8-2010

Applications of Linear Programming to Coding Theory

Nathan Axvig
s-naxvig1@math.unl.edu

APPLICATIONS OF LINEAR PROGRAMMING TO CODING THEORY

by

Nathan Axvig

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Mathematics

Under the Supervision of Professor Judy L. Walker

Lincoln, Nebraska

August, 2010

APPLICATIONS OF LINEAR PROGRAMMING TO CODING THEORY

Nathan Axvig, Ph. D.

University of Nebraska, 2010

Adviser: Judy L. Walker

Maximum-likelihood decoding is often the optimal decoding rule one can use, but it is very costly to implement in a general setting. Much effort has therefore been dedicated to find efficient decoding algorithms that either achieve or approximate the error-correcting performance of the maximum-likelihood decoder. This dissertation examines two approaches to this problem.

In 2003 Feldman and his collaborators defined the linear programming decoder, which operates by solving a linear programming relaxation of the maximum-likelihood decoding problem. As with many modern decoding algorithms, is possible for the linear programming decoder to output vectors that do not correspond to codewords; such vectors are known as pseudocodewords. In this work, we completely classify the set of linear programming pseudocodewords for the family of cycle codes.

For the case of the binary symmetric channel, another approximation of maximum-likelihood decoding was introduced by Omura in 1972. This decoder employs an iterative algorithm whose behavior closely mimics that of the simplex algorithm. We generalize Omura's decoder to operate on any binary-input memoryless channel, thus obtaining a soft-decision decoding algorithm. Further, we prove that the probability of the generalized algorithm returning the maximum-likelihood codeword approaches 1 as the number of iterations goes to infinity.

DEDICATION

For Dad, who taught me when to keep at it and when to go home for the day.

ACKNOWLEDGMENTS

My first thanks go to my adviser Judy Walker. As I reflect on the past three years, I see that her efforts have had the goal of training me not only how to conjecture and prove, but also how to be a better teacher, colleague, and mathematical citizen. I am certain that her example will be one that I turn to in the years to come.

Second, I would like to thank Lance Pérez, Jamie Radcliffe, and Mark Walker for serving on my committee. Special thanks are due to Lance, who has been an invaluable authority on the engineering side of coding theory, not to mention good beer and wine. Stephen Hartke was not a formal member of my committee, but he might as well have been. He introduced me to the field of linear optimization and helped me to develop the technical skills necessary to prove many of the results in this dissertation.

It would be remiss not to acknowledge my fellow students and friends who have served with me in the mathematical trenches. My many conversations with Deanna Dreher, Katie Morrison, and Eric Psota have helped shape and solidify many of the ideas contained in this dissertation. In addition to my collaborators, I would like to say thanks to Chris Ahrendt, my long-time officemate and good friend.

A special thank-you goes to my all my friends at the University Lutheran Chapel, who have been my second family during my time in Lincoln. In particular, I am grateful for PB, Ryan, Matthew, and Adam and their collective ability to supply non-mathematical distractions.

I give my final thanks to my parents. Although they may not understand the finer points of my work (and Dad still doesn't believe in infinity), their support and encouragement were essential to the completion of this dissertation. Thank you so much – I love you.

Contents

Chapter 1

Introduction

All communication channels have some level of unreliability: storms interfere with radio signals, memory discs get scratched, and packets of information sent over the internet get lost or destroyed. A brute-force method of increasing reliability is to transmit a message with more power, but this can be impractical in terms of the energy required. Another approach to achieving more reliable communication comes in the form of repetition: send the intended message n times in a row. In this way, the receiver can determine the original message with high probability by a simple majority vote. This method can improve reliability, but it does so by sacrificing the rate at which information is transmitted.

In his landmark 1948 paper, Shannon [24] proved that is possible to achieve arbitrarily reliable communication while maintaining a non-vanishing rate of information transfer. To accomplish this, *codes* were introduced. In the broadest sense, a code is a (usually strict) subset of all possible messages. By transmitting only codewords, the information rate of a communication system is reduced. However, this restriction can be thought of as adding a certain amount of redundancy to a message, much as was illustrated earlier with the repetition scheme. This redundancy can then be

exploited at the receiver, where a distorted message can be *decoded* into a reliable estimate of the transmitted codeword.

This brief description illustrates two of the central questions addressed by coding theory. What sort of redundancy should be added to the original message? And how should one use this redundancy to interpret a noisy received message? Shannon [24] handles the first question in a remarkable yet non-constructive manner: by considering a probability distribution on ensembles of codes, he proves that "good" codes must exist. The search for concrete examples of these good codes can be regarded as the question of code design, an area that continues to prove both fruitful and surprising.

The second question is that of the decoding problem: once a code has been chosen, how can the redundancy of that code be used by the decoder form a reliable estimate of the transmitted codeword? Assuming that all messages are equally likely to be transmitted, the optimal decoder with respect to word-error rate is the maximum-likelihood (ML) decoder. Maximum-likelihood decoding is, however, very costly to implement in a general setting, and in certain situations the decision problem corresponding to ML decoding is known to be NP-complete [4]. Much effort has therefore been dedicated to finding efficient decoding algorithms that either achieve or approximate the error-correcting performance of the ML decoder. This dissertation examines two approaches to this problem, each through the lens of linear optimization.

1.1 Optimal Decoding

A maximum-likelihood decoder works by solving a simply-stated discrete optimization problem: given a received sequence $\mathbf{y}$, find a codeword $\mathbf{c}$ that maximizes the probability of receiving $\mathbf{y}$ conditioned on the event that $\mathbf{c}$ was transmitted. A naïve implementation of ML decoding involves searching through a list of astronomical

proportions. There are, of course, certain situations where a code's structure can be exploited to permit efficient ML decoding, e.g., decoding convolutional codes with the Viterbi algorithm. Since the discovery of a polynomial-time algorithm for ML decoding of general binary linear codes would imply that P = NP [4], a common attack on the ML decoding problem is to design an efficient algorithm whose error-correcting performance approximates that of the optimal ML decoder.

In his PhD thesis, Feldman [13] recasts the problem of maximum-likelihood decoding as an integer linear program. The *linear programming decoder* is then defined as a linear programming relaxation of the ML integer program. The linear programming decoder is provably suboptimal, but the source of this suboptimality is known to be the presence of non-integer extreme points in the underlying polytope, which are known as nontrivial pseudocodewords. In this dissertation, we present a new characterization of these pseudocodewords for the family of cycle codes. This characterization is in terms of the *normal graph* of the code, and the techniques used in this specific situation show promise of shedding light on the general case.

For the special case of the *binary symmetric channel*, another approximation of maximum-likelihood decoding is introduced by Omura [22]. This decoder employs an iterative algorithm whose behavior closely mimics that of the simplex algorithm. While encouraging simulation results are presented, no theoretical results on the performance of this algorithm are given in [22]. We present a generalization of Omura's algorithm that is capable of operating on the diverse class of binary-input memoryless channels (e.g., the *additive white Gaussian noise channel*, the *binary erasure channel*, or the *Z-channel*). Further, by modeling this generalized algorithm as a Markov chain we show that the probability of the algorithm returning the ML codeword approaches 1 as the number of iterations is allowed to grow without bound.

1.2 Thesis Outline

This thesis is organized as follows. Chapter 2 gives a brief introduction to coding theory, introducing terminology and results necessary to the ensuing chapters. Most importantly, this chapter reduces the ML decoding problem to Problem (2.1) of Theorem 2.3.1. This rephrased problem statement is the base from which both the linear programming decoder and the generalized Omura decoder are derived.

Chapter 3 introduces Feldman's linear programming decoder. It also contains a crash-course in basic linear programming. Two presentations of the *fundamental polytope* are given, and these equivalent notions will be used in the proofs of Chapter 4.

Chapter 4 presents a graphical characterization of linear programming pseudocodewords for cycle codes. This characterization is highly visual, and is shown to be useful in both the identification and construction of pseudocodewords. We also compare and contrast our characterization to Dreher's characterization of *minimal linear programming pseudocodewords* [12].

In Chapter 5 we generalize Omura's decoding algorithm from the binary symmetric channel to any binary-input memoryless channel. This new soft-decision decoder encompasses past generalizations of Omura's work. The main contribution of this chapter is a proof that the probability of the generalized Omura decoder outputting the ML codeword approaches 1 as the number of iterations grows without bound. This helps to explain the good error-correcting performance of Omura's algorithm observed in computer simulations.

Chapter 2

Background on Coding Theory

This chapter collects well-known coding theory terminology and results that can be found, for example, in [17, 18, 21]. Though one can define codes over arbitrary finite fields and even rings, we focus exclusively on binary linear codes.

Definition 2.0.1. *A* binary linear code*, or, more simply, a* code*, is a linear subspace $\mathcal{C}$ of $\mathbb{F}_2^n$. A* codeword *is an element* $\mathbf{c}$ *of* $\mathcal{C}$*. The* length *of* $\mathcal{C}$ *is the dimension* n *of the ambient vector space, and the* dimension *of* $\mathcal{C}$ *is its vector space dimension* k*. The* rate *of* $\mathcal{C}$ *is the ratio* k/n*. A code of length* n *and dimension* k *is denoted as an* $[n, k]$ *code.*

Since a binary linear code is a linear subspace of $\mathbb{F}_2^n$, one can describe it compactly with matrices.

Definition 2.0.2. *A* generator matrix G *for a code* $\mathcal{C}$ *is a matrix whose row space is* $\mathcal{C}$*. A* parity-check matrix H *for a code* $\mathcal{C}$ *is a matrix whose null space is* $\mathcal{C}$*.*

It is important to note that, in general, a code $\mathcal{C}$ has many distinct generator and parity-check matrices. Also, a given generator matrix or parity-check matrix need not have full rank: there may be redundant rows in either. The results herein

require knowledge of the specific parity-check representation being used, so a code is taken to be a binary linear code $\mathcal{C}$ equipped with a fixed parity-check matrix H. It is important for the reader to note that this is a marked departure in terminology from many standard texts and publications in coding theory where a code is described intrinsically as a subspace, not extrinsically as the kernel of a specific matrix.

One of the most fundamental concepts of coding theory is that of a *channel*, i.e., a medium over which information is transmitted. Examples of channels in physical systems include telephone lines, ethernet cables, compact discs, and even the earth's atmosphere. While actual channels come in myriad forms, we model them mathematically as a random transformation of the channel input.

Definition 2.0.3. *Let $\mathcal{Y}$ be some output alphabet, which can be either finite or infinite. A* binary-input channel*, or, more simply, a* channel*, is a random transformation from $\mathbb{F}_2^n$ to $\mathcal{Y}^n$, characterized by a conditional probability function $P(\mathbf{y}|\mathbf{x})$, where $\mathbf{x} \in \mathbb{F}_2^n$ and $\mathbf{y} \in \mathcal{Y}^n$.*

The case where the channel affects the components of vectors $\mathbf{x} \in \mathbb{F}_2^n$ independently is of particular interest; such a channel is called *memoryless*. If a channel affects the transmitted bits 0 and 1 in symmetric fashion, then the channel is called *symmetric*.

Definition 2.0.4. *A* binary-input memoryless channel *is a channel whose conditional probability function $P(\mathbf{y}|\mathbf{x})$ can be factored as*

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|x_i).$$

A binary-input memoryless channel with output alphabet $\mathcal{Y}$ is called output-symmetric*, or, more simply,* symmetric*, if $\mathcal{Y}$ can be partitioned into pairs $\{a, b\}$ (possibly with*

$a = b$*) such that* $P(y = a \,|\, x = 0) = P(y = b \,|\, x = 1)$ *and* $P(y = a \,|\, x = 1) = P(y = b \,|\, x = 0)$.

In addition to the commonly used modifiers of memoryless and symmetric, we introduce the term "completely non-deterministic" to described channels for which no output completely determines the channel input.

Definition 2.0.5. *A binary-input memoryless channel with output alphabet* $\mathcal{Y}$ *is called* completely non-deterministic *if* $P(y|0) > 0$ *and* $P(y|1) > 0$ *for all* $y \in \mathcal{Y}$. *A binary-input memoryless channel that is not completely non-deterministic is said to be* partially deterministic.

Remark 2.0.6. *While it may be more appropriate to refer to perfect channels as completely deterministic, such situations are of no interest to coding theory. All communication channels in this dissertation are therefore assumed to be imperfect channels, and so the dichotomy introduced by Definition 2.0.5 is appropriate.*

For many channels, the probability $P(\mathbf{y}|\mathbf{x})$ is closely related to the *Hamming distance* between $\mathbf{y}$ and $\mathbf{x}$. The notions of Hamming distance and the corresponding minimum distance are ubiquitous in coding theory, since they aid in describing the limits of a code's error-correcting capabilities.

Definition 2.0.7. *Let* $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$, *and let* $\mathbf{0}$ *denote the all-zeros vector. The* Hamming distance *between* $\mathbf{x}$ *and* $\mathbf{y}$ *is* $d(\mathbf{x}, \mathbf{y}) := |\mathrm{supp}(\mathbf{x} - \mathbf{y})|$, *where* $\mathrm{supp}(\mathbf{a})$ *is the* support *of the vector* $\mathbf{a}$, *i.e., the set of all indices where* $\mathbf{a}$ *is non-zero. The* Hamming weight, *or simply the* weight *of a vector* $\mathbf{x}$ *is* $\mathrm{wt}(\mathbf{x}) := d(\mathbf{x}, \mathbf{0}) = |\mathrm{supp}(\mathbf{x})|$. *The* minimum distance *of a code* $\mathcal{C}$ *is the minimum Hamming distance* $d(\mathbf{x}, \mathbf{y})$ *over all pairs of codewords* $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ *such that* $\mathbf{x} \neq \mathbf{y}$. *If the minimum distance of an* $[n, k]$ *code* $\mathcal{C}$ *is known to be* d, *we say that* $\mathcal{C}$ *is an* $[n, k, d]$ *code.*

The next two propositions can be found in any standard text on coding theory.

Proposition 2.0.8. *The Hamming distance is a metric on* $\mathbb{F}_2^n$.

Proposition 2.0.9. *The minimum distance of a binary linear code* $\mathcal{C}$ *is equal to the minimum weight of a non-zero codeword.*

2.1 LPDC Codes: Codes on Graphs

Since the discovery of turbo codes [5] in 1993, there has been a renewed interest in graph-based codes. Besides turbo codes themselves, the low-density parity-check (LDPC) codes of Gallager [15] have been the object of much research. Much of this interest can be credited to the fact that the graphical structure of the codes admits efficient decoding algorithms that in simulations have been shown to have outstanding error-correcting capabilities. Iterative message-passing decoding of LDPC codes is a vast topic, and we will not discuss it here. The graphical structure of LDPC codes, however, provides a useful framework when discussing Feldman's *linear programming decoder* [13]. It is for this reason that we provide some basic definitions and terminology.

A low-density parity-check code is a code whose parity-check matrix is "sparse," a term that can be made precise only when considering infinite ensembles of codes. While the relative number of 0's and 1's in a parity-check matrix can have a significant impact on the complexity of implementing certain decoding algorithms, all results herein apply to binary linear codes without regard to sparsity.

Definition 2.1.1. *A* Tanner graph *is a bipartite graph with bipartition* $\mathcal{I} \cup \mathcal{J}$ *and edge set* E*. The elements of* $\mathcal{I}$ *are called* variable nodes*, and the elements of* $\mathcal{J}$ *are called* check nodes.

Given a parity-check matrix $H = (h_{j,i})$, one can construct the Tanner graph $T = T(H)$ of H as follows. Let $\mathcal{I}$ be the set of columns of H and let $\mathcal{J}$ be the set of rows. Define $T(H)$ to be the graph whose vertex set is $\mathcal{I} \cup \mathcal{J}$, with $i \in \mathcal{I}$ adjacent to $j \in \mathcal{J}$ if and only if $h_{j,i} = 1$. In other words, $T(H)$ is the bipartite graph whose bipartite adjacency matrix is H. Conversely, one can derive a parity-check matrix from a Tanner graph: index rows by vertices in $\mathcal{J}$ and columns by vertices in $\mathcal{I}$. Set $h_{j,i} = 1$ precisely when i and j are adjacent in T. In this manner one can see that there is a bijective correspondence between Tanner graphs and parity-check matrices.

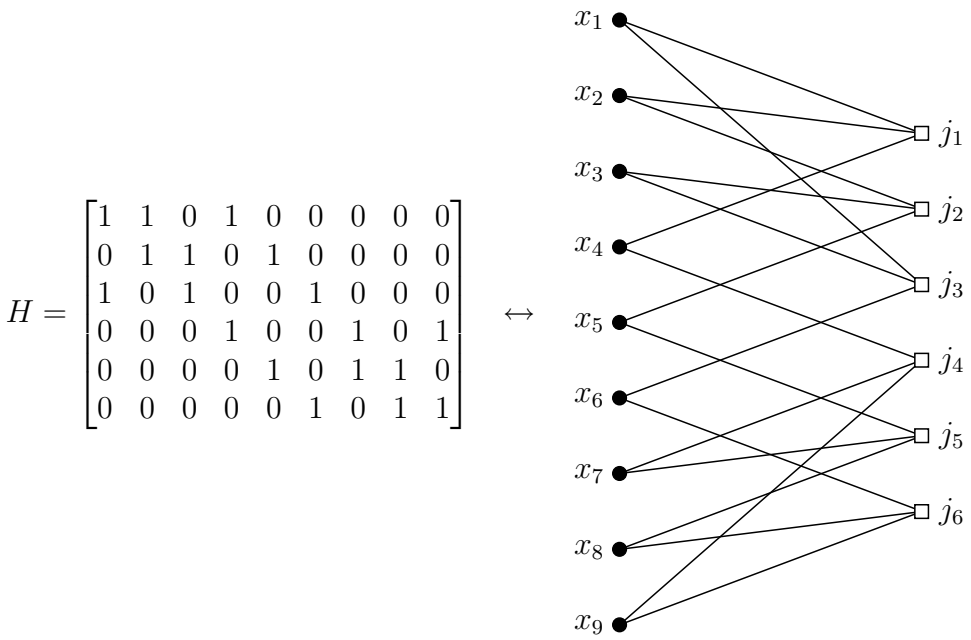


Figure 2.1: A parity-check matrix H and the corresponding Tanner graph T.

The Tanner graph provides a highly visual environment in which to work with codes.

Definition 2.1.2. *Let T be a Tanner graph, and let $\mathbf{x}$ be an assignment of binary values to the variable nodes of T. The assignment $\mathbf{x}$ is called a* valid configuration *on T provided that for each $j \in \mathcal{J}$ the quantity $\sum_{i \in N(j)} x_i$ is even, where, as is standard,*

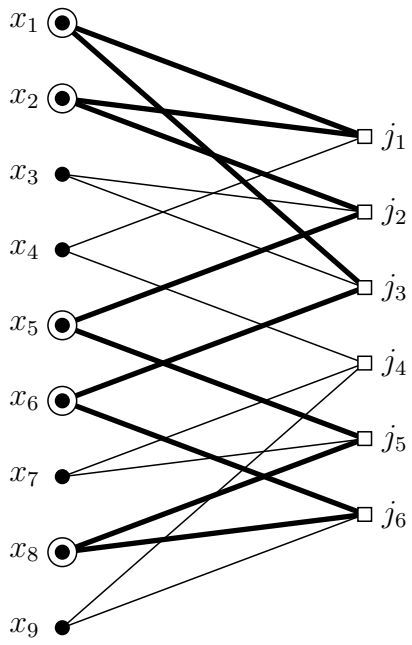


Figure 2.2: A valid configuration on the Tanner graph of Figure 2.1. Circled nodes receive a binary value of 1, and uncircled nodes receive a binary value of 0.

$N(j)$ denotes the neighborhood of j in T, i.e., the set of all $i \in \mathcal{I}$ that are adjacent to j.

Proposition 2.1.3 (see, e.g., [2])**.** *Let $\mathcal{C}$ be a code with parity-check matrix H, and let $T = T(H)$ be the corresponding Tanner graph. The set of valid configurations on T is precisely the set $\mathcal{C}$ of codewords.*

By Proposition 2.1.3, codewords of $\mathcal{C}$ may be viewed as valid binary configurations on a Tanner graph. This graphical realization of codewords is further displayed in the family of *cycle codes*.

Definition 2.1.4. *A* cycle code *is a code $\mathcal{C}$ equipped with a parity-check matrix H that has uniform column weight 2. Equivalently, every variable node in the associated Tanner graph has degree 2. The* normal graph *N of $\mathcal{C}$ is the (multi)graph whose vertex-edge incidence matrix is H.*

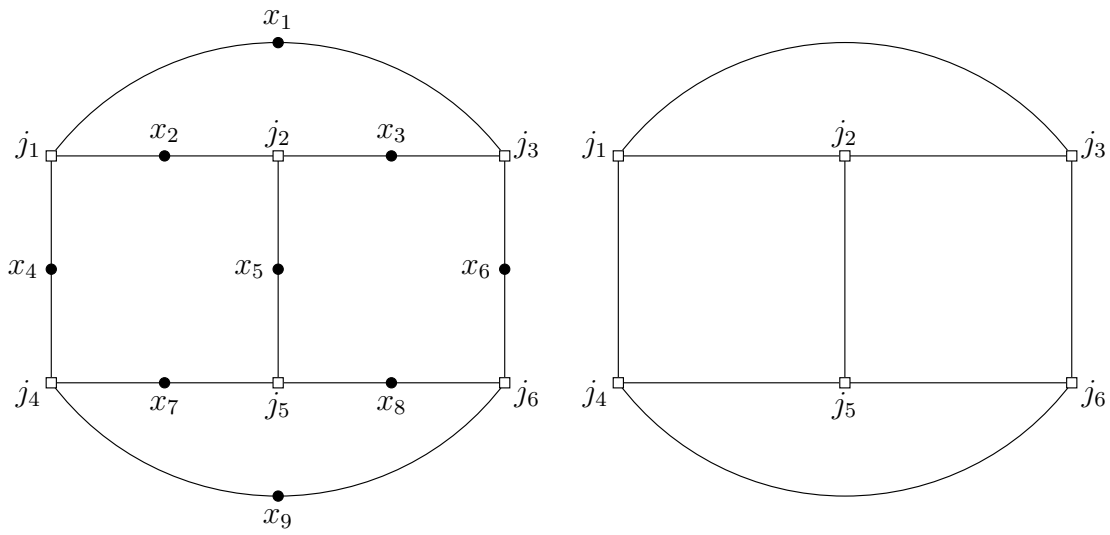


Figure 2.3: The Tanner graph of Figure 2.1 and the corresponding normal graph N.

The parity-check matrix of Figure 2.1 has uniform column weight two, and hence defines a cycle code. Figure 2.3 shows a Tanner graph isomorphic to that given in Figure 2.1 and the corresponding normal graph, illustrating the general fact that the normal graph can be obtained from the Tanner graph by simply turning variable nodes into edges.

The class of cycle codes, in general, has poor distance properties: the minimum distance is equal to the girth of the normal graph, which only grows logarithmically with the length of the code. However, cycle codes are often much easier to analyze than general codes because of their special structure. Let $\mathcal{C}$ be a cycle code with Tanner graph T and normal graph N. Proposition 2.1.3 states that $\mathcal{C}$ is the set of valid configurations on T. By identifying variable nodes of T with edges of N, we may regard $\mathcal{C}$ as the set of all valid edge labelings on N, i.e., all binary assignments to the edges of N such that each vertex of N is incident to an even number of 1's. By [26, Proposition 1.2.27], every codeword is an indicator vector for an edge-disjoint union of simple cycles in N. The converse is also true: the indicator vector of any

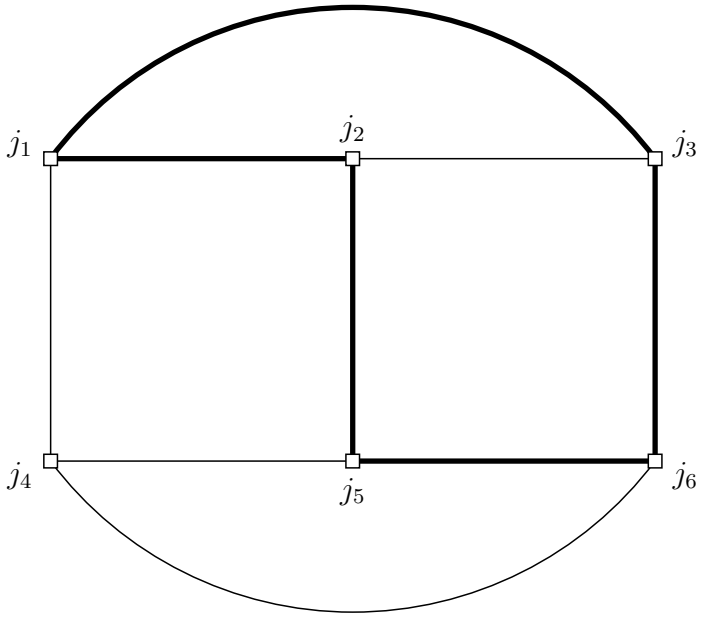


Figure 2.4: A different view of the configuration indicated in Figure 2.2. Bold edges receive a value of 1, and all other edges receive a value of 0.

edge-disjoint union of simple cycles is a codeword. This is illustrated in Figure 2.4.

2.2 Channel Models and Decoding

Let $\mathcal{C}$ be a code, and consider transmission over a binary-input memoryless channel. The task of a decoder is to translate the channel output into an educated guess as to what the transmitted codeword truly was. When a decoder fails to decode or returns an estimate that does not match the transmitted codeword, this error in interpretation is called a *word* or *block error*. The probability of a word error occurring is called the *word-error rate*, and it is sometimes denoted as P_w. A natural and important question to address is how to design a decoder that minimizes this word-error rate.

Two common decoders are the *maximum a posteriori (MAP)* decoder and the *maximum-likelihood (ML)* decoder. Given a received vector $\mathbf{y}$, the MAP decoder

computes

$$\mathbf{c}^{\mathrm{MAP}} = \underset{\mathbf{c}\in\mathcal{C}}{\operatorname{argmax}}\, P(\mathbf{c}|\mathbf{y})$$

while the ML decoder computes

$$\mathbf{c}^{\mathrm{ML}} = \underset{\mathbf{c}\in\mathcal{C}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{c}).$$

A codeword $\mathbf{c}^{\mathrm{ML}}$ that maximizes $P(\mathbf{y}|\mathbf{c})$ is called a *maximum-likelihood codeword*. The MAP decoder is theoretically optimal in that it minimizes P_w. The ML decoder does not necessarily minimize the word-error rate; however, in the case where each codeword is equally likely to be transmitted (a common assumption made in coding theory) MAP decoding and ML decoding coincide. Being able to go either way under the assumption of equally likely codewords, we take ML decoding as our optimal decoder, for it is somewhat more natural to compute based on the channel's probability function.

Maximum-likelihood decoding can be implemented by performing an exhaustive search over all 2^k codewords, but this is not practical for even moderately high values of k. Much effort has therefore been dedicated to finding more efficient ways of decoding while still achieving or approaching the optimal word-error rate of the ML decoder. This section introduces five binary-input memoryless channels and describes algorithms that achieve ML performance for four of them. The purpose of providing these examples is two-fold. First, it reviews some common channels and decoding methods. Second, it illustrates the dependence that specialized decoding strategies have on the channel for which they were developed. The generalized Omura decoder introduced in this dissertation is capable of decoding on any of the following channels.

2.2.1 The Binary Symmetric Channel

The *binary symmetric channel (BSC) with crossover probability* p is perhaps the simplest nontrivial example of a communication channel. It is both symmetric and completely non-deterministic. Its output alphabet is $\mathbb{F}_2$, and its conditional probability function is given by

$$P(y|x) = \begin{cases} 1-p & \text{if } y = x \\ p & \text{if } y \neq x. \end{cases}$$

A depiction of the channel is given in Figure 2.5.

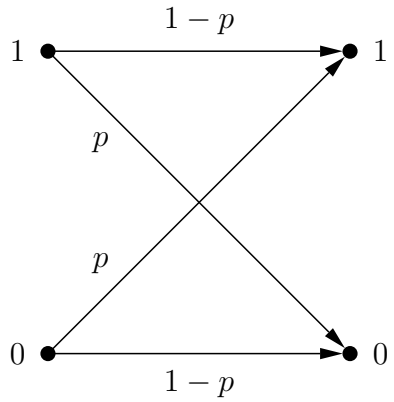


Figure 2.5: The binary symmetric channel.

When working with the binary symmetric channel, it is customary to assume that $p < \frac{1}{2}$, since if $p > \frac{1}{2}$ one can simply invert the output sequence at the receiver and get back to the $p < \frac{1}{2}$ case. The case $p = \frac{1}{2}$ is of no interest, since reliable communication is then impossible. Assuming $p < \frac{1}{2}$, the following proposition points to a method of ML decoding.

Proposition 2.2.1. *Let $\mathcal{C}$ be a binary linear code, and consider transmission over the binary symmetric channel with crossover probability $p < \frac{1}{2}$. If $\mathbf{y}$ is received, then the*

set of maximum-likelihood codewords is precisely the set of codewords that are closest in Hamming distance to $\mathbf{y}$.

Said colloquially, maximum-likelihood decoding on the binary symmetric channel can be accomplished by employing a *nearest-neighbor* decoder, i.e., one that finds the closest codeword to the received vector. Using this fact and Proposition 2.0.8, one can show the classical result that an ML decoder on the BSC is guaranteed to correctly decode a received sequence provided that at most $\lfloor (d-1)/2 \rfloor$ positions were flipped by the channel, where d is the minimum distance of the code.

Given a received vector $\mathbf{y}$, the task of nearest-neighbor decoding can be accomplished by first finding a least-weight error vector $\mathbf{e}_* \in \mathbb{F}_2^n$ satisfying $\mathbf{y} + \mathbf{e}_* \in \mathcal{C}$ and then adding $\mathbf{e}_*$ to $\mathbf{y}$ to obtain the codeword nearest to $\mathbf{y}$. Finding such an error vector is equivalent to finding a least-weight vector satisfying $H(\mathbf{y}+\mathbf{e}) = \mathbf{0}$, or, more simply, $H\mathbf{e} = H\mathbf{y}$. With this reformulation of nearest-neighbor decoding in mind, we now describe *syndrome decoding*.

Let $\mathcal{C}$ be an $[n, k]$ code with parity-check matrix H. Before any decoding takes place, create a table of all possible 2^{n-k} *syndromes* and their corresponding *coset leaders*.

Definition 2.2.2. *Let* $\mathcal{C}$ *be an* $[n, k]$ *code with parity-check matrix* H. *The* syndrome $\mathbf{s}$ *of an* n*-dimensional vector* $\mathbf{y}$ *is* $\mathbf{s} = H\mathbf{y}$. *A* coset leader $\mathbf{e} \in \mathbb{F}_2^n$ *for a syndrome* $\mathbf{s}$ *is a minimum-weight vector satisfying* $H\mathbf{e} = \mathbf{s}$.

Remark 2.2.3. *There is a natural bijective correspondence between syndrome vectors and cosets of* $\mathcal{C}$. *Indeed, a coset is uniquely determined by the syndrome of any of its elements. From this viewpoint, one can see why coset leaders are so named.*

Once a vector $\mathbf{y}$ is received, find its syndrome $\mathbf{s} = H\mathbf{y}$. Search the previously constructed table to find the coset leader $\mathbf{e}_*$ whose syndrome matches $\mathbf{s}$. Finally, return

the estimate $\mathbf{c}_* = \mathbf{y} + \mathbf{e}_*$, which, by the preceding discussion, is an ML codeword.

2.2.2 The AWGN Channel

The *additive white Gaussian noise (AWGN) channel* is a binary-input memoryless channel with output alphabet $\mathbb{R}$, and it is both symmetric and completely non-deterministic. When a codeword $\mathbf{c} \in \mathbb{F}_2^n$ is sent through the AWGN channel, it is first *modulated* via the coordinate-wise map $\mathfrak{m}(c_i) = 2c_i - 1$. This modulation serves to model a digital communication system, where bits are transmitted as antipodal waveforms. The set $\mathfrak{m}(\mathcal{C}) = \{\mathfrak{m}(\mathbf{c}) : \mathbf{c} \in \mathcal{C}\} \subseteq \mathbb{R}^n$ of all modulated codewords is called the *signal constellation* of the code $\mathcal{C}$.

Once a codeword is modulated, the additive white Gaussian noise channel is modeled by adding independent, identically distributed Gaussian random variables to each coordinate of the modulated vector. Thus, the received vector is $\mathbf{y} = \mathfrak{m}(\mathbf{c}) + \boldsymbol{\eta}$, where the components of $\boldsymbol{\eta}$ are drawn independently from a Gaussian random variable of mean 0 and variance σ^2. This model gives rise to the following conditional probability functions:

$$P(y \,|\, c = 0) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y+1)^2}{2\sigma^2}}$$
$$P(y \,|\, c = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-1)^2}{2\sigma^2}}.$$

Proposition 2.2.4. *Let $\mathcal{C}$ be a binary linear code, and consider transmission over the additive white Gaussian noise channel. Suppose that the vector $\mathbf{y}$ is received. The set of maximum-likelihood codewords is precisely the set of codewords whose points in the signal constellation are closest to $\mathbf{y}$ with respect to Euclidean distance.*

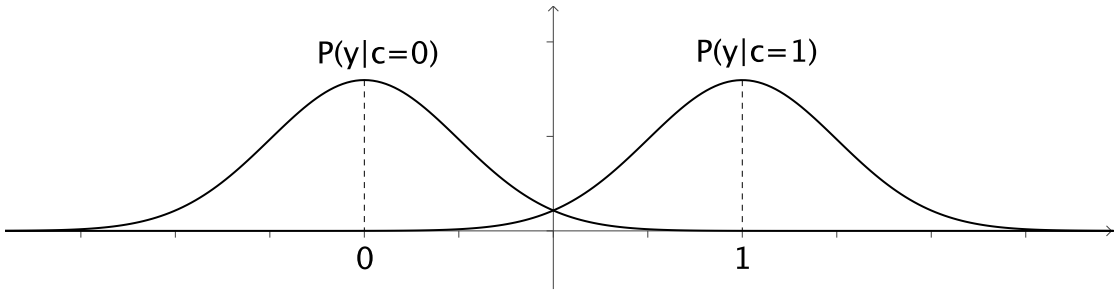


Figure 2.6: The additive white Gaussian noise channel.

Proposition 2.2.4 implies that we may implement maximum-likelihood decoding on the additive white Gaussian noise channel by first computing the Euclidean distance between the received vector and all points in the signal constellation and then selecting the codeword whose corresponding point in the signal constellation is nearest to the received vector. Performing ML decoding in this nearest neighbor fashion still involves an exhaustive search through the 2^k points in the signal constellation, which quickly becomes computationally infeasible as the dimension of the code grows.

2.2.3 The Binary Erasure Channel

The *binary erasure channel (BEC) with probability ϵ of erasure* is a binary-input memoryless channel with output alphabet $\{0, e, 1\}$. The basic premise of the BEC model is that if a bit is received, it must be the correct bit (i.e., the transmitted bit). Bits are erased on occassion, in which case the transmitted bit is equally likely to be a 0 or a 1. The conditional probability function presented in Figure 2.7 makes this explicit.

The binary erasure channel is symmetric, but, unlike the binary symmetric channel and the additive white Gaussian noise channel, it is partially non-deterministic since $P(y = 0 \,|\, x = 1) = 0$ and $P(y = 1 \,|\, x = 0) = 0$. This combination of complete certainty (if $y = 0$ or 1) with complete uncertainty (if $y = e$) results in a ML decod-

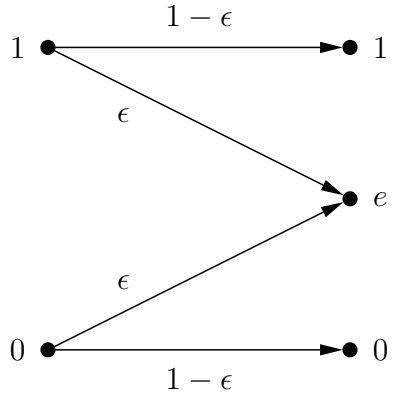


Figure 2.7: The binary erasure channel.

ing algorithm that resembles a *feasibility problem*. In other words, if any plausible solution is found, it must be optimal. This idea is made clear in Proposition 2.2.5, to which we provide a proof since the author was unable to locate a satisfactory reference. Nonetheless, the basic idea behind the proof is certainly not original to this dissertation.

Proposition 2.2.5. *Let $\mathcal{C}$ be a code of length n, and consider transmission over the binary erasure channel with probability of erasure ϵ. If the vector $\mathbf{y}$ is received, then the set of maximum-likelihood codewords is the collection of all codewords that agree with $\mathbf{y}$ on all but the erased positions.*

Proof. Let $\mathfrak{U} = \{i \in \{1, 2, \ldots, n\} \,|\, y_i = e\}$ be the set of indices of erased positions in $\mathbf{y}$, and let $\mathfrak{C} = \{1, 2, \ldots, n\} \setminus \mathfrak{U}$. Define $A := \{\mathbf{c} \in \mathcal{C} \,|\, c_i = y_i \text{ for all } i \in \mathfrak{C}\}$, so that A is the set of codewords that match $\mathbf{y}$ in all but the erased positions. If $\mathbf{c} \in \mathcal{C} \setminus A$, then there exists some index $\ell \in \mathfrak{C}$ such that $c_\ell \neq y_\ell$. This implies that $P(y_\ell|c_\ell) = 0$, and so

$$P(\mathbf{y}|\mathbf{c}) = \prod_{i=1}^{n} P(y_i|c_i) = 0.$$

On the other hand, if $\mathbf{c} \in A$ we have

$$\begin{aligned} P(\mathbf{y}|\mathbf{c}) &= \prod_{i=1}^{n} P(y_i|c_i) \\ &= \left(\prod_{i\in\mathfrak{U}} P(y_i|c_i)\right) \cdot \left(\prod_{i\in\mathfrak{C}} P(y_i|c_i)\right) \\ &= \epsilon^{|\mathfrak{U}|}(1-\epsilon)^{n-|\mathfrak{U}|}. \end{aligned}$$

Thus, the codewords in A are exactly the codewords that maximize $P(\mathbf{y}|\mathbf{c})$. □

Let $\mathcal{C}$ be a code with parity-check matrix H, and suppose that $\mathbf{y}$ is received over the binary erasure channel. Set $\mathfrak{C} = \{i \,|\, y_i \in \{0,1\}\}$ and $\mathfrak{U} = \{i \,|\, y_i = e\}$. We interpret $\mathfrak{C}$ as the set of coordinates with *certainty* and $\mathfrak{U}$ as the set of coordinates with *uncertainty*. In the following, $\mathbf{x}^A$ is used to denote the projection of a vector $\mathbf{x}$ onto the coordinates indicated by $A \subseteq \{1, 2, \ldots, n\}$. In a similar manner, H^A denotes the projection of the matrix H onto the columns indicated by A.

By Proposition 2.2.5, we can achieve ML decoding if we can find a codeword that matches $\mathbf{y}$ on the positions of $\mathfrak{C}$. We therefore seek a binary vector $\mathbf{c}$ such that $H\mathbf{c} = 0$ and $\mathbf{c}^{\mathfrak{C}} = \mathbf{y}^{\mathfrak{C}}$. Such a vector $\mathbf{c}$ can be found by solving the linear system $H^{\mathfrak{C}}\mathbf{y}^{\mathfrak{C}} + H^{\mathfrak{U}}\mathbf{c}^{\mathfrak{U}} = 0$ for $\mathbf{c}^{\mathfrak{U}}$.

2.2.4 The Z-Channel

The *Z-channel with cross-over probability p* is a binary-input memoryless channel with output alphabet $\mathbb{F}_2$. Its conditional probability function is depicted in Figure 2.8.

Unlike the binary symmetric channel, the additive white Gaussian noise channel, and the binary erasure channel, the Z-channel is not symmetric with respect to the input bit. Like the BEC, the Z-channel is partially deterministic: if a 0 is received,

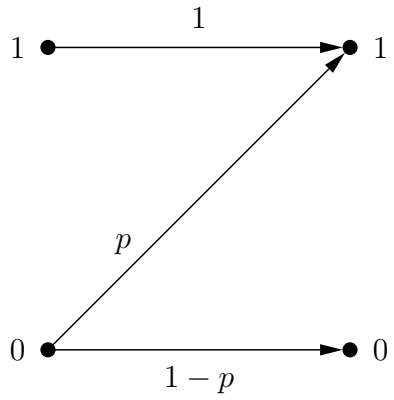


Figure 2.8: The Z-channel.

it is known that a 0 was sent.

For completeness, we present a maximum-likelihood decoding algorithm for this channel model. For any $T \subseteq \{1, 2, \ldots, n\}$, let $\mathcal{C}(T)$ denote the code obtained by first considering only those codewords that are equal to 0 on the positions of T and then projecting these codewords onto the complement of T (this procedure is known as *shortening* the code on T). Given a received vector $\mathbf{y}$, project $\mathbf{y}$ onto $\mathrm{supp}(\mathbf{y})$ and perform nearest-neighbor decoding in $\mathcal{C}(\{1, 2, \ldots, n\} \setminus \mathrm{supp}(\mathbf{y}))$. Lift the result back to a length n vector to obtain an ML codeword.

2.2.5 The Binary Asymmetric Channel

The *binary asymmetric channel with crossover probabilities p and q* is the binary input memoryless channel illustrated in Figure 2.9. The binary asymmetric channel models communication channels where transition probabilities can vary according to the input bit. As such, the binary asymmetric channel subsumes both the BSC (by setting $p = q$) and the Z-channel (by setting $q = 0$).

It is unclear how to best implement maximum-likelihood decoding for the binary

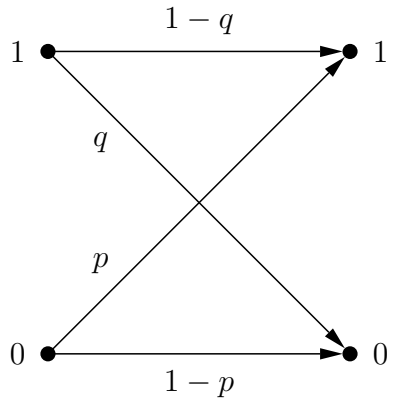


Figure 2.9: The binary asymmetric channel.

asymmetric channel, and it is for this reason that this channel is introduced. We seek a method of performing ML decoding on any binary-input memoryless channel. In Section 2.3 we recast the ML decoding problem as an integer program. It is from this point of view that Feldman develops the linear programming decoder [13], and it is also from this perspective that we develop a generalization of Omura's decoding algorithm [22]. The linear programming decoder can operate on any binary-input memoryless channel that is completely non-deterministic, while our generalized Omura decoder is capable of decoding on any binary-input memoryless channel. Both decoders are suboptimal in that they do not guarantee ML performance, but the generality that they admit warrants attention.

2.3 General Binary Memoryless Channels

We now recast the maximum-likelihood decoding problem on general binary-input memoryless channels, a class of channels that subsumes all of the channels previously discussed. Let $\mathcal{C}$ be a binary linear code presented by a parity-check matrix H,

not necessarily of full rank. Consider transmission over a binary-input memoryless channel and suppose that the vector $\mathbf{y}$ is received.

To allow for the treatment of partially deterministic channels, we first examine the situation in which some received symbols completely determine the corresponding codeword bits. Define

$$\mathfrak{U} := \{i \in \{1, 2, \ldots, n\} \mid P(y_i|0) > 0 \text{ and } P(y_i|1) > 0\}.$$

In other words, $\mathfrak{U}$ is the set of positions where there is some uncertainty as to what y_i ought to be. Set $\mathfrak{C} := \{1, 2, \ldots, n\} \setminus \mathfrak{U}$. The set $\mathfrak{C}$ represents the positions where y_i determines the codeword bit c_i. Let $\mathbf{d} = (d_i)_{i \in \mathfrak{C}}$ be such that for each $i \in \mathfrak{C}$ the bit d_i is the unique binary value such that $P(y_i|d_i) > 0$; note that the case where some position is such that $P(y_i|0) = P(y_i|1) = 0$ has probability 0. The vector $\mathbf{d}$ is the vector of *determined codeword bits*. For each $i \in \mathfrak{U}$, define the vector $\boldsymbol{\lambda}$ of *log-likelihood ratios* component-wise by

$$\lambda_i = \log\left(\frac{P(y_i|0)}{P(y_i|1)}\right).$$

By definition, the output of the ML decoder is

$$\operatorname*{argmax}_{\mathbf{c} \in \mathcal{C}} P(\mathbf{y}|\mathbf{c}).$$

Since the channel is memoryless, we can express $P(\mathbf{y}|\mathbf{c})$ as $\prod_{i=1}^{n} P(y_i|c_i)$, so the expression above becomes

$$\operatorname*{argmax}_{\mathbf{c} \in \mathcal{C}} \prod_{i=1}^{n} P(y_i|c_i).$$

If a codeword $\mathbf{c}$ is such that $\prod_{i=1}^{n} P(y_i|c_i) > 0$, it must be that $\mathbf{c}^{\mathfrak{C}} = \mathbf{d}$. The output

of ML decoding can therefore be expressed as

$$\begin{aligned}
\operatorname*{argmax}_{\mathbf{c}\in\mathcal{C}} \prod_{i=1}^{n} P(y_i|c_i) &= \operatorname*{argmax}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \prod_{i=1}^{n} P(y_i|c_i) \\
&= \operatorname*{argmax}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \prod_{i\in\mathfrak{U}} P(y_i|c_i) \\
&= \operatorname*{argmax}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \sum_{i\in\mathfrak{U}} \log(P(y_i|c_i)) \\
&= \operatorname*{argmin}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \sum_{i\in\mathfrak{U}} \log\left(\frac{1}{P(y_i|c_i)}\right) \\
&= \operatorname*{argmin}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \left(\sum_{i\in\mathfrak{U}} \log\left(\frac{1}{P(y_i|c_i)}\right) + \sum_{i\in\mathfrak{U}} \log\left(P(y_i|0)\right)\right) \\
&= \operatorname*{argmin}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \sum_{i\in\mathfrak{U}} \log\left(\frac{P(y_i|0)}{P(y_i|c_i)}\right) \\
&= \operatorname*{argmin}_{\substack{\mathbf{c}\in\mathcal{C}\\ \mathbf{c}^{\mathfrak{C}}=\mathbf{d}}} \boldsymbol{\lambda}^T \mathbf{c}^{\mathfrak{U}},
\end{aligned}$$

where the final equivalence can be observed by considering the cases of $c_i = 0$ and $c_i = 1$ separately. We have proven the following well-known theorem.

Theorem 2.3.1. *Let $\mathcal{C}$ be a code and suppose that $\mathbf{y}$ is received over a binary input memoryless channel. Set $\mathfrak{U} = \{i \in \{1, 2, \ldots, n\} \mid P(y_i|0) > 0 \text{ and } P(y_i|1) > 0\}$ and $\mathfrak{C} = \{1, 2, \ldots, n\} \setminus \mathfrak{U}$, and let $\boldsymbol{\lambda}$ be the vector of log-likelihood ratios on $\mathfrak{U}$. Let $\mathbf{d} = (d_i)_{i\in\mathfrak{C}}$ be such that for each $i \in \mathfrak{C}$ the bit d_i is the unique binary value such that $P(y_i|d_i) > 0$. The set of maximum-likelihood codewords is precisely the solution set to the following optimization problem:*

$$\begin{aligned}
&\textit{minimize} && \boldsymbol{\lambda}^T \mathbf{c}^{\mathfrak{U}} \\
&\textit{subject to} && \mathbf{c} \in \mathcal{C} \\
& && \mathbf{c}^{\mathfrak{C}} = \mathbf{d}.
\end{aligned} \tag{2.1}$$

Corollary 2.3.2 follows immediately from Theorem 2.3.1. It has significant relevance, since it applies to the BSC and the AWGN channel.

Corollary 2.3.2 ([13], Theorem 2.1)**.** *Let $\mathcal{C}$ be a code and consider transmission over a completely non-deterministic binary input memoryless channel. Suppose that $\mathbf{y}$ is received, and let $\boldsymbol{\lambda}$ be the vector of log-likelihood ratios. The set of maximum-likelihood codewords is precisely the solution set to the following minimization problem:*

$$\begin{aligned} &\textit{minimize} \quad \boldsymbol{\lambda}^T \mathbf{c} \\ &\textit{subject to} \quad \mathbf{c} \in \mathcal{C}. \end{aligned} \tag{2.2}$$

Chapter 3

Background on Linear Programming Decoding

A linear program is a problem in which a linear functional of the form $\mathbf{c}^T\mathbf{x}$ is to be optimized over all real vectors $\mathbf{x}$ that satisfy a finite set of linear equality and linear (non-strict) inequality constraints[1]. Imposing the additional constraint that $\mathbf{x}$ be an integer vector gives rise to integer linear programs. Many important problems can be phrased in terms of integer linear programs, one of the most famous being the Traveling Salesman Problem [6]. While solutions to these problems are often of great interest, the task of solving general integer linear programs is quite difficult. On the other hand, solutions to linear programs can often be found quickly (using, e.g., the simplex algorithm), and they can even be found in polynomial time (using, e.g., the ellipsoid method). A common approach to solving integer linear programs therefore is to solve a related linear program (or a series of linear programs) and then interpret the solution in light of the original problem.

In his thesis [13], Feldman recasts the maximum-likelihood decoding problem as

[1]Unless stated otherwise, in the context of linear programming we use the convention that inequalities are not strict, i.e., we assume the use of "$\geq$" and "$\leq$" instead of "$>$" and "$<$".

an integer linear program. In an attempt to find solutions to this integer program efficiently, a linear programming relaxation of the original integer program is introduced. The linear programming (LP) decoder is defined to be the decoder that outputs a solution to this relaxed optimization problem. The LP decoder has been the subject of much research since its introduction in the early 2000's, partially because of intuitive ties between it and iterative message-passing decoding of LDPC codes [25]. In this dissertation we examine the LP decoder for its own sake.

This chapter offers a brief introduction to Feldman's linear programming decoder, setting the stage for Chapter 4. We first introduce basic vocabulary and concepts from linear programming in Section 3.1. Section 3.2 introduces the LP decoder, and the fundamental polytope is defined and discussed in Section 3.3. We conclude in Section 3.4 with a brief discussion of $\mathcal{C}$-*symmetry*, a property of the fundamental polytope that is both immediately useful and interesting in its own right.

3.1 Background on Linear Programming

This section gives a brief introduction to linear programming. Most material is taken from Bertsimas and Tsitsiklis's excellent text [6].

In this dissertation, we define a *linear program* as being an optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{a}_j^T\mathbf{x} \geq b_j \quad j \in M_1 \\ & \mathbf{a}_j^T\mathbf{x} \leq b_j \quad j \in M_2 \\ & \mathbf{a}_j^T\mathbf{x} = b_j \quad j \in M_3 \\ & x_i \geq 0 \qquad i \in N_1 \\ & x_i \leq 0 \qquad i \in N_2, \end{aligned} \tag{3.1}$$

where $\mathbf{c}, \mathbf{x}$, and $\mathbf{a}_j$ are n-dimensional real vectors and the index sets M_1, M_2, M_3, N_1, and N_2 are finite. The vector $\mathbf{c}$ is called the *cost function* of the linear program. The linear functional $\mathbf{c}^T\mathbf{x}$ is the *objective function*, though it is sometimes also referred to as the *cost function*. The vectors $\mathbf{a}_j$ are called *constraint vectors*, or simply *constraints*. Also called constraints are the (in)equalities of the form $\mathbf{a}_j^T\mathbf{x} \geq b_j, \mathbf{a}_j^T\mathbf{x} \leq b_j$, or $\mathbf{a}_j^T\mathbf{x} = b_j$.

Remark 3.1.1. *In general, a linear program can be either a maximization or a minimization problem. We restrict to minimization problems to simplify definitions and discussion. There is, however, no essential loss of generality in using this convention: maximizing* $\mathbf{c}^T\mathbf{x}$ *is equivalent to minimizing* $(-\mathbf{c})^T\mathbf{x}$.

Any vector $\mathbf{x}$ that satisfies all of the linear program's constraints is called a *feasible solution*, and the set of all feasible solutions is the *feasible set*. A feasible solution $\mathbf{x}_*$ that minimizes the objective function is an *optimal feasible solution*, or simply an *optimal solution*. Note that while the optimality of a vector $\mathbf{x}$ depends on both the constraints and the cost function, the feasibility of a solution $\mathbf{x}$ depends only on the constraints. Since the feasible set for a linear program is an intersection of closed half-spaces in $\mathbb{R}^n$, it is a *polyhedron*. A polyhedron that is also bounded is called a *polytope*. When dealing with polyhedra in the context of linear programming, particular attention is paid to the *extreme points*.

Definition 3.1.2. *Let* $\mathcal{M}$ *be a nonempty polyhedron defined by linear equality and inequality constraints. A point* $\mathbf{x} \in \mathcal{M}$ *is an* extreme point *provided that it is not in the convex hull of* $\mathcal{M} \setminus \{\mathbf{x}\}$. *In other words,* $\mathbf{x}$ *is an extreme point of* $\mathcal{M}$ *if it cannot be written as a convex sum* $\alpha\mathbf{y} + (1-\alpha)\mathbf{z}$ *for* $\mathbf{y}, \mathbf{z} \in \mathcal{M} \setminus \{\mathbf{x}\}$ *and* $\alpha \in (0,1)$.

Remark 3.1.3. *Extreme points are often referred to as vertices in linear programming literature. We refrain from applying the term "vertex" to polyhedra to avoid confusion*

when speaking of both graphs and polyhedra simultaneously. Thus, the terms "node" and "vertex" are reserved for graphs, and "extreme point" is reserved for polyhedra.

In general, a polyhedron need not contain any extreme points. For a bounded polyhedron, however, at least one extreme point is guaranteed to exist [6, Corollary 2.2]. The importance of extreme points is summarized in the next two theorems.

Theorem 3.1.4 ([6], Theorem 2.7)**.** *Suppose that a linear program has feasible set $\mathcal{M}$. If $\mathcal{M}$ has an extreme point and an optimal solution to the linear program exists, then there exists an optimal solution that is an extreme point of $\mathcal{M}$.*

Theorem 3.1.4 implies that we may assume that an optimal solution to a linear program occurs at an extreme point. In particular, when solving a linear program whose underlying polyhedron is bounded, it suffices to consider only extreme points as candidate solutions. On the other hand, given an extreme point $\mathbf{e}$ of a polyhedron $\mathcal{M}$ there is always some linear program with $\mathcal{M}$ as its feasible set whose unique solution is $\mathbf{e}$:

Theorem 3.1.5 ([6], Theorem 2.3)**.** *Let $\mathcal{M}$ be a polyhedron defined by linear equality and inequality constraints. If $\mathbf{e}$ is an extreme point of $\mathcal{M}$, then there exists a cost function $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{e}$ is the unique optimal solution to the following linear program:*

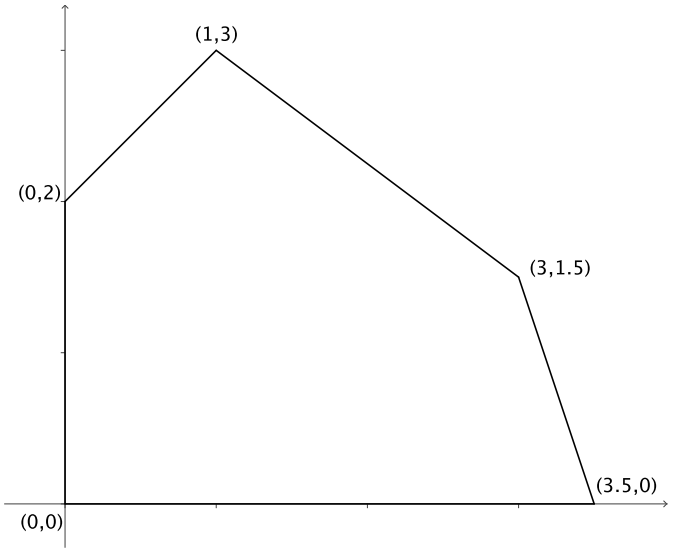
$$\begin{aligned} &\textit{minimize} && \mathbf{c}^T\mathbf{x} \\ &\textit{subject to} && \mathbf{x} \in \mathcal{M}. \end{aligned}$$

In other words, $\mathbf{c}^T\mathbf{e} < \mathbf{c}^T\mathbf{x}$ for all $\mathbf{x} \in \mathcal{M} \setminus \{\mathbf{e}\}$.

Example 3.1.6. *Consider the linear program given by*

$$\begin{aligned} \textit{minimize} \quad & -2x - y \\ \textit{subject to} \quad & -x + y \leq 2 \\ & 3x + y \leq \tfrac{21}{2} \\ & \tfrac{3}{4}x + y \leq \tfrac{15}{4} \\ & x,\ y \geq 0 \end{aligned}$$

The feasible set is the convex region in $\mathbb{R}^2$ *depicted below. This set has five extreme points.*



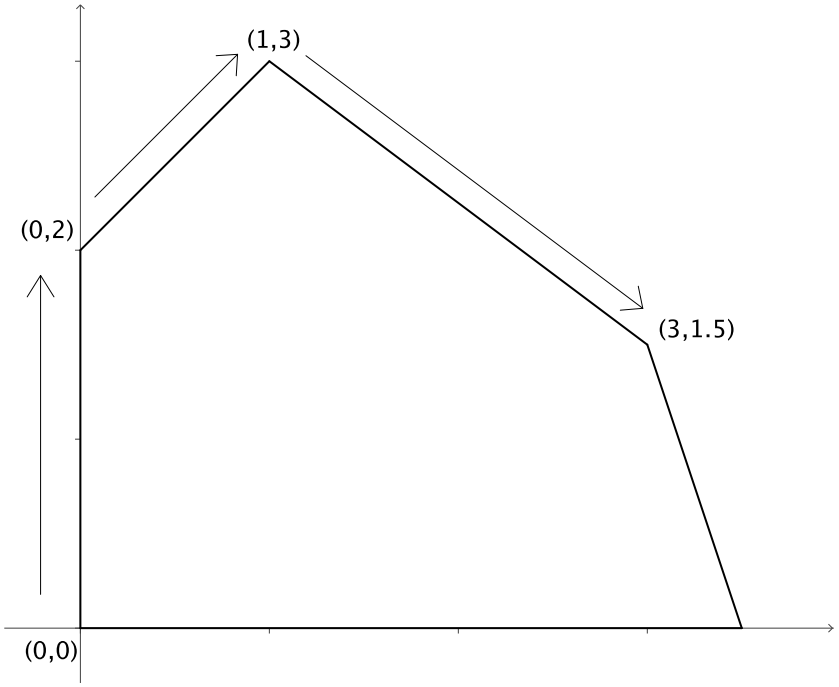
Since the number of extreme points of a polyhedron is guaranteed to be finite [6, Corollary 2.1], one might be tempted to solve a given linear program by first enumerating all extreme points and then computing the cost of each to find the optimal solution. The task of enumerating vertices of general polyhedra can, however, take an amount of time that is exponential in the dimension of the feasible set and the number of constraints. This is the case, for example, when enumerating the vertices of the unit hypercube $[0, 1]^n$. Fortunately, there is a practical method for solving

linear programs that forgoes this brute-force search: the simplex algorithm.

Though a complete and concrete development of the simplex algorithm is not necessary for this work, a high-level understanding of how it operates will be useful. The simplex algorithm can be accurately viewed as a guided walk on the underlying polyhedron. The algorithm is initialized at some extreme point $\mathbf{x}_0$ of $\mathcal{M}$. It then examines all neighboring extreme points (i.e., those that lie on a common edge of $\mathcal{M}$ with $\mathbf{x}_0$) to see if any offers an improvement in cost over $\mathbf{x}_0$. If it finds such an extreme point $\mathbf{x}_1$, the simplex algorithm "walks" to $\mathbf{x}_1$ along an edge of $\mathcal{M}$. The process continues until an optimal solution is found, which is guaranteed to occur after a finite number of steps.

Example 3.1.7. *This example is a continuation of Example 3.1.6. If the simplex algorithm is initialized at* $(0, 0)$*, it is possible for the algorithm to follow the route*

$$(0, 0) \to (0, 2) \to (1, 3) \to (3, 1.5).$$



The corresponding sequence of costs is

$$0 \mapsto -2 \mapsto -5 \mapsto -7.5.$$

The extreme point $(3, 1.5)$ *is an optimal solution to this linear program.*

In order to implement the simplex algorithm on paper or in computer software, it is convenient to represent the feasible set of a linear program algebraically instead of geometrically. We say that a constraint of the form $\mathbf{a}^T\mathbf{x} \geq b, \mathbf{a}^T\mathbf{x} \leq b$, or $\mathbf{a}^T\mathbf{x} = b$ is *active* at $\boldsymbol{\omega}$ if $\mathbf{a}^T\boldsymbol{\omega} = b$, i.e., if the constraint is met with equality. If the polyhedron $\mathcal{M}$ is defined by linear equality and inequality constraints and $\mathbf{x}$ is an element of $\mathbb{R}^n$, then we say that $\mathbf{x}$ is a *basic solution* if all equality constraints of $\mathcal{M}$ are satisfied and the set of active constraint vectors spans all of $\mathbb{R}^n$. If $\mathbf{x}$ is a basic solution that satisfies all of the constraints, it is called a *basic feasible solution*. Using these algebraic notions, Theorem 3.1.8 gives an alternative characterization of extreme points.

Theorem 3.1.8 ([6], Theorem 2.3)**.** *Let* $\mathcal{M}$ *be a nonempty polyhedron defined by linear equality and inequality constraints, and let* $\mathbf{x} \in \mathcal{M}$*. The following are equivalent:*

(a) $\mathbf{x}$ *is an extreme point of* $\mathcal{M}$*,*

(b) $\mathbf{x}$ *is a basic feasible solution of* $\mathcal{M}$*.*

Theorem 3.1.8 relates a geometric invariant to the specific algebraic representation of a polyhedron. It also points to a method of finding extreme points: suppose that $\mathbf{a}_1, \ldots, \mathbf{a}_n$ is a set of n linearly independent constraint vectors, and let $b_1, \ldots, b_n$ be the associated constraint values. Let A be the matrix whose rows are $\mathbf{a}_1, \ldots, \mathbf{a}_n$, and let $\mathbf{b}$ be a column vector whose entries are $b_1, \ldots, b_n$. Since the rows of A are linearly independent, there is a solution to $A\mathbf{x} = \mathbf{b}$, namely $\mathbf{x} = A^{-1}\mathbf{b}$. By the construction

of $\mathbf{x}$, it satisfies the constraints $\mathbf{a}_1, \ldots, \mathbf{a}_n$ with equality. If $\mathbf{x}$ turns out to satisfy the rest of the constraints of the polyhedron, then $\mathbf{x}$ is a basic feasible solution and hence, by Theorem 3.1.8, an extreme point.

Conversely, if $\mathbf{x}$ is an extreme point of a polyhedron, then Theorem 3.1.8 implies that there exists a set of n linearly independent constraint vectors that are active at $\mathbf{x}$; call them $\mathbf{a}_1, \ldots, \mathbf{a}_n$. With A and $\mathbf{b}$ as in the preceding paragraph, the extreme point $\mathbf{x}$ must be the unique solution to $A\mathbf{x} = \mathbf{b}$. This interpretation of an extreme point as a solution to a linear system of equations gives the proof of the following well-known theorem.

Theorem 3.1.9. *Let $\mathcal{M}$ be a non-empty polyhedron. If every constraint vector defining $\mathcal{M}$ has rational coefficients, then every extreme point of $\mathcal{M}$ is a rational vector.*

3.2 Linear Programming Decoding

Let $\mathcal{C}$ be a code, and consider transmission over a binary-input memoryless channel that is completely non-deterministic. Corollary 2.3.2 states that ML decoding can be accomplished by finding a solution to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T \mathbf{c} \\ \text{subject to} \quad & \mathbf{c} \in \mathcal{C}, \end{aligned} \tag{3.2}$$

where $\boldsymbol{\lambda}$ is the vector of log-likelihoods determined by the channel output. By considering codewords as elements of the unit hypercube in $\mathbb{R}^n$ instead of vectors in $\mathbb{F}_2^n$, we can restate this problem as

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{x} \in \text{conv}(\mathcal{C}), \end{aligned}$$

where $\text{conv}(\mathcal{C})$ denotes the set of all convex combinations of elements of $\mathcal{C}$. Since it is the convex hull of a finite set of points, $\text{conv}(\mathcal{C})$ is a polytope [6, Corollary 2.6], and hence can be described in terms of linear equality and inequality constraints. However, as Feldman points out, the number of constraints needed to describe $\text{conv}(\mathcal{C})$ is likely to be quite large [13].

To get around this problem, Feldman defines a relaxation $\mathcal{P}$ of $\text{conv}(\mathcal{C})$. This relaxed polytope $\mathcal{P}$, known as the *fundamental polytope*, is a function of the specific parity-check matrix defining $\mathcal{C}$ (this is the main reason we make the convention that a code is a subspace of $\mathbb{F}_2^n$ paired with a fixed parity-check matrix describing it). The polytope $\mathcal{P}$ is a subset of $[0,1]^n$ that contains the polytope $\text{conv}(\mathcal{C})$, and so every codeword is a vertex of $\mathcal{P}$. Furthermore, the fundamental polytope is more practical to represent with linear equality and inequality constraints than $\text{conv}(\mathcal{C})$ [13]. Section 3.3 gives a precise definition of the fundamental polytope. For the time being, however, we have the vocabulary necessary to define the linear programming decoder.

Definition 3.2.1 ([13])**.** *Let $\mathcal{C}$ be a linear code with parity-check matrix H, and let $\mathcal{P}$ be its fundamental polytope. Suppose that $\mathbf{y}$ is a received vector from a binary-input memoryless channel that is completely non-deterministic, and let $\boldsymbol{\lambda}$ be the associated vector of log-likelihood ratios. The* linear programming (LP) decoder *is the decoder that outputs an optimal solution to*

$$\begin{aligned} &\textit{minimize} \quad \boldsymbol{\lambda}^T \mathbf{x} \\ &\textit{subject to} \quad \mathbf{x} \in \mathcal{P}. \end{aligned} \tag{3.3}$$

By Theorem 3.1.4, we may assume without loss of generality that the linear programming decoder always returns an extreme point of $\mathcal{P}$ as its output. Operating under the paradigm that the output of any decoding algorithm, however strange or

nonsensical, is a *pseudocodeword*, we make the following definition.

Definition 3.2.2. *Let $\mathcal{C}$ be a code with parity-check matrix H and fundamental polytope $\mathcal{P}$. A vector $\boldsymbol{\omega}$ is a* linear programming pseudocodeword *if $\boldsymbol{\omega}$ is an extreme point of $\mathcal{P}$. A linear programming pseudocodeword that is not a zero–one vector corresponding to a codeword is a* nontrivial linear programming pseudocodeword.

In principle, one could define a decoding rule however one wishes. Given a received vector $\mathbf{y}$, one could decode $\mathbf{y}$ by completely disregarding the vector and instead returning whatever happens to be the catch-of-the-day at the local fish market. This is completely well-defined, but an output of "halibut" makes little sense when one is expecting something along the lines of "1011011." In the same way, we would like to have some assurance that the outputs of the LP decoder have some meaningful relation with the transmitted sequence. Such assurance lies in Feldman's fundamental polytope $\mathcal{P}$, which we now present explicitly.

3.3 The Fundamental Polytope

We begin this section with a clarification: what we term the fundamental polytope Feldman refers to as the projected polytope. The reason for this is that the first polytope $\mathcal{Q}$ developed for the purposes of linear programming decoding in [13] is "bigger" than it needs to be; its definition involves many auxiliary variables that never directly influence the objective function of the linear program. When these auxiliary variables are stripped away through projection, the result is the fundamental or projected polytope. We include discussion of this *extended polytope* both to illustrate the relaxation of the integer program to a linear program, and also because it will play a central role in the characterization of linear programming pseudocodewords in Chapter 4.

Definition 3.3.1 ([13]). *Let $\mathcal{C}$ be a code with parity-check matrix H and Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$. For each variable node i in T, create a variable x_i. For each check node j, let E_j denote the collection of all even-sized subsets of $N(j)$. For every pair (j, S) with $S \in E_j$, create a variable $w_{j,S}$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{w} = (w_{j,S})_{j \in \mathcal{J}, S \in E_j}$. The* extended polytope $\mathcal{Q} = \mathcal{Q}(H)$ *is the set of all real vectors $(\mathbf{x}, \mathbf{w})$ satisfying the following constraints:*

(a) $0 \leq x_i \leq 1$ *for all* $i = 1, 2, \ldots, n$,

(b) $w_{j,S} \geq 0$ *for all checks* j *and subsets* $S \in E_j$,

(c) $\sum_{S \in E_j} w_{j,S} = 1$ *for all checks* j, *and*

(d) $\sum_{\{S \in E_j \mid i \in S\}} w_{j,S} = x_i$ *for all* $j \in \mathcal{J}$ *and variable nodes* $i \in N(j)$.

Feldman gives motivation for Definition 3.3.1 in [13], which we repeat here for completeness. Suppose that $\mathbf{x}$ is a codeword. Then $x_i \in \{0, 1\}$ for all i, and $H\mathbf{x} = \mathbf{0} \in \mathbb{F}_2^n$. In other words, the projection $\mathbf{x}^{N(j)}$ of $\mathbf{x}$ onto the neighborhood of every check node j must have an even number of 1's. For each check j, define $S_j = \text{supp}(\mathbf{x}^{N(j)})$, and then set $w_{j,S_j} = 1$ and all other $w_{j,S} = 0$. One can see that the vector formed in this way satisfies all the constraints of $\mathcal{Q}$. The variables x_i indicate the bit values of the variable nodes, and the variables $w_{j,S}$ indicate how the bit nodes satisfy the local parity constraint at check j.

Definition 3.3.2 ([13]). *Let $\mathcal{C}$ be a code with parity-check matrix H and Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$. The* fundamental polytope $\mathcal{P} = \mathcal{P}(H)$ *of $\mathcal{C}$ is the set of all vectors $\mathbf{x} \in \mathbb{R}^n$ satisfying the following constraints:*

(a) $0 \leq x_i \leq 1$ *for all* $i = 1, 2, \ldots, n$, *and*

(b) $\sum_{i \in S} x_i + \sum_{i \in N(j) \setminus S} (1 - x_i) \leq |N(j)| - 1$ *for all pairs* (j, S)*, where* $j \in \mathcal{J}$ *and* S *is a subset of* $N(j)$ *with odd cardinality.*

Feldman offers a heuristic justification for the constraints defining $\mathcal{P}$, which again we reproduce here for the sake of completeness. Since all codewords lie in $\{0,1\}^n$, we do not wish for the components of $\mathbf{x} \in \mathcal{P}$ to be able to assume values outside the interval $[0,1]$, but since linear programs require convexity we permit these variables to assume intermediate values. The other nontrivial family of constraints is best explained in the integer case, for then these constraints can be seen as forbidding any "bad" configurations. To be explicit, suppose that $\mathbf{x} \in \{0,1\}^n$ is not a codeword. Then there is a check node $j \in \mathcal{J}$ such that $\mathbf{x}$ assigns an odd number of ones to the variable nodes of $N(j)$ in T. Letting S denote this subset, we see that $\mathbf{x}$ is such that

$$\begin{aligned}
\sum_{i \in S} x_i + \sum_{i \in N(j) \setminus S} (1 - x_i) &= \sum_{i \in S} 1 + \sum_{i \in N(j) \setminus S} 1 \\
&= |S| + |N(j) \setminus S| \\
&= |N(j)| \\
&> |N(j)| - 1.
\end{aligned}$$

Hence, the non-codeword integer vector $\mathbf{x}$ is excluded from $\mathcal{P}$.

Although the definitions of the extended polytope $\mathcal{Q}(H)$ and the fundamental polytope $\mathcal{P}(H)$ appear to be quite different, there is a tight link between these two polytopes.

Theorem 3.3.3 ([13], Theorem 5.1)**.** *Let* $\mathcal{C}$ *be a code with parity-check matrix* H*. Then* $\mathcal{P}(H) = \{\mathbf{x} \mid$ *there exists a vector* $\mathbf{w}$ *such that* $(\mathbf{x}, \mathbf{w}) \in \mathcal{Q}(H)\}$*.*

As mentioned in Section 3.2, the properties of the polytope over which the linear

programming decoder is defined can make or break this decoder in terms of its usefulness. Fortunately, the fundamental polytope has a key property relating its integer points back to codewords.

Definition 3.3.4 ([13]). *Let $\mathcal{C}$ be a code. A polytope $\mathcal{M} \subseteq [0,1]^n$ is called* proper *provided that $\mathcal{M} \cap \{0,1\}^n = \mathcal{C}$, i.e., if the set of integral points in $\mathcal{M}$ is exactly the set of codewords.*

Theorem 3.3.5 ([13]). *Let $\mathcal{C}$ be a code with parity-check matrix H. Then the fundamental polytope $\mathcal{P} = \mathcal{P}(H)$ of $\mathcal{C}$ is proper.*

Using the properness of the fundamental polytope, Feldman [13] is able to derive one of the main properties of the linear programming decoder: the *ML-certificate property* [13]. Suppose that the linear programming decoder outputs an integer vector $\mathbf{z}$ such that $\boldsymbol{\lambda}^T\mathbf{z} < \boldsymbol{\lambda}^T\mathbf{x}$ for all $\mathbf{x} \in \mathcal{P} \setminus \{\mathbf{z}\}$. Since the fundamental polytope is proper, the zero–one vector $\mathbf{z}$ must be a codeword. Further, the properness of the polytope implies that $\text{conv}(\mathcal{C}) \subseteq \mathcal{P}$. Thus, $\boldsymbol{\lambda}^T\mathbf{z} < \boldsymbol{\lambda}^T\mathbf{x}$ for all $\mathbf{x} \in \text{conv}(\mathcal{C}) \setminus \{\mathbf{z}\}$, so $\mathbf{z}$ is a solution to Problem (3.2) and hence is an ML codeword [13].

By the ML-certificate property, an integer output of the linear programming decoder is guaranteed to be an optimal maximum-likelihood codeword. This does not, however, imply that the LP decoder attains ML performance. In addition to codewords, there are usually fractional extreme points of $\mathcal{P}$ as well (since the constraints defining $\mathcal{P}$ are linear with integer coefficients, Theorem 3.1.9 implies that non-integer extreme points are rational). Suppose that $\boldsymbol{\omega}$ is a fractional extreme point of $\mathcal{P}$. By Theorem 3.1.5, there exists a cost function $\boldsymbol{\lambda}$ such that the solution to Problem (3.3) is $\boldsymbol{\omega}$. In such a case, the LP decoder declares a decoding error while the ML decoder provides an optimal solution. The presence of these fractional extreme points, or

nontrivial LP pseudocodewords, is therefore exactly what prevents the LP decoder from attaining ML performance.

3.4 Notes on $\mathcal{C}$-Symmetry

When studying the performance of linear codes, analysis is often simplified when one can make the *all-zeros assumption*, i.e., when one may assume that the transmitted codeword is $\mathbf{0}$ without changing the word-error rate. Feldman [13] shows that the all-zeros assumption is valid for the LP decoder provided that the channel is both completely non-deterministic and symmetric. To do this, the notion of *$\mathcal{C}$-symmetry* is introduced. Loosely speaking, a $\mathcal{C}$-symmetric polytope is a polytope whose symmetry group admits the additive group structure of the code.

We introduce $\mathcal{C}$-symmetry mainly to utilize it in the construction of nontrivial linear programming pseudocodewords in Chapter 4. We also make explicit the relationship between the symmetry of the fundamental polytope and the additive structure of a code: if $\mathrm{Sym}(\mathcal{P})$ is the group of rotational and reflective symmetries of $\mathcal{P}$, then $\mathcal{C}$ is isomorphic to a subgroup of $\mathrm{Sym}(\mathcal{P})$. We do not fully develop this latter idea, but leave it as an object of future pursuit.

Definition 3.4.1 ([13])**.** *Let $\mathbf{f} \in [0,1]^n$ and $\mathbf{y} \in \{0,1\}^n$ be given. The* relative point $\mathbf{f}^{[\mathbf{y}]}$ of $\mathbf{f}$ with respect to $\mathbf{y}$ *is the point whose coordinates are given by $f_i^{[\mathbf{y}]} = |f_i - y_i|$ for all $i = 1, 2, \ldots, n$.*

Definition 3.4.2 ([13])**.** *Let $\mathcal{C}$ be a code with parity-check matrix H. A proper polytope $\mathcal{M}$ is said to be* $\mathcal{C}$-symmetric *provided that for all points $\mathbf{x} \in \mathcal{M}$ and all codewords $\mathbf{c} \in \mathcal{C}$ the relative point $\mathbf{x}^{[\mathbf{c}]}$ is also an element of $\mathcal{M}$.*

With these definitions, Feldman proves the following two theorems.

Theorem 3.4.3 ([13], Theorem 4.6)**.** *Let $\mathcal{C}$ be a code, and let $\mathcal{M}$ be a $\mathcal{C}$-symmetric polytope. For all extreme points $\mathbf{x}$ of $\mathcal{M}$ and for all codewords $\mathbf{c} \in \mathcal{C}$, the relative point $\mathbf{x}^{[\mathbf{c}]}$ is also an extreme point of $\mathcal{M}$.*

Theorem 3.4.4 ([13], Theorem 5.4)**.** *Let $\mathcal{C}$ be a code with parity-check matrix H. The fundamental polytope $\mathcal{P} = \mathcal{P}(H)$ of $\mathcal{C}$ is $\mathcal{C}$-symmetric.*

Example 3.4.5. *Let $\mathcal{C}$ be the code presented by the parity-check matrix H of Figure 2.1:*

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

Let $\mathcal{P}$ be the fundamental polytope of $\mathcal{C}$. Since $\mathbf{c}_1 = (1,1,1,0,0,0,1,1,1)$ and $\mathbf{c}_2 = (1,0,1,1,1,0,0,1,1)$ are both codewords, the properness of $\mathcal{C}$ implies that both are elements of $\mathcal{P}$. We have that $\mathbf{c}_1^{[\mathbf{c}_2]} = \mathbf{c}_2^{[\mathbf{c}_1]} = (0,1,0,1,1,0,1,0,0)$. Note that $\mathbf{c}_1^{[\mathbf{c}_2]}$ is identical to the binary sum of $\mathbf{c}_1$ and $\mathbf{c}_2$ and hence is also a codeword.

Consider now $\mathbf{c}_3 = (1,1,1,0,0,0,0,0,0), \mathbf{c}_4 = (0,1,0,1,1,0,1,0,0)$, and $\mathbf{c}_5 = (1,0,0,1,0,1,0,0,1)$. The points $\mathbf{c}_3, \mathbf{c}_4$, and $\mathbf{c}_5$ are codewords and hence are extreme points of $\mathcal{P}$. As we will show in Section 4.4, the point $\boldsymbol{\omega} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 1, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ is also an extreme point of $\mathcal{P}$. Since $\mathcal{P}$ is convex, the point $\mathbf{p} = \frac{1}{2}\mathbf{c}_3 + \frac{1}{6}\mathbf{c}_4 + \frac{1}{3}\boldsymbol{\omega} = (\frac{2}{3}, \frac{5}{6}, \frac{2}{3}, \frac{1}{2}, \frac{1}{6}, 0, \frac{1}{3}, \frac{1}{6}, \frac{1}{6})$ is an element of $\mathcal{P}$. By Theorem 3.4.4, the point $\mathbf{p}^{[\mathbf{c}_5]} = (\frac{1}{3}, \frac{5}{6}, \frac{2}{3}, \frac{1}{2}, \frac{1}{6}, 1, \frac{1}{3}, \frac{1}{6}, \frac{5}{6})$ is also in $\mathcal{P}$.

Let $\mathcal{C}$ be a code with parity-check matrix H and fundamental polytope $\mathcal{P}$, and let $\mathbf{c} \in \mathcal{C}$ be a codeword. Define a map $\phi_{\mathbf{c}} : \mathcal{P} \to [0,1]^n$ by $\phi_{\mathbf{c}}(\mathbf{x}) := \mathbf{x}^{[\mathbf{c}]}$.

Proposition 3.4.6. *The map $\phi_{\mathbf{c}}$ is an element of Sym($\mathcal{P}$).*

Proof. Define $\phi_i : [0,1]^n \to [0,1]^n$ by reflection through the plane $x_i = \frac{1}{2}$. Note that $\phi_i(\mathbf{x})$ has ith coordinate equal to $1 - x_i$ and is otherwise equal to $\mathbf{x}$. Using this fact, we see that ϕ_i^2 is the identity map and that $\phi_i \circ \phi_j = \phi_j \circ \phi_i$.

Enumerating the support of $\mathbf{c}$ as $\{i_1 < i_2 < \cdots < i_m\}$, we have that $\phi_{\mathbf{c}}(\mathbf{x}) = \mathbf{x}^{[\mathbf{c}]} = \left(\phi_{i_1} \circ \phi_{i_2} \circ \cdots \circ \phi_{i_m}\right)(\mathbf{x})$. If $\mathbf{x} \in \mathcal{P}$, Theorem 3.4.4 implies that $\mathbf{x}^{[\mathbf{c}]}$ and hence $\left(\phi_{i_1} \circ \phi_{i_2} \circ \cdots \circ \phi_{i_m}\right)(\mathbf{x})$ is also in $\mathcal{P}$. Thus, $\phi_{\mathbf{c}}$ maps $\mathcal{P}$ into $\mathcal{P}$. Since $\phi_{\mathbf{c}} = \phi_{i_1} \circ \phi_{i_2} \circ \cdots \circ \phi_{i_m}$ is a composition of reflections, it is itself a rotation or reflection (depending on the parity of $|\text{supp}(\mathbf{c})|$). We conclude that $\phi_{\mathbf{c}}$ is an element of Sym($\mathcal{P}$). □

Theorem 3.4.7. *Let $\mathcal{C}$ be a code with parity-check matrix H and fundamental polytope $\mathcal{P}$. The map $\Phi : \mathcal{C} \to Sym(\mathcal{P})$ given by $\mathbf{c} \mapsto \phi_{\mathbf{c}}$ is an injective group homomorphism.*

Proof. Let $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$ be given. By Proposition 3.4.6, both $\Phi(\mathbf{c}_1) = \phi_{\mathbf{c}_1}$ and $\Phi(\mathbf{c}_2) = \phi_{\mathbf{c}_2}$ are elements of Sym($\mathcal{P}$). We also have that $\phi_{\mathbf{c}_1} = \phi_{\mathbf{c}_2}$ if and only if $\text{supp}(\mathbf{c}_1) = \text{supp}(\mathbf{c}_2)$, if and only if $\mathbf{c}_1 = \mathbf{c}_2$ (since $\mathcal{C}$ is a binary code). Thus, the map Φ is injective. It remains to be shown that $\Phi(\mathbf{c}_1 + \mathbf{c}_2) = \Phi(\mathbf{c}_1)\Phi(\mathbf{c}_2)$.

By Definition 3.4.1, we have that $\mathbf{x}^{[\mathbf{c}_1+\mathbf{c}_2]} = \left(\mathbf{x}^{[\mathbf{c}_2]}\right)^{[\mathbf{c}_1]}$. This can be restated as $\phi_{\mathbf{c}_1+\mathbf{c}_2} = \phi_{\mathbf{c}_1} \circ \phi_{\mathbf{c}_2}$, and finally as $\Phi(\mathbf{c}_1 + \mathbf{c}_2) = \Phi(\mathbf{c}_1)\Phi(\mathbf{c}_2)$. □

Combining Theorem 3.4.7 with the fact that elements of Sym($\mathcal{P}$) preserve the spatial relationships between points of $\mathcal{P}$, we have that Φ induces a faithful group action of $\mathcal{C}$ on the set of extreme points of $\mathcal{P}$ (this is essentially a more formal statement of the conclusion of Theorem 3.4.3). The action of $\mathcal{C}$ on the set of extreme points of $\mathcal{P}$ will serve a practical purpose in Chapter 4 when we discuss how to generate a complete set of extreme points for the fundamental polytope of a cycle

code. Though at the present we only use this group action as a method of jumping from one extreme point to another, it would be interesting to study for its own sake. In particular, it would be interesting to examine what, if any, information about a fractional extreme point can be gleaned from its stabilizer subgroup or from its orbit.

Chapter 4

A Characterization of Linear Programming Pseudocodewords for Cycle Codes

As discussed in Section 3.2, we may take the output of the linear programming decoder to be an extreme point of the fundamental polytope. If the output is an integer vector, then the properness of the fundamental polytope implies that the output is a codeword, and the ML-certificate property implies that this codeword is the ML codeword. On the other hand, if the output of the LP decoder is a fractional extreme point of the fundamental polytope, then a decoding error is declared. To compare the performance of the LP decoder to the optimal ML decoder, it is therefore important to identify sources of nontrivial LP pseudocodewords. Since the fundamental polytope is a function of the parity-check matrix defining the code, it is reasonable to begin this task by examining structures of the parity-check matrix and the corresponding Tanner graph that give rise to these fractional extreme points.

In this chapter we characterize the set of linear programming pseudocodewords in

the case of cycle codes. We first introduce some terminology that will aid in stating the characterization succinctly.

Definition 4.0.1. *Let $G = (V, E)$ be a (multi)graph, and recall that every edge $e \in E$ can be regarded as a set consisting of two vertices $v_1, v_2 \in V$ (if $v_1 = v_2$, then the edge is a loop). For any $A \subseteq E$, define the* subgraph of G spanned by A*, or, more simply, the* subgraph spanned by A*, to be subgraph of G with vertex set $\left(\bigcup_{e \in A} e\right)$ and edge set A.*

Recall from Section 3.3 that for a code $\mathcal{C}$ with Tanner graph T and fundamental polytope $\mathcal{P}$, points in $\mathcal{P}$ are indexed by the variable nodes in T. We therefore view points in $\mathcal{P}$ as numeric labelings on the variable nodes of the Tanner graph; this is analogous to identifying codewords with valid configurations (see Section 2.1). For cycle codes, we may also view points in the fundamental polytope as numeric labels on the edges of the normal graph. With this in mind, the main result of this chapter is given in Theorem 4.0.2.

Theorem 4.0.2. *Let $\mathcal{C}$ be a cycle code of length n with normal graph N and fundamental polytope $\mathcal{P}$. A vector $\boldsymbol{\omega} \in \mathbb{R}^n$ is a linear programming pseudocodeword of $\mathcal{P}$ if and only if the following four conditions hold:*

(a) $\boldsymbol{\omega} \in \mathcal{P}$.

(b) $\boldsymbol{\omega} \in \{0, \frac{1}{2}, 1\}^n$.

(c) The subgraph Γ of N spanned by the edges that are assigned a value of $\frac{1}{2}$ by $\boldsymbol{\omega}$ is 2-regular. Equivalently, Γ is a vertex-disjoint union of simple cycles $\gamma_1, \gamma_2, \ldots, \gamma_\ell$.

(d) For every simple cycle γ_i in Γ, the number of edges in N incident to γ_i that are assigned a value of 1 by $\boldsymbol{\omega}$ is odd.

The proof of Theorem 4.0.2 offered herein makes use of both presentations of the fundamental polytope given in Section 3.3. As such, the proof is divided into two main parts: Sections 4.2 and 4.3. Section 4.4 discusses how to use Theorem 4.0.2 in conjunction with $\mathcal{C}$-symmetry to obtain a complete list of LP pseudocodewords for a given cycle code. Finally, Section 4.5 discusses the relation between this characterization of LP pseudocodewords and Dreher's characterization of *minimal* LP pseudocodewords [12]. We begin by proving an important technical lemma in Section 4.1.

4.1 A Technical Lemma on Active Constraints

To prove the forward direction of Theorem 4.0.2 in Section 4.2, we adopt a viewpoint suggested by Theorem 3.1.8 – namely, we view an extreme point of a polytope as a solution to a linear system of equations. In our case, the equations forming this system come from the constraint vectors of the fundamental polytope that are satisfied with equality. In this section we develop vocabulary particular to the fundamental polytope that aids in discussing whether and how a vector satisfies a given constraint. We then prove a key lemma that is used in Section 4.2.

Let $\mathcal{C}$ be a code defined by parity-check matrix H, and let $T = (\mathcal{I} \cup \mathcal{J}, E)$ be its Tanner graph. Without loss of generality, we may assume that $\mathcal{I} = \{1, 2, \ldots, n\}$. Recall from Section 3.3 that the fundamental polytope $\mathcal{P} = \mathcal{P}(H)$ is defined as the

set of all $\mathbf{x} \in \mathbb{R}^n$ satisfying the following set of inequalities:

$$\begin{aligned} x_i &\leq 1 && i = 1, 2, \ldots, n \\ x_i &\geq 0 && i = 1, 2, \ldots, n \\ \sum_{i \in S} x_i + \sum_{i \in N(j) \setminus S} (1 - x_i) &\leq |N(j)| - 1 && \text{for all } j \in \mathcal{J} \text{ and for all } S \subseteq N(j) \text{ such that } |S| \text{ is odd} \end{aligned}$$

We may rearrange this to put it in the standard form presented in Section 3.1:

$$x_i \leq 1 \qquad i = 1, 2, \ldots, n \tag{4.1}$$

$$x_i \geq 0 \qquad i = 1, 2, \ldots, n \tag{4.2}$$

$$\sum_{i \in S} x_i + \sum_{i \in N(j) \setminus S} -x_i \leq |S| - 1 \qquad \text{for all } j \in \mathcal{J} \text{ and for all } S \subseteq N(j) \text{ such that } |S| \text{ is odd} \tag{4.3}$$

Constraints in the form of Inequalities (4.1) and (4.2) simply keep the fundamental polytope from escaping the unit hypercube. Of more interest are constraints in the form of Inequality (4.3) that are imposed by the check nodes of the Tanner graph. There is a natural bijection between constraint vectors in the form of Inequality (4.3) and pairs (j, S), where S is an odd-sized subset of $N(j)$. If S is an odd-sized subset of $N(j)$, there is a corresponding constraint vector of $\mathcal{P}$ that assigns coefficients of $+1$'s to $i \in S$, -1's to $i \in N(j) \setminus S$, and 0's to all other i. Similarly, given a constraint vector for $\mathcal{P}$ that is not of the form $x_i \leq 1$ or $x_i \geq 0$, one can easily find an odd-sized subset S of $N(j)$ for some j. We therefore use the notation (j, S) to denote the constraint vector of $\mathcal{P}$ corresponding to the check j and the odd-sized subset $S \subseteq N(j)$.

Definition 4.1.1. *Let $\mathcal{C}$ be a code presented by a parity-check matrix H. Let $T =$*

$(\mathcal{I} \cup \mathcal{J}, E)$ be its Tanner graph, and let $\mathcal{P}$ be its fundamental polytope. For a check node $j \in \mathcal{J}$, a subset $S \subseteq N(j)$, and a vector $\mathbf{x} \in [0,1]^n$, the cost *of S at j relative to $\mathbf{x}$ is given by*

$$\kappa_{j,\mathbf{x}}(S) = \sum_{i \in S} x_i + \sum_{i \in N(j) \setminus S} (1 - x_i).$$

If there exists an odd-sized subset $S \subseteq N(j)$ such that $\kappa_{j,\mathbf{x}}(S) = |N(j)| - 1$, then we say that the check j is active *at $\mathbf{x}$ and that the set S is* active *for $\mathbf{x}$ at j.*

Note that a vector $\mathbf{x} \in [0,1]^n$ is in the fundamental polytope if and only if $\kappa_{j,\mathbf{x}}(S) \leq |N(j)| - 1$ for all pairs (j, S) such that S is an odd-sized subset of $N(j)$. The following lemma follows directly from Definition 4.1.1 in conjunction with the definition of the fundamental polytope.

Lemma 4.1.2. *Let $\mathcal{C}$ be a code presented by a parity-check matrix H. Let $T = (\mathcal{I} \cup \mathcal{J}, E)$ be its Tanner graph, and let $\mathcal{P}$ be its fundamental polytope. If $j \in \mathcal{J}$ is active at some $\mathbf{x} \in \mathcal{P}$ and $\alpha \in (0,1)$, then $\mathbf{x}$ can assume the value of α in at most $\max\{\frac{1}{\alpha}, \frac{1}{1-\alpha}\}$ positions of $N(j)$. In particular, $\mathbf{x}$ can assume the value of $\frac{1}{2}$ at most twice in $N(j)$.*

Proof. Let $D \subseteq \mathcal{I}$ be the set of variable nodes in $N(j)$ that are assigned a value of α by $\mathbf{x}$, and let $d := |D|$. Since j is active at $\mathbf{x}$, there exists an odd-sized subset $S \subseteq N(j)$ such that S is active for $\mathbf{x}$ at j. We consider two cases.

If $\alpha \geq \frac{1}{2}$, then $\alpha \geq 1 - \alpha$. We therefore have

$$|N(j)| - 1 = \kappa_{j,\mathbf{x}}(S) \leq |N(j)| - d(1 - \alpha),$$

which implies that $d \leq \frac{1}{1-\alpha} = \max\{\frac{1}{\alpha}, \frac{1}{1-\alpha}\}$.

On the other hand, if $\alpha < \frac{1}{2}$, then $\alpha < 1 - \alpha$. We therefore have

$$|N(j)| - 1 = \kappa_{j,\mathbf{x}}(S) \leq |N(j)| - d\alpha,$$

which implies that $d \leq \frac{1}{\alpha} = \max\{\frac{1}{\alpha}, \frac{1}{1-\alpha}\}$. □

Recalling that the symmetric difference between two sets S_1 and S_2 is $S_1 \triangle S_2 := (S_1 \cup S_2) \setminus (S_1 \cap S_2)$, we now prove the main result of this section.

Lemma 4.1.3. *Let $\mathcal{C}$ be a code presented by a parity-check matrix H. Let $T = (\mathcal{I} \cup \mathcal{J}, E)$ be its Tanner graph, and let $\mathcal{P}$ be its fundamental polytope. Fix $\mathbf{x} \in \mathcal{P}$ and $j \in \mathcal{J}$. If j is active at $\mathbf{x}$ and $\mathbf{x}$ is not integral on $N(j)$ (i.e., there is some $i \in N(j)$ such that $0 < x_i < 1$), then there are at most two active sets for $\mathbf{x}$ at j. Moreover, in the case that the number of active sets for $\mathbf{x}$ at j is exactly two, the symmetric difference of these two sets consists of exactly two variable nodes, and these two variable nodes are exactly the neighbors of j at which $\mathbf{x}$ is not integral.*

Proof. Let $\mathbf{x} \in \mathcal{P}$ and $j \in \mathcal{J}$ be given such that $\mathbf{x}^{N(j)}$ is not integral and j is active at $\mathbf{x}$. For each $i \in N(j)$, the value x_i appears in the expression for $\kappa_{j,\mathbf{x}}(S)$ as x_i if $i \in S$ or as $1 - x_i$ if $i \in N(j) \setminus S$. The basic idea of this proof is to address the following question: how can x_i make the largest contribution to $\kappa_{j,\mathbf{x}}(S)$? It is clear that $x_i > 1 - x_i$ if and only if $x_i > \frac{1}{2}$ and that $x_i < 1 - x_i$ if and only if $x_i < \frac{1}{2}$. From this, we see that $\kappa_{j,\mathbf{x}}(S)$ is maximized exactly when S consists of all indices i such that $x_i > \frac{1}{2}$, and possibly some indices i with $x_i = \frac{1}{2}$. In a search for sets S that are active for $\mathbf{x}$ at j we can only consider odd-sized subsets of $N(j)$. This parity-based issue is highly dependent on $\mathbf{x}$ itself, about which we know little. We therefore consider several cases, each of which takes the greedy solutions that ignore parity and mashes them into solutions that respect this parity condition.

Define the following sets:

- $\mathcal{L} := \{i \in N(j) \mid x_i < \frac{1}{2}\}$
- $\mathcal{E} := \{i \in N(j) \mid x_i = \frac{1}{2}\}$
- $\mathcal{G} := \{i \in N(j) \mid x_i > \frac{1}{2}\}$
- $\mathcal{Q} := \{i \in N(j) \mid |\frac{1}{2} - x_i| \leq |\frac{1}{2} - x_{i'}| \text{ for all } i' \in N(j)\}$.

The set $\mathcal{Q}$ contains all positions i such that x_i is closest to $\frac{1}{2}$ among all x_ℓ with $\ell \in N(j)$; note that $\mathcal{Q} = \mathcal{E}$ if $\mathcal{E} \neq \emptyset$.

Since j is active at $\mathbf{x}$, by Lemma 4.1.2 we know that $0 \leq |\mathcal{E}| \leq 2$. Suppose first that $|\mathcal{E}| = 2$, and write $\mathcal{Q} = \mathcal{E} = \{q_1, q_2\}$. If $|\mathcal{G}|$ is odd, then the sets $S_1 := \mathcal{G}$ and $S_2 := \mathcal{G} \cup \mathcal{Q}$ are the only two odd-sized subsets of $N(j)$ that maximize $\kappa_{j,\mathbf{x}}$ over all odd-sized subsets of $N(j)$. Moreover, since j is active at $\mathbf{x}$, the maximum value achieved by $\kappa_{j,\mathbf{x}}$ over all odd-sized subsets of $N(j)$ is precisely $|N(j)| - 1$. Having $|\mathcal{E}| = 2$ therefore forces all values on $N(j) \setminus \mathcal{Q}$ to be integral. Since S_1 and S_2 satisfy $S_1 \triangle S_2 = \mathcal{Q}$ and $|S_1 \triangle S_2| = 2$, the lemma is proved in this case. If $|\mathcal{G}|$ is even, then the sets $S_1 := \mathcal{G} \cup \{q_1\}$ and $S_2 := \mathcal{G} \cup \{q_2\}$ are the only two odd-sized subsets that maximize $\kappa_{j,\mathbf{x}}$. As in the previous case, the maximum value achieved by $\kappa_{j,\mathbf{x}}$ over all odd-sized subsets of $N(j)$ is $|N(j)| - 1$, so $|\mathcal{E}| = 2$ again forces all values on $N(j) \setminus \mathcal{Q}$ to be integral. Since $S_1 \triangle S_2 = \mathcal{Q}$, we are done in this case as well.

Now suppose that $|\mathcal{E}| = 1$. If $|\mathcal{G}|$ is odd, then $S = \mathcal{G}$ is the unique maximizer of $\kappa_{j,\mathbf{x}}$ over all odd-sized subsets of $N(j)$. If $|\mathcal{G}|$ is even, then $S = \mathcal{G} \cup \mathcal{E}$ is the unique maximizer of $\kappa_{j,\mathbf{x}}$ over all odd-sized subsets of $N(j)$. In either situation there is at most one active set at j, and so the lemma is proved in this case.

Finally, assume that $|\mathcal{E}| = 0$. If $|\mathcal{G}|$ is odd, then $S = \mathcal{G}$ is the unique maximizer of $\kappa_{j,\mathbf{x}}$ over all odd-sized subsets of $N(j)$ and the lemma is proved. If $|\mathcal{G}|$ is even, then

the collection of all active sets for $\mathbf{x}$ at j is given by $\{S_q \mid q \in \mathcal{Q}\}$, where S_q is defined as follows:

$$S_q := \begin{cases} \mathcal{G} \cup \{q\} & \text{if } q \notin \mathcal{G} \\ \mathcal{G} \setminus \{q\} & \text{if } q \in \mathcal{G}. \end{cases}$$

If $|\mathcal{Q}| = 1$, there is a unique set S_q that maximizes $\kappa_{j,\mathbf{x}}(S)$ and we are done. So suppose that $|\mathcal{Q}| > 1$, and let q_1 and q_2 be distinct elements of $\mathcal{Q}$. By the definition of $\mathcal{Q}$, we have $x_{q_1} = x_{q_2}$ if $q_1, q_2 \in \mathcal{L}$ or $q_1, q_2 \in \mathcal{G}$, and $x_{q_1} = 1 - x_{q_2}$ if $q_1 \in \mathcal{L}$ and $q_2 \in \mathcal{G}$ or vice-versa. We will soon deduce that $x_i \in \{0, 1\}$ for all $i \in N(j) \setminus \{q_1, q_2\}$, but first we do a computation for each of these four cases.

If $q_1, q_2 \in \mathcal{L}$, then

$$\begin{aligned} |N(j)| - 1 &= \kappa_{j,\mathbf{x}}(S_{q_1}) \\ &= \sum_{i \in S_{q_1}} x_i + \sum_{i \in S_{q_1}^c} (1 - x_i) \\ &= \sum_{i \in S_{q_1} \setminus \{q_1\}} x_i + x_{q_1} + (1 - x_{q_2}) + \sum_{i \in S_{q_1}^c \setminus \{q_2\}} (1 - x_i) \\ &= \sum_{i \in S_{q_1} \setminus \{q_1\}} x_i + 1 + \sum_{i \in S_{q_1}^c \setminus \{q_2\}} (1 - x_i). \end{aligned}$$

If $q_1, q_2 \in \mathcal{G}$, we have again that $\kappa_{j,\mathbf{x}}(S_{q_1}) = |N(j)| - 1$. This implies that

$$\begin{aligned} |N(j)| - 1 &= \kappa_{j,\mathbf{x}}(S_{q_1}) \\ &= \sum_{i \in S_{q_1}} x_i + \sum_{i \in S_{q_1}^c} (1 - x_i) \\ &= \sum_{i \in S_{q_1} \setminus \{q_2\}} x_i + x_{q_2} + (1 - x_{q_1}) + \sum_{i \in S_{q_1}^c \setminus \{q_1\}} (1 - x_i) \\ &= \sum_{i \in S_{q_1} \setminus \{q_2\}} x_i + 1 + \sum_{i \in S_{q_1}^c \setminus \{q_1\}} (1 - x_i). \end{aligned}$$

If $q_1 \in \mathcal{L}$ and $q_2 \in \mathcal{G}$, we have

$$\begin{aligned}
|N(j)| - 1 &= \kappa_{j,\mathbf{x}}(S_{q_1}) \\
&= \sum_{i \in S_{q_1}} x_i + \sum_{i \in S_{q_1}^c} (1 - x_i) \\
&= \sum_{i \in S_{q_1} \setminus \{q_1, q_2\}} x_i + x_{q_1} + x_{q_2} + \sum_{i \in S_{q_1}^c} (1 - x_i) \\
&= \sum_{i \in S_{q_1} \setminus \{q_1, q_2\}} x_i + x_{q_1} + (1 - x_{q_1}) + \sum_{i \in S_{q_1}^c} (1 - x_i) \\
&= \sum_{i \in S_{q_1} \setminus \{q_1, q_2\}} x_i + 1 + \sum_{i \in S_{q_1}^c} (1 - x_i).
\end{aligned}$$

Finally, if $q_1 \in \mathcal{G}$ and $q_2 \in \mathcal{L}$, we have

$$\begin{aligned}
|N(j)| - 1 &= \kappa_{j,\mathbf{x}}(S_{q_1}) \\
&= \sum_{i \in S_{q_1}} x_i + \sum_{i \in S_{q_1}^c} (1 - x_i) \\
&= \sum_{i \in S_{q_1}} x_i + (1 - x_{q_1}) + (1 - x_{q_2}) + \sum_{i \in S_{q_1}^c \setminus \{q_1, q_2\}} (1 - x_i) \\
&= \sum_{i \in S_{q_1}} x_i + x_{q_2} + (1 - x_{q_2}) + \sum_{i \in S_{q_1}^c \setminus \{q_1, q_2\}} (1 - x_i) \\
&= \sum_{i \in S_{q_1}} x_i + 1 + \sum_{i \in S_{q_1}^c \setminus \{q_1, q_2\}} (1 - x_i).
\end{aligned}$$

In each of these four cases, we conclude that $\mathbf{x}$ must be integral on $N(j) \setminus \{q_1, q_2\}$; otherwise, it would not be possible for $\kappa_{j,\mathbf{x}}(S_{q_1})$ to attain the value $|N(j)| - 1$. By assumption we have that $\mathbf{x}$ must be non-integral in at least one position of $N(j)$, so we conclude that either $0 < x_{q_1} < 1$ or $0 < x_{q_2} < 1$. It follows from this and from the definition of $\mathcal{Q}$ that $\mathcal{Q} = \{q_1, q_2\}$ and that $0 < x_{q_1}, x_{q_2} < 1$. Thus, there are exactly two active sets S_{q_1} and S_{q_2}, these sets satisfy $|S_{q_1} \triangle S_{q_2}| = 2$, and $S_{q_1} \triangle S_{q_2}$ is exactly

the set of indices where $\mathbf{x}$ is not integral. □

4.2 LP Pseudocodewords of Cycle Codes are Half-Integral

Let $\mathcal{C}$ be a code of length n with parity-check matrix H, Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$, and fundamental polytope $\mathcal{P}$. Let $\boldsymbol{\omega} \in \mathcal{P}$ be an LP pseudocodeword, i.e., an extreme point of $\mathcal{P}$. By Theorem 3.1.8, $\boldsymbol{\omega}$ is a basic feasible solution. Thus, the span of the constraint vectors of $\mathcal{P}$ that are active at $\boldsymbol{\omega}$ has dimension n. For each node to which $\boldsymbol{\omega}$ assigns either a 0 or a 1 there is an active constraint of the form (4.1) or (4.2). This set of constraint vectors is linearly independent, so we may extend it by other active constraint vectors, which will necessarily be of the form (j, S), to obtain a set of n linearly independent constraint vectors that are active at $\boldsymbol{\omega}$. Up to a permutation of rows and columns, we may write these n constraint vectors in matrix form as

$$L_{\boldsymbol{\omega}} := \begin{bmatrix} I_{n-m} & \mathbf{0} \\ U_{\boldsymbol{\omega}} & R_{\boldsymbol{\omega}} \end{bmatrix},$$

where the last m columns represent the variable nodes in $\mathcal{F}_{\boldsymbol{\omega}} := \{i \mid 0 < \omega_i < 1\}$. Note that since the last m rows of $L_{\boldsymbol{\omega}}$ come from constraints of the form (j, S), each entry of $L_{\boldsymbol{\omega}}$ is either -1, 0, or $+1$. Because $\det(L_{\boldsymbol{\omega}}) = \det(R_{\boldsymbol{\omega}})$, the fact that $L_{\boldsymbol{\omega}}$ is invertible implies that the square submatrix $R_{\boldsymbol{\omega}}$ is also invertible.

Since each row of $L_{\boldsymbol{\omega}}$ is a constraint vector that is active at $\boldsymbol{\omega}$, we have that $L_{\boldsymbol{\omega}}\boldsymbol{\omega} = \mathbf{z}_{\boldsymbol{\omega}}$, where $\mathbf{z}_{\boldsymbol{\omega}}$ is an integer vector determined from the right-hand sides of constraints (4.1) – (4.3). If we knew that $L_{\boldsymbol{\omega}}^{-1}$ takes entries only from $\frac{1}{2}\mathbb{Z}$, we could write $\boldsymbol{\omega} = L_{\boldsymbol{\omega}}^{-1}\mathbf{z}_{\boldsymbol{\omega}}$ and conclude that $\boldsymbol{\omega} \in \{0, \frac{1}{2}, 1\}^n$. Our next goal is therefore to show

that $L_{\boldsymbol{\omega}}^{-1}$ has entries only in $\frac{1}{2}\mathbb{Z}$. Because of the block structure of $L_{\boldsymbol{\omega}}$, this amounts to showing that $R_{\boldsymbol{\omega}}^{-1}$ has entries only in $\frac{1}{2}\mathbb{Z}$. To show this algebraic fact about $R_{\boldsymbol{\omega}}$, we turn to graphical methods.

Definition 4.2.1. *Let $\mathcal{C}$ be a code with parity-check matrix H, Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$ and fundamental polytope $\mathcal{P}$. Let $\boldsymbol{\omega}$ be a nontrivial linear programming pseudocodeword, define $\mathcal{F}_{\boldsymbol{\omega}} := \{i \in \mathcal{I} \mid 0 < \omega_i < 1\}$, and set $m = |\mathcal{F}_{\boldsymbol{\omega}}|$. Let $R_{\boldsymbol{\omega}}$ be an $m \times m$ matrix formed as above, and define $\mathcal{J}_{R_{\boldsymbol{\omega}}}$ to be the set of all check nodes $j \in J$ such that a constraint of the form (j, S) is represented in the rows of $R_{\boldsymbol{\omega}}$. Define $T_{R_{\boldsymbol{\omega}}}$ to be the subgraph of T whose vertex set is $\mathcal{F}_{\boldsymbol{\omega}} \cup \mathcal{J}_{R_{\boldsymbol{\omega}}}$ with $i \in \mathcal{F}_{\boldsymbol{\omega}}$ adjacent to $j \in \mathcal{J}_{R_{\boldsymbol{\omega}}}$ if and only if one of the rows of $R_{\boldsymbol{\omega}}$ arising from j has a non-zero entry in the ith position.*

Lemma 4.2.2. *Let $\mathcal{C}$ be a code with parity-check matrix H, Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$ and fundamental polytope $\mathcal{P}$. Fix $\mathbf{x} \in \mathcal{P}$ and $j \in \mathcal{J}$. If j is incident to a variable node $i \in \mathcal{F}_{\mathbf{x}} := \{i \in \mathcal{I} \mid 0 < x_i < 1\}$, then it is incident to at least two such nodes in $\mathcal{F}_{\mathbf{x}}$.*

Proof. Assume that j is incident to one and only one variable node $i_0 \in \mathcal{F}_{\mathbf{x}}$. Then every variable node in $N(j) \setminus \{i_0\}$ is assigned a value of 0 or 1 by $\mathbf{x}$. Let $\mathcal{O}_j := \{i \in N(j) \mid x_i = 1\}$. If $|\mathcal{O}_j|$ is even, then $S := \mathcal{O}_j \cup \{i_0\}$ has odd cardinality. Notice that $\kappa_{j,\mathbf{x}}(S) > |S| + |N(j) \setminus S| - 1 = |N(j)| - 1$, which means that $\mathbf{x}$ does not satisfy constraint (j, S) of $\mathcal{P}$ given by

$$\kappa_{j,\mathbf{x}}(S) = \sum_{i \in S} x_i + \sum_{i \in N(j) \setminus S} (1 - x_i) \leq |N(j)| - 1.$$

Hence, $\mathbf{x}$ is not an element of $\mathcal{P}$.

If $|\mathcal{O}_j|$ is odd, set $S = \mathcal{O}_j$ and observe that $\kappa_{j,\mathbf{x}}(S) > |S| + |N(j) \backslash S| - 1 = |N(j)| - 1$. Again, $\mathbf{x}$ fails to satisfy constraint (j, S) of $\mathcal{P}$, so $\mathbf{x}$ is not an element of $\mathcal{P}$. □

Remark 4.2.3. *Lemma 4.2.2 implies that for any $\mathbf{x} \in \mathcal{P}$, $\mathcal{F}_{\mathbf{x}}$ is a* stopping set. *By definition, a stopping set is a set of variable nodes U such that if a check j is adjacent to some $u \in U$, then it is adjacent to at least two distinct elements of U. Stopping sets are significant in the study of iterative message-passing decoding on the binary erasure channel: the belief propagation algorithm fails to decode if and only if the set of erased bits contains a stopping set [9].*

We now restrict to the case where $\mathcal{C}$ is a cycle code.

Lemma 4.2.4. *Let $\mathcal{C}$ be a cycle code with Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$ and fundamental polytope $\mathcal{P}$. For any nontrivial linear programming pseudocodeword $\boldsymbol{\omega} \in \mathcal{P}$ we have $\mathcal{J}_{R_\omega} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$, $|\mathcal{F}_{\boldsymbol{\omega}}| = |\mathcal{J}_{R_\omega}|$, and T_{R_ω} is 2-regular.*

Proof. We first show that $\mathcal{J}_{R_\omega} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$. Let $j \in \mathcal{J}_{R_\omega}$ be given. There must be a corresponding $S \subseteq N(j)$ such that (j, S) is a row of $L_{\boldsymbol{\omega}}$. Since $R_{\boldsymbol{\omega}}$ is non-singular, this row must involve some variable nodes in $\mathcal{F}_{\boldsymbol{\omega}}$. Thus, $j \in N(\mathcal{F}_{\boldsymbol{\omega}})$, so $\mathcal{J}_{R_\omega} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$.

We now show that $|\mathcal{J}_{R_\omega}| = |\mathcal{F}_{\boldsymbol{\omega}}|$. This amounts to showing that no check is represented in the rows of $[U_{\boldsymbol{\omega}}\, R_{\boldsymbol{\omega}}]$ more than once. Clearly, a single constraint vector (j, S) associated with j cannot appear in the rows of $[U_{\boldsymbol{\omega}}\, R_{\boldsymbol{\omega}}]$ more than once since otherwise $R_{\boldsymbol{\omega}}$ would not be invertible.

Suppose now that for some $j \in \mathcal{J}_{\boldsymbol{\omega}}$ there are two distinct subsets $S_1, S_2 \subseteq N(j)$ such that the constraints (j, S_1) and (j, S_2) are both rows of $[U_{\boldsymbol{\omega}}\, R_{\boldsymbol{\omega}}]$. Then both S_1 and S_2 are active for $\boldsymbol{\omega}$ at j. In general, the vectors (j, S_1) and (j, S_2) are such that $(j, S_1) = -(j, S_2)$ when we restrict to those positions of $S_1 \triangle S_2$, and $(j, S_1) = (j, S_2) = \mathbf{0}$ on the positions of $\mathcal{I} \setminus N(j)$. By applying Lemma 4.1.3 to this check node

we see that $S_1 \triangle S_2 = \{i \in N(j) \mid 0 < \omega_i < 1\}$. This means that $(j, S_1) = -(j, S_2)$ on $S_1 \triangle S_2 = N(j) \cap \mathcal{F}_{\boldsymbol{\omega}}$ and $(j, S_1) = (j, S_2) = \mathbf{0}$ on $\mathcal{F}_{\boldsymbol{\omega}} \setminus N(j)$. It follows that the projections of (j, S_1) and (j, S_2) onto the positions of $\mathcal{F}_{\boldsymbol{\omega}}$ are scalar multiples of one another, which contradicts in the fact that $R_{\boldsymbol{\omega}}$ is invertible. We conclude that $|\mathcal{J}_{R_{\boldsymbol{\omega}}}| = |\mathcal{F}_{\boldsymbol{\omega}}|$.

To prove the third and final claim of the lemma, we bound the number e of edges in $T_{R_{\boldsymbol{\omega}}}$ in two different ways. Since $\mathcal{C}$ is a cycle code, each variable node has degree exactly 2 in T, so every variable node in $T_{R_{\boldsymbol{\omega}}}$ has degree at most 2. Thus, $e \leq 2|\mathcal{F}_{\boldsymbol{\omega}}|$. On the other hand, since $\mathcal{J}_{R_{\boldsymbol{\omega}}} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$ Lemma 4.2.2 yields $e \geq 2|\mathcal{J}_{R_{\boldsymbol{\omega}}}|$. Since $|\mathcal{F}_{\boldsymbol{\omega}}| = |\mathcal{J}_{R_{\boldsymbol{\omega}}}|$, the result follows. □

Proposition 4.2.5. *Let $\mathcal{C}$ be a cycle code with fundamental polytope $\mathcal{P}$. For any nontrivial linear programming pseudocodeword $\boldsymbol{\omega} \in \mathcal{P}$ and corresponding matrix $R_{\boldsymbol{\omega}}$, the matrix $R_{\boldsymbol{\omega}}^{-1}$ has entries only in $\{-\frac{1}{2}, 0, +\frac{1}{2}\}$.*

Proof. By Lemma 4.2.4, $T_{R_{\boldsymbol{\omega}}}$ is a 2-regular bipartite graph. Thus, each connected component is a cycle of even length. By Definition 4.2.1, the non-zero entries of $R_{\boldsymbol{\omega}}$ give the incidence structure of $T_{R_{\boldsymbol{\omega}}}$. More formally, the matrix $|R_{\boldsymbol{\omega}}|$ obtained by taking the coordinate-wise absolute value of each entry in $R_{\boldsymbol{\omega}}$ is the incidence matrix for $T_{R_{\boldsymbol{\omega}}}$. We may therefore assume (by permuting columns and rows) that $R_{\boldsymbol{\omega}}$ is a block diagonal matrix

$$R_{\boldsymbol{\omega}} = \begin{bmatrix} D_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & D_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & D_b \end{bmatrix}$$

where each block is square and has the form

$$D = \begin{bmatrix} d_{1,1} & d_{1,2} & 0 & \cdots & 0 \\ 0 & d_{2,2} & d_{2,3} & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & d_{\ell-1,\ell-1} & d_{\ell-1,\ell} \\ d_{\ell,1} & 0 & \ldots & 0 & d_{\ell,\ell} \end{bmatrix},$$

with $d_{p,q} \in \{-1, +1\}$ for all p, q. To show that $R_{\boldsymbol{\omega}}^{-1}$ has entries only in $\{-\frac{1}{2}, 0, +\frac{1}{2}\}$, it suffices to show that each block D of $R_{\boldsymbol{\omega}}$ is such that D^{-1} takes entries only in $\{-\frac{1}{2}, 0, +\frac{1}{2}\}$.

Let D be a block of $R_{\boldsymbol{\omega}}$. Since $R_{\boldsymbol{\omega}}$ is invertible, D is also invertible. This implies the existence of a unique solution $\mathbf{a}$ to the equation $D\mathbf{a} = \boldsymbol{\epsilon}_p$, where $\boldsymbol{\epsilon}_p$ is the pth standard basis vector. The equation $D\mathbf{a} = \boldsymbol{\epsilon}_p$ gives rise to a set of relations that must hold between the entries of $\mathbf{a}$. In the following we take subscripts modulo ℓ to respect the cyclic nature of D. For all $q \neq p$, we have $a_q d_{q,q} + a_{q+1} d_{q,q+1} = 0$. Since $d_{q,q+1}, d_{q,q} \in \{-1, +1\}$ we may rearrange to get $a_q = -\frac{d_{q,q+1}}{d_{q,q}} a_{q+1}$, and so $a_q = \pm a_{q+1}$ for all $q \neq p$. This in turn implies that all of the entries of $\mathbf{a}$ are the same up to sign.

The equation $D\mathbf{a} = \boldsymbol{\epsilon}_p$ also implies that $a_p d_{p,p} + a_{p+1} d_{p,p+1} = 1$. We know from the previous paragraph that a_{p+1} is either a_p or $-a_p$. Thus, we have to consider two possible equations: $a_p(d_{p,p} + d_{p,p+1}) = 1$ or $a_p(d_{p,p} - d_{p,p+1}) = 1$. In either case, since $d_{p,p+1}, d_{p,p} \in \{-1, +1\}$ we have that a_p is either $-\frac{1}{2}$ or $+\frac{1}{2}$. Combining this with the previous paragraph, we see that all the entries of $\mathbf{a}$ are in $\{-\frac{1}{2}, +\frac{1}{2}\}$.

Renaming the unique solution to the equation $D\mathbf{a} = \boldsymbol{\epsilon}_p$ to be $\mathbf{a}_p$, we see that

$$D^{-1} = \begin{bmatrix} | & | & \dots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_\ell \\ | & | & \dots & | \end{bmatrix}$$

It follows that D^{-1} takes entries only in $\{-\frac{1}{2}, \frac{1}{2}\}$. □

The next corollary follows immediately from Proposition 4.2.5 and the discussion at the beginning of this section.

Corollary 4.2.6. *Let $\mathcal{C}$ be a cycle code with fundamental polytope $\mathcal{P}$. If $\boldsymbol{\omega}$ is an extreme point of $\mathcal{P}$, then $L_{\boldsymbol{\omega}}^{-1}$ has entries only in $\frac{1}{2}\mathbb{Z}$.*

We now have the tools to prove the following theorem.

Theorem 4.2.7. *Let $\mathcal{C}$ be a cycle code with Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$ and fundamental polytope $\mathcal{P}$. If $\boldsymbol{\omega}$ is a nontrivial linear programming pseudocodeword of $\mathcal{P}$, then $\boldsymbol{\omega} \in \{0, \frac{1}{2}, 1\}^n$ and the subgraph $T_{\mathcal{F}_{\boldsymbol{\omega}}}$ of T induced by the set $\mathcal{F}_{\boldsymbol{\omega}} = \{i \mid \omega_i = \frac{1}{2}\}$ and its neighborhood $N(\mathcal{F}_{\boldsymbol{\omega}})$ is precisely $T_{R_{\boldsymbol{\omega}}}$, which is 2-regular.*

Proof. By Corollary 4.2.6, $L_{\boldsymbol{\omega}}^{-1}$ has entries in $\frac{1}{2}\mathbb{Z}$. By the discussion at the beginning of this section, we have that $L_{\boldsymbol{\omega}}\boldsymbol{\omega} = \mathbf{z}_{\boldsymbol{\omega}}$, where $\mathbf{z}_{\boldsymbol{\omega}}$ is an integer vector. Thus, $\boldsymbol{\omega} = L^{-1}\mathbf{z}$ must have entries that come only from $\frac{1}{2}\mathbb{Z}$. Since any point in the fundamental polytope can only assume values between 0 and 1, $\boldsymbol{\omega} \in \{0, \frac{1}{2}, 1\}^n$.

Lemma 4.2.4 implies that $\mathcal{J}_{R_{\boldsymbol{\omega}}} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$, $|\mathcal{F}_{\boldsymbol{\omega}}| = |\mathcal{J}_{R_{\boldsymbol{\omega}}}|$, and $T_{R_{\boldsymbol{\omega}}}$ is 2-regular. From $\mathcal{J}_{R_{\boldsymbol{\omega}}} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$ and Definition 4.2.1 we see that $T_{R_{\boldsymbol{\omega}}}$ is a subgraph of $T_{\mathcal{F}_{\boldsymbol{\omega}}}$. Letting e denote the number of edges in $T_{\mathcal{F}_{\boldsymbol{\omega}}}$, we have that $e = 2|\mathcal{F}_{\boldsymbol{\omega}}|$ since $\mathcal{C}$ is a cycle code.

Lemma 4.2.2 implies that $2|N(\mathcal{F}_{\boldsymbol{\omega}})| \leq e$. Using the fact that $\mathcal{J}_{R_{\boldsymbol{\omega}}} \subseteq N(\mathcal{F}_{\boldsymbol{\omega}})$, we have

$$2|\mathcal{J}_{R_{\boldsymbol{\omega}}}| \leq 2|N(\mathcal{F}_{\boldsymbol{\omega}})| \leq e = 2|\mathcal{F}_{\boldsymbol{\omega}}|.$$

Since $|\mathcal{F}_{\boldsymbol{\omega}}| = |\mathcal{J}_{R_{\boldsymbol{\omega}}}|$, these inequalities must be tight. Thus $T_{\mathcal{F}_{\boldsymbol{\omega}}}$ has the same number of edges as $T_{R_{\boldsymbol{\omega}}}$ and contains $T_{R_{\boldsymbol{\omega}}}$ as a subgraph. We conclude that $T_{\mathcal{F}_{\boldsymbol{\omega}}} = T_{R_{\boldsymbol{\omega}}}$. □

Theorem 4.2.7 gives most of the forward direction of Theorem 4.0.2. In Section 4.3, we derive a parity-based condition that allows us to complete this forward direction as well as to prove the reverse implication.

4.3 A Set of Sufficient Suppositions: The Reverse Implication

Throughout this section we continue with the assumption that $\mathcal{C}$ is a cycle code with parity-check matrix H, Tanner graph $T = (\mathcal{I} \cup \mathcal{J}, E)$, normal graph N, and fundamental polytope $\mathcal{P}$. We also identify variable nodes of the Tanner graph with edges in the normal graph. In this way, we use x_e to denote the value $\mathbf{x} \in \mathcal{P}$ assigns to edge $e \in N$.

It is important to note that in shifting from Tanner graphs to normal graphs, we must appropriately alter the manner in which we present the constraints for the fundamental polytope. For a generic graph G with vertex v, we use $E(v)$ to denote the set of edges incident to v. Thus $N(j)$, the neighborhood of check node j in the Tanner graph, is identified with $E(j)$ in the normal graph. For a cycle code $\mathcal{C}$ with normal graph N, we can therefore say that the fundamental polytope is the set of all real vectors $\mathbf{x}$ satisfying

(a) $0 \leq x_e \leq 1$ for all edges e, and

(b) $\sum_{e \in S} x_e + \sum_{e \in E(j) \setminus S} (1 - x_e) \leq |E(j)| - 1$ for all checks j in N, and for all odd-sized subsets S of $E(j)$.

We also appropriately adapt the notion of the cost $\kappa_{j,\mathbf{x}}(S)$, where $S \subseteq E(j)$, to mean

$$\kappa_{j,\mathbf{x}}(S) = \sum_{e \in S} x_e + \sum_{e \in E(j) \setminus S} (1 - x_e).$$

We now finish the proof of Theorem 4.0.2. To do so, we first prove two supporting lemmas.

Lemma 4.3.1. *Let $\mathcal{C}$ be a cycle code with fundamental polytope $\mathcal{P}$, and let j be a check node in the normal graph N. Suppose that a point $\mathbf{x} \in \mathcal{P}$ assigns 0's and 1's to all but possibly two edges incident to j, say e_1 and e_2. Let $\mathcal{O}_j^*$ denote the set of edges in $E(j) \setminus \{e_1, e_2\}$ that are assigned a value of 1 by $\mathbf{x}$.*

(a) If $|\mathcal{O}_j^*|$ is odd, then $x_{e_1} + x_{e_2} = 1$.

(b) If $|\mathcal{O}_j^*|$ is even, then $x_{e_1} = x_{e_2}$.

Proof. In the following, $\mathcal{Z}_j^*$ is the set of edges in $E(j) \setminus \{e_1, e_2\}$ that are assigned a value of 0 by $\mathbf{x}$. We first prove part (a). Assume that $|\mathcal{O}_j^*|$ is odd, and set $S = \mathcal{O}_j^*$. Since $\mathbf{x} \in \mathcal{P}$, we know that $\kappa_j(S)$ is at most $|E(j)| - 1$:

$$\sum_{i \in \mathcal{O}_j^*} 1 + \sum_{i \in \mathcal{Z}_j^*} (1 - 0) + (1 - x_{e_1}) + (1 - x_{e_2}) \leq |E(j)| - 1.$$

Since $|\mathcal{O}_j^* \cup \mathcal{Z}_j^*| = |E(j)| - 2$, we have

$$1 \leq x_{e_1} + x_{e_2}.$$

For the other inequality, set $S = \mathcal{O}_j^* \cup \{e_1, e_2\}$. Using this set S, we have the following constraint:

$$x_{e_1} + x_{e_2} + \sum_{i \in \mathcal{O}_j^*} 1 + \sum_{i \in \mathcal{Z}_j^*} (1 - 0) \leq |E(j)| - 1.$$

This can be rearranged to obtain

$$x_{e_1} + x_{e_2} \leq 1.$$

Thus, $x_{e_1} + x_{e_2} = 1$, and part (a) is proved.

For part (b), assume that $|\mathcal{O}_j^*|$ is even, and set $S = \mathcal{O}_j^* \cup \{e_1\}$. Since $|S|$ is odd, we have the constraint

$$x_{e_1} + \sum_{i \in \mathcal{O}_j^*} 1 + \sum_{i \in \mathcal{Z}_j^*} (1 - 0) + (1 - x_{e_2}) \leq |E(j)| - 1.$$

This can be rearranged to get $x_{e_1} - x_{e_2} \leq 0$, or, more simply, $x_{e_1} \leq x_{e_2}$. By setting $S = \mathcal{O}_j^* \cup \{e_2\}$ and performing a symmetric argument, one finds that $x_{e_2} \leq x_{e_1}$, which completes the proof. □

Lemma 4.3.2. *Let $\mathcal{C}$ be a cycle code with normal graph N and fundamental polytope $\mathcal{P}$. Fix $\mathbf{x} \in \{0, \frac{1}{2}, 1\}^n$ and let Γ be the subgraph of N spanned by the edges of N that are assigned values of $\frac{1}{2}$ by $\mathbf{x}$. If Γ is 2-regular, then the following are equivalent:*

(a) $\mathbf{x} \in \mathcal{P}$.

(b) If j is a check node of N that is incident to an odd number of edges assigned a value of 1 by $\mathbf{x}$, then j must also be incident to exactly two edges assigned values of $\frac{1}{2}$.

Proof. Throughout the proof, we use $\mathcal{O}_j$ and $\mathcal{H}_j$ to denote the sets of edges incident to the check j that are assigned values of 1 and $\frac{1}{2}$ by $\mathbf{x}$, respectively.

Since Γ is 2-regular, this means that $|\mathcal{H}_j| = 0$ or 2 for all checks j. If there exists some check j with $|\mathcal{O}_j|$ odd and $|\mathcal{H}_j| \neq 2$, we must therefore have $\mathcal{H}_j = \emptyset$. Letting $S = \mathcal{O}_j$, we see that $\mathbf{x}$ violates the (j, S) constraint of the fundamental polytope, and so $\mathbf{x} \notin \mathcal{P}$. We have shown that (a) implies (b).

To show that (b) implies (a), we make use of Theorem 3.3.3, which establishes the equivalence of Feldman's extended polytope and the fundamental polytope. This equivalence implies that a point $\mathbf{y}$ is in the fundamental polytope if and only if for every check j, the local configuration of $\mathbf{y}$ at j is a convex sum of local codewords. In other words, the projection $\mathbf{y}^{E(j)}$ of $\mathbf{y}$ onto the edges incident to j must be a convex sum of even-weight zero-one vectors. Thus, to show $\mathbf{x} \in \mathcal{P}$ it is sufficient to write $\mathbf{x}^{E(j)}$ as a convex combination of local codewords at j.

Suppose that (b) holds, i.e., for all checks j with $|\mathcal{O}_j|$ odd, we also have $|\mathcal{H}_j| = 2$. Let j be an arbitrary check node of N. If $|\mathcal{O}_j|$ is even, let $\mathbf{a} = \text{ind}(\mathcal{O}_j \cup \mathcal{H}_j)$ and $\mathbf{b} = \text{ind}(\mathcal{O}_j)$, where $\text{ind}(A)$ is used to denote the indicator vector of the set A. Both $\mathbf{a}$ and $\mathbf{b}$ are local codewords at j, and we have $\frac{1}{2}\mathbf{a} + \frac{1}{2}\mathbf{b} = \mathbf{x}^{E(j)}$. If, on the other hand, $|\mathcal{O}_j|$ is odd, we can write $\mathcal{H}_j = \{e_1, e_2\}$. Set $\mathbf{a} = \text{ind}\,(\mathcal{O}_j \cup \{e_1\})$ and $\mathbf{b} = \text{ind}\,(\mathcal{O}_j \cup \{e_2\})$. As defined, both $\mathbf{a}$ and $\mathbf{b}$ are local codewords at j, and $\frac{1}{2}\mathbf{a} + \frac{1}{2}\mathbf{b} = \mathbf{x}^{E(j)}$. Since for all checks j of N we have shown that $\mathbf{x}^{E(j)}$ is a sum of local codewords at j, we conclude that $\mathbf{x}$ is an element of the fundamental polytope. □

Theorem 4.3.3. *Let $\mathcal{C}$ be a cycle code with normal graph N and fundamental polytope $\mathcal{P}$. Let $\boldsymbol{\omega} \in \mathcal{P} \cap \{0, \frac{1}{2}, 1\}^n$ be given, and let Γ be the subgraph of N spanned by the edges of N that are assigned values of $\frac{1}{2}$ by $\boldsymbol{\omega}$. Assume that Γ is 2-regular; equivalently, that Γ is a vertex-disjoint union of simple cycles $\gamma_1, \gamma_2, \ldots, \gamma_\ell$. Then $\boldsymbol{\omega}$ is an extreme*

point of $\mathcal{P}$ if and only if for every simple cycle γ in Γ, the number of edges in N that are incident to γ and are assigned a value of 1 by $\boldsymbol{\omega}$ is odd.

Proof. We begin by assuming that every simple cycle γ in Γ is incident to an odd number of edges assigned a value of 1. Suppose that there are $\mathbf{x}, \mathbf{y} \in \mathcal{P}$ and a $\lambda \in (0,1)$ such that $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} = \boldsymbol{\omega}$. For this to occur, it must be that $\mathbf{x}$ and $\mathbf{y}$ both assign 0's to the edges that $\boldsymbol{\omega}$ does, and that $\mathbf{x}$ and $\mathbf{y}$ both assign 1's to the edges that $\boldsymbol{\omega}$ does. Thus, the only edges where $\mathbf{x}$ and $\mathbf{y}$ can possibly disagree with $\boldsymbol{\omega}$ are those edges in Γ.

Let $\gamma \in \Gamma$ be a simple cycle, and let C_γ be the set of check nodes in γ that are incident with an odd number of edges to which $\boldsymbol{\omega}$ assigns a 1. By assumption, $|C_\gamma|$ must be odd. Let $j_1, j_2, \ldots, j_{2m+1}$ be an enumeration of C_γ. Without loss of generality, assume that these check nodes are indexed in such a way that respects the cyclic structure of γ. Partition the edges of γ into groups E_ℓ such that E_ℓ is the set of all edges between j_ℓ and $j_{\ell+1}$, where subscripts are taken modulo $2m+1$. We say that two such subsets E_k and E_ℓ are *adjacent* if $k \equiv \ell \pm 1 \pmod{2m+1}$. Lemma 4.3.1 implies that $\mathbf{x}$ is constant on each E_ℓ, and also that the common values assigned to any two adjacent sets must add to 1. Since there are an odd number of these sets and their adjacency structure is cyclic, we must have that each edge in the cycle is assigned a value of $\frac{1}{2}$ by $\mathbf{x}$. See Figure 4.1 for an illustration.

Thus, $\mathbf{x} = \boldsymbol{\omega}$. By a symmetric argument, $\mathbf{y} = \boldsymbol{\omega}$. Since $\boldsymbol{\omega}$ cannot be written as a convex sum of elements of $\mathcal{P} \setminus \{\boldsymbol{\omega}\}$, we conclude that $\boldsymbol{\omega}$ is an extreme point of $\mathcal{P}$.

For the converse, suppose that there is a simple cycle $\gamma \in \Gamma$ that is incident to an even number of edges receiving 1 from $\boldsymbol{\omega}$. Let C_γ be the set of check nodes in γ that are incident with an odd number of edges to which $\boldsymbol{\omega}$ assigns a 1, and let $j_1, j_2, \ldots, j_{2m}$ be an enumeration of the checks in C_γ that respects the cyclic structure of γ. With

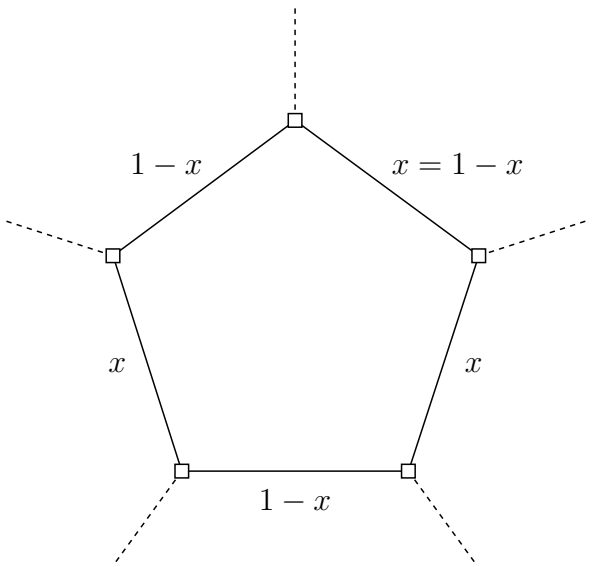


Figure 4.1: A case where $|C_\gamma|$ is odd.

the sets E_ℓ defined as before, set $A = \bigcup_{\ell \text{ odd}} E_\ell$ and $B = \bigcup_{\ell \text{ even}} E_\ell$. Define a vector $\mathbf{x} \in \mathbb{R}^n$ by making it equal to 1 on all edges of A, 0 on all edges of B, and otherwise identical to $\boldsymbol{\omega}$. Define $\mathbf{y}$ to be 0 on A, 1 on B, and otherwise equal to $\boldsymbol{\omega}$. It is easy to see that $\frac{1}{2}\mathbf{x} + \frac{1}{2}\mathbf{y} = \boldsymbol{\omega}$: Figure 4.2 provides an illustration.

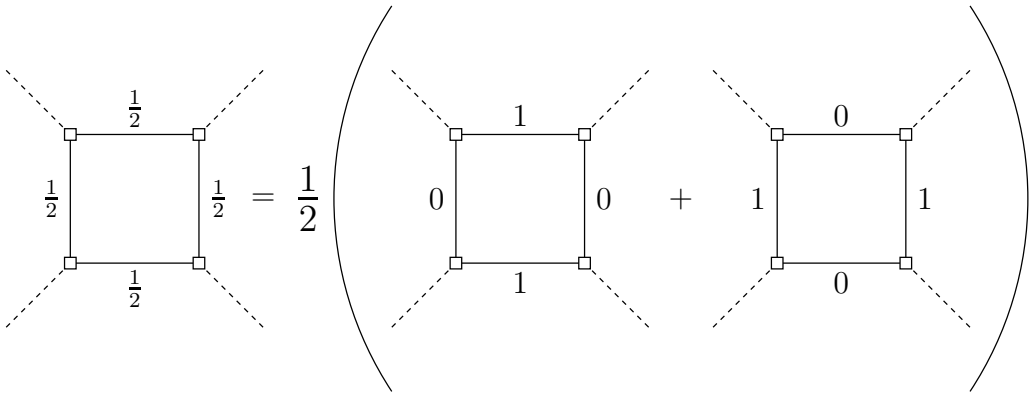


Figure 4.2: A case where $|C_\gamma|$ is even.

Lemma 4.3.2 implies that $\mathbf{x}, \mathbf{y} \in \mathcal{P}$, and so $\boldsymbol{\omega}$ is not an extreme point of $\mathcal{P}$. $\square$

Combining Theorem 4.2.7 and Theorem 4.3.3, we obtain Theorem 4.0.2.

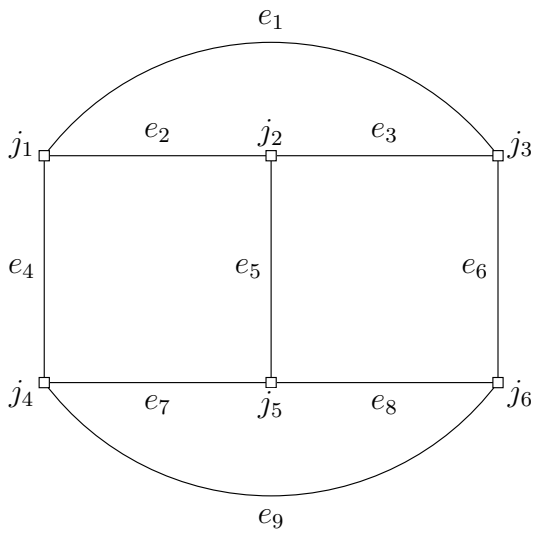
4.4 Construction of LP Pseudocodewords

Given a cycle code $\mathcal{C}$ with normal graph N, Theorem 4.0.2 provides an immediate litmus test to check whether a vector is a linear programming pseudocodeword. This test, while highly graphical, is not a major result. Indeed, for general polytopes one can test a candidate point to see if it is an extreme point by computing the rank of the matrix of active constraint vectors (see Theorem 3.1.8). The true worth of Theorem 4.0.2 is that it completely describes all LP pseudocodewords in a clear-cut manner, and in doing so suggests a method of constructing them.

To write down a pseudocodeword, begin with a subgraph $\Gamma = \big(V(\Gamma), E(\Gamma)\big)$ of $N = (V, E)$ that is a vertex-disjoint union of simple cycles in N. Next, select a set P of edge-disjoint paths[1] from $N \setminus \Gamma := \big(V, E \setminus E(\Gamma)\big)$, each of which starts and ends at some simple cycle of Γ. Ensure that each simple cycle of Γ is incident to an odd number of edges in P. Note that it is permissible for a path in P to start and end at the *same* simple cycle of Γ, as well as for a path in P to be incident with more than two simple cycles of Γ. Assign values of 1 to the edges of P and values of $\frac{1}{2}$ to the edges in Γ. To all other edges, assign values of 0. Letting $\boldsymbol{\omega}$ be the vector built up in this way, Lemma 4.3.2 guarantees that $\boldsymbol{\omega} \in P$, and Theorem 4.0.2 implies that $\boldsymbol{\omega}$ is an LP pseudocodeword.

[1] We use the convention that a *path* is a walk that does not repeat any vertex.

As an example, let $\mathcal{C}$ be the cycle code presented by the following normal graph.



The only possible nontrivial vertex-disjoint union of simple cycles is $\Gamma = \{e_1e_2e_3, e_7e_8e_9\}$. For a set of paths P satisfying the desired conditions, we have four choices: $P = \{e_4\}$, $P = \{e_5\}$, $P = \{e_6\}$, or $P = \{e_4, e_5, e_6\}$. The nontrivial LP pseudocodewords we obtain in this way are

$$\left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 1, 0, 0, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right)$$
$$\left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 0, 1, 0, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right)$$
$$\left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 0, 0, 1, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right)$$
$$\left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 1, 1, 1, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right).$$

This list of four vectors is a complete list of all the fractional extreme points of the fundamental polytope for this code.

The method described thus far can only produce nontrivial pseudocodewords, along with the all-zeros vector (if the number of cycles in Γ is chosen to be zero). The properness of $\mathcal{P}$, however, implies that each of the 16 codewords is also an extreme point of the fundamental polytope. This is not a contradiction to Theorem 4.0.2; indeed, every codeword satisfies the conditions of Theorem 4.0.2 since the empty graph

is trivially 2-regular. This shortcoming in our construction method is patched by allowing for $\mathcal{C}$-symmetry: Proposition 4.4.2 below shows that any LP pseudocodeword is either a codeword or lies in the $\mathcal{C}$-symmetry orbit of a pseudocodeword that is obtainable by the cycle-path method outlined earlier in this section. The proof of Proposition 4.4.2 requires the following lemma.

Lemma 4.4.1. *Let G be an acyclic graph. The edge set E of G can be partitioned as $\bigcup_{\ell=1}^{d} P_\ell$, where each P_ℓ is the edge set of a path in G whose endpoints have odd degree.*

Proof. Let V and E denote the vertex set and the edge set of G, respectively. We proceed by induction on $|E|$. If $|E| = 1$, the result is trivial. So suppose that $|E| > 1$.

Since G is acyclic, each connected component of G is a tree, and hence each connected component has at least two leaf nodes. Let $v_1, v_2 \in V$ be distinct vertices of degree 1 in the same connected component of G. Let ρ_0 be the unique path connecting v_1 and v_2 in G, and let P_0 be the edge set of ρ_0.

Let G' be the graph with vertex set $V \setminus \{v_1, v_2\}$ and edge set $E \setminus P_0$. By induction, there is a partition $\bigcup_{\ell=1}^{d} P_\ell$ of $E \setminus P_0$ such that each P_ℓ is the edge set of a path ρ_ℓ in G' with the property that ρ_ℓ begins and ends at vertices with odd degree in G'. We have that $P_0 \cup P_1 \cup \cdots \cup P_d$ is therefore a partition of E. For every vertex v in G', the degree of v in G' either has the same degree as v did in G or has degree two less than it did in G. Thus, the parity of the degree of v in G' is the same as the parity of the degree of v in G. It follows that every path $\rho_1, \rho_2, \ldots, \rho_d$ begins and ends at a vertex of odd degree in G, and the result is proved. □

Proposition 4.4.2. *Let $\mathcal{C}$ be a cycle code with normal graph N and fundamental polytope $\mathcal{P}$. For any linear programming pseudocodeword $\boldsymbol{\omega} \in \mathcal{P}$, we have that $\boldsymbol{\omega}$ lies in the $\mathcal{C}$-symmetry orbit of either $\mathbf{0}$ (the all-zeros codeword) or a nontrivial linear*

programming pseudocodeword $\boldsymbol{\omega}'$ that is constructible by the cycle-path method outlined at the beginning of this section.

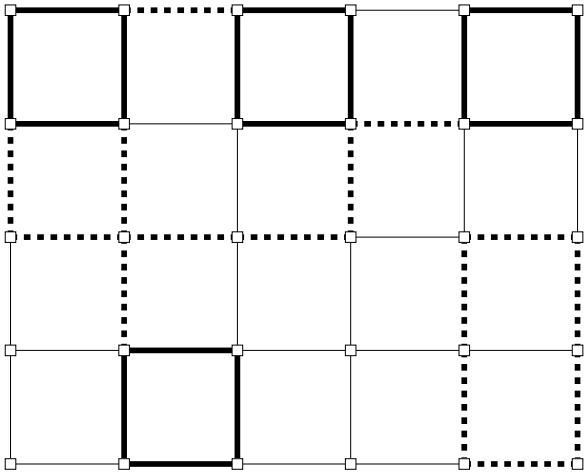
Proof. By Theorem 4.0.2, $\boldsymbol{\omega} \in \{0, \frac{1}{2}, 1\}^n$. Let Γ be the subgraph of N spanned by those edges assigned a value of $\frac{1}{2}$ by $\boldsymbol{\omega}$. If Γ contains no edges, $\boldsymbol{\omega}$ is an integer vector and the properness of $\mathcal{P}$ implies that $\boldsymbol{\omega} \in \mathcal{C}$. We therefore have $\mathbf{0}^{[\omega]} = \boldsymbol{\omega}$, i.e., $\boldsymbol{\omega}$ lies in the $\mathcal{C}$-symmetry orbit of $\mathbf{0}$, and we are done.

Now suppose that Γ has at least one edge. By Theorem 4.0.2, Γ must be a vertex-disjoint union of simple cycles $\gamma_1, \gamma_2, \ldots, \gamma_\ell$. For any vector $\mathbf{x}$ of length n, let $N_{\mathbf{x}=1}$ be the subgraph of N spanned by the edges of N that receive a value of 1 from $\mathbf{x}$. Let A be the set of codewords in $\mathcal{C}$ whose support is contained in the edge set of $N_{\boldsymbol{\omega}=1}$. Choose $\mathbf{c} \in A$ to have maximal support, i.e., such that for all $\mathbf{c}' \in A \setminus \{\mathbf{c}\}$ we have that $\mathrm{supp}(\mathbf{c}) \not\subseteq \mathrm{supp}(\mathbf{c}')$. By Theorems 3.4.3 and 3.4.4, we have that $\boldsymbol{\omega}' := \boldsymbol{\omega}^{[\mathbf{c}]}$ is also an LP pseudocodeword. Moreover, the maximality of $\mathbf{c}$ implies that $N_{\boldsymbol{\omega}'=1}$ is acyclic.

We claim that $\boldsymbol{\omega}'$ is constructible by the cycle-path method outlined at the beginning of this section. Apply Lemma 4.4.1 to $N_{\boldsymbol{\omega}'=1}$ to obtain a set of paths $\rho_1, \rho_2, \ldots, \rho_d$ whose edges partition the edge set of $N_{\boldsymbol{\omega}'=1}$ and with the property that each ρ_ℓ begins and ends at a node of $N_{\boldsymbol{\omega}'=1}$ that has odd degree in $N_{\boldsymbol{\omega}'=1}$. By Lemma 4.3.2, every vertex of $N_{\boldsymbol{\omega}'=1}$ with odd degree in $N_{\boldsymbol{\omega}'=1}$ must be on some cycle γ_i of Γ. It follows that $\boldsymbol{\omega}'$ is constructible by the cycle-path method using Γ for the cycles and $\rho_1, \rho_2, \ldots, \rho_d$ for the paths. Since $\boldsymbol{\omega} = \left(\boldsymbol{\omega}^{[\mathbf{c}]}\right)^{[\mathbf{c}]} = (\boldsymbol{\omega}')^{[\mathbf{c}]}$, we are done. □

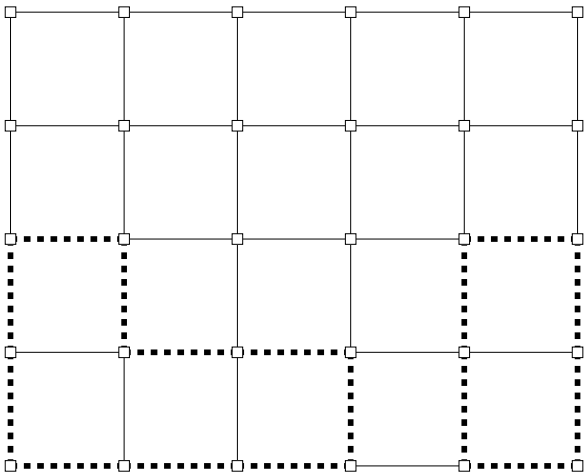
Example 4.4.3. *Let $\boldsymbol{\omega}$ be the configuration indicated on the following normal graph: dashed edges receive a value of 1, bold edges receive a value of $\frac{1}{2}$, and all other edges*

receive a value of 0.

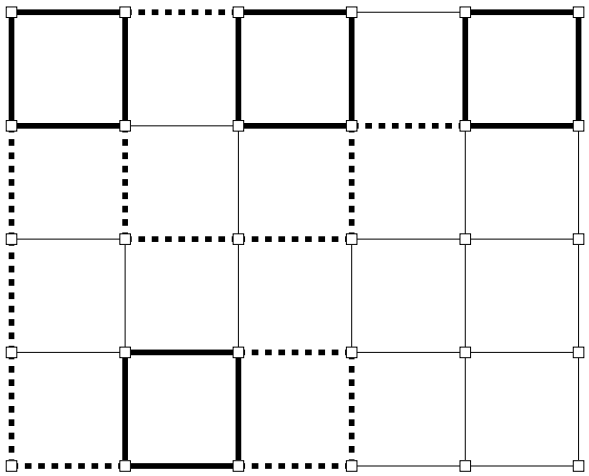


Lemma 4.3.2 guarantees that $\boldsymbol{\omega} \in P$*, and Theorem 4.0.2 implies that* $\boldsymbol{\omega}$ *is an LP pseudocodeword. The dashed cycle on the lower right prevents this pseudocodeword from being constructed in the manner discussed at the beginning of this section.*

Let $\mathbf{c}$ *be the codeword defined graphically by the following picture.*



With $\mathbf{c}$ *as given, we have that* $\boldsymbol{\omega}^{[\mathbf{c}]}$ *is the configuration described by the following.*



This latter pseudocodeword $\boldsymbol{\omega}^{[\mathbf{c}]}$ *can be obtained via the cycle-path construction at the beginning of this section. Finally, we observe that* $\boldsymbol{\omega} = \left(\boldsymbol{\omega}^{[\mathbf{c}]}\right)^{[\mathbf{c}]}$*, and so* $\boldsymbol{\omega}$ *is in the* $\mathcal{C}$*-symmetry orbit of* $\boldsymbol{\omega}^{[\mathbf{c}]}$*.*

4.5 A Note on Minimal LP Pseudocodewords

The set of linear programming pseudocodewords constitutes the entire set of output vectors that the linear programming decoder can return. In [25], it is shown that the performance of LP decoding is governed primarily by the set of *minimal pseudocodewords*, much as the performance of a classical ML decoder is determined largely by the codewords of minimum weight.

Definition 4.5.1 ([25])**.** *Let* $\mathcal{C}$ *be a code with parity-check matrix* H *and fundamental polytope* $\mathcal{P}$*. Define the* fundamental cone $\mathcal{K}(H) = \mathcal{K}$ *to be the conic hull of* $\mathcal{P}$*; i.e.,*

$$\mathcal{K} := \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} = \alpha\mathbf{y} \text{ for some } \alpha \in [0, \infty) \text{ and } \mathbf{y} \in \mathcal{P}\}.$$

An extreme point $\boldsymbol{\omega} \in \mathcal{P}$ is said to be a minimal linear programming pseudocodeword*, or more simply a* minimal pseudocodeword*, if it lies on an edge of $\mathcal{K}$. A minimal pseudocodeword that is also a codeword is a* trivial *minimal pseudocodeword, and a minimal pseudocodeword that is not a codeword is a* nontrivial *minimal pseudocodeword.*

Remark 4.5.2. *Since scaling in the fundamental cone is in some sense immaterial, other authors (e.g., Vontobel and Koetter [25]) define minimal pseudocodewords as being any point on an edge of the fundamental cone.*

With this definition, it is intuitively plausible that the minimal pseudocodewords play a major role in determining the performance of the LP decoder: suppose that $\mathbf{0}$ is the transmitted codeword. If the codeword is received perfectly, i.e., without any noise, the cost function $\boldsymbol{\lambda}$ of the LP decoder should "point at" $\mathbf{0}$. As noise can be interpreted as a perturbation of this cost function [13], the most likely error to be made by the LP decoder is for it to decide on one of $\mathbf{0}$'s neighboring extreme points in $\mathcal{P}$, i.e., a minimal pseudocodeword.

In her PhD thesis, Dreher [12] provides an elegant graphical characterization of minimal pseudocodewords for cycle codes.

Theorem 4.5.3 ([12], see Theorem 5.2.21)**.** *Let $\mathcal{C}$ be a cycle code of length n with normal graph N and fundamental polytope $\mathcal{P}$. A vector $\boldsymbol{\omega} \in \mathbb{R}^n$ is a trivial minimal linear programming pseudocodeword if and only if the following three conditions hold:*

(a) $\boldsymbol{\omega} \in \mathcal{P}$.

(b) $\boldsymbol{\omega} \in \{0, 1\}^n$.

(c) The subgraph of N spanned by the edges of N that are assigned a value of 1 *by $\boldsymbol{\omega}$ consists of a single simple cycle.*

A vector $\boldsymbol{\omega} \in \mathbb{R}^n$ *is a nontrivial minimal linear programming pseudocodeword if and only if the following four conditions hold:*

(a) $\boldsymbol{\omega} \in \mathcal{P}$.

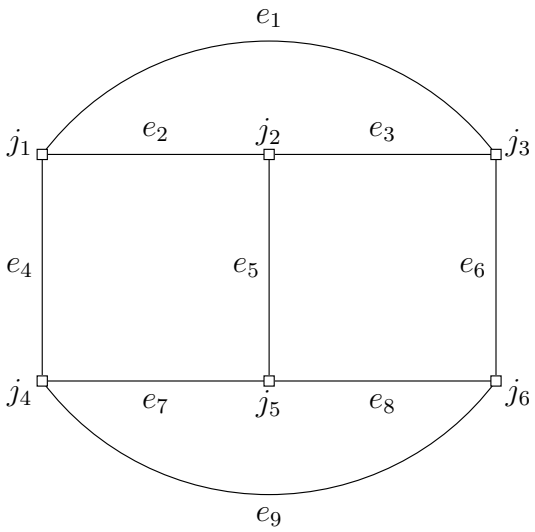
(b) $\boldsymbol{\omega} \in \{0, \frac{1}{2}, 1\}^n$.

(c) The subgraph Γ *of* N *spanned by the edges of* N *that are assigned a value of* $\frac{1}{2}$ *by* $\boldsymbol{\omega}$ *is a vertex-disjoint union of two simple cycles* γ_1 *and* γ_2.

(d) The subgraph P *of* N *spanned by the edges of* N *that are assigned a value of* 1 *by* $\boldsymbol{\omega}$ *is a path that begins at* γ_1 *and ends at* γ_2 *but is otherwise internally disjoint from* γ_1 *and* γ_1.

The similarity between Theorem 4.5.3 and Theorem 4.0.2 is apparent, though there are important differences. Theorem 4.0.2 classifies all pseudcodewords for cycle codes, yet it does not distinguish between minimal and non-minimal pseudocodewords. By comparison, the set of minimal pseudocodewords is usually a proper subset of all the pseudocodewords, as is illustrated in the following example.

Example 4.5.4. *Let* $\mathcal{C}$ *be the cycle code presented by the following normal graph.*



In Section 4.4 the nontrivial linear programming pseudocodewords were found to be

$$\begin{aligned}
\boldsymbol{\omega}_1 &= \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 1, 0, 0, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right) \\
\boldsymbol{\omega}_2 &= \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 0, 1, 0, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right) \\
\boldsymbol{\omega}_3 &= \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 0, 0, 1, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right) \\
\boldsymbol{\omega}_4 &= \left(\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, 1, 1, 1, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}\right).
\end{aligned}$$

The pseudocodewords $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2$, and $\boldsymbol{\omega}_3$ are minimal pseudocodewords by Theorem 4.5.3. The vector $\boldsymbol{\omega}_4$ is not, since it violates condition (c) of Theorem 4.5.3.

Although the vector $\boldsymbol{\omega}_4$ of Example 4.5.4 is not minimal, it is in the $\mathcal{C}$-symmetry orbit of the minimal pseudocodeword $\boldsymbol{\omega}_1$. Indeed, we have that $\boldsymbol{\omega}_1^{[\mathbf{c}]} = \boldsymbol{\omega}_4$ for $\mathbf{c} = (0, 0, 1, 0, 1, 1, 0, 1, 0) \in \mathcal{C}$. Thus, in this example each nontrivial pseudocodeword is either minimal or it is in the orbit of a minimal pseudocodeword under the action of $\mathcal{C}$-symmetry. A natural question to ask is whether this occurs in general; that is, whether the minimal pseudocodewords "span" all pseudocodewords under the action of $\mathcal{C}$-symmetry. This is not, however, the case: an LP pseudocodeword can have four or more simple cycles in the normal graph whose edges receive values of $\frac{1}{2}$, whereas by Theorem 4.5.3 a minimal pseudocodeword can have at most two such cycles. For a concrete illustration, refer to Example 4.4.3.

Chapter 5

A Generalization of Omura's Decoding Algorithm

Let $\mathcal{C}$ be an $[n, k]$ code presented by the parity-check matrix H, and consider transmission over the binary symmetric channel with crossover probability $p < \frac{1}{2}$. If the binary vector $\mathbf{y}$ is received, it is well known that ML decoding on this channel can be achieved by the method discussed in Section 2.2.1: first, compute the syndrome $\mathbf{s} = H\mathbf{y}$ of the received vector. Next, find the smallest weight vector $\mathbf{e}_*$ satisfying $H\mathbf{e} = \mathbf{s}$. Finally, add $\mathbf{e}_*$ to $\mathbf{y}$. The result is an ML codeword.

A typical way of performing the second step in this process is to create, *a priori*, a look-up table of all possible syndrome vectors and the corresponding minimum weight error vectors (coset leaders) that produce them. Decoding then amounts to searching this array to find the minimum-weight error vector whose syndrome matches that of the received vector. For even moderately large values of n and k (e.g., $n = 100$ and $k = 50$), however, this naïve approach becomes computationally intractable. For this reason Omura [22] dispenses with this look-up table and designs a decoder that handles the task of finding the minimum weight vector $\mathbf{e}_*$ in a much different manner.

Development of this new decoder begins by recasting the problem as

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^T\mathbf{e} && (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H\mathbf{e} = \mathbf{s} && (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^n \end{aligned} \tag{5.1}$$

where $\mathbf{1}$ denotes a column vector of all 1's.

At first glance, Problem (5.1) looks like a linear program in standard form, but the function to be minimized is being computed in the reals while the constraints are binary. This prevents us from using the simplex method [6] for solving the problem. Omura [22] has developed a method that is guaranteed to solve this problem in finite time provided that H has full rank. This is accomplished by using an iterative algorithm whose behavior closely mimics that of the simplex algorithm. A practical snag is that this algorithm can be computationally prohibitive (this is not too much of a surprise, since the decision problem corresponding to ML decoding on the BSC is NP-complete [4]). Omura recognizes this drawback and compensates by randomizing the algorithm in such a way as to make it more efficient. Simulation results suggest that the corresponding loss of performance is not large, yet no proof that Omura's randomized algorithm always converges to the ML codeword is given in [22].

In this chapter, we generalize Omura's algorithm to decode any fixed binary linear code over an arbitrary binary-input memoryless channel (e.g., the binary symmetric channel, the binary erasure channel, the additive white Gaussian noise channel, the Z-channel, etc.). Moreover, we demonstrate that the generalized version of Omura's randomized algorithm can be modeled by a finite-state Markov chain and use this to show that the probability of decoding to an ML codeword approaches 1 as the number of iterations goes to infinity.

This chapter is organized as follows: Section 5.1 gives a brief survey of past work

on soft-decision decoding algorithms, as well as work directly following from Omura's original paper. Section 5.2 develops a key reduction of ML decoding on an arbitrary binary-input memoryless channel. Section 5.3 examines the reduction of Section 5.2 in the case of the BSC, the BEC, and the AWGN channel. We develop the generalized Omura algorithms in Section 5.4. Section 5.5 focuses on analyzing the performance of the randomized version of the generalized Omura decoder. Finally, Section 5.6 gives simulation results of the generalized Omura decoder for several codes, which serve as empirical evidence of the results proven in Section 5.5.

5.1 The Generalized Omura Decoder in Context

5.1.1 A Survey of Soft-Decision Decoders

The generalized Omura decoder can be categorized as a *soft-decision* decoder. Decoders in this class account for the reliability of received bits as well as the hard-decision vector (i.e., the vector obtained from the channel output by making coordinate-wise optimal choices). The use of this "soft" information can improve decoding performance by as much as 3 decibels over hard-decision decoding [21], making soft-decision decoding an attractive choice. In the following we outline three soft-decision decoders. As with the generalized Omura decoder, these algorithms can be used to decode any binary linear code over any channel for which the reliability of the received codeword bits can be computed (e.g., for completely non-deterministic channels). There are, however, notable differences between these decoders and the generalized Omura decoder.

One of the first soft-decision decoders was the *generalized minimum distance (GMD) decoder* introduced by Forney [16]. Given a received sequence $\mathbf{y}$, the GMD

decoder forms the hard decision vector $\widehat{\mathbf{y}}$ along with a vector $\mathbf{r}$ of reliability values, where $r_i \geq 0$ for all i. By making hard decisions, errors are likely to occur. Moreover, most of these errors will lie in the positions with lowest reliability. Erasures are therefore systematically introduced in the least reliable positions of $\mathbf{v}$, and the resulting vectors are each decoded with a hard-decision algebraic decoder that is capable of correcting both errors (for the relatively few reliable positions in error) and erasures (for the unreliable positions the decoder erased). This process results in a list of candidate codewords from which the most probable is chosen. In the case of a fixed code on the AWGN channel, the performance of GMD decoding approaches that of the optimal ML decoder as the signal-to-noise ratio (SNR) goes to infinity [16].

Three related soft-decision decoders are introduced by Chase in [10]. The three *Chase decoding algorithms* operate in a manner similar to the GMD decoder. Given a received vector $\mathbf{y}$, the Chase decoding algorithms first compute the hard-decision vector $\widehat{\mathbf{y}}$ and the vector of reliability values. A certain number of low-weight error patterns are introduced to $\widehat{\mathbf{y}}$ in the least reliable positions (compare this method to the GMD decoder that erases such bits). The manner of choosing which low-weight error patterns to form is determined by which of the three decoders is being used. Each resulting vector is decoded with a hard-decision errors-only decoding algorithm, and the best codeword from the final list of candidates is selected as the output. As with the GMD decoder, in the case of a fixed code on the AWGN channel the performance of each of the Chase decoding algorithms approaches ML performance as the SNR goes to infinity [10].

In contrast to the generalized minimum distance decoder or the Chase decoding algorithms, which make use of the least reliable received bits, the *ordered-statistic decoder* [14] of Fossorier and Lin makes use of the most reliable positions. Given a received vector $\mathbf{y}$, compute the reliability of each code-bit position. Assigning these

reliabilities as weights to the columns of a generator matrix for the code, one can use a greedy search to find a *most reliable basis* $\mathcal{B}$, i.e., a set of k linearly independent columns of G with maximum weight[1] (see Section 10.8 of [18] for an efficient method of finding such a basis). Let $\widehat{\mathbf{m}}$ denote the length k vector obtained by making hard decisions at each code-bit position in $\mathcal{B}$. Intuitively, $\widehat{\mathbf{m}}$ ought to differ from the transmitted codeword in relatively few positions since the positions in $\mathcal{B}$ were chosen to have high reliability. The ordered statistic decoder operates according to the following general procedure: first, make some number of low-weight alterations to the vector $\widehat{\mathbf{m}}$ to obtain a set of vectors $\mathcal{M}$. Using the columns of G indicated by $\mathcal{B}$ as an information set, encode each vector $\mathbf{m}' \in \mathcal{M}$ as a codeword $\mathbf{y}'$ and compute the likelihood of receiving $\mathbf{y}'$ given that $\mathbf{m}'$ was transmitted. Select the vector with the highest likelihood.

It is through the lens of performance that we see the first major difference between the generalized Omura decoder and the aforementioned decoders. For a fixed code, the GMD decoder, the three Chase decoders, and the generalized Omura algorithm are all asymptotically ML on the AWGN channel, but they are asymptotic with respect to different parameters. For the GMD and the three Chase decoders to approach ML performance, the SNR must be allowed to grow without bound - essentially, this means that the channel itself is becoming completely reliable. This is in contrast to the generalized Omura decoder, which by Theorem 5.5.11 is guaranteed to approach ML performance *for a fixed SNR* as long as the decoder is permitted to use an arbitrarily large number of iterations.

It is the author's belief that the difference in performance results highlighted in

[1]If A is a matrix with column set E, then the collection of linearly independent subsets of E forms a *matroid* on the ground set E. For any nonnegative weight function on E, the greedy algorithm is guaranteed to produce a maximal linearly independent set of maximum total weight. For more reading on matroids, see, e.g., [26].

the previous paragraph is due primarily to an algorithmic difference between the generalized Omura decoder and the other decoders. Generalized minimum distance decoding, Chase decoding, and ordered statistic decoding all work by fixing an ordering on the received bits (using reliability information) and proceeding with strict respect to this order – the idea being that errors are more likely to occur in positions with low reliability. Still, it is possible for positions with high reliability to be received in error and likewise for positions with low reliability to be received correctly. Moreover, this intuition of relative likelihood loses its potency for received vectors where the range of reliability values is small (an extreme example is the BSC, where reliability is uniform). As will be made clear in Section 5.4, the generalized Omura decoder takes into account reliability information, but, given the opportunity, it is able to flip the value of any received bit – regardless of that bit's individual reliability – to arrive at a better solution.

5.1.2 Derivatives and Other Generalizations of Omura's Algorithms

Given the number of years that have passed since the 1972 publication date of Omura's paper, there has been relatively little work done on extending the methods of Omura. In [11], Omura's decoder is adapted to the AWGN channel[2]. This decoder is more or less identical to the decoder obtained by specializing the generalized Omura decoder that we will develop in Section 5.4.2 to the AWGN channel. To discuss the significance of the work presented in this chapter, it is useful to describe a common model into which these decoding algorithms can be placed.

As will be shown in Section 5.5, these Omura-type algorithms can be modeled

[2]The Omura-type algorithm of [11] is also explored in [7].

accurately as guided walks on the *matroid basis graph* [19] of the parity-check matrix H. The matroid basis graph of a matrix M has as its vertices every possible maximal linearly independent set of columns taken from M. As such, these vertices are called *bases*. Two bases B_1 and B_2 are adjacent if and only if $|B_1 \cap B_2| = \text{rank}(M) - 1$; in other words, if they differ in only one element. The reason for considering this specific graph is that there is a natural map (made explicit in Definition 5.4.3) from the set of bases to the set of candidate codewords. Each algorithm is initialized at some basis B_0. It then examines all neighboring bases to see if one offers an improved solution. If it finds such a neighbor, it moves to this new basis and the process continues. The end goal is to arrive at a basis that corresponds to an optimal, or nearly optimal, solution to the ML decoding problem.

Using intuition similar to that of the ordered statistic decoder, the decoding algorithm of [11] makes use of the *least-reliable basis* – a basis for the columns of H that has minimal reliability. Using a least reliable basis has the effect of giving this algorithm a head-start over the generalized Omura decoder, which manifests itself in very good simulation results (see [7]). A heuristic explanation of this good decoding performance is given in [11], but a precise analysis (i.e., an actual theorem regarding performance) is still lacking.

The main contribution of the work herein is two-fold. First, we explicitly describe how to adapt Omura's algorithm to any binary-input memoryless channel, including those that are partially deterministic. Second and most importantly, we abandon the notion of most and least reliable bases. This is not to say that we cannot use a least reliable basis to initialize the generalized Omura decoder – doing so just brings us back to previous attacks on the problem. Rather, this carefree point of view allows us to prove that the generalized Omura decoder, and hence Omura's original algorithm [22] and the decoder of [11], will converge in probability to the ML codeword *regardless*

of the initial basis.

On a final note, we observe that some of the same ideas utilized by Omura have been applied in the context of cryptanalysis. Canteaut and Chabaud [8] develop an algorithm not unlike Omura's for the purpose of breaking McEliece's cryptosystem. As opposed to operating on the fundamental hardness of computing the discrete logarithm or factoring large integers, McEliece's cryptosystem relies on the hardness of decoding a code that has seemingly random structure [20]. Attacks on such systems must therefore be capable of achieving relatively good decoding performance while utilizing only the linearity of the code, which is exactly what Omura's original algorithm is designed to do. In light of Theorem 5.5.11, further analysis of the convergence rate of the generalized Omura decoder will aid in determining its efficacy as an attack on code-based cryptosystems.

5.2 Generalization to an Arbitrary Binary-Input Memoryless Channel.

Let $\mathcal{C}$ be a binary linear code with parity-check matrix H, not necessarily of full rank. Consider transmission over a binary-input, memoryless channel and suppose that the vector $\mathbf{y}$ is received. By Theorem 2.3.1, the output of the ML decoder can be taken to be any solution to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T \mathbf{c}^{\mathfrak{U}} \\ \text{subject to} \quad & \mathbf{c} \in \mathcal{C} \\ \text{and} \quad & \mathbf{c}^{\mathfrak{C}} = \mathbf{d} \end{aligned} \tag{5.2}$$

where $\mathfrak{U} = \{i \,|\, P(y_i|0) > 0 \text{ and } P(y_i|1) > 0\}$ is the set of uncertain code-bit positions, $\mathfrak{C} = \{1, 2, \ldots, n\} \setminus \mathfrak{U}$ is the set of certain code-bit positions, $\boldsymbol{\lambda}$ is the vector of log-

likelihood ratios on $\mathfrak{U}$, and $\mathbf{d}$ is the vector of codeword bits determined by $\mathfrak{C}$.

The requirement that $\mathbf{c} \in \mathcal{C}$ and $\mathbf{c}^{\mathfrak{C}} = \mathbf{d}$ is equivalent to looking only at those binary vectors $\mathbf{x}$ of length $u := |\mathfrak{U}|$ such that $H^{\mathfrak{C}}\mathbf{d} + H^{\mathfrak{U}}\mathbf{x} = 0$, which is the same as $H^{\mathfrak{U}}\mathbf{x} = H^{\mathfrak{C}}\mathbf{d}$ where, as in Chapter 2, we use $H^{\mathfrak{U}}$ and $H^{\mathfrak{C}}$ to denote the matrix H projected onto the columns indicated by $\mathfrak{U}$ and $\mathfrak{C}$, respectively. We may therefore rephrase Problem (5.2) as

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T\mathbf{x} && (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H^{\mathfrak{U}}\mathbf{x} = H^{\mathfrak{C}}\mathbf{d} && (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{x} \in \{0,1\}^u. \end{aligned} \tag{5.3}$$

Define the vector $\widehat{\mathbf{y}} \in \{0,1\}^u$ coordinate-wise by

$$\widehat{y}_i = \begin{cases} 0 & \text{if } \lambda_i \geq 0 \\ 1 & \text{if } \lambda_i < 0. \end{cases}$$

The vector $\widehat{\mathbf{y}}$ is the hard-decision vector based on $\mathbf{y}^{\mathfrak{U}}$; for example, on the BSC with $p < \frac{1}{2}$ we have $\mathfrak{C} = \emptyset$ and $\widehat{\mathbf{y}} = \mathbf{y}$. It is clear that $\widehat{\mathbf{y}}$ minimizes $\boldsymbol{\lambda}^T\mathbf{x}$ over all vectors $\mathbf{x} \in \{0,1\}^u$; however, $\widehat{\mathbf{y}}$ need not satisfy the constraint $H^{\mathfrak{U}}\widehat{\mathbf{y}} = H^{\mathfrak{C}}\mathbf{d}$ specified by Problem (5.3). Since any solution to Problem (5.3) must satisfy this constraint, we are left to find a perturbation $\mathbf{v}$ of $\widehat{\mathbf{y}}$ with $H^{\mathfrak{U}}\mathbf{v} = H^{\mathfrak{C}}\mathbf{d}$ such that the difference between $\boldsymbol{\lambda}^T\mathbf{v}$ and $\boldsymbol{\lambda}^T\widehat{\mathbf{y}}$ is as small as possible.

Set $\mathbf{s} = H^{\mathfrak{U}}\widehat{\mathbf{y}} + H^{\mathfrak{C}}\mathbf{d}$. All solutions $\mathbf{v}$ to $H^{\mathfrak{U}}\mathbf{v} = H^{\mathfrak{C}}\mathbf{d}$ can be generated by first finding a solution $\mathbf{e}$ to $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$ and then setting $\mathbf{v} = \widehat{\mathbf{y}} + \mathbf{e}$. Let "$\odot$" denote the coordinate-wise (Hadamard) product of two vectors. Keeping in mind that $\widehat{\mathbf{y}}, \mathbf{e} \in \{0,1\}^u$, we have that $\widehat{\mathbf{y}} + \mathbf{e}$ is symbolically the same as $\widehat{\mathbf{y}} + \mathbf{e} - 2(\mathbf{e} \odot \widehat{\mathbf{y}})$, where the first sum is computed modulo 2 and the the latter sum is computed in the real

numbers. Using this, Problem (5.3) reduces to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T \widehat{\mathbf{y}} + \boldsymbol{\lambda}^T \mathbf{e} - 2\boldsymbol{\lambda}^T (\mathbf{e} \odot \widehat{\mathbf{y}}) \quad (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H^{\mathfrak{U}} \mathbf{e} = \mathbf{s} \qquad\qquad\qquad (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^u. \end{aligned} \tag{5.4}$$

The $\boldsymbol{\lambda}^T \widehat{\mathbf{y}}$ term in Problem (5.4) above does not affect which $\mathbf{e} \in \{0,1\}^u$ solves the problem. We may therefore drop this term to obtain

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\lambda}^T \mathbf{e} - 2\boldsymbol{\lambda}^T (\mathbf{e} \odot \widehat{\mathbf{y}}) \quad (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H^{\mathfrak{U}} \mathbf{e} = \mathbf{s} \qquad\qquad\qquad (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^u. \end{aligned} \tag{5.5}$$

Define the vector $\boldsymbol{\beta} \in \mathbb{R}^u$ by $\beta_i = |\lambda_i|$ for all $i \in \mathfrak{U}$. Notice that $\boldsymbol{\beta} = \boldsymbol{\lambda} - 2(\boldsymbol{\lambda} \odot \widehat{\mathbf{y}})$ since $\widehat{y}_i = 1$ when λ_i is negative and $\widehat{y}_i = 0$ otherwise. We have

$$\begin{aligned} \boldsymbol{\lambda}^T \mathbf{e} - 2\boldsymbol{\lambda}^T (\widehat{\mathbf{y}} \odot \mathbf{e}) &= \boldsymbol{\lambda}^T \mathbf{e} - 2(\boldsymbol{\lambda} \odot \widehat{\mathbf{y}})^T \mathbf{e} \\ &= (\boldsymbol{\lambda} - 2(\boldsymbol{\lambda} \odot \widehat{\mathbf{y}}))^T \mathbf{e} \\ &= \boldsymbol{\beta}^T \mathbf{e}, \end{aligned}$$

and Problem (5.5) can be further reduced to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\beta}^T \mathbf{e} \qquad (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H^{\mathfrak{U}} \mathbf{e} = \mathbf{s} \qquad (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^u. \end{aligned} \tag{5.6}$$

Adopting terminology from linear programming, we say that $\boldsymbol{\beta}$ is the *cost function* of Problem (5.6). We do this because β_i represents the "cost" or "penalty" one must

pay in order to set $e_i = 1$.

In summary, we have reduced the problem of maximum-likelihood decoding on an arbitrary binary-input memoryless channel to the following: given a received vector, find the sets $\mathfrak{C}$ and $\mathfrak{U}$ and compute the vectors $\mathbf{d}$, $\boldsymbol{\lambda}$, $\widehat{\mathbf{y}}$, $\mathbf{s}$, and $\boldsymbol{\beta}$. Find a vector $\mathbf{e}_* \in \{0,1\}^u$ that minimizes $\boldsymbol{\beta}^T\mathbf{e}$ subject to the constraint that $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$. Add this vector $\mathbf{e}_*$ to $\widehat{\mathbf{y}}$ and extend this sum to a vector of length n by padding it with the entries of $\mathbf{d}$. The result is an ML codeword.

5.3 Reality Check, and Something New

The process of performing maximum-likelihood decoding as outlined at the end of Section 5.2 involves much notation to handle the full generality of an arbitrary binary-input memoryless channel. This method does, however, reduce to familiar methods of decoding when we consider specific channels.

First, consider the case of the binary symmetric channel with crossover probability $p < \frac{1}{2}$. We observe that on the BSC the set $\mathfrak{C}$ is always empty – there is always a chance that a received bit may have been flipped. Thus, the vector $\mathbf{d}$ does not exist and $\widehat{\mathbf{y}}$ is simply $\mathbf{y}$. The vector $\mathbf{s}$ is the usual syndrome $H\mathbf{y}$. The vector $\boldsymbol{\lambda}$ of log-likelihood ratios is a vector in $\left\{\pm\log\left(\frac{p}{1-p}\right)\right\}^n$, and the corresponding $\boldsymbol{\beta} = |\boldsymbol{\lambda}|$ is a constant vector that can be scaled to the all-ones vector. Putting this together shows that on the BSC, Problem (5.6) reduces to Problem (5.1), i.e., the problem that Omura originally considered [22].

Next, assume that transmission occurs over the binary erasure channel. The set $\mathfrak{U}$ denotes the set of erased positions, and $\mathbf{d}$ is the vector of bits that did not get erased by the channel. Since an erased bit has an equal probability of being 0 or 1,

the vector $\boldsymbol{\lambda}$ of log-likelihood ratios is the all-zeros vector. Problem (5.6) becomes

$$\begin{aligned} \text{minimize} \quad & 0 && (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H^{\mathfrak{U}}\mathbf{e} = \mathbf{s} && (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^u. \end{aligned}$$

Since the quantity to be minimized is constant, solving this problem amounts to finding a solution to $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$. Said another way, we need to find binary values for $\mathbf{e}$, the erased bits, such that when we combine these bits with $\mathbf{d}$ we get a codeword. This is the classical decoding rule for the BEC outlined in Section 2.2.3.

Finally, we consider the case of the additive white Gaussian noise channel. With this channel model, Problem (5.6) becomes equivalent to the problem considered in [11]. Elaborating on this, assume binary antipodal modulation of codeword bits via the map $c_i \mapsto 2c_i - 1$ and let $\mathbf{y}$ be a received vector. The set of certain codeword bits $\mathfrak{C}$ is empty, and $\boldsymbol{\lambda}$ is proportional to $-\mathbf{y}$:

$$\begin{aligned} \lambda_i &= \log\left(\frac{P(y_i|c_i=0)}{P(y_i|c_i=1)}\right) \\ &= \log\left(\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y_i+1)^2}{2\sigma^2}}\right) / \left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y_i-1)^2}{2\sigma^2}}\right)\right) \\ &= \log\left(\left(e^{-\frac{(y_i+1)^2}{2\sigma^2}+\frac{(y_i-1)^2}{2\sigma^2}}\right)\right) \\ &= -\frac{2y_i}{\sigma^2}. \end{aligned}$$

Thus, $\widehat{y_i} = 1$ if and only if $y_i > 0$, and the cost function $\boldsymbol{\beta}$ is $|\mathbf{y}|$. We note that $\boldsymbol{\beta}$ could be *any real-valued vector* whose entries are all positive, although small entries are certainly more probable than larger ones.

Of course, this generalization is of little use if we have no method of solving Problem (5.6). In Section 5.4, we generalize two of Omura's algorithms. Algorithm 1 is

guaranteed to solve Problem (5.6), but as is the case with the corresponding algorithm in [22], this method is too computationally intense to be used in general. Algorithm 2 is a randomized version of Algorithm 1, and is less costly to implement. Algorithm 2, however, is not guaranteed to solve Problem (5.6) in a finite number of iterations. In Section 5.5 we present a detailed analysis of Algorithm 2, which shows that the probability of Algorithm 2 returning the ML codeword approaches 1 as the number of iterations grows without bound.

5.4 Algorithm Development

5.4.1 The Full-Rank Assumption

Before developing any algorithms, we first show that without loss of generality we may assume that the matrix $H^{\mathfrak{U}}$ of Problem (5.6) has full rank. This full-rank assumption is crucial to both Algorithm 1 and Algorithm 2, which will be introduced in Section 5.4.3.

Proposition 5.4.1. *Let $\mathcal{C}$ be a code presented by some $t \times n$ parity-check matrix H, and consider transmission over a binary-input memoryless channel. Let $\mathbf{y}$ be a received vector, and let $\mathfrak{U}$, $\boldsymbol{\beta}$, and $\mathbf{s}$ be the set of uncertain code-bit positions, the cost function, and the syndrome vector computed from $\mathbf{y}$. When solving*

$$\begin{aligned} \textit{minimize} \quad & \boldsymbol{\beta}^T \mathbf{e} && (\textit{over } \mathbb{R}) \\ \textit{subject to} \quad & H^{\mathfrak{U}} \mathbf{e} = \mathbf{s} && (\textit{over } \mathbb{F}_2) \\ \textit{over all} \quad & \mathbf{e} \in \{0,1\}^u \end{aligned}$$

one may assume without loss of generality that the projected $t \times u$ matrix $H^{\mathfrak{U}}$ has full rank.

Proof. Let $r \leq t$ be the rank of $H^{\mathfrak{U}}$. Then there is then a collection of r linearly independent rows of $H^{\mathfrak{U}}$ that are indexed by some set $\mathcal{A} := \{a_1 < a_2 < \cdots < a_r\}$. Let $M = (m_{i,j})$ be the $r \times t$ matrix with $m_{i,j} = 1$ if and only if $j = a_i$. Then $MH^{\mathfrak{U}}$ is the full-rank $r \times u$ matrix consisting of the linearly independent rows of $H^{\mathfrak{U}}$ specified by $\mathcal{A}$.

We claim that the solution sets of $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$ and $MH^{\mathfrak{U}}\mathbf{e} = M\mathbf{s}$ coincide. We first argue that there must be at least one solution to $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$. Let $\mathfrak{C} = \{1, 2, \ldots, n\} \setminus \mathfrak{U}$, let $\mathbf{d}$ be the vector of length $n-u$ of codeword bits determined by $\mathbf{y}^{\mathfrak{C}}$, and, as usual, let $\widehat{\mathbf{y}}$ be the hard-decision vector of length u based on $\mathbf{y}^{\mathfrak{U}}$. Then there exists a codeword $\mathbf{c} \in \mathcal{C}$ such that $\mathbf{c}^{\mathfrak{C}} = \mathbf{d}$. This implies that $H^{\mathfrak{U}}\mathbf{c}^{\mathfrak{U}} = H^{\mathfrak{C}}\mathbf{d}$. Set $\mathbf{e} = \mathbf{c}^{\mathfrak{U}} + \widehat{\mathbf{y}}$, and recall that $\mathbf{s}$ is defined as $H^{\mathfrak{U}}\widehat{\mathbf{y}} + H^{\mathfrak{C}}\mathbf{d}$. Then

$$\begin{aligned} H^{\mathfrak{U}}\mathbf{e} &= H^{\mathfrak{U}}\widehat{\mathbf{y}} + H^{\mathfrak{U}}\mathbf{c}^{\mathfrak{U}} \\ &= H^{\mathfrak{U}}\widehat{\mathbf{y}} + H^{\mathfrak{C}}\mathbf{d} \\ &= \mathbf{s}. \end{aligned}$$

Now let $\mathbf{e}_*$ be a solution of $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$, and let $\mathcal{N}$ denote the null space of $H^{\mathfrak{U}}$. The solution set $\mathcal{S}$ of $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$ is exactly $\{\mathbf{e}_* + \mathbf{n} \mid \mathbf{n} \in \mathcal{N}\}$. By the rank-nullity theorem, $|\mathcal{S}| = 2^{u-r}$. Let $\mathcal{S}^M$ denote the solution set of $MH^{\mathfrak{U}}\mathbf{e} = M\mathbf{s}$. It is clear that $\mathcal{S} \subseteq \mathcal{S}^M$. By the rank-nullity theorem, $|\mathcal{S}^M| = 2^{u-r}$, so $\mathcal{S} = \mathcal{S}^M$. Thus, the solution sets of $H^{\mathfrak{U}}\mathbf{e} = \mathbf{s}$ and $MH^{\mathfrak{U}}\mathbf{e} = M\mathbf{s}$ coincide. It follows that any vector that is a solution to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\beta}^T\mathbf{e} && (\text{over } \mathbb{R}) \\ \text{subject to} \quad & H^{\mathfrak{U}}\mathbf{e} = \mathbf{s} && (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^u \end{aligned}$$

must also be a solution to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\beta}^T \mathbf{e} && (\text{over } \mathbb{R}) \\ \text{subject to} \quad & MH^{\mathfrak{U}}\mathbf{e} = M\mathbf{s} && (\text{over } \mathbb{F}_2) \\ \text{over all} \quad & \mathbf{e} \in \{0,1\}^u. \end{aligned}$$

Since $MH^{\mathfrak{U}}$ has full rank, the proposition is proved. □

5.4.2 Elementary Results

We now develop some elementary yet important results that will aid us in solving Problem (5.6). This subsection is essentially a reproduction of the work done by Omura [22], borrowing much notation and generalizing many results. It is necessary that we include this reproduction because in shifting from the BSC to general binary-input memoryless channels we are no longer able to assume the cost function $\boldsymbol{\beta}$ (see Section 5.2) is constant. We do, however, know that $\boldsymbol{\beta}$ only takes on non-negative values, a fact that turns out to be the key ingredient in generalizing Omura's theorems.

Throughout this section, let $\mathbf{y}$ be a vector received from a binary-input memoryless channel. Any objects dependent on the received vector, such as $\mathfrak{U}$, $\mathbf{s}$, $\boldsymbol{\beta}$, etc., are assumed to be associated with the fixed vector $\mathbf{y}$ unless otherwise stated. For simplicity, we reset the notation of Problem 5.6 so that $H := H^{\mathfrak{U}}$ is an $(n-k) \times n$ binary matrix. By Proposition 5.4.1, we may assume that H has full rank. We let $\mathbf{h}_\ell$ denote the ℓth column of H. Let B be an $(n-k) \times (n-k)$ square matrix formed by some collection of $(n-k)$ linearly independent columns of H (such a matrix exists, since we assume H to have full rank). We call any such matrix B a *basic matrix* of H, and the columns of B are referred to as the *basic columns*. Let $I = (i_1, i_2, \ldots, i_{n-k})$ be an ordered $(n-k)$-tuple such that $\mathbf{h}_{i_\ell}$ is the ℓth column of B. Similarly, let

$J = (j_1, j_2, \ldots, j_k)$ denote an ordered k-tuple indexing the columns of H not used by B. We use $I(\ell)$ and $J(p)$ to refer to the ℓth and pth entries of I and J, respectively. Define R as the $(n-k) \times k$ matrix formed by the columns indicated by J. The columns of R are referred to as the *nonbasic columns*. If $\mathbf{x}$ is a length n vector, let $\mathbf{x}^B$ denote the projection of $\mathbf{x}$ onto the coordinates associated with the basic columns and $\mathbf{x}^R$ denote the projection of $\mathbf{x}$ onto the coordinates associated with the nonbasic columns. We define the *cost* of a length n vector $\mathbf{e} \in \{0,1\}^n$ as $\text{cost}(\mathbf{e}) := \boldsymbol{\beta}^T\mathbf{e}$, where $\boldsymbol{\beta}$ is the cost function of Problem (5.6), i.e., the vector of absolute values of the log-likelihood ratios. With this notation, we may rephrase Problem (5.6) as

$$\begin{array}{rll} \text{minimize} & \text{cost}(\mathbf{e}) & (\text{over } \mathbb{R}) \\ \text{subject to} & H\mathbf{e} = \mathbf{s} & (\text{over } \mathbb{F}_2) \\ \text{over all} & \mathbf{e} \in \{0,1\}^n. & \end{array} \tag{5.7}$$

At times it will be useful to consider the cost of a vector associated only with the basic or nonbasic columns, so we define $\text{cost}^B(\mathbf{u}) := (\boldsymbol{\beta}^B)^T\mathbf{u}$ for vectors $\mathbf{u}$ of length $(n-k)$ and $\text{cost}^R(\mathbf{v}) := (\boldsymbol{\beta}^R)^T\mathbf{v}$ for vectors $\mathbf{v}$ of length k .

If $H\mathbf{e} = \mathbf{s}$, we have

$$\mathbf{s} = B\mathbf{e}^B + R\mathbf{e}^R.$$

Since B is invertible, we may write

$$B^{-1}\mathbf{s} = \mathbf{e}^B + B^{-1}R\mathbf{e}^R.$$

The vector $B^{-1}\mathbf{s}$ and the matrix $B^{-1}R$ are important enough to warrant names of their own: following [22], set $\bar{\mathbf{e}}^B := B^{-1}\mathbf{s}$ and $Y := B^{-1}R$. With this notation, the

previous equation becomes

$$\bar{\mathbf{e}}^B = \mathbf{e}^B + Y\mathbf{e}^R.$$

We write $\mathbf{y}_p$ for the pth column of Y and $y_{\ell,p}$ for the (ℓ, p)th entry of Y.

Example 5.4.2. *Consider the* $[7, 4, 3]$ *Hamming code, as described by the following parity check matrix:*

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

Let $I = (2, 1, 4)$ *be the indices of the basic columns of* H *and let* $J = (3, 5, 6, 7)$ *be the indices of the nonbasic columns. The corresponding basic matrix and matrix of nonbasic columns are*

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

and

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

Note that $B^{-1} = B$. *Suppose that* $\mathbf{s} = (1, 0, 1)^T$. *Then*

$$\bar{\mathbf{e}}^B = B^{-1}\mathbf{s} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

and

$$Y = B^{-1}R = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Definition 5.4.3 ([22])**.** *The* basic solution *associated with a basic matrix B of H is the solution of*

$$H\mathbf{e} = \mathbf{s}$$

given by constructing $\mathbf{e}$ *such that*

$$\mathbf{e}^B = \bar{\mathbf{e}}^B = B^{-1}\mathbf{s}$$

and

$$\mathbf{e}^R = \mathbf{0}.$$

As an example of Definition 5.4.3, the basic solution associated with the basic matrix of Example 5.4.2 is $\mathbf{e} = (1, 0, 0, 1, 0, 0, 0)^T$. The next proposition establishes that in trying to solve Problem (5.7), it suffices to consider only basic solutions.

Proposition 5.4.4 (compare to [22], Theorem 1)**.** *If* $\mathbf{e}$ *satisfies* $H\mathbf{e} = \mathbf{s}$*, then there exists a basic solution* $\mathbf{e}_*$ *such that* $\text{cost}(\mathbf{e}_*) \leq \text{cost}(\mathbf{e})$.

Proof. Let A denote the set of columns of H indicated by the non-zero positions of $\mathbf{e}$. Let M be a maximal set of vectors in A that add to $\mathbf{0}$ (note that M could be the empty set). Then $A \setminus M$ is linearly independent, so we may supplement $A \setminus M$ with other columns of H so as to form a basic matrix B. Let $\mathbf{e}_*$ be the basic solution associated to B. Abusing notation, identify the sets A and M with the indices of the columns of H they contain. Since $\sum_{\ell \in M} \mathbf{h}_\ell = \mathbf{0}$, we have that $\mathbf{s} = H\mathbf{e} = \sum_{i \in A \setminus M} \mathbf{h}_i + \sum_{\ell \in M} \mathbf{h}_\ell = \sum_{i \in A \setminus M} \mathbf{h}_i$. This last expression is a linear

combination of columns of B that adds to $\mathbf{s}$, so it must be that $A \setminus M = \text{supp}(\mathbf{e}_*)$ since B is invertible. Finally, we note that since $\boldsymbol{\beta}$ is a vector with all entries non-negative and $\text{supp}(\mathbf{e}_*) \subseteq \text{supp}(\mathbf{e})$, we have $\text{cost}(\mathbf{e}_*) = \boldsymbol{\beta}^T \mathbf{e}_* \leq \boldsymbol{\beta}^T \mathbf{e} = \text{cost}(\mathbf{e})$. □

Proposition 5.4.4 motivates the following definition.

Definition 5.4.5. *The* cost *of a basic matrix B is defined to be* $\text{cost}(B) := \text{cost}(\mathbf{e}) = \text{cost}^B(\bar{\mathbf{e}}^B)$*, where* $\mathbf{e}$ *is the basic solution associated to B. We say that a basic matrix B is* optimal *if the basic solution associated to B is an optimal solution to Problem (5.7).*

We now introduce the notion of the *tableau* associated to a basic matrix. As in the so-called "full tableau" implementation of the simplex method (see, e.g., [6]), the tableau provides an object well-suited to the storage of information. Moreover, Proposition 5.4.8 and Corollary 5.4.9 below demonstrate that changing from one tableau to another can be as easy as performing elementary row operations. This fact that will be put to good use when implementing Algorithm 2, which will be introduced in Section 5.4.3.

Definition 5.4.6. *Let B be a basic matrix of H. The* tableau *associated to B is the $(n-k) \times (n+1)$ array given by*

$$tab(B) = \left[B^{-1}\mathbf{s}, \quad B^{-1}H \right] = \left[\bar{\mathbf{e}}^B, \quad B^{-1}H \right].$$

The columns of $tab(B)$ are numbered from left to right as $0, 1, 2, \ldots, n$.

Note that all $(n-k)$ standard basis vectors may be found in the columns of the tableau: they appear in the locations indicated by I. The columns corresponding to J are exactly the columns of $Y = B^{-1}R$, and the cost of column 0, $\text{cost}^B(\bar{\mathbf{e}}^B)$, is precisely the cost of the basic solution associated to B. These observations are made concrete in the following example.

Example 5.4.7. *This example is a continuation of Example 5.4.2. Recall that* $I = (2, 1, 4)$, $J = (3, 5, 6, 7)$,

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

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and

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

Suppose that the vector of log-likelihood ratios is

$$\boldsymbol{\lambda} = (1, 2, 2, 3, -0.5, 2, 2)^T.$$

The corresponding vectors are

$$\widehat{\mathbf{y}} = (0, 0, 0, 0, 1, 0, 0)^T$$

$$\mathbf{s} = (1, 0, 1)^T$$

$$\boldsymbol{\beta} = (1, 2, 2, 3, 0.5, 2, 2)^T.$$

The tableau associated to B is therefore

$$tab(B) = \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{cccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right] \end{array}.$$

Column 0 of $tab(B)$ is $\bar{\mathbf{e}}^B = B^{-1}\mathbf{s}$. The first, second, and third standard basis vectors may be found in columns 2, 1, and 4, respectively, as was predicted by I. Columns 3, 5, 6, and 7 are the columns of Y, as was predicted by J. Finally, Definition 5.4.5 states that we may compute $\text{cost}(B)$ *as follows:*

$$\begin{aligned} \text{cost}(B) &= (\boldsymbol{\beta}^B)^T \bar{\mathbf{e}}^B \\ &= (2,1,3)(0,1,1)^T \\ &= 4. \end{aligned}$$

This value of 4 is identical to the cost of the basic solution $(1,0,0,1,0,0,0)$ *associated to B.*

Proposition 5.4.8 (compare to [22], Theorem 2)**.** *Let B be a basic matrix with associated matrix Y, and let J denote the nonbasic columns with respect to B. By replacing the ℓth column of B with column $\mathbf{h}_{J(p)}$ of H, the new matrix B_* we obtain is a basic matrix if and only if $y_{\ell,p} = 1$. Moreover, when $y_{\ell,p} = 1$ we have $B_*^{-1} = AB^{-1}$, where A is the matrix that performs elementary row operations on $\mathbf{y}_p$ to obtain the ℓth standard basis vector.*

Proof. The vector $\mathbf{y}_p = B^{-1}\mathbf{h}_{J(p)}$ gives the unique linear combination of columns of

B that produces $\mathbf{h}_{J(p)}$, since

$$\mathbf{h}_{J(p)} = B(B^{-1}\mathbf{h}_{J(p)}) = B\mathbf{y}_p = \sum_{\ell=1}^{n-k} y_{\ell,p}\mathbf{h}_{I(\ell)}.$$

If $y_{\ell,p} = 0$, then there is a linear dependence between $\mathbf{h}_{J(p)}$ and columns $1, 2, \ldots, \ell - 1, \ell+1, \ldots, n-k$ of B. Conversely, if $y_{\ell,p} = 1$, then the ℓth column of B is required in the linear combination of columns of B that produces $\mathbf{h}_{J(p)}$, and no linear dependence can exist between $\mathbf{h}_{J(p)}$ and columns $1, 2, \ldots, \ell - 1, \ell + 1, \ldots, n - k$ of B. Thus, by replacing the ℓth column of B with column $\mathbf{h}_{J(p)}$ of H we obtain a new basic matrix B_* if and only if $y_{\ell,p} = 1$.

Suppose now that $y_{\ell,p} = 1$. For $j \neq \ell$, the jth column of $B^{-1}B_*$ is the jth standard basis vector, due to the fact that the jth columns of B and B_* coincide. The ℓth column of $B^{-1}B_*$ is $B^{-1}\mathbf{h}_{J(p)} = \mathbf{y}_p$. Let A be the matrix that performs elementary row operations on $\mathbf{y}_p$ to obtain the ℓth standard basis vector. Then $AB^{-1}B_*$ is the identity matrix, which implies that $B_*^{-1} = AB^{-1}$. □

Combining Definition 5.4.6 with Proposition 5.4.8, we obtain the following corollary.

Corollary 5.4.9. *Let B be a basic matrix with Y its associated matrix, and let J denote the nonbasic columns with respect to B. If $y_{\ell,p} = 1$ and B_* is the basic matrix formed by replacing the ℓth column of B with $\mathbf{h}_{J(p)}$, then tab(B_*) can be found by row-reducing tab(B) in such a way as to make column $J(p)$ the ℓth standard basis vector.*

Theorem 5.4.10 (compare to [22], Theorem 3)**.** *Let B be a basic matrix with Y its associated matrix, and let J denote the nonbasic columns with respect to B. If $y_{\ell,p} = 1$ and B_* is the basic matrix formed by replacing the ℓth column of B with $\mathbf{h}_{J(p)}$, then*

the cost of the basic solution associated to B_* is given by

$$\text{cost}^{B_*}(\bar{\mathbf{e}}^{B_*}) = \begin{cases} \text{cost}^B(\bar{\mathbf{e}}^B) & \text{if } \bar{e}_\ell^B = 0 \\ \text{cost}^B(\bar{\mathbf{e}}^B) + \beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) & \text{if } \bar{e}_\ell^B = 1, \end{cases}$$

where $\beta_{J(p)}$ is the $J(p)$th entry of $\boldsymbol{\beta}$, the cost function of Problem (5.7).

Proof. By Corollary 5.4.9, we may obtain $\text{tab}(B_*)$ from $\text{tab}(B)$ by row-reducing in such a way as to transform column $J(p)$ of $\text{tab}(B)$ into the ℓth standard basis vector. If $\bar{e}_\ell^B = 0$, this process of row-reduction has no effect on column 0 of $\text{tab}(B)$, and the basic solution associated with B is the same as the basic solution associated with B_*.

If $\bar{e}_\ell^B = 1$, the row reduction makes the first column of $\text{tab}(B_*)$ equal to $\bar{\mathbf{e}}^{B_*} = \bar{\mathbf{e}}^B + \mathbf{y}_p + \boldsymbol{\epsilon}_\ell$, where the sum is computed modulo 2 and $\boldsymbol{\epsilon}_\ell$ denotes the ℓth standard basis vector. As noted in Section 5.2, we may write $\bar{\mathbf{e}}^{B_*}$ as the real sum $\bar{\mathbf{e}}^B + \mathbf{y}_p - 2\bar{\mathbf{e}}^B \odot \mathbf{y}_p + \boldsymbol{\epsilon}_\ell$. Therefore

$$\text{cost}^{B_*}(\bar{\mathbf{e}}^{B_*}) = \text{cost}^{B_*}(\bar{\mathbf{e}}^B) + \text{cost}^{B_*}(\mathbf{y}_p) - 2\text{cost}^{B_*}(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) + \text{cost}^{B_*}(\boldsymbol{\epsilon}_\ell).$$

Let I denote the tuple indicating the columns of B. Since the columns of B_* are the same as the columns of B with the exception of the ℓth column that is now $\mathbf{h}_{J(p)}$, we have

$$\begin{aligned} \text{cost}^{B_*}(\bar{\mathbf{e}}^B) &= \text{cost}^B(\bar{\mathbf{e}}^B) - \beta_{I(\ell)} + \beta_{J(p)} \\ \text{cost}^{B_*}(\mathbf{y}_p) &= \text{cost}^B(\mathbf{y}_p) - \beta_{I(\ell)} + \beta_{J(p)} \\ -2\text{cost}^{B_*}(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) &= -2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) + 2\beta_{I(\ell)} - 2\beta_{J(p)} \\ \text{cost}^{B_*}(\boldsymbol{\epsilon}_\ell) &= \beta_{J(p)}. \end{aligned}$$

Summing these equations, we find the desired formula. □

Example 5.4.11. *This example is a continuation of Examples 5.4.2 and 5.4.7. Recall that* $\boldsymbol{\beta} = (1, 2, 2, 3, 0.5, 2, 2)^T$, $I = (2, 1, 4)$, $J = (3, 5, 6, 7)$, *and*

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

Since $y_{3,2} = 1$, *Proposition 5.4.8 implies that we may exchange the third column of the basic matrix* B *for the second column of* R *to obtain a new basic matrix*

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

To compute tab(B_*) *from tab*(B), *we follow Corollary 5.4.9 and row-reduce tab*(B) *to make column* $J(2) = 5$ *the third standard basis vector:*

$$tab(B_*) = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Since $\bar{e}_3^B = 1$, *Theorem 5.4.10 states that the difference between* $\text{cost}(B_*)$ *and* $\text{cost}(B)$ *is precisely* $\beta_5 + 4 - 2(4) = -3.5$. *This is verified by observing that* $\text{cost}(B_*) = (2, 1, 0.5)(0, 0, 1)^T = 0.5$ *and* $\text{cost}(B) = (2, 1, 3)(0, 1, 1)^T = 4$.

Theorem 5.4.10 may be restated as follows: if $y_{\ell,p} = 1$, then the net change in the cost of the basic solution imposed by exchanging $\mathbf{h}_{I(\ell)}$ for $\mathbf{h}_{J(p)}$ in the basis is 0 if

$\bar{e}_\ell^B = 0$, and the net change is $\beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p)$ if $\bar{e}_\ell^B = 1$. We note that the quantity $\beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p)$ is a function of $\mathbf{y}_p$ and can be calculated independently of ℓ. From this, we see that if a basic solution associated to B is optimal, then $\beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) \geq 0$ for all columns $\mathbf{y}_p$ of Y. This condition does not, however, classify all optimal basic matrices. For a necessary and sufficient condition, one needs to consider nonbasic columns many at a time, not just one at a time. This is made explicit in Theorem 5.4.13 below, whose proof requires Lemma 5.4.12.

Lemma 5.4.12. *Let B be any basic matrix for H, and let $\mathcal{S}$ be the set of solutions to $H\mathbf{e} = \mathbf{s}$. The map $\phi : \mathcal{S} \to \mathbb{F}_2^k$ given by $\phi(\mathbf{e}) = \mathbf{e}^R$ is a bijection.*

Proof. The inverse function ϕ^{-1} is given by taking $\mathbf{v} \in \mathbb{F}_2^k$ to the vector $\mathbf{e}$ such that $\mathbf{e}^B = B^{-1}(\mathbf{s} - R\mathbf{v})$ and $\mathbf{e}^R = \mathbf{v}$. □

Theorem 5.4.13 (compare to [22], Theorem 4)**.** *Let B be a basic matrix. The basic solution associated to B is an optimal solution if and only if*

$$2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R) \leq \text{cost}^B(\mathbf{e}^R) + \text{cost}^R(Y\mathbf{e}^R)$$

for all k-dimensional vectors $\mathbf{e}^R$.

Proof. Suppose $\mathbf{e}$ is a solution to $H\mathbf{e} = \mathbf{s}$. We have that $\text{cost}(\mathbf{e}) = \text{cost}^B(\mathbf{e}^B) + \text{cost}^R(\mathbf{e}^R)$. Since $\mathbf{s} = H\mathbf{e} = B\mathbf{e}^B + R\mathbf{e}^R$, we have $\bar{\mathbf{e}}^B = B^{-1}\mathbf{s} = \mathbf{e}^B + Y\mathbf{e}^R$, which can be rearranged to obtain $\mathbf{e}^B = \bar{\mathbf{e}}^B + Y\mathbf{e}^R$. Thus,

$$\text{cost}(\mathbf{e}) = \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(\bar{\mathbf{e}}^B) + \text{cost}^B(Y\mathbf{e}^R) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R). \tag{5.8}$$

Since $\text{cost}^B(\bar{\mathbf{e}}^B)$ is the cost of the basic solution associated to B, the basic solution

associated to B is optimal if and only if $\text{cost}^B(\bar{\mathbf{e}}^B) \leq \text{cost}(\mathbf{e})$ for all solutions to $H\mathbf{e} = \mathbf{s}$. Equation (5.8) implies that the basic solution associated to B is optimal if and only if $0 \leq \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(Y\mathbf{e}^R) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R)$, or $2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R) \leq \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(Y\mathbf{e}^R)$, for all solutions to $H\mathbf{e} = \mathbf{s}$. By Lemma 5.4.12, there is a bijective correspondence between solutions to $H\mathbf{e} = \mathbf{s}$ and the projected vectors $\mathbf{e}^R$. Since the condition $2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R) \leq \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(Y\mathbf{e}^R)$ only involves the non-basic coordinates $\mathbf{e}^R$ of $\mathbf{e}$, we conclude that the basic solution associated to B is optimal if and only if $2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R) \leq \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(Y\mathbf{e}^R)$ for all k-dimensional vectors $\mathbf{e}^R$. □

Theorem 5.4.13 motivates the following definition.

Definition 5.4.14 (see [22])**.** *Given a basic matrix B and a k-dimensional vector $\mathbf{e}^R$, the* gradient *of $\mathbf{e}^R$ is*

$$G^B(\mathbf{e}^R) := \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(Y\mathbf{e}^R) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R).$$

Proposition 5.4.15. *If $H\mathbf{e} = \mathbf{s}$, then for any basic matrix B we have* $\text{cost}(\mathbf{e}) = \text{cost}^B(\bar{\mathbf{e}}^B) + G^B(\mathbf{e}^R)$.

Proof. This is a restatement of Equation (5.8) in the notation of Definition 5.4.14. □

By Theorem 5.4.13, the basic solution associated with a basic matrix B is optimal if and only if the gradient of all k-dimensional vectors is positive. Let $\boldsymbol{\epsilon}_p$ denote the pth standard basis vector. In the case where $\mathbf{e}^R = \boldsymbol{\epsilon}_p$ for some $p \in \{1, 2, \ldots, k\}$, i.e., the Hamming weight of $\mathbf{e}^R$ is 1, we recover the necessary condition for optimality obtained earlier from Theorem 5.4.10: $\beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) \geq 0$. Since we will frequently only be concerned with the gradient of weight-one vectors, we define

the gradient [22] of a single column $\mathbf{y}_p$ of Y to be

$$g^B(\mathbf{y}_p) := G^B(\boldsymbol{\epsilon}_p) = \beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p).$$

Example 5.4.16. *This example is a continuation of Examples 5.4.2, 5.4.7, and 5.4.11. Suppose now that the vector of log-likelihoods is*

$$\boldsymbol{\lambda} = (1, 3, 2, 2, 2, -0.3, -0.2)^T.$$

As before, we take $I = (2, 1, 4)$ *and* $J = (3, 5, 6, 7)$*, so that*

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

and

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

We have $\boldsymbol{\beta} = (1, 3, 2, 2, 2, 0.3, 0.2)^T$*, so* $\boldsymbol{\beta}^B = (3, 1, 2)^T$ *and* $\boldsymbol{\beta}^R = (2, 2, 0.3, 0.2)^T$*. Also,* $\widehat{\mathbf{y}} = (0, 0, 0, 0, 0, 1, 1)^T$ *and* $\mathbf{s} = H\widehat{\mathbf{y}} = (1, 0, 0)^T$*. Multiplying* $\mathbf{s}$ *by* B^{-1}*, we get* $\bar{\mathbf{e}}^B = (0, 1, 0)^T$*.*

The gradient of the fourth column $\mathbf{y}_4$ *of* Y *is*

$$\begin{aligned} g^B(\mathbf{y}_4) &= \beta_{J(4)} + \text{cost}^B(\mathbf{y}_4) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_4) \\ &= \beta_7 + (3,1,2)\cdot(1,1,1)^T - 2(3,1,2)\cdot(0,1,0)^T \\ &= 0.2 + 6 - 2(1) \\ &= 4.2. \end{aligned}$$

In a similar manner one can compute that the gradients of the first, second, and third columns of Y *are* 4, 3, *and* 5.3 *respectively. Thus, there is no single column of* Y *that has a negative gradient.*

Let $\mathbf{e}^R = (0,0,1,1)^T$. *We have*

$$\begin{aligned} G^B(\mathbf{e}^R) &= \text{cost}^R(\mathbf{e}^R) + \text{cost}^B(Y\mathbf{e}^R) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}^R) \\ &= (2,2,0.3,0.2)\cdot(0,0,1,1)^T + (3,1,2)\cdot(0,1,0)^T - 2(3,1,2)\cdot(0,1,0)^T \\ &= 0.3 + 0.2 + 1 - 2 \\ &= -0.5. \end{aligned}$$

By Theorem 5.4.13, we conclude that the basic matrix B *is not optimal, despite the fact that no single column of* Y *has a negative gradient.*

Theorem 5.4.13 allows us to design an algorithm that is guaranteed to converge to a maximum-likelihood solution in a finite number of steps. Loosely speaking, this algorithm starts with some basic matrix and examines k-dimensional vectors until one of negative gradient is found. It then changes to a new basic matrix with lower cost, and it repeats this procedure until an optimal basic matrix is found. (This is analogous to Omura's first algorithm [22].) To formalize this algorithm, we first prove

Theorem 5.4.17. This result, which is implicit in [22], elaborates on how to change basic matrices once a vector of negative gradient is discovered.

Theorem 5.4.17. *Let B be a basic matrix with Y its associated matrix, and suppose that B is not optimal. Of all k-dimensional vectors $\mathbf{e}^R$ with $G^B(\mathbf{e}^R) < 0$, let $\mathbf{e}^R_*$ be one of smallest Hamming weight. Then there exists a basic matrix B_* that includes the nonbasic columns associated to the support of $\mathbf{e}^R_*$ and the basic columns associated to the support of $\bar{\mathbf{e}}^B + Y\mathbf{e}^R_*$. Moreover,* $\text{cost}(B_*) = \text{cost}(B) + G^B(\mathbf{e}^R_*) < \text{cost}(B)$.

Proof. By Lemma 5.4.12, there is a unique $(n-k)$-dimensional vector $\mathbf{e}^B_*$ such that $\mathbf{s} = B\mathbf{e}^B_* + R\mathbf{e}^R_*$, i.e. $\bar{\mathbf{e}}^B = \mathbf{e}^B_* + Y\mathbf{e}^R_*$. Rearranging, we obtain $\mathbf{e}^B_* = \bar{\mathbf{e}}^B + Y\mathbf{e}^R_*$. We deduce that the columns of H associated with the support of $\bar{\mathbf{e}}^B + Y\mathbf{e}^R_*$ are the same as those associated with the support of $\mathbf{e}^B_*$. Let A be the set of basic columns of H associated to $\mathbf{e}^B_*$, and let N be the set of nonbasic columns of H associated to $\mathbf{e}^R_*$.

We claim that $A \cup N$ is linearly independent. Suppose, for a contradiction, that there exists a linear dependence between vectors in $A \cup N$. Let M denote a non-empty subset of $A \cup N$ such that the vectors in M add to the zero vector. Note that the set A is linearly independent, since its elements come from the current basis. Thus, M must include at least one element of N. Let $\tilde{\mathbf{e}}^B_*$ and $\tilde{\mathbf{e}}^R_*$ denote indicator vectors for $A \setminus M$ and $N \setminus M$ respectively, and note that the Hamming weight of $\tilde{\mathbf{e}}^R_*$ is strictly less than the Hamming weight of $\mathbf{e}^R_*$.

Let $\mathbf{e}_*$ be the solution to $H\mathbf{e} = \mathbf{s}$ given by combining $\mathbf{e}^B_*$ and $\mathbf{e}^R_*$. Similarly, let $\tilde{\mathbf{e}}_*$ be the solution obtained by combining $\tilde{\mathbf{e}}^B_*$ and $\tilde{\mathbf{e}}^R_*$. Since $\text{supp}(\tilde{\mathbf{e}}_*) \subseteq \text{supp}(\mathbf{e}_*)$ and the coordinates of $\boldsymbol{\beta}$ are nonnegative, $\text{cost}(\tilde{\mathbf{e}}_*) \leq \text{cost}(\mathbf{e}_*)$. By Proposition 5.4.15, this implies $G^B(\tilde{\mathbf{e}}^R_*) \leq G^B(\mathbf{e}^R_*) < 0$, which contradicts the minimal weight assumption on $\mathbf{e}^R_*$. Thus $A \cup N$ is linearly independent, so we can augment this set by columns of B to obtain a basic matrix B_*.

Finally, we have

$$\begin{aligned}
\text{cost}(B_*) &= \text{cost}(\mathbf{e}_*) \\
&= \text{cost}^B(\mathbf{e}_*^B) + \text{cost}^R(\mathbf{e}_*^R) \\
&= \text{cost}^B(\bar{\mathbf{e}}^B + Y\mathbf{e}_*^R) + \text{cost}^R(\mathbf{e}_*^R) \\
&= \text{cost}^B(\bar{\mathbf{e}}^B) + \text{cost}^B(Y\mathbf{e}_*^R) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}_*^R) + \text{cost}^R(\mathbf{e}_*^R) \\
&= \text{cost}^B(\bar{\mathbf{e}}^B) + G^B(\mathbf{e}_*^R) \\
&= \text{cost}(B) + G^B(\mathbf{e}_*^R),
\end{aligned}$$

as was to be shown. □

5.4.3 Generalizations of Omura's Algorithms

Following [22], we now use the results of Section 5.4.2 to develop an iterative algorithm that is guaranteed to solve Problem (5.7) in a finite number of steps. Let B be any basic matrix of H. Systematically check the gradients of all k-dimensional vectors $\mathbf{e}^R$, first examining those of Hamming weight one, then those of weight two, and so on. If B is optimal, then by Theorem 5.4.13 the exhaustive search of all k-dimensional vectors will fail to produce one of negative gradient. If, however, we find a vector $\mathbf{e}_*^R$ with $G^B(\mathbf{e}_*^R) < 0$, then B is not optimal. Since the search of k-dimensional vectors was done in the order of increasing Hamming weight, we know that $\mathbf{e}_*^R$ has minimum Hamming weight among all vectors of negative gradient. By Theorem 5.4.17, there is a basic matrix B_* with $\text{cost}(B_*) = \text{cost}(B) + G^B(\mathbf{e}_*^R) < \text{cost}(B)$; one can find B_* by extending the set of columns given by $\text{supp}(\mathbf{e}_*^R) \cup \text{supp}(\bar{\mathbf{e}}^B + Y\mathbf{e}_*^R)$. Switch to this new basic matrix, and repeat the process.

In each iteration of the above algorithm, one of two things happens. In one

case, the algorithm finds that all k-dimensional vectors have non-negative gradient, concludes that the current basic matrix is optimal, and outputs an optimal solution to Problem (5.7). In the case where the current basic matrix is not optimal, the algorithm moves to a new basic matrix that yields a solution to $H\mathbf{e} = \mathbf{s}$ of strictly smaller cost. Since there are only a finite number of basic matrices for a given parity-check matrix, this algorithm must eventually find an optimal solution. The algorithm is presented formally below.

Algorithm 1: A generalization of Omura's algorithm.

Input: A parity-check matrix H, a basic matrix B, a syndrome $\mathbf{s}$, and a cost function $\boldsymbol{\beta}$.

Output: An optimal solution $\mathbf{e}_*$ to Problem (5.7).

Compute $\bar{\mathbf{e}}^B$ and Y.

Set $L = 1$.

while $L \leq k$ **do**

  if $G^B(\mathbf{e}^R) \geq 0$ *for all length* k*, Hamming weight* L *vectors* $\mathbf{e}^R$ **then**

    ▶ Set $L = L + 1$.

  else

    ▶ Choose a vector $\mathbf{e}_*^R$ that minimizes $G^B(\mathbf{e}^R)$ over all weight L vectors.

    ▶ Reset the basic matrix B to be a matrix that includes the columns associated to the nonzero components of $\mathbf{e}_*^R$ and of $\bar{\mathbf{e}}^B + Y\mathbf{e}_*^R$.

    ▶ Use this new basic matrix to recompute $\bar{\mathbf{e}}^B$ and Y.

    ▶Set $L = 1$.

  end

end

Output the basic solution $\mathbf{e}_*$ associated to the current basic matrix B.

As previously observed, Algorithm 1 will always find an optimal solution to Problem (5.7) in a finite number of steps. This finite number, however, could be very large,

given that the algorithm must examine all $2^k - 1$ nonzero k-dimensional vectors at least once before convergence. We therefore follow Omura's lead in reducing the complexity of the algorithm by bringing columns into the basis one at a time, rather than many at a time [22]. More specifically, we modify Algorithm 1 so as to only consider single columns of Y. If one of negative gradient is found, we bring it into the basis so as to reduce the cost of the current basic matrix (Theorem 5.4.10). If no single column of Y has a negative gradient, instead of moving on to consider higher-weight combinations of columns of Y we randomly select a pair (ℓ, p) such that $y_{\ell,p} = 1$ and $\bar{e}_\ell^B = 0$ and then exchange the ℓth basis vector for $\mathbf{h}_{J(p)}$. We refer to an operation of the latter type as a "dry pivot" on position (ℓ, p) of Y. By Theorem 5.4.10, the execution of a dry pivot has no effect on the cost of the current basis. Finally, if all columns of Y have nonnegative gradient and no dry pivot is possible, then the algorithm terminates and outputs the current basic solution.

Algorithm 2: A generalization of Omura's randomized algorithm.

Input: A parity-check matrix H, a basic matrix B, a syndrome $\mathbf{s}$, a cost function $\boldsymbol{\beta}$, and the maximum number of iterations $N_{\max}$.

Output: A low-cost solution $\mathbf{e}_*$ of $H\mathbf{e} = \mathbf{s}$.

for $N = 1$ **to** N_{max} **do**

Compute $\bar{\mathbf{e}}^B$ and Y.

if *there exists a column* $\mathbf{y}$ *of* Y *such that* $g^B(\mathbf{y}) < 0$ **then**

▶ Select a column $\mathbf{y}_p$ uniformly at random from those columns of Y that minimize $g^B(\mathbf{y})$.

▶ Select an index ℓ such that $\bar{e}^B_\ell = y_{\ell,p} = 1$ uniformly at random, and exchange the ℓth column of B for $\mathbf{h}_{J(p)}$.

else if *there exists an integer* ℓ *and an index* p *such that* $y_{\ell,p} = 1$ *and* $\bar{e}^B_\ell = 0$ **then**

▶ Find all columns $\mathbf{y}$ of Y such that there exists an integer ℓ that satisfies $y_\ell = 1$ and $\bar{e}^B_\ell = 0$.

▶ From these columns, select one $\mathbf{y}_p$ uniformly at random.

▶ Pick an index ℓ with $y_{\ell,p} = 1$ and $\bar{e}^B_\ell = 0$ uniformly at random.

▶ Exchange the ℓth column of B for $\mathbf{h}_{J(p)}$.

else

▶ Break

end

end

Output the basic solution $\mathbf{e}_*$ associated to the current basis matrix B.

A clear advantage that Algorithm 2 holds over Algorithm 1 is that the number of computations done between changes of basic matrices is drastically reduced. Also, by Corollary 5.4.9 the task of changing basic matrices for Algorithm 2 can be accomplished with a simple pivoting operation on a tableau. This gain in speed and

simplicity is balanced by the loss of assurance that the output of the algorithm is an optimal solution to Problem (5.7). This can be attributed to the fact that this algorithm only tests a necessary condition for optimality, i.e., only weight-one combinations of columns of Y instead of all possible combinations. Another issue to tackle is whether Algorithm 2 will ever prematurely enter the "Break" stage, i.e., whether the loop may be broken before $N_{\max}$ iterations and result in a suboptimal output.

In Section 5.5 we show that if the algorithm breaks the loop before $N_{\max}$ iterations have passed, then the output is an optimal solution to Problem (5.7). Moreover, we show that despite the lack of a hard guarantee of optimality, the probability of the algorithm returning an optimal solution to Problem (5.7) approaches one as the number of iterations $N_{\max}$ goes to infinity.

5.5 Analysis of Algorithm 2

Section 5.4 was dedicated to generalizing and clarifying the work of Omura [22]. Indeed, the work of [22] is completely recovered by specializing Section 5.4 to the BSC. In this section we depart from generalizing past work and present a novel analysis of Algorithm 2. To the author's knowledge, the work below constitutes the first precise explanation of why Omura-type algorithms perform well in simulations.

This section is organized as follows. We first prove two technical lemmas that, when combined in Theorem 5.5.3, show that there is a non-zero probability of Algorithm 2 moving to a basic solution of strictly lower cost. A detailed illustration of these results is provided by a continuation of the examples from Section 5.4. We then segue into viewing Algorithm 2 as a Markov chain whose state space is the set of basic matrices, and we show in Theorem 5.5.10 that the ergodic vertices of this Markov chain are precisely the optimal basic matrices. From this and the theory of

Markov chains, we conclude by proving Theorem 5.5.11, which states that the probability of Algorithm 2 returning a basic solution $\mathbf{e}_*$ corresponding to an ML codeword approaches 1 as the number of iterations goes to infinity.

5.5.1 Technical Lemmas

Lemma 5.5.1. *Let B be the current basis with Y its associated matrix. Suppose that B is not optimal, and let $\mathbf{e}_*^R$ have minimum Hamming weight among all k-dimensional vectors $\mathbf{e}^R$ satisfying $G^B(\mathbf{e}^R) < 0$. If the weight of $\mathbf{e}_*^R$ is at least 2, then there is an integer $\ell \in \{1, 2, \ldots, n-k\}$ and an index $p \in \mathrm{supp}(\mathbf{e}_*^R)$ such that*

(a) $\bar{e}_\ell^B = 0$,

(b) $y_{\ell,p} = 1$, *and*

(c) $\sum_{q \in \mathrm{supp}(\mathbf{e}_*^R) \setminus \{p\}} y_{\ell,q} = 1$.

Proof. As before, let $\boldsymbol{\epsilon}_p$ denote the pth standard basic vector of length k. Since the Hamming weight m of $\mathbf{e}_*^R$ is minimal and $m \geq 2$, we have that both $G^B(\boldsymbol{\epsilon}_p) \geq 0$ and $G^B(\mathbf{e}_*^R - \boldsymbol{\epsilon}_p) \geq 0$ for all $p \in \mathrm{supp}(\mathbf{e}_*^R)$. Since $\mathrm{cost}^R(\boldsymbol{\epsilon}_p) = \beta_{J(p)}$ and $Y\boldsymbol{\epsilon}_p = \mathbf{y}_p$, the condition $G^B(\boldsymbol{\epsilon}_p) \geq 0$ can be restated as

$$2\mathrm{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) \leq \beta_{J(p)} + \mathrm{cost}^B(\mathbf{y}_p). \tag{5.9}$$

Similarly, since $\mathrm{cost}^R(\mathbf{e}_*^R - \boldsymbol{\epsilon}_p) = \sum_{q \in \mathrm{supp}(\mathbf{e}_*^R) \setminus \{p\}} \beta_{J(q)} := \sum \beta_{J(q)}$ and $Y(\mathbf{e}_*^R - \boldsymbol{\epsilon}_p) = \sum_{q \in \mathrm{supp}(\mathbf{e}_*^R) \setminus \{p\}} \mathbf{y}_q := \sum \mathbf{y}_q$, the condition $G^B(\mathbf{e}_*^R - \boldsymbol{\epsilon}_p) \geq 0$ can be restated as

$$2\mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \sum \mathbf{y}_q\right) \leq \sum \beta_{J(q)} + \mathrm{cost}^B\left(\sum \mathbf{y}_q\right). \tag{5.10}$$

The rest of the proof will proceed as follows. We first show that there must exist a pair (ℓ, p) satisfying (a) and (b). Building on this preliminary existence result, we dig deeper to show that there must be a pair (ℓ, p) satisfying not only (a) and (b) but also (c). In each of these two parts, we start by assuming that no such pair (ℓ, p) exists. We then use Inequalities (5.9) and (5.10) along with the assumption that $G^B(\mathbf{e}_*^R) < 0$ to derive a contradiction.

We begin by showing that there is a pair (ℓ, p) that satisfies (a) and (b). Suppose for a contradiction that for all $\ell \in \{1, 2, \ldots, n-k\}$ and $p \in \text{supp}(\mathbf{e}_*^R)$, if $\bar{e}_\ell^B = 0$ then $y_{\ell,p} = 0$. This implies that $\text{supp}(\mathbf{y}_p) \subseteq \text{supp}(\bar{\mathbf{e}}^B)$ for all $p \in \text{supp}(\mathbf{e}_*^R)$. Fix $p \in \text{supp}(\mathbf{e}_*^R)$. Since $\text{supp}(\mathbf{y}_q) \subseteq \text{supp}(\bar{\mathbf{e}}^B)$ for all $q \in \text{supp}(\mathbf{e}_*^R) \setminus \{p\}$, we also have $\text{supp}(\sum \mathbf{y}_q) \subseteq \text{supp}(\bar{\mathbf{e}}^B)$. The inclusions $\text{supp}(\mathbf{y}_p) \subseteq \text{supp}(\bar{\mathbf{e}}^B)$ and $\text{supp}\left(\sum \mathbf{y}_q\right) \subseteq \text{supp}(\bar{\mathbf{e}}^B)$ imply that $\bar{\mathbf{e}}^B \odot \mathbf{y}_p = \mathbf{y}_p$ and $\bar{\mathbf{e}}^B \odot \left(\sum \mathbf{y}_q\right) = \sum \mathbf{y}_q$. These statements allow us to rewrite Inequalities (5.9) and (5.10) as

$$2\text{cost}^B(\mathbf{y}_p) \leq \beta_{J(p)} + \text{cost}^B(\mathbf{y}_p)$$

and

$$2\text{cost}^B\left(\sum \mathbf{y}_q\right) \leq \sum \beta_{J(q)} + \text{cost}^B\left(\sum \mathbf{y}_q\right).$$

Performing the obvious cancellations yields

$$\text{cost}^B(\mathbf{y}_p) \leq \beta_{J(p)}$$

and

$$\text{cost}^B\left(\sum \mathbf{y}_q\right) \leq \sum \beta_{J(q)}.$$

Summing these inequalities, we obtain

$$\text{cost}^B(\mathbf{y}_p) + \text{cost}^B\left(\sum \mathbf{y}_q\right) \leq \beta_{J(p)} + \sum \beta_{J(q)}. \tag{5.11}$$

By assumption, $G^B(\mathbf{e}_*^R) < 0$. This is equivalent to

$$2\text{cost}^B(\bar{\mathbf{e}}^B \odot Y\mathbf{e}_*^R) > \text{cost}^R(\mathbf{e}_*^R) + \text{cost}^B(Y\mathbf{e}_*^R). \tag{5.12}$$

Since $Y\mathbf{e}_*^R = \mathbf{y}_p + \sum \mathbf{y}_q$ and each of $\text{supp}(\mathbf{y}_p)$ and $\text{supp}(\sum \mathbf{y}_q)$ is a subset of $\text{supp}(\bar{\mathbf{e}}^B)$, we have $\bar{\mathbf{e}}^B \odot Y\mathbf{e}_*^R = \mathbf{y}_p + \sum \mathbf{y}_q$. Also, note that $\text{cost}^R(\mathbf{e}_*^R) = \beta_{J(p)} + \sum \beta_{J(q)}$. This allows us to simplify (5.12) to

$$\text{cost}^B\left(\mathbf{y}_p + \sum \mathbf{y}_q\right) > \beta_{J(p)} + \sum \beta_{J(q)}. \tag{5.13}$$

Since

$$\text{cost}^B\left(\mathbf{y}_p + \sum \mathbf{y}_q\right) \leq \text{cost}^B\left(\mathbf{y}_p\right) + \text{cost}^B\left(\sum \mathbf{y}_q\right), \tag{5.14}$$

we may combine (5.11), (5.13), and (5.14) to obtain

$$\text{cost}^B\left(\mathbf{y}_p\right) + \text{cost}^B\left(\sum \mathbf{y}_q\right) < \text{cost}^B\left(\mathbf{y}_p\right) + \text{cost}^B\left(\sum \mathbf{y}_q\right),$$

which is absurd.

We conclude from the preceding contradiction that there must exist some integer ℓ and some index $p \in \text{supp}(\mathbf{e}_*^R)$ such that $\bar{e}_\ell^B = 0$ and $y_{\ell,p} = 1$, i.e., that (a) and (b) hold. To complete the proof, we must show that there exists such a pair with the additional property that $\sum y_{\ell,q} = 1$. To that end, we make the following claim. Let $p \in \text{supp}(\mathbf{e}_*^R)$ be such that there exists some integer ℓ with $\bar{e}_\ell^B = 0$ and $y_{\ell,p} = 1$. Then there exists an integer ℓ' satisfying $\bar{e}_{\ell'}^B = 0$, $y_{\ell',p} = 1$, and $\sum y_{\ell',q} = 1$. Note that we

are actually proving something stronger than we need: for all p such that there exists an integer ℓ satisfying (a) and (b), there exists an ℓ' satisfying (c) as well.

We prove this claim by contradiction. Fix $p \in \mathrm{supp}(\mathbf{e}_*^R)$ such that there exists some integer ℓ with $\bar{e}_\ell^B = 0$ and $y_{\ell,p} = 1$. Assume that for any ℓ' such that $\bar{e}_{\ell'}^B = 0$ and $y_{\ell',p} = 1$, we also have $\sum y_{\ell',q} = 0$. Setting $\mathcal{Z} = \{1, 2, \ldots, n-k\} \setminus \mathrm{supp}(\bar{\mathbf{e}}^B)$, this implies that $\mathcal{Z} \cap \mathrm{supp}(\mathbf{y}_p) \cap \mathrm{supp}\left(\sum \mathbf{y}_q\right) = \emptyset$. In the following, we let $\mathrm{cost}^{\mathcal{Z}}$ denote the restriction of the cost function cost^B to those coordinates indicated by $\mathcal{Z}$.

The assumption that $G^B(\mathbf{e}_*^R) < 0$ implies

$$\begin{aligned} 2\mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \left(\mathbf{y}_p + \sum \mathbf{y}_q\right)\right) &> \beta_{J(p)} + \sum \beta_{J(q)} + \mathrm{cost}^B\left(\mathbf{y}_p + \sum \mathbf{y}_q\right) \\ &= \beta_{J(p)} + \sum \beta_{J(q)} \\ &\quad + \mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \left(\mathbf{y}_p + \sum \mathbf{y}_q\right)\right) + \mathrm{cost}^{\mathcal{Z}}\left(\mathbf{y}_p + \sum \mathbf{y}_q\right). \end{aligned}$$

Canceling, we get

$$\mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \left(\mathbf{y}_p + \sum \mathbf{y}_q\right)\right) > \beta_{J(p)} + \sum \beta_{J(q)} + \mathrm{cost}^{\mathcal{Z}}\left(\mathbf{y}_p + \sum \mathbf{y}_q\right). \tag{5.15}$$

Because the supports of $\mathbf{y}_p$ and $\sum \mathbf{y}_q$ on $\mathcal{Z}$ are disjoint, we have $\mathrm{cost}^{\mathcal{Z}}\left(\mathbf{y}_p + \sum \mathbf{y}_q\right) = \mathrm{cost}^{\mathcal{Z}}\left(\mathbf{y}_p\right) + \mathrm{cost}^{\mathcal{Z}}\left(\sum \mathbf{y}_q\right)$. Since

$$\mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \left(\mathbf{y}_p + \sum \mathbf{y}_q\right)\right) \leq \mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \mathbf{y}_p\right) + \mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \sum \mathbf{y}_q\right),$$

we obtain from (5.15) that

$$\begin{aligned} \mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \mathbf{y}_p\right) + \mathrm{cost}^B\left(\bar{\mathbf{e}}^B \odot \sum \mathbf{y}_q\right) &> \beta_{J(p)} + \sum \beta_{J(q)} \\ &\quad + \mathrm{cost}^{\mathcal{Z}}\left(\mathbf{y}_p\right) + \mathrm{cost}^{\mathcal{Z}}\left(\sum \mathbf{y}_q\right). \end{aligned} \tag{5.16}$$

Since the inequalities of (5.9) and (5.10) hold, we also have

$$\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) \leq \beta_{J(p)} + \text{cost}^{\mathcal{Z}}(\mathbf{y}_p)$$

and

$$\text{cost}^B\left(\bar{\mathbf{e}}^B \odot \sum \mathbf{y}_q\right) \leq \sum \beta_{J(q)} + \text{cost}^{\mathcal{Z}}\left(\sum \mathbf{y}_q\right).$$

Summing, we get

$$\begin{aligned}\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p) + \text{cost}^B\left(\bar{\mathbf{e}}^B \odot \sum \mathbf{y}_q\right) \leq \beta_{J(p)} + \sum \beta_{J(q)} \\ + \text{cost}^{\mathcal{Z}}(\mathbf{y}_p) + \text{cost}^{\mathcal{Z}}\left(\sum \mathbf{y}_q\right).\end{aligned} \tag{5.17}$$

Combining (5.16) with (5.17), we obtain the desired contradiction. □

Lemma 5.5.2. *Let B be the current basic matrix with J indicating the nonbasic columns with respect to B. Let $\mathbf{e}^R$ be a k-dimensional vector of Hamming weight at least 2, and suppose there exists an integer $\ell \in \{1, 2, \ldots, n-k\}$ and an index $p \in \text{supp}(\mathbf{e}^R)$ such that $\bar{e}^B_\ell = 0$ and $y_{\ell,p} = \sum_{q \in \text{supp}(\mathbf{e}^R)\setminus\{p\}} y_{\ell,q} = 1$. If B' is the basic matrix obtained from B by setting the ℓth column equal to $\mathbf{h}_{J(p)}$, then $G^{B'}(\mathbf{e}^R - \boldsymbol{\epsilon}_p) = G^B(\mathbf{e}^R)$, where, as usual, $\boldsymbol{\epsilon}_p$ denotes the pth standard basis vector.*

Proof. Let Y and Y' be the "Y" matrices associated with B and B' respectively. By Corollary 5.4.9, we know that $Y' = AY_p$, where A is the matrix that row reduces $\mathbf{y}_p$ to $\boldsymbol{\epsilon}_\ell$ and Y_p is the matrix obtained from Y by replacing the pth column with $\boldsymbol{\epsilon}_\ell$.

By definition,

$$\begin{aligned} G^{B'}(\mathbf{e}^R - \boldsymbol{\epsilon}_p) &= \text{cost}^{R'}(\mathbf{e}^R - \boldsymbol{\epsilon}_p) \\ &\quad + \text{cost}^{B'}\left(Y'(\mathbf{e}^R - \boldsymbol{\epsilon}_p)\right) \\ &\quad - 2\text{cost}^{B'}\left(\bar{\mathbf{e}}^{B'} \odot Y'(\mathbf{e}^R - \boldsymbol{\epsilon}_p)\right). \end{aligned} \tag{5.18}$$

We now examine each of these three terms. Since J', the vector of nonbasic indices relative to B', is identical to J except in the pth position, we have that $\text{cost}^{R_2}(\mathbf{e}^R - \boldsymbol{\epsilon}_p) = \sum \beta_{J(q)}$ (as in the proof of Lemma 5.5.1, we suppress the index set $\text{supp}(\mathbf{e}^R) \setminus \{p\}$ of the summation).

We have

$$\begin{aligned} Y'(\mathbf{e}^R - \boldsymbol{\epsilon}_p) &= A\big(Y_p(\mathbf{e}^R - \boldsymbol{\epsilon}_p)\big) \\ &= A\left(\sum \mathbf{y}_q\right) \\ &= \left(\sum \mathbf{y}_q\right) + \mathbf{y}_p + \boldsymbol{\epsilon}_\ell \\ &= Y\mathbf{e}^R + \boldsymbol{\epsilon}_\ell. \end{aligned}$$

The penultimate equality can be verified by recalling that $y_{\ell,p} = \sum y_{\ell,q} = 1$ and that A is the matrix that row reduces $\mathbf{y}_p$ to $\boldsymbol{\epsilon}_\ell$: the action of A on $\sum \mathbf{y}_q$ is recovered by first adding $\mathbf{y}_p$ to $\sum \mathbf{y}_q$ and then placing a 1 back into the pivot position. The equality above yields

$$\begin{aligned} \text{cost}^{B'}\left(Y'(\mathbf{e}^R - \boldsymbol{\epsilon}_p)\right) &= \text{cost}^{B'}(Y\mathbf{e}^R + \boldsymbol{\epsilon}_\ell) \\ &= \text{cost}^{B'}(Y\mathbf{e}^R) + \text{cost}^{B'}(\boldsymbol{\epsilon}_\ell) \\ &= \text{cost}^{B}(Y\mathbf{e}^R) + \beta_{J(p)}. \end{aligned}$$

By Corollary 5.4.9, we have that $\bar{\mathbf{e}}^{B'} = \bar{\mathbf{e}}^{B}$, since $\bar{e}^{B}_{\ell} = 0$. Thus,

$$\begin{aligned}
\text{cost}^{B'}\left(\bar{\mathbf{e}}^{B'} \odot Y'(\mathbf{e}^R - \boldsymbol{\epsilon}_p)\right) &= \text{cost}^{B'}\left(\bar{\mathbf{e}}^{B} \odot (Y\mathbf{e}^R + \boldsymbol{\epsilon}_\ell)\right) \\
&= \text{cost}^{B'}(\bar{\mathbf{e}}^{B} \odot Y\mathbf{e}^R + \bar{\mathbf{e}}^{B} \odot \boldsymbol{\epsilon}_\ell) \\
&= \text{cost}^{B'}(\bar{\mathbf{e}}^{B} \odot Y\mathbf{e}^R + \mathbf{0}) \\
&= \text{cost}^{B'}(\bar{\mathbf{e}}^{B} \odot Y\mathbf{e}^R) \\
&= \text{cost}^{B}(\bar{\mathbf{e}}^{B} \odot Y\mathbf{e}^R)
\end{aligned}$$

where the final equality holds because B and B' are identical except in the ℓth column. Substituting the results from the last three paragraphs into Equation (5.18) shows that $G^{B'}(\mathbf{e}^R - \boldsymbol{\epsilon}_p) = G^{B}(\mathbf{e}^R)$. □

Theorem 5.5.3. *Let B be the current $(n-k)\times(n-k)$ basic matrix, and suppose that B is not optimal. Then there is a non-zero probability of Algorithm 2 moving to a basic matrix of cost strictly lower than B in at most k iterations.*

Proof. Since the basic solution associated to B is not optimal, Theorem 5.4.13 states that there is at least one binary k-dimensional vector $\mathbf{e}^R$ such that $G^B(\mathbf{e}^R) < 0$. Of those k-dimensional vectors with negative gradient, consider those of smallest Hamming weight; call this weight m. From these weight-m vectors with negative gradient, choose $\mathbf{e}^R_*$ to have the least gradient. We will show by induction on m that there is a non-zero probability that Algorithm 2 will move from B to a basic matrix of lower cost in at most m iterations. Since $m \leq k$, this will complete the proof.

Let Y be the matrix associated with B, and let J denote the nonbasic columns with respect to B. If $m = 1$, then $\mathbf{e}^R_* = \boldsymbol{\epsilon}_p$ for some p and we have

$$0 > G^B(\mathbf{e}^R_*) = g^B(\mathbf{y}_p) = \beta_{J(p)} + \text{cost}^B(\mathbf{y}_p) - 2\text{cost}^B(\bar{\mathbf{e}}^B \odot \mathbf{y}_p).$$

Hence, $\operatorname{supp}(\bar{\mathbf{e}}^B \odot \mathbf{y}_p)$ is not empty, and so there is an ℓ such that $\bar{e}^B_\ell = y_{\ell,p} = 1$. Thus, there is a non-zero probability that Algorithm 2 will choose to include $\mathbf{h}_{J(p)}$ in the basis while excluding $\mathbf{h}_{I(\ell)}$, which by Theorem 5.4.10 will lower the cost of the current basis by a positive amount.

Now suppose that $m > 1$. By Lemma 5.5.1, there is an integer ℓ and an index $p \in \operatorname{supp}(\mathbf{e}^R_*)$ such that $\bar{e}^B_\ell = 0$ and $y_{\ell,p} = 1 = \sum y_{\ell,q}$, where as before we write $\sum \mathbf{y}_q$ for $\sum_{q \in \operatorname{supp}(\mathbf{e}^R_*) \setminus \{p\}} \mathbf{y}_q$. Algorithm 2 has a non-zero probability of exchanging the ℓth column of B for $\mathbf{h}_{J(p)}$ to obtain a new basis matrix B'. The vector $\mathbf{e}^R_* - \boldsymbol{\epsilon}_p$ has Hamming weight strictly less than m, and $G^{B'}(\mathbf{e}^R_* - \boldsymbol{\epsilon}_p) < 0$ by Lemma 5.5.2. By induction there is a non-zero probability of Algorithm 2 moving from B' to a basis with strictly lower cost in no more than $m - 1$ iterations. Thus, there is a non-zero probability of Algorithm 2 moving from B to a basis of strictly lower cost in no more than m iterations. □

Example 5.5.4. *This example is a continuation of Examples 5.4.2, 5.4.7, 5.4.11, and 5.4.16. Again let* $\boldsymbol{\lambda} = (1, 3, 2, 2, 2, -0.3, -0.2)^T$, $I = (2, 1, 4)$, *and* $J = (3, 5, 6, 7)$.

Consider the following array:

$$\begin{array}{c} 1 \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccccccc} * & * & * & 4 & * & 3 & 5.3 & 4.2 \end{array} \\ \left[\begin{array}{cccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right] \end{array}.$$

The lower-right-hand portion of this array is tab(B). *The first row indicates the gradients of the corresponding columns of* Y, *and the first column indicates the costs of the basic columns. The entry in the upper left-hand corner indicates the cost of the current basic solution.*

It was shown in Example 5.4.16 that the binary vector $\mathbf{e}^R = (0,0,1,1)^T$ *satisfies* $G^B(\mathbf{e}^R) = -0.5$. *Lemma 5.5.1 implies that there is a pair* (ℓ, p) *such that* $\bar{e}^B_\ell = 0$ *and* $y_{\ell,p} = \sum y_{\ell,q} = 1$. *The pair* $\ell = 1$, $p = 4$ *fits the bill.*

The probability of Algorithm 2 exchanging the first column of B *for the fourth column of* Y *is* $\frac{1}{6}\cdot\frac{1}{2} = \frac{1}{12}$. *Supposing that it chooses to make this particular exchange, the array above will be updated by first performing elementary row operations on* $tab(B)$ *so as to make column* $J(4) = 7$ *the first standard basis vector (via Corollary 5.4.9) and then updating the first row and column of the big array:*

$$\begin{array}{c} 1 \\ 0.2 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccccccc} * & * & 4.2 & 4.2 & * & 3 & -0.5 & * \end{array} \\ \left[\begin{array}{cccccccc} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}.$$

The cost of the current basic solution has not changed, but a column of negative gradient has been "revealed." Indeed, letting B_* *denote this new basis, we have* $G^{B_*}((0,0,1,0)^T) = G^B((0,0,1,1)^T) = -0.5$, *as guaranteed by Lemma 5.5.2. Algorithm 2 will immediately exchange the second column of the current basis for the sixth column of* H *to obtain*

$$\begin{array}{c} 0.5 \\ 0.2 \\ 0.3 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccccccc} * & 0.5 & 4.7 & 3.8 & * & 3.5 & * & * \end{array} \\ \left[\begin{array}{cccccccc} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}.$$

In fact, the basic solution $\mathbf{e} = (0,0,0,0,0,1,1)^T$ *associated with this final array is the optimal solution to Problem (5.7).*

5.5.2 Modeling Algorithm 2 as a Markov Chain

At each iteration, Algorithm 2 jointly selects one basic column and one non-basic column, where the selection is made randomly from a set of permissible candidate pairs. The criterion for selection ensures that the algorithm can exchange the non-basic column for the basic column to obtain a new basic matrix for H. The key observation we make is that the probability of augmenting the current basic matrix in a given manner depends *only* upon the current basic matrix and is independent of the current iteration number. In other words, Algorithm 2 can be accurately modeled as a Markov chain.

Any finite-state Markov chain is completely specified by its *transition digraph* [23], a directed graph whose vertices are the states of the Markov chain and whose edges are weighted by the probability of moving from one state to another. In order to describe the transition digraph of Algorithm 2, we define an equivalence relation $\sim$ between basic matrices so that $B_1 \sim B_2$ if and only if one can be obtained from the other by a permutation of columns. Let V be the collection of all basic matrices of H under this equivalence relation, and let D be the digraph whose vertex set is V. Place a directed edge from B_1 to B_2 in D and write $B_1 \rightarrow B_2$ if and only if Algorithm 2 initialized at B_1 has a non-zero probability of moving from B_1 to B_2 in a single iteration. For basic matrices causing Algorithm 2 to prematurely enter the "Break" stage, we add a self-loop whose associated probability is 1. Labeling each directed edge with its associated probability, we see that D is the transition digraph for Algorithm 2.

As was suggested by the preceding paragraph, the advantage of viewing of Algorithm 2 as a Markov chain is that it allows us to succinctly discuss the behavior of the algorithm in the context of graphs. To this end, we introduce some terminology

and notation from graph theory and Markov chain theory, material which is largely adapted from [23]. Let G be a digraph, and let v_1, v_2 be two vertices of G. We write $v_1 \rightsquigarrow v_2$ if there is a directed path in G from v_1 to v_2. Define an equivalence relation $v_1 \sim v_2$ if and only $v_1 \rightsquigarrow v_2$ and $v_2 \rightsquigarrow v_1$. An equivalence class under $\sim$ is called a *strongly connected component* of G.

A set S of vertices is *closed* provided that, whenever $v \in S$ and $v \rightsquigarrow w$, we have $w \in S$. Said colloquially, once you are in a closed set you stay inside. We define the *closure* of S to be

$$\overline{S} := \{v \in V(G) \ : \ \text{there exists an } s \in S \text{ such that } s \rightsquigarrow v\}.$$

One can check that $\overline{S}$ is always closed and that $S \subseteq \overline{S}$ with equality if and only if S is closed. Also, if $S \subseteq Q$, then $\overline{S} \subseteq \overline{Q}$. An *ergodic set* is a minimal closed set, and an *ergodic vertex* is an element of an ergodic set. With these definitions, one can see that a set S is ergodic if and only if S is closed and constitutes an entire strongly connected component of G. Also, a vertex v is ergodic if and only if its strongly connected component $[v]$ is closed. The importance of ergodic vertices is summarized in the following theorem.

Theorem 5.5.5 ([23], Theorem 5.4)**.** *Let M be a homogenous Markov chain with finitely many states, let G be its transition digraph, and let v be a vertex of G. If M is initialized at v and run for t steps, then the probability of the process terminating at an ergodic vertex of G approaches 1 as t goes to infinity.*

We now explore the nature of the digraph D with the goal of showing that Algorithm 2 converges to an optimal solution with high probability. We first prove some results on what strongly connected components of D must look like. These results,

along with Theorem 5.5.3, are combined in Theorem 5.5.10 to show that the set of optimal basic matrices of H and the set of ergodic vertices of D coincide. Using this characterization in tandem with Theorem 5.5.5 we obtain Theorem 5.5.11, which is the main result of this chapter.

Lemma 5.5.6. *If B and B' are two basic matrices in the same strongly connected component of D, then* $\text{cost}(B) = \text{cost}(B')$.

Proof. The basic matrices B and B' are in the same strongly connected component of D if and only if there is a non-zero probability that Algorithm 2 initialized at B will move to B' in a finite number of iterations, and vice-versa. From iteration to iteration, the cost of the current basic solution can never increase. Thus, $\text{cost}(B) \geq \text{cost}(B') \geq \text{cost}(B)$, and so we have equality. □

Lemma 5.5.7. *Let B be a basic matrix with Y its associated matrix. If there exists a column of Y that has negative gradient, then the strongly connected component of B in D is $\{B\}$.*

Proof. Since there is a column of Y that has negative gradient, Algorithm 2 will always choose to move from B to a basic matrix of strictly lower cost. Thus for any B_* with $B \rightsquigarrow B_*$, we must have that $B_* \notin [B]$ by Lemma 5.5.6. □

Proposition 5.5.8. *Let B and B' be two basic matrices, and let Y and Y' be their associated matrices. Suppose that* $\text{cost}^B(\mathbf{y}) \geq 0$ *for all columns* $\mathbf{y}$ *of Y and that* $\text{cost}^{B'}(\mathbf{y}') \geq 0$ *for all columns* $\mathbf{y}'$ *of Y'. If $B \to B'$, then $B' \to B$.*

Proof. Since $\text{cost}^B(\mathbf{y}) \geq 0$ for all columns $\mathbf{y}$ of Y and $B \to B'$, it must be that there is an integer ℓ and an index p such that Algorithm 2 can move from B to B' through a dry pivot on position (ℓ, p) of Y. This implies that $\bar{e}_\ell^B$ and $\bar{e}_\ell^{B'}$ must both be equal to 0. By Corollary 5.4.9, we know that $Y' = AY_p$, where A is the matrix that row

reduces $\mathbf{y}_p$ to $\boldsymbol{\epsilon}_\ell$ and Y_p is the matrix obtained from Y by replacing the pth column with $\boldsymbol{\epsilon}_\ell$. Thus the pth columns of Y and Y' are identical. In particular, position (ℓ, p) of Y' is equal to 1. Combining this with the facts that $\bar{e}_\ell^{B'} = 0$ and that no column of Y' has negative gradient, we see that Algorithm 2 initialized at B' has a non-zero probability of moving to B. Thus, $B' \to B$. □

Corollary 5.5.9. *Let B and B' be two distinct basic matrices in the same strongly connected component of D. If $B \to B'$, then $B' \to B$.*

Proof. Since $[B]$ contains at least two elements, Lemma 5.5.7 implies that all columns of Y must have non-negative gradient. A symmetric argument implies that all columns of Y' must have non-negative gradient. The result follows from Proposition 5.5.8. □

Theorem 5.5.10. *A vertex B of D is ergodic if and only if it is an optimal basic matrix.*

Proof. Suppose that B is optimal. To show that B is ergodic, we will show that its strongly connected component $[B]$ is closed. Let $B_1 \in [B]$ be given, and let B_2 be such that $B_1 \to B_2$ in D. Since Algorithm 2 always moves to a basic matrix whose cost is less than or equal to that of the current basic matrix, we have that B_1 and B_2 are optimal as well. This optimality allows us to apply Proposition 5.5.8, so we have $B_2 \to B_1$. Thus, $[B]$ is closed, so B is ergodic.

Now suppose that B is not optimal. By Theorem 5.5.3, there is a non-zero probability of Algorithm 2 moving from B to a basic matrix B_* of strictly smaller cost. This means that $B \rightsquigarrow B_*$ in D. By Lemma 5.5.6, it must be that B and B_* belong to different strongly connected components, so $B_* \not\rightsquigarrow B$. It follows that the strongly connected component of B is not closed, so B is not ergodic. □

Theorem 5.5.11. *Let B be any basic matrix of H. If Algorithm 2 is initialized at B and allowed to perform n iterations, then the probability that the output yields a maximum-likelihood codeword approaches* 1 *as $n \to \infty$.*

Proof. Initializing Algorithm 2 at B and allowing it to run for n iterations is equivalent to initializing the Markov chain whose transition digraph is D at B and allowing it to run for n steps. Theorem 5.5.5 states that the probability of this process terminating at an ergodic vertex of D approaches 1 as $n \to \infty$. By Theorem 5.5.10, any ergodic vertex gives an optimal solution to Problem (5.7) and therefore yields an ML codeword. □

5.6 Simulation Results

The word-error rate P_w of the generalized Omura decoder using Algorithm 2 was simulated for three binary linear codes: a $[35, 9]$ low-density parity-check code, a randomly generated $[200, 160]$ code, and the $[32, 16, 8]$ Reed-Muller code. Each of these three codes was simulated on the BSC and on the AWGN channel, and the Reed-Muller code was also simulated on the binary asymmetric channel and on the Z-channel. The output of the generalized Omura decoder was sampled at various iterations to see how the number of iterations affected performance. As predicted by Theorem 5.5.11, allowing more iterations only improves performance. When feasible, ML decoding was also performed to see if the generalized Omura decoder did indeed approach ML performance with increased iterations. We observe that in all such examples ML decoding was steadily approximated by the generalized Omura decoder.

5.6.1 A [35,9] LDPC Code

Figure 5.1 shows the simulated word-error rates of the $[35, 9]$ turbo-based low-density parity-check code of [1] and [3] on the binary symmetric channel. Both generalized Omura decoding and ML decoding were performed, with the output of the generalized Omura decoder sampled at 10, 20, 30, and 80 iterations. The performance of the generalized Omura decoder at 10 iterations differs from ML performance by a wide margin, but as the number of iterations is increased performance does likewise. At 80 iterations, the word-error rate of the generalized Omura decoder is almost identical to that of the ML decoder.

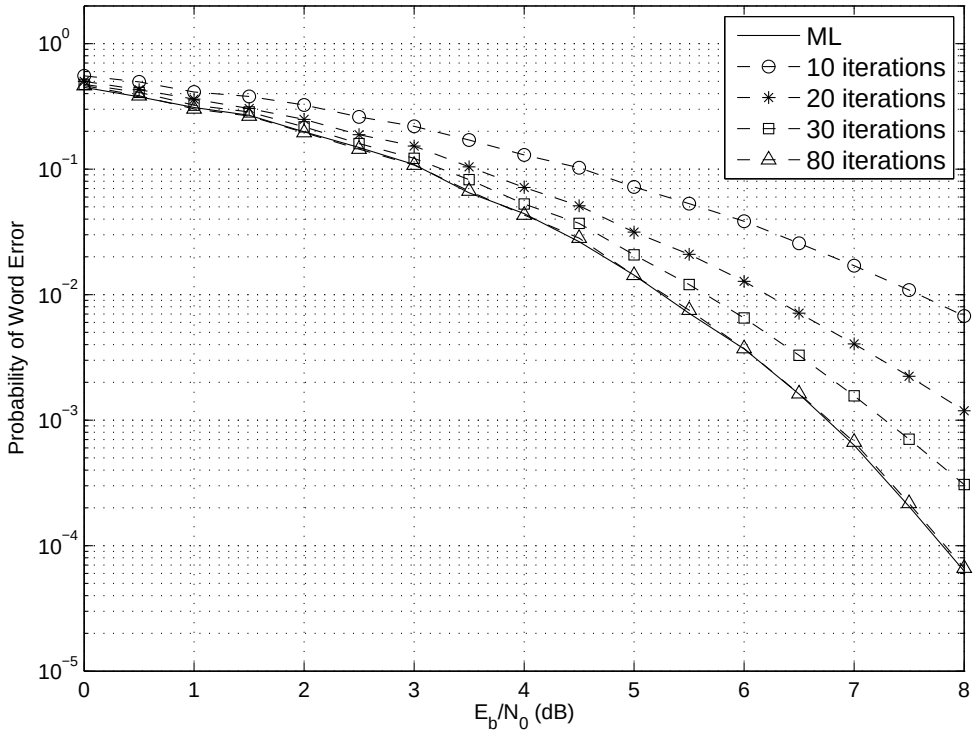


Figure 5.1: P_w for a $[35, 9]$ turbo-based LDPC code on the BSC.

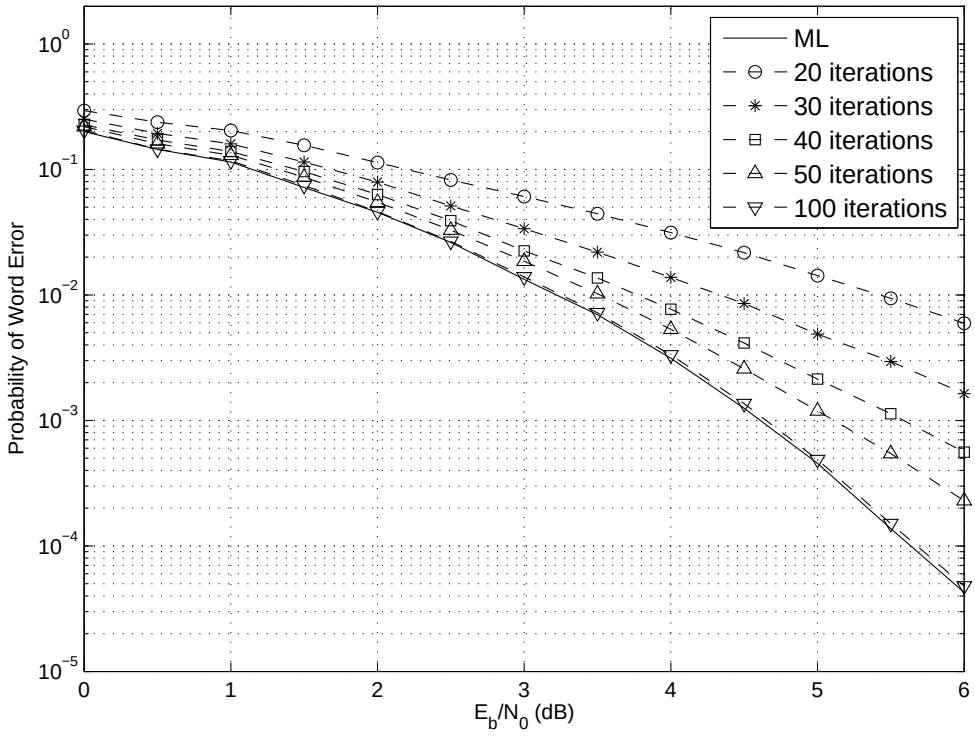


Figure 5.2: P_w for a $[35, 9]$ turbo-based LDPC code on the AWGN channel.

Figure 5.2 shows the same $[35, 9]$ code on the additive white Gaussian noise channel. The performance curve of the generalized Omura decoder sampled at 20 iterations roughly follows the curve of the ML decoder, but the gap between word-error rates grows along with the signal-to-noise ratio (SNR). Nonetheless, we observe that as the number of iterations is increased, the corresponding performance curves converge to ML performance.

5.6.2 A Randomly Generated $[200, 160]$ Code

Since the generalized Omura decoder was designed for use on any binary linear code, its performance was tested on a randomly generated code of moderate length. More specifically, a 40×200 binary parity-check matrix H was generated at random, and it was verified that the resulting code had dimension 160. All simulations in this subsection were done with respect to the fixed parity-check matrix H. Due to the prohibitive computational demands made by ML decoding at this length and dimension, we include results only for generalized Omura decoding.

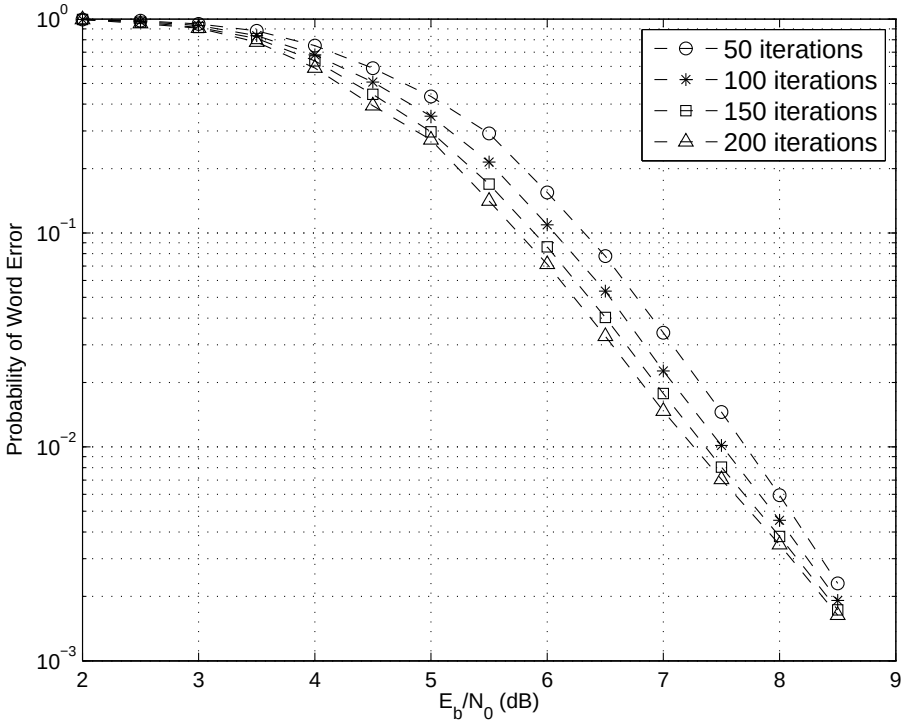


Figure 5.3: P_w for a randomly generated $[200, 160]$ code on the BSC.

The random $[200, 160]$ code was first simulated on the binary symmetric channel; the results are shown in Figure 5.3. As was the case in Section 5.6.1, the performance

curves are stacked neatly on top of one another, ordered according to the number of iterations. While the performance curves tend to diverge from one another at low to moderate SNRs, as they did in Section 5.6.1, they begin to converge together again at higher SNRs. This behavior is especially curious given that the same phenomenon is not observed for the same code on the AWGN channel (see Figure 5.4).

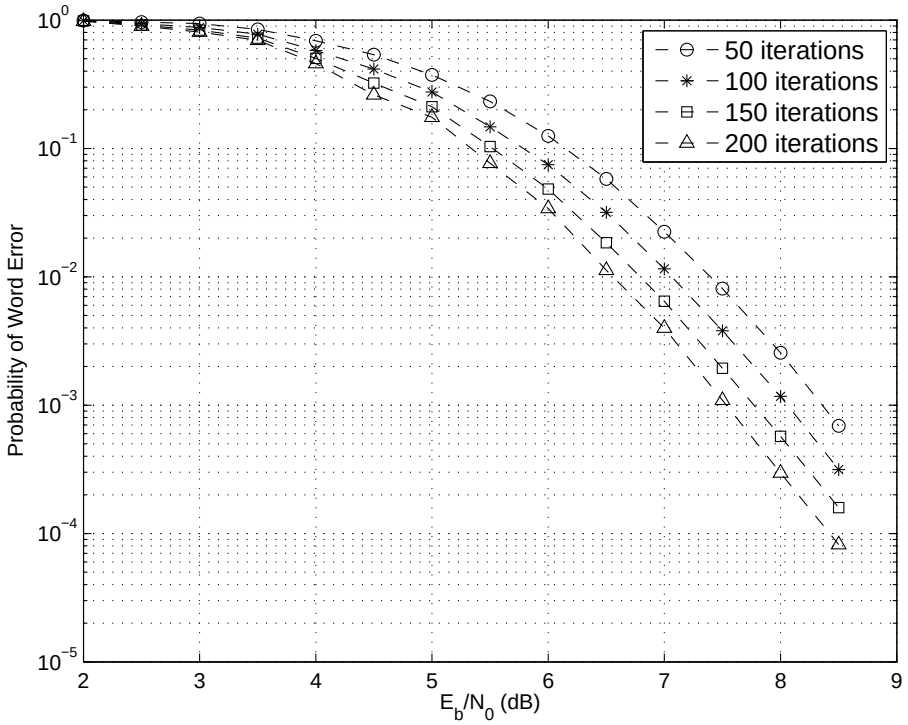


Figure 5.4: P_w for a randomly generated $[200, 160]$ code on the AWGN channel.

Figure 5.4 shows the word-error rates of the $[200, 160]$ code on the additive white Gaussian noise channel. There is no indication of the performance curves piling up next to each other; indeed, there seems to be a fairly uniform increase in performance for every additional 50 iterations. This seems to indicate that at 200 iterations the generalized Omura decoder is still a good distance away from achieving ML performance.

5.6.3 The $[32, 16, 8]$ Reed-Muller Code

The $[32, 16, 8]$ second order Reed-Muller Code (see, e.g., [17, pages 33-36]) was simulated on several channels. Figure 5.5 displays the data from a simulation on the BSC. The generalized Omura decoder sampled at 100 iterations achieves practically optimal word-error rates on the BSC. By comparison, we see from Figure 5.6 that on the AWGN channel 100 iterations are not enough to converge to ML performance.

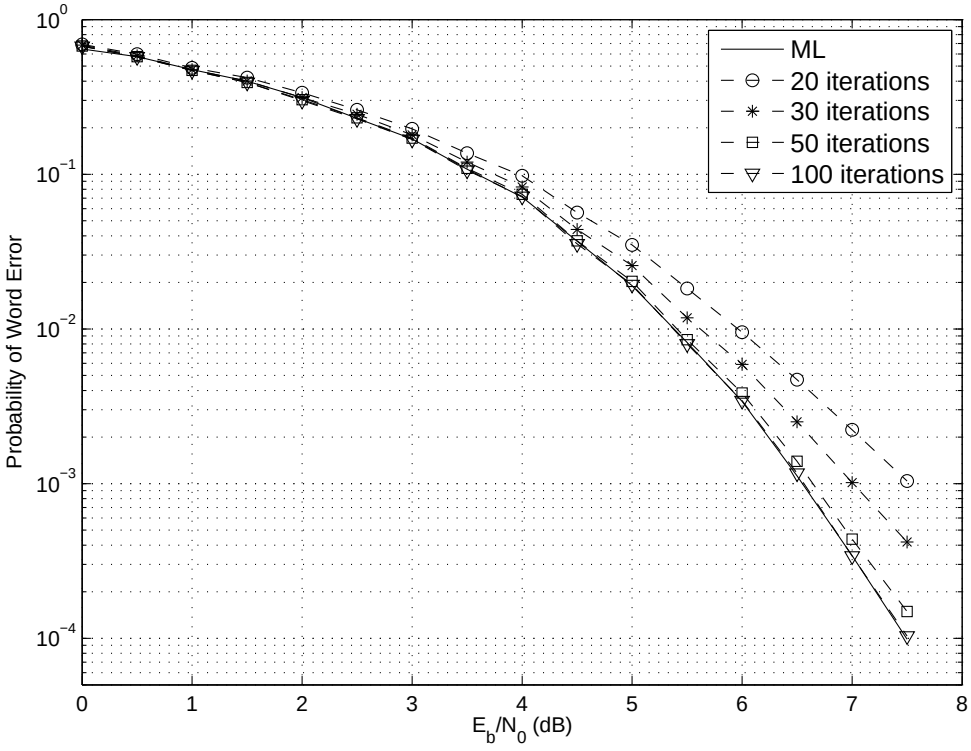


Figure 5.5: P_w for the $[32, 16, 8]$ Reed-Muller code on the BSC.

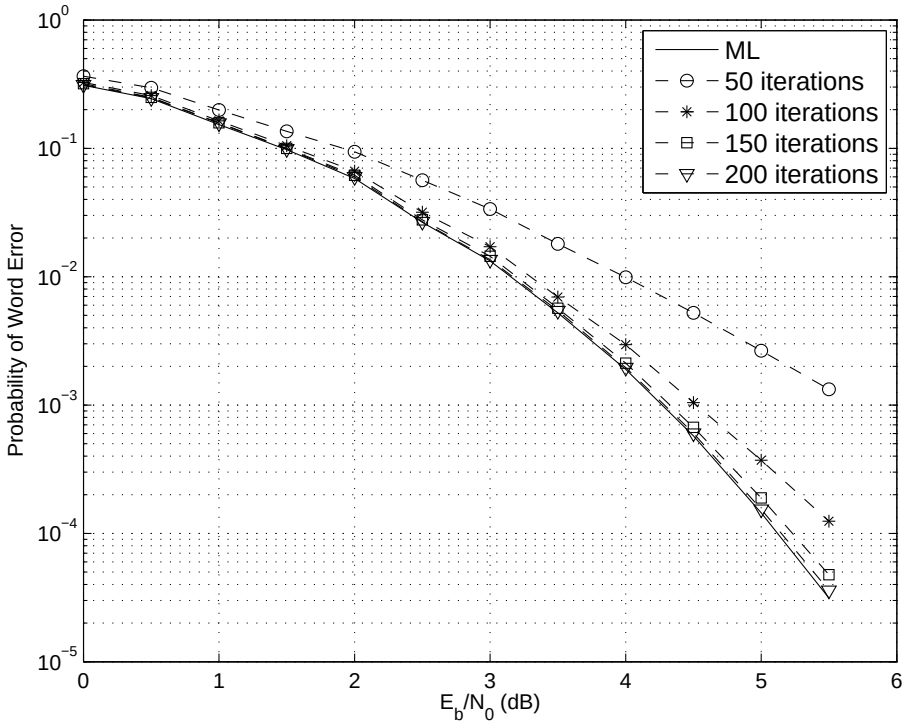
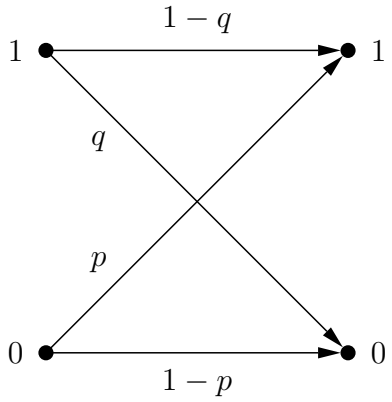


Figure 5.6: P_w for the $[32, 16, 8]$ Reed-Muller code on the AWGN channel.

Recall from Section 2.2.5 that the binary asymmetric channel's conditional probability function can be depicted by the following diagram.



As was noted in Section 2.2.5, the Z-channel is the special case of the binary asym-

metric channel obtained by setting $q = 0$.

Remark 5.6.1. *Initial simulations of the generalized Omura decoder on the binary asymmetric channel resulted in erratic performance curves when specialized to the binary symmetric channel. To overcome this problem, in all simulations on the binary asymmetric channel small amounts of Gaussian noise were added to all components of the log-likelihood vector* $\boldsymbol{\lambda}$ *before its use in both generalized Omura decoding and ML decoding. In all instances, each component of this noise was drawn independently from a Gaussian random variable with mean 0 and standard deviation* $|\lambda_{min}| \cdot 10^{-6}$*, where* λ_{min} *denotes the component of* $\boldsymbol{\lambda}$ *closest to zero in the Euclidean metric. The use of this additional noise corrected the problem, and the degradation in performance incurred by its use, as compared to the performance of the same decoders using the true log-likelihood vector, is negligible for two of our three simulations.*

Elaborating on this, let $\boldsymbol{\lambda}' = \boldsymbol{\lambda} + |\lambda_{min}| \cdot 10^{-6} \cdot \boldsymbol{\eta}$ *with all components of* $\boldsymbol{\eta}$ *being drawn independently from a standard normal distribution, and let* E *be the event in which a codeword that is not an ML codeword under* $\boldsymbol{\lambda}$ *becomes an ML codeword under* $\boldsymbol{\lambda}'$*. For our simulations of the* $[32, 16, 8]$ *Reed-Muller code on the binary asymmetric channel with* $q = P(0 \,|\, 1) \approx 0.0377$ *and with* $q = P(0 \,|\, 1) \approx 0.000783$*, the probability of the event* E *can be upper bounded by* $P\left(\chi^2(32) \geq 12,000\right)$*, the probability of a chi-square random variable with 32 degrees of freedom exceeding 12,000. MATLAB® evaluates* $P\left(\chi^2(32) \geq 12,000\right)$ *as being exactly 0. Thus, for the* $[32, 16, 8]$ *Reed-Muller code on the binary asymmetric channel with* $q \approx 0.0377$ *and* $q \approx 0.000783$*, our simulations of ML decoding, and hence of generalized Omura decoding, should experience very little degradation in performance.*

We were unable to draw a similar conclusion for our simulation of the $[32, 16, 8]$ *Reed-Muller code on the binary asymmetric channel with* $q \approx 0.159$*, though we con-*

jecture that the added noise does not affect decoding performance on this particular channel much either.

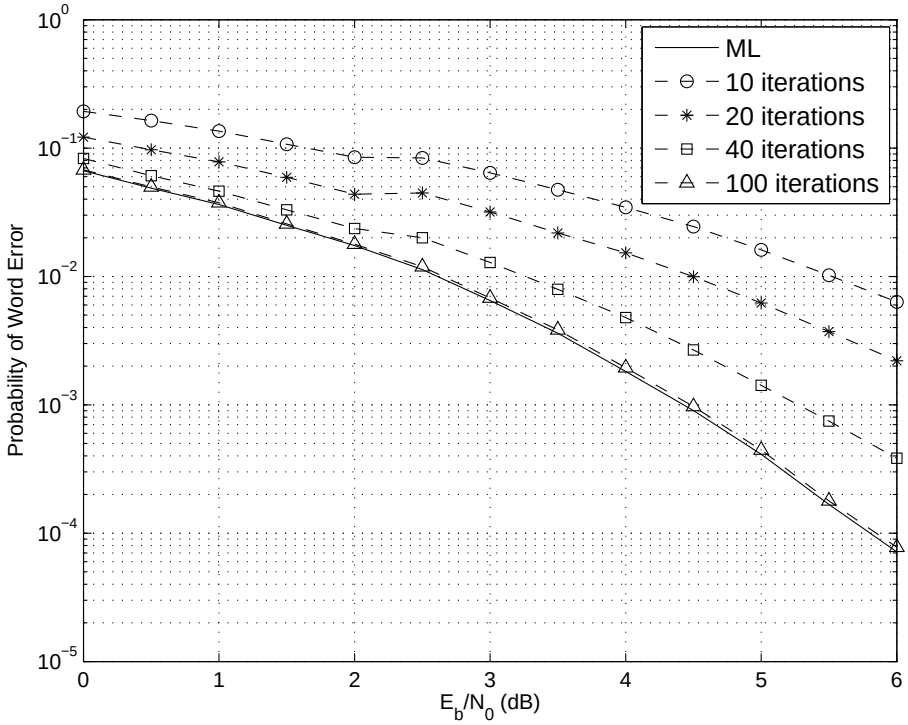


Figure 5.7: P_w for the $[32, 16, 8]$ Reed-Muller code on the binary asymmetric channel with $q := P(0 \mid 1) \approx 0.000783$. The horizontal axis measures $p := P(1 \mid 0)$.

Figure 5.7 shows P_w for the $[32, 16, 8]$ Reed-Muller code on the binary asymmetric channel with $q := P(0 \mid 1)$ fixed at a value of approximately 0.000783. The performance curves for both 10 and 20 iterations fall at the same rate as the ML curve until 2.5 decibels, at which point there is a marked increase in the gap between these two curves and ML. This jump is less pronounced at 40 iterations, and it vanishes outright at 100 iterations, at which point the generalized Omura decoder attains practically optimal performance. This serves as yet another illustration of the eventual convergence predicted by Theorem 5.5.11.

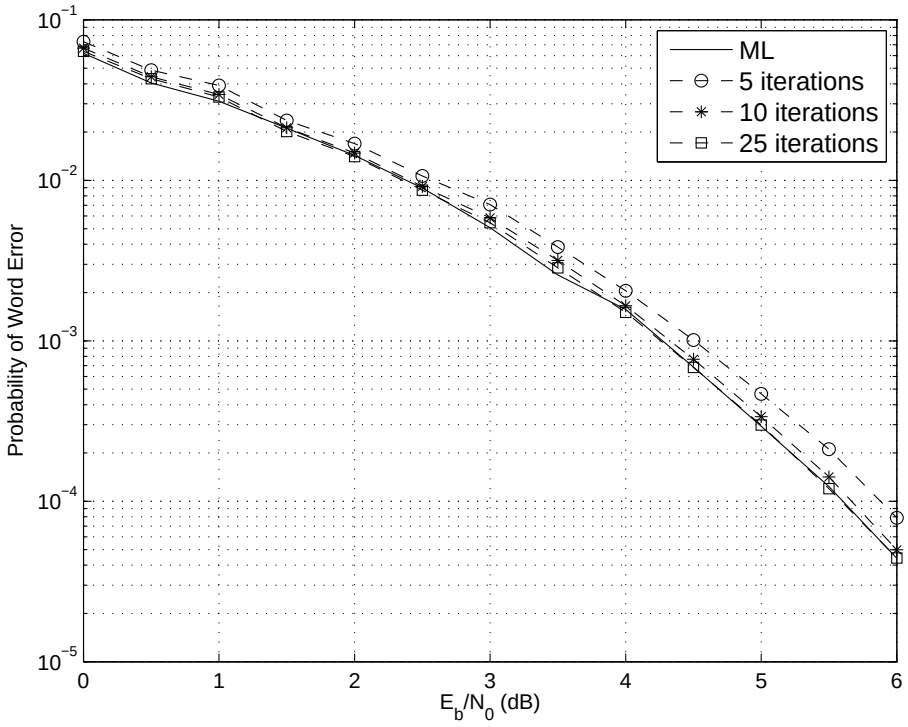


Figure 5.8: P_w for the $[32, 16, 8]$ Reed-Muller code on the Z-channel. The horizontal axis measures $p := P(1 \mid 0)$.

Figure 5.8 displays the results of a simulation on the Z-channel. We note that even though the Z-channel is a special case of the binary asymmetric channel, the generalized Omura decoder must go through additional procedures (i.e., computing a new, smaller parity-check matrix based on the received vector) to account for the partial determinism of the Z-channel. At 25 iterations we see that ML performance is achieved, which is a faster rate of convergence than was observed on the BSC. This is likely due to the fact that, on the Z-channel, Algorithm 2 is actually dealing with parity-check matrices smaller than the original parity-check matrix defining the code.

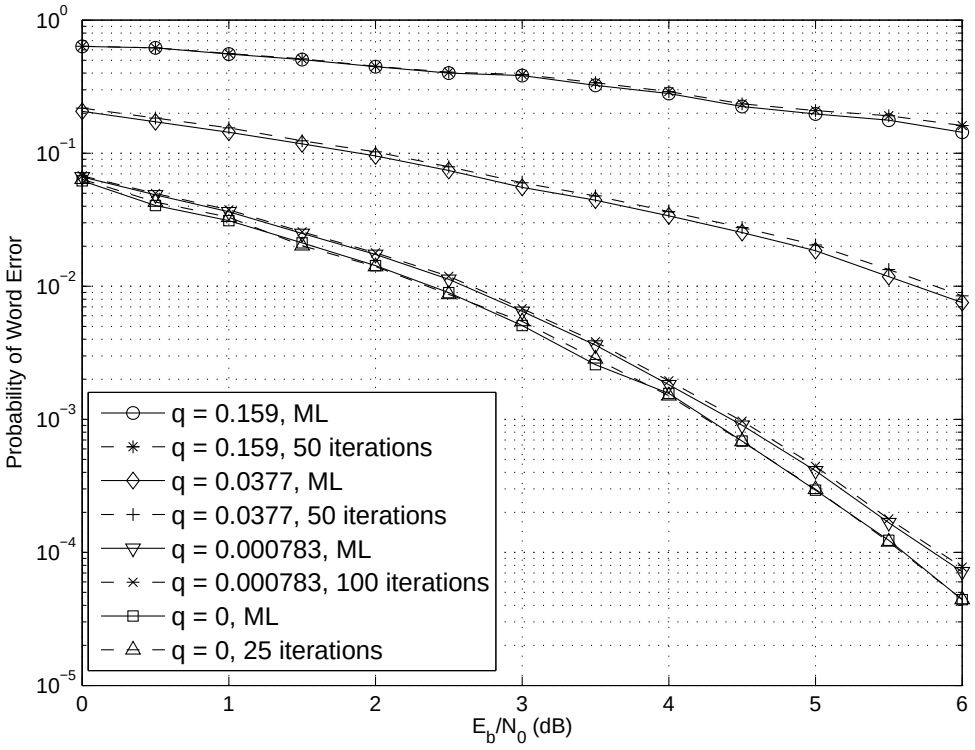


Figure 5.9: A comparison of word-error rates for the $[32, 16, 8]$ Reed-Muller code on the binary asymmetric channel for several values of $q := P(0 \mid 1)$. A value of 0 for q indicates the Z-channel. The horizontal axis measures $p := P(1 \mid 0)$.

We conclude with Figure 5.9, which shows the performance of the $[32, 16, 8]$ Reed-Muller code on the binary asymmetric channel for several values of q, as well as on the Z-channel. Not only do word-error rates drop in tandem with q, but we see that both generalized Omura decoding and ML decoding on the binary asymmetric channel approach the the error rates of the Z-channel with diminishing values of q.

Bibliography

[1] N. Axvig, D. Dreher, K. Morrison, E. Psota, L.C. Pérez, and J.L. Walker. A universal theory of decoding and pseudocodewords. SGER Technical Report 0801, University of Nebraska-Lincoln, July 2008. Available online at `http://www.math.unl.edu/~jwalker7`.

[2] N. Axvig, D. Dreher, K. Morrison, E. Psota, L.C. Pérez, and J.L. Walker. Analysis of connections between pseudocodewords. *IEEE Transactions on Information Theory*, 55(9):4099–4107, September 2009.

[3] N. Axvig, E. Price, E. Psota, D. Turk, L.C. Pérez, and J.L. Walker. A universal theory of pseudocodewords. In *Proceedings of the 45th Annual Allerton Conference on Communication, Control, and Computing*, pages 336–343, September 2007.

[4] E.R. Berlekamp, R.J. McEliece, and H.C.A. van Tilborg. On the inherent intractibility of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–6, May 1978.

[5] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo codes 1. In *Proceedings of the 1993 IEEE International Conference on Communications*, pages 1064–1070, Geneva, Switzerland, 1993.

[6] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[7] M. Breitbach, M. Bossert, R. Lucas, and C. Kempter. Soft-decision decoding of linear block codes as optimization problem. *European Transactions on Telecommunications*, 9(3):289–293, May-June 1998.

[8] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, January 1998.

[9] D. Changyan, D. Proietti, I.E. Telatar, T.J. Richardson, and R.L. Urbanke. Finite length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Transactions on Information Theory*, 48:1570–1579, June 2002.

[10] D. Chase. A class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, 18(1):170–182, January 1972.

[11] G.C. Clark, Jr. and J.B. Cain. *Error-Correction Coding for Digital Communications*. Plenum Press, New York, NY, 1981.

[12] D. Dreher. *Pseudocodewords on Graph Covers and Computation Trees*. PhD thesis, University of Nebraska-Lincoln, 2010.

[13] J. Feldman. *Decoding Error-Correcting Codes via Linear Programming*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[14] M.P.C. Fossorier and S. Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, 41(5):1379–1396, September 1995.

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

[16] G.D. Forney, Jr. Generalized minimum distance decoding. *IEEE Transactions on Information Theory*, 12(2):125–131, April 1966.

[17] W.C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[18] S. Lin and D.J. Costello, Jr. *Error Control Coding*. Pearson Prentice Hall, Upper Saddle River, NJ, second edition, 2004.

[19] S.B. Maurer. Matroid basis graphs. I. *Journal of Combinatorial Theory. Series B*, 14:216–240, 1973.

[20] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. DSN Progress Report 42-44, January and February 1978.

[21] T.K. Moon. *Error Correction Coding*. John Wiley & Sons, Inc., 2005.

[22] J.K. Omura. Iterative decoding of linear codes by a modulo-2 linear program. *Discrete Mathematics*, 3:193–208, 1972.

[23] F.S. Roberts. *Discrete Mathematical Models with Applications to Social, Biological, and Environmental Problems*. Prentice Hall, Inc., 1976.

[24] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[25] P. Vontobel and R. Koetter. Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes. To appear in *IEEE Transactions on Information Theory*.

[26] D.B. West. *Introduction to Graph Theory*. Prentice-Hall Inc., Upper Saddle River, NJ, 2001.