future Work

Future work related to both developed algorithms is necessary for the contributions of this work to realize their full potential. For the finite block length LDPC code space enumeration algorithm, the obvious next step is to develop an objective function and corresponding optimization method that searches over the discrete space generated by the enumeration algorithm for the degree distribution pair which provides the best objective function value. If the developed objective function is comparable to an objective function commonly used for asymptotic LDPC code optimization methods, it would be interesting to see if the discrete optimization method utilizing the enumeration algorithm could outperform an asymptotic optimization method for small block lengths. For the improved timing offset estimator, the next step is to generalize the estimators to include a frequency estimator in addition to the timing and channel estimators. Following that generalization, the estimators could be tested on a real-life system as was done in [27].

Bibliography

[1] M. Rice, J. Palmer, C. Lavin, and T. Nelson. Space-time coding for aeronautical telemetry: Part I—Estimators. *IEEE Transactions on Aerospace and Electronic Systems*, 53(4):1709–1731, 2017.

[2] Charles Audet and Warren Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing AG, Gewerbestrasse 11, 6330 Cham, Switzerland, 2017.

[3] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Jon Dattorro. *Convex optimization & Euclidean distance geometry*. Lulu. com, 2010.

[5] Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–336, 2017.

[6] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[7] R Gary Parker and Ronald L Rardin. *Discrete optimization*. Elsevier, 2014.

[8] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423, 623–656, July, Oct. 1948.

[9] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *IEEE International Conference on Communications*, volume 2, pages 1064–1070, 1993.

[10] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, 2009.

[11] D. J. C. MacKay and R. M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 33(6):457–458, Mar. 1997.

[12] M. A. Jensen, M. D. Rice, and A. L. Anderson. Aeronautical telemetry using multiple-antenna transmitters. *IEEE Transactions on Aerospace and Electronic Systems*, 43(1):262–272, 2007.

[13] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[14] S. Giddens, M. A. C. Gomes, J. P. Vilela, J. L. Santos, and W. K. Harrison. Enumeration of the degree distribution space for finite block length LDPC codes. Under Review.

[15] R. G. Gallager. *Low-Density Parity-Check Codes*. MIT Press, Cambridge, MA, 1963.

[16] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In Colin Boyd, editor, *Cryptography and Coding 5th IMA Conf.*, number 1025 in Lecture Notes in Computer Science, pages 100–111. Springer, Berlin, Germany, 1995.

[17] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inf. Theory*, 45(2):399–431, Mar. 1999.

[18] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, Sep. 1981.

[19] T. J. Richardson and R. L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, 2001.

[20] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory*. Cambridge University Press, New York, NY, 2008.

[21] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., Hoboken, NJ, 2006.

[22] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, Feb 2001.

[23] S. ten Brink, G. Kramer, and A. Ashikhmin. Design of low-density parity-check codes for modulation and detection. *IEEE Transactions on Communications*, 52(4):670–678, 2004.

[24] M. Ivkovic, S. K. Chilappagari, and B. Vasic. Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers. *IEEE Transactions on Information Theory*, 54(8):3763–3768, 2008.

[25] Sae-Young Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 5(2):58–60, Feb 2001.

[26] S. Abu-Surra, D. Divsalar, and W. E. Ryan. Enumerators for protograph-based ensembles of LDPC and generalized LDPC codes. *IEEE Transactions on Information Theory*, 57(2):858–886, 2011.

[27] M. Rice, T. Nelson, J. Palmer, C. Lavin, and K. Temple. Space-time coding for aeronautical telemetry: Part II—Decoder and system performance. *IEEE Transactions on Aerospace and Electronic Systems*, 53(4):1732–1754, 2017.

[28] Proakis. *Digital Communications 5th Edition*. McGraw Hill, 2007.

[29] E. Perrins and M. Rice. Optimal and reduced complexity receivers for m-ary multi-h cpm. In *2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No.04TH8733)*, volume 2, pages 1165–1170 Vol.2, 2004.